Proceedings of the 34th Canadian Conference on Computational Geometry (CCCG 2022)

August 25-27, 2022
Toronto Metropolitan University*
Toronto, Ontario
Canada

Logo designed by Amin Bahoo.

* Formerly Ryerson University

Sponsors

Faculty of Science

Department of Computer Science
Faculty of Science

Department of Mathematics
Faculty of Science

Preface

This volume contains the papers presented at the
CCCG2022: 34th Canadian Conference on Computational Geometry
held on August 25-27, 2022 at the Toronto Metropolitan University in Toronto, Canada. There were 65 submissions. Each submission was reviewed by 3 program committee members. The committee decided to accept 45 papers. The program also included the following invited talks:

- Bernard Chazelle, Princeton University (Paul Erdős Memorial Lecture); Particle Systems, Geometry, and Dynamic Networks
- Subhash Suri, University of California, Santa Barbara (Godfried Toussaint Memorial Lecture); Obstacle Avoidance
- Jorge Urrutia, Universidad Nacional Autónoma de México (Ferran Hurtado Memorial Lecture); Art Gallery Problems

We gratefully acknowledge financial support from the following sponsors:

- Fields Institute for Research in Mathematical Sciences
- Pacic Institute for the MathematicalSciences (PIMS)
- Atlantic Association for Research in the Mathematical Sciences (AARMS)
- Faculty of Science, Toronto Metropolitan University
- Department of Computer Science, Toronto Metropolitan University
- Department of Mathematics, Toronto Metropolitan University

August 29, 2022
Toronto

Yeganeh Bahoo
Konstantinos Georgiou
CCCG2022 Program Committee co-Chairs

Program Committee

Hugo Akitaya	Tufts University
Yeganeh Bahoo	Ryerson University
Prosenjit Bose	Carleton University
Giordano Da Lozzo	Roma Tre University
David Eppstein	University of California, Irvine
William Evans	The University of British Columbia
Konstantinos Georgiou	Ryerson University
Shahin Kamali	York University
Akitoshi Kawamura	Kyoto University
Anil Maheshwari	Carleton University
Debajyoti Mondal	University of Saskatchewan
Pat Morin	Carleton University
Lucia Moura	University of Ottawa
Wolfgang Mulzer	Freie Universität Berlin
Ian Munro	University of Waterloo
Joseph O'Rourke	Smith College
Denis Pankratov	Concordia University
Maria Saumell	The Czech Academy of Sciences
Thomas Shermer	Simon Fraser University
Shakhar Smorodinsky	Ben Gurion University
Bettina Speckmann	Eindhoven University of Technology
Csaba Toth	California State University Northridge
Ryuhei Uehara	Japan Advanced Institute of Science and Technology
Haitao Wang	University of Utah
Sue Whitesides	University of Victoria

Additional Reviewers

Ackerman, Eyal
Alegría, Carlos
Angelini, Patrizio
Arevalo Loyola, Alma

Bazargani, Saman

D'Angelo, Anthony
Damian, Mirela

Esteban, Guillermo

Fekete, Sándor
Frati, Fabrizio
Fukuzawa, Shion

Giachoudis, Nikolaos
Gonczi, Andrei

Har-Peled, Sariel
Hull, Thomas

Jartoux, Bruno

Kawase, Yasushi
Keil, Mark
Keszegh, Balázs
Khodaee, Amin
Kisfaludi-Bak, Sándor
Klute, Fabian
Ku, Jason S.
Kupavskii, Andrey

Lin, Rebecca
Lynch, Jayson

Masařík, Tomáš
Miltzow, Till
Mohammad Lavasani, Ali

Nakano, Shin-Ichi
Naredla, Anurag Murty
Nishat, Rahnuma Islam

Odak, Saeed
Okamoto, Yoshio
Oliveira, Fabiano
Ophelders, Tim
Otachi, Yota

Parada, Irene

Silveira, Rodrigo
Suzuki, Akira

Tabatabaee, Seyed Ali
Tiwary, Hans Raj
Tuttle, Tyler

Verbeek, Kevin

Wu, Kaiyu

Yamanaka, Katsuhisa

Çağirici, Onur

Local Organization Committee

Yeganeh Bahoo	co-Chair	Toronto Metropolitan University
Konstantinos Georgiou	co-Chair	Toronto Metropolitan University

With many thanks to Toronto Metropolitan University HQPs:

Onur Çağırıcı
Woojin Jang
Christopher Kolios
Sean Leizerovich
Kody Manastyrski
Somnath Kundu
Rahnuma Islam Nishat
Roni Sherman
Harjas Singh
Rudaba Syed
Xin Wang
Pengfei Wu

Table of Contents

Optimally Tracking Labels on an Evolving Tree

Aditya Acharya* David M. Mount†

Abstract

Motivated by the problem of maintaining data structures for a large sets of points that are evolving over the course of time, we consider the problem of maintaining a set of labels assigned to the vertices of a tree. We study the problem in the evolving data framework, where labels continuously change over time due to the action of an agent called the evolver. An algorithm, which can only track these changes by explicitly probing the individual vertices, is tasked with maintaining an approximate sketch of the underlying tree. Such a framework necessitates an algorithm which is fast enough to keep up with the changes, while simultaneously being accurate enough to maintain a close approximation. We present an algorithm that allows for both randomized and adversarial evolution of the data, subject to allowing a constant speedup factor over the evolver. We show that in the limit, it is possible to maintain labels to within an average distance of $O(1)$ of their actual locations. We also present nearly matching lower bounds, both on the distance, and the speed-up factor.

1 Introduction

Many modern data sets are characterized by two qualities: massive size and dynamic variation with time. The combination of size and dynamics makes maintaining them extremely challenging. Algorithms that recompute the structure can be prohibitively expensive, owing to scale of the data set. Standard models for dynamic structures (e.g., [6]) may not be applicable because we may not know where or when changes occur within the structure. These qualities together challenge the traditional single-input/single-output model used in the field of algorithm design.

Anagnostopoulos *et al.* [1] proposed the *evolving data framework* to capture the salient aspects of such data sets. In this framework, the structure varies continuously through the actions of an *evolver*, which makes small, random changes to the structure behind the scenes. Instead of taking a single input and producing a single output, an algorithm judiciously *probes* the current state of the structure and attempts to continuously maintain a view of the structure that is as close as possible to its actual state.

In this paper, we consider the problem of maintaining a tree with n distinct labeled nodes in this framework. The tree topology is assumed to be fixed over time, but the evolver changes label locations by swapping the labels of two adjacent vertices. We consider the problem both in the classical evolving framework, where swaps are chosen uniformly at random, and an adversarial framework, where the evolver's swaps are arbitrary. To probe the structure's current state, we assume the existence of an *oracle*, which given a pair consisting of a label and a vertex, either reports that the label truly resides at this vertex, or it returns an edge incident to the vertex indicating the first edge along the path leading from the probed vertex to the vertex where the label currently resides.

We model our current state by means of a *hypothesized labeling*, that is, a mapping of labels to the vertices. Unlike the actual labeling, the mapping need not be 1–1. Our update algorithm is extremely simple. With each step, it queries a label-vertex pair. If the label is not at this vertex, it moves the label hypothesis one vertex closer to its actual location in the tree. To measure how close our hypothesis is to the truth, we define a *distance function*, which is just the sum of distances over all the labels between their hypothesized and true locations. Note that the evolver moves two labels with each step, while our algorithm moves only one. For this reason we provide our algorithm with a *speedup factor* $c \geq 1$ (not necessarily an integer), which allows our algorithm to perform multiple steps for each single action of the evolver. (Further details are given in Section 2.)

We present four main results. We first show that, even in the most benign case of a uniform random evolver and any constant speedup, the steady-state distance over a bounded degree tree is $\Omega(n)$ (Theorem 3). Second, we show that given a speedup factor of $c = 2$ and a uniform random evolver, there exists a simple algorithm that achieves a steady-state distance of $O(n)$, for any bounded degree tree (Theorem 7). Next, we show that given a speedup factor of $c > 2$, for any evolver, the same simple algorithm achieves a steady-state distance of $O(n)$ (Theorem 9). Finally we show that for any speedup $c < 2$, there exists a tree, and an adversarial evolver, such that the steady state distance is not in $o(n^2)$ (Theorem 10).

*Department of Computer Science, University of Maryland,College Park MD, USA, adach@umd.edu

†Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park MD, USA, mount@umd.edu

1.1 Related Work

The problem we consider here falls under the general category of pebble motion problems. Given a graph $G = \{V, E\}$, a set of labels $L = \{l_1, l_2, \ldots, l_k\}$, a labeling configuration is defined as a mapping $M : L \to V$, such that $M(l_i) \neq M(l_j)$, for $l_i \neq l_j$. A single move of a label l, where $M(l) = v$, can be defined as updating the mapping to $M(l) = u$, where u is a neighbor of v.

Given two such label assignments M_1 and M_2 on a common graph, the problem of deciding whether there is a sequence of moves to transform M_1 to M_2, was first referred to as the *pebble motion problem* by Kornhauser *et al.* [9]. Under the restriction that a label can only be moved to an unmapped neighboring vertex, Goraly and Hassin *et al.* [7] show that the feasibility problem can be decided in linear time. Ratner *et al.* [13] proved that the associated problem of finding the optimal sequence of moves is NP-hard.

A variant of pebble motion that is more closely related to this paper is the problem of *token swapping*. Again we have a graph with n vertices, and there are n distinct labels. A single move involves swapping the labels of two neighboring vertices. It is easy to see that on a simple path, transforming one configuration to another is akin to sorting the path, and therefore such a sequence of swaps can be generated by a variant of bubble sort. Yamanaka *et al.* [14] showed that there exists a polynomial time 2-approximation when the graph is tree. Miltzow *et al.* [11] generalized this to a polynomial time 4-approximation on general graphs. Graf considered a very similar problem of moving objects along a tree by a robot and presents an excellent collection of similar problems [8, Section 6].

Another related line of work involves algorithms for evolving data sets, which was first introduced by Anagnostopoulos *et al.* [1]. In their framework, the input data set is constantly changing through the actions of a random evolving agent, or *evolver*, and an algorithm is tasked with maintaining an output that is close to the one corresponding to the current data. The algorithm can only access the data set through a series of probes, each of which returns some relevant local information. They considered the problem of maintaining a sorted order of points, where the true ranking of points evolves over time. Besa *et al.* [4] gave an optimal algorithm that maintains an approximate ordering with only $O(n)$ inversions. They showed that a repeated run of an $O(n^2)$ time sorting algorithm like the *insertion sort* suffices.

Researchers have considered other problems in the evolving context, including path connectivity, minimum spanning trees [2], shortest paths [16], and page rank [3], among others. A common theme across these papers is the evolution of the list of edges of the graph, either through introducing a new edge, and deleting an existing one, or by changing the ranking of the edge weights.

1.2 A New Framework for Evolving Data

Our framework differs from the standard evolving data framework in few significant aspects. The first involves the behavior of the evolver. An important characteristic of the evolving model introduced in [1] is that the evolver acts randomly, and algorithms in this model exploit the fact that the evolver will occasionally improve matters. In this paper we consider both uniformly random evolvers as well as evolvers that are non-uniform, possibly deterministic, which may act in an adversarial manner.

The second difference is that our structure is more general in that the mapping of labels to vertices need not be 1–1. We think of the structure that the evolver acts on as a "real world" object, which has capacity constraints on the number of labels each vertex can hold. In contrast, we think of our hypothesized labeled point set as a theoretical model of this real-world structure, which is not constrained by real-world limitations. We also provide our algorithm with a constant speed-up factor, to handle cases when each step of the evolver effects a bigger change than that of the algorithm. In compensation for this asymmetry, our algorithms and analyses are much simpler.

The final difference is the nature of the oracle. We can view our problem as a generalization of evolutionary sorting, but where the domain is a tree structure, rather than a linear list. In sorting, the oracle determines whether two objects are out of order, but this is not really meaningful in our tree-based setting. Instead, our oracle provides a directional pointer to the current location of the label.

2 Problem Formulation

In this section we provide the specifics of our evolving token/label swapping problem. We are given a fixed undirected tree $T = (V, E)$ with n vertices and maximum degree k. Each vertex of the tree is assigned a unique label from the set of labels $L = \{l_1, \ldots, l_n\}$, that is, there is a bijective mapping $M_T : L \to V$. At any time, let $\mathcal{T} = \{T, M_T\}$ denote the current "true" labeled tree (see Figure 1(a)).

The *evolver*, denoted $\mathcal{E}$, introduces changes to the labelings. Each time it runs it selects a pair of adjacent vertices in T and swaps their labels. The evolver may either be *random* or *adversarial*. In the former case the pair to be swapped is chosen uniformly at random, and in the latter the adjacent pair can be chosen arbitrarily, deterministically or adversarially. In Figure 1(a) and (b), the evolver swaps labels X and G.

Our algorithm maintains a model of current labeled tree in the form of a structure we call a *hypothesis tree*, denoted $\mathcal{H} = \{T, M_H\}$, where T is the same tree, and $M_H : L \to V$ is a (not necessarily bijective) mapping

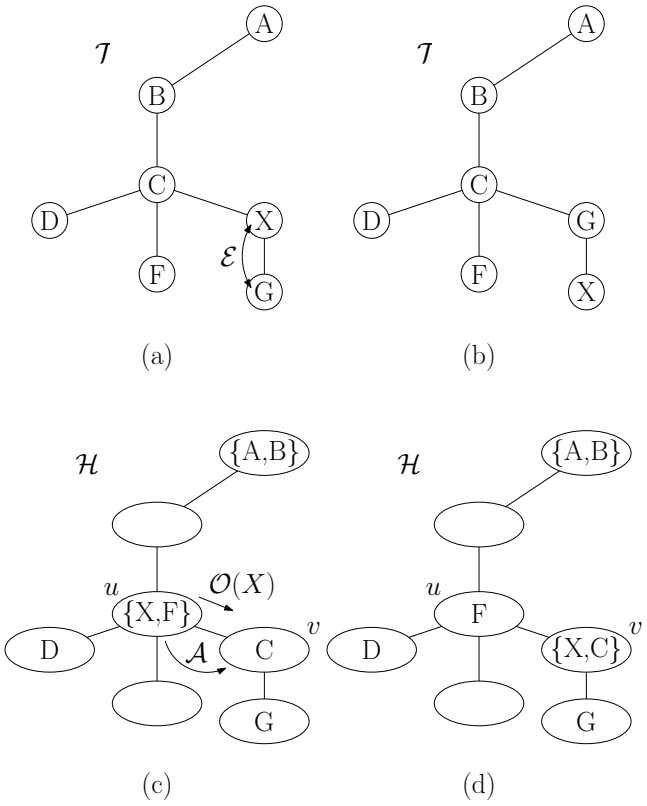


Figure 1: The action of the algorithm on a labeled tree $\mathcal{T}$, evolver $\mathcal{E}$, a labeled hypothesis tree $\mathcal{H}$, and oracle $\mathcal{O}$. (a): The current state of the underlying labeled tree $\mathcal{T}$. (b): The state of $\mathcal{T}$ after the evolver swapped labels across a pair of adjacent nodes. (c): A single step in our algorithm $\mathcal{A}$ on $\mathcal{H}$—Query label X, find that the oracle is pointing us to the location of X on $\mathcal{T}$, and then move the label X to the adjacent node in the returned direction. (d): The final state of our hypothesis tree $\mathcal{H}$ after a single step of $\mathcal{A}$.

from labels to vertices. Note that M_H may assign multiple labels to a vertex of T (see Figure 1(c)).

In order to probe the current actual state, we assume the existence of *oracle*, denoted $\mathcal{O}$. Each query to the oracle is presented in the form of a pair (l_i, u), where l_i is a label and u is a vertex. If l_i is currently located at u, the oracle returns a special value *null*. Otherwise, it returns the edge incident to u that lies on the shortest path from u to $M_T(l_i)$, the vertex that contains l_i in the true labeling. (In Figure 1(c), the query $\mathcal{O}(\mathrm{X}, u)$ returns the edge (u, v) because in the actual tree, the path to the node w containing X contains this edge.)

Each single step of algorithm $\mathcal{A}$ involves the following actions: $\mathcal{A}$ selects a label l and a vertex u. Then queries the oracle to find $\mathcal{O}(l, u)$ and then is free to move the label l from $M_H(l)$ to any adjoining node in the tree. A step of one such algorithm is illustrated in Figure 1(c) and (d), where the algorithm is applied to label X. The query $\mathcal{O}(\mathrm{X}, u)$ returns (u, v), and the algorithm moves label X to v. We define $\mathcal{C}$ as the class of such algorithms, and throughout this paper we only consider algorithms from this class.

To measure how close our hypothesized labeling is to the true labeling we introduce a natural *distance function*. Given two vertices u and v in T, define their distance $d(u, v) = d_T(u, v)$ to be the tree distance, i.e., the length (number of edges) of the path between them. Given the true labeling $\mathcal{T}$ and the hypothesized labeling $\mathcal{H}$ and any label l_i, let $D_i = d(M_T(l_i), M_H(l_i))$ denote the distance between the assigned label positions. Define the overall distance to be $D(\mathcal{T}, \mathcal{H}) = \sum_{l_i \in L} D_i$. *Remark:* $D(\mathcal{T}, \mathcal{H})$ is a metric since it is the sum of tree distances, which are themselves metrics for a particular label.

Observe that with each step the evolver can affect the overall distance by at most 2, moving each of the labels being swapped one node farther from our current hypothesis. Since we have n vertices and the maximum distance between two nodes on the tree is $n-1$, we have $D(\mathcal{T}, \mathcal{H}) \in O(n^2)$. It is easy to see that there exists a tree T and a sequence of swaps by the evolver, which results in $D(\mathcal{T}, \mathcal{H}) \in \Omega(n^2)$. Specifically, consider the case where T is a path and the labels are swapped in a sequence to result in a labeled path with the labels sorted in the opposite order. On the other hand, our algorithm clearly satisfies the following invariant: Every step of an algorithm from class $\mathcal{C}$ reduces the overall distance $D(\mathcal{T}, \mathcal{H})$ by at most 1.

Given the disparity between the evolver's and our algorithm's effect on $D(\mathcal{T}, \mathcal{H})$, we will allow our algorithm a modest *speedup factor*. We denote this by a constant $c \geq 1$. This means that the time taken by a single step of the evolver is c times as that of the algorithm. Or in other words, over a large enough time interval if the algorithm takes m steps, the evolver takes m/c steps.

The problem considered for a given speedup factor c and any arbitrary starting configuration of $\mathcal{H}$: Does there exist an algorithm with this speedup factor such that, in the steady state, after arbitrarily long execution sequences, $D(\mathcal{T}, \mathcal{H}) = o(n^2)$? We will in fact show that (depending on the nature of the evolver) that there exists a deterministic algorithm and an associated speedup factor such that $D(\mathcal{T}, \mathcal{H}) = O(n)$ from the underlying labeled tree, after some sufficiently large time and with high probability.

3 Probabilistic Tools

In this section we mention the probabilistic tools we use through out the paper. First, as a concentration bound, we use a weak version of Chernoff's inequality (Theorem 4.5 in [12]).

Lemma 1 (Chernoff Bound) *Let $X_1, X_2, \ldots, X_n$ be independent random indicator variables, let $X = \sum_i X_i$, and let $\mu = E[X]$. Then,* $\Pr\left[X \leq \frac{\mu}{2}\right] \leq \exp(-\frac{\mu}{8})$.

Next we use a concept called Poisson approximation. Suppose $X_1, X_2, \ldots, X_n$, are the random variables indicating the number of balls in the i^{th} bin, when m balls are thrown into n bins uniformly at random. We call this the exact case.

Let $Y_1, Y_2, \ldots, Y_n$ be independent Poisson random variables with $\Pr[Y_i = k] = e^{-\lambda}\frac{\lambda^k}{k!}$, where $\lambda = m/n$. In other words, Y_i represents the load in a bin, when the number of balls in each of them is a Poisson distribution with parameter λ. We note the following on any event that is a function of the loads of each bin. (Corollary 5.9 [12].)

Lemma 2 (Poisson Approximation) *Any event that takes place with probability p in the Poisson case takes place with probability at most $p\ e\ \sqrt{m}$ in the exact case.*

4 Lower Bounds on the Distance

We first prove a lower bound on $D(\mathcal{T}, \mathcal{H})$, when the maintaining algorithm is in the class $C(\mathcal{A})$ as defined in Section 2 and for any constant speedup factor c. Our proof follows the same structure as a similar proof by Anagnostopoulos *et al.*[1]. We prove the following for $D(\mathcal{T}, \mathcal{H})_{(t)}$, for a sufficiently large t, where $D(\mathcal{T}, \mathcal{H})_{(t)}$ denotes $D(\mathcal{T}, \mathcal{H})$ at time t.

Theorem 3 *For any speedup factor $c \geq 1$ and for all sufficiently large t, irrespective of the algorithm $\mathcal{A}$, $D(\mathcal{T}, \mathcal{H})_{(t)} = \Omega(n)$ with high probability, even in the case of a random evolver.*

Proof. For ease of analysis we let our algorithm $\mathcal{A}$ run a single step every time unit, and the evolver, which runs c times slower perform a swap every c time units. Consider the time interval $[t-n/w, t]$, where w is a large constant. The algorithm and the evolver can reduce $D(\mathcal{T}, \mathcal{H})$ by at most n/w and $2n/cw$ during this time interval, respectively. So if $D(\mathcal{T}, \mathcal{H})_{(t-n/w)}$ was at least $n/w + 2n/cw + \Omega(n)$, then $D(\mathcal{T}, \mathcal{H})_{(t)}$ remains $\Omega(n)$.

Next, let us assume $D(\mathcal{T}, \mathcal{H})_{(t-n/w)}$ is at most $n/w + 2n/cw + o(n)$. That implies there are at most $n/w + 2n/cw + o(n)$ labels displaced from their true location at time $t - n/w$. Let L' denote the set of displaced labels, that is, $L' = \{l_i \mid D_i > 0\}$. We define $V' = \{M_T(l_i) \mid l_i \in L'\}$, as the set of corresponding vertices on T. And then the set of incident edges as $E' = \{(u, v) \mid u \in V' \vee v \in V'\}$. Since the degree of the T is k, we have $|E'| \leq k(n/w + 2n/cw + o(n))$.

In the same time frame, the algorithm $\mathcal{A}$ can act on at most n/w labels. Call that set of labels $L_{\mathcal{A}}$. Define $V_{\mathcal{A}} = \{M_T(l_i) \mid l_i \in l_{\mathcal{A}}\}$, as the set of corresponding vertices on T. And then the set of incident edges as $E_{\mathcal{A}} = \{(u, v) \mid u \in V_{\mathcal{A}} \vee v \in V_{\mathcal{A}}\}$. Now, $|E_{\mathcal{A}}| \leq kn/w$.

Next we look at the set of edges that were unaltered at time, $t-n/w$, and were not affected by the algorithm throughout the time interval. Call it $E^* = E \backslash (E' \cup E_{\mathcal{A}})$. Now, $|E^*| \geq n - 2kn/w - 2nk/cw - k \cdot o(n) \geq n\gamma$, for some sufficiently large w, and $\gamma = (1 - 2k/w - 2k/cw - k/w)$. The evolver picking any edge from E^* exactly once, guarantees that the labels stay swapped at the end of the time interval.

Let X_e be the indicator variable, representing the fact that e is picked by the evolver exactly once. We use the Poisson approximation scheme from Lemma 2. The evolver chooses n/cw edges at random from the n available ones. Therefore $\lambda = (n/cw)/n = 1/cw$, which is a constant. Hence, $\Pr[Y_e = 1] = \lambda e^{-\lambda} = s$, a constant. That implies, $\mathrm{E}[\sum_{e \in E^*} Y_e] \geq s\gamma n$. Using a Chernoff bound (Lemma 1), we have

$$\Pr\left[\sum_{e \in E^*} Y_e \leq s\gamma n/2\right] \leq e^{-\Omega(n)}.$$

Using Lemma 2 again, we have

$$\begin{aligned}
&\Pr\left[\sum_{e \in E^*} X_e \leq s\gamma n/2\right] \\
&\leq e\sqrt{\frac{n}{cw}} \Pr\left[\sum_{e \in E^*} Y_e \leq s\gamma n/2\right] \\
&\leq e\sqrt{\frac{n}{cw}} e^{-\Omega(n)} \leq e^{-\Omega(n)}
\end{aligned}$$

Therefore with exponentially high probability, the evolver picks at least $s\gamma n/2$ edges from E^*, ensuring that those edges stay swapped at the end of the interval. Therefore, $D(\mathcal{T}, \mathcal{H})_{(t)} \geq s\gamma n \in \Omega(n)$, as desired. □

5 Algorithm

Here, we describe a simple algorithm to track the labels. We use the same algorithm in both the cases of a random and an adversarial evolver. Recall the set of labels $L = \{l_1, \ldots, l_n\}$, and the definition of the oracle from Section 2.

Intuitively, the algorithm works as follows. For each $l_i \in L$, we query the oracle on $(l_i, M_H(l_i))$ and update its location by moving it one step in the direction returned by the oracle. We keep doing this until the oracle returns *null*, that is, when l_i is in its true location. We then move on to the next label, repeating the process indefinitely. A single pass over all the labels is called an *iteration* of the algorithm.

Algorithm 1 Tracking Labels

```
/*Continuously run the algorithm*/
for j ← 1, 2, ..., ∞ do
    /*For every label in order*/
    for i ← 1 to n do
        /*Until the label is in its true location*/
        while (O(l_i, M_H(l_i)) ≠ null) do
            /*Query the oracle to find the direction*/
            (u, v) ← O(l_i, M_H(l_i))
            /*Update the location of the label*/
            M_H(l_i) ← v
        end while
    end for
end for
```

6 Analysis

Again for ease of analysis, we let our algorithm $\mathcal{A}$ run a single step every time unit, and the evolver, which runs c times slower, perform a swap every c time units.

Let t_0 be the time when the algorithm starts. Let t_j be the time when the jth iteration of the algorithm ends. Let $D(\mathcal{T}, \mathcal{H})$ at the start of the jth iteration be $D(\mathcal{T}, \mathcal{H})_j$. And for a specific label l_i we denote the distance at the start of the jth iteration to be $D_{i,j}$.

We set the total number of moves effected on l_i, by the algorithm in the j^{th} iteration as $\mathcal{A}_{i,j}$. Therefore the total decrease in D_i, the distance with respect to label l_i, in the j^{th} iteration is $\mathcal{A}_{i,j}$. We define the total decrease in $D(\mathcal{T}, \mathcal{H})$ due to the algorithm, in the j^{th} iteration as $\mathcal{A}_j$, $\mathcal{A}_j = \sum_{l_i \in L} \mathcal{A}_{i,j}$.

We note the following about Δt_j, the time taken by the j^{th} iteration.

Lemma 4 $\Delta t_j = t_j - t_{j-1} = n + \mathcal{A}_j$.

Proof. Every step of the algorithm either moves a label in the direction of its true location, or fixes it, i.e., finds the label is in its true location. Since there are n labels, and $\mathcal{A}_j$ is the total moves effected by the algorithm, we have the result □

Next we show a lower bound for the time taken by the j^{th} iteration.

Lemma 5 $\Delta t_j \geq \frac{c}{2+c}(D(\mathcal{T}, \mathcal{H})_j + n)$.

Proof. For a specific label l_i, our algorithm reduces its distance by $\mathcal{A}_{i,j}$, then finds that the label is at its true location, and then moves on to the next label. This implies that for some subset of steps taken by the evolver, the distance associated with l_i was reduced by $D_{i,j} - \mathcal{A}_{i,j}$. Otherwise, the algorithm would not have moved on to the next label.

This further implies that in the j^{th} iteration for some subset of its steps, the evolver reduced the overall distance by at least $\sum_i (D_{i,j} - \mathcal{A}_{i,j}) = D(\mathcal{T}, \mathcal{H})_j - \mathcal{A}_j$. That takes the evolver at least $(D(\mathcal{T}, \mathcal{H})_j - \mathcal{A}_j)/2$ steps, or at least $(c/2)(D(\mathcal{T}, \mathcal{H})_j - \mathcal{A}_j)$ time.

Therefore we have $\Delta t_j \geq (c/2)(D(\mathcal{T}, \mathcal{H})_j - \mathcal{A}_j)$. Using Lemma 4, we have $\Delta t_j \geq \frac{c}{2}(D(\mathcal{T}, \mathcal{H})_j - \Delta t_j + n)$. Simplifying the inequality gives us the desired result □

6.1 Random Evolver and Speedup 2

In this section we prove the following: In the case of a random evolver, where the evolver $\mathcal{E}$ picks an edge at random and swaps its labels, an algorithm that runs at least twice as fast as the evolver maintains an optimal distance. Or in other words, we show that for $c \geq 2$, our algorithm ensures $D(\mathcal{T}, \mathcal{H}) \in O(n)$ with high probability. Using Theorem 3, we can conclude that our algorithm is optimal for $c \geq 2$ and a random evolver.

As in [4], we first prove an interesting result about the random evolver. We show that a constant fraction of the steps taken by the random evolver do not increase the overall distance $D(\mathcal{T}, \mathcal{H})$.

Lemma 6 *For $c = 2$ and degree k, there exists a constant ϵ, $0 < \epsilon < 1$, such that for all j, the random evolver does not increase the overall distance in at least $\epsilon \Delta t_j$ steps in the j^{th} iteration, with high probability.*

Proof. From Lemma 4, we know Δt_j is at least n. We look at the first $n/10k$ steps of this particular iteration. The algorithm can process at most $n/10k$ nodes in this time. The number of edges incident on these nodes is at most $n/10$. Let E' denote the set of edges left unaltered by the algorithm in this time interval. Then $|E'| \geq 9n/10$. In the same time period, the evolver picks edges at random from the edge set E, $n/20k$ times with replacement.

For every edge e in E, we set $X_e = 1$, if $e \in E'$, and the evolver picks e, at least twice in the time-frame, but picks none of the edges incident on e.

We use the Poisson approximation scheme from Lemma 2. The evolver chooses $n/20k$ edges from the n available ones. Therefore $\lambda = (n/20k)/n = 1/20k$, which is a constant. Now let Y_e be the independent Poisson approximations of X_e, with $\lambda = 1/20k$.

Next we find $\Pr[Y_e = 1]$. That represents the event when e is picked from E', and e is picked twice but none of the edges incident on e are picked. In the Poisson approximation scenario, each edge is picked j times with a probability $e^{-\lambda}\lambda^j/j!$. Therefore, the probability that an edge is picked at least twice is $(1 - e^{-\lambda} - \lambda e^{-\lambda})$, and the probability that it is not picked whatsoever is $e^{-\lambda}$. Since at most $2k$ edges can be incident on e, we have

$$\Pr[Y_e = 1] \;\geq\; \frac{9}{10}\left(1 - e^{-\lambda} - \lambda e^{-\lambda}\right) e^{-2k\lambda}.$$

Since the right hand side is a constant, there exists $s = O(1)$ such that $\Pr[Y_e = 1] \geq s$. Therefore,

$\mathrm{E}[\sum_{e\in E} Y_e] \geq sn$. Using a Chernoff bound (Lemma 1), we have

$$\Pr\left[\sum_{e\in E} Y_e \leq sn/2\right] \;\leq\; e^{-\Omega(n)}.$$

Using Lemma 2 again, we have

$$\begin{aligned}\Pr\left[\sum_{e\in E} X_e \leq sn/2\right] &\leq\; e\sqrt{\frac{n}{20k}}\Pr\left[\sum_{e\in E} Y_e \leq sn/2\right]\\ &\leq\; e\sqrt{\frac{n}{20k}}e^{-\Omega(n)} \;\leq\; e^{-\Omega(n)}.\end{aligned}$$

We note that if $X_e = 1$, then e is left unaltered by the algorithm, but it is altered at least twice by the evolver. That further means, one of those steps by the evolver either decreases the overall distance $D(\mathcal{T},\mathcal{H})$ or leaves it unchanged. And since the number of such edges e, with $X_e = 1$, is at least $sn/2$ with exponentially high probability, we conclude that in at least $sn/2$ of the evolver steps, in the first $n/10k$ steps of the iteration, $D(\mathcal{T},\mathcal{H})$ does not increase. Dividing the iteration into chunks of $n/10k$ steps, we obtain the desired result. □

Finally we prove one of the main theorems of this paper, that for a long enough passage of time, $D(\mathcal{T},\mathcal{H})$ converges to $O(n)$, in the case of $c = 2$, and a random evolver.

Theorem 7 *Given a tree of size n and a constant degree, and a random evolver, there exists z (a function of n) such that for all $j > z$, Algorithm 1 achieves $D(\mathcal{T},\mathcal{H})_j \in O(n)$, with a speed-up factor $c = 2$.*

Proof. Consider the j^{th} iteration. From Lemma 6, the evolver increases $D(\mathcal{T},\mathcal{H})$ by at most $(1-\epsilon)\Delta t_j$. In the same iteration the algorithm reduces $D(\mathcal{T},\mathcal{H})$ by $\mathcal{A}_j$. Therefore, with high probability:

$$\begin{aligned}&D(\mathcal{T},\mathcal{H})_{j+1}\\ &\leq\; D(\mathcal{T},\mathcal{H})_j + (1-\epsilon)\Delta t_j - \mathcal{A}_j\\ &\leq\; D(\mathcal{T},\mathcal{H})_j + n - \epsilon\Delta t_j && \text{[Lemma 4]}\\ &\leq\; \left(1-\frac{\epsilon}{2}\right)D(\mathcal{T},\mathcal{H})_j + \left(1-\frac{\epsilon}{2}\right)n\\ & && \text{[$c=2$ in Lemma 5]}\\ &=\; \left(1-\frac{\epsilon}{2}\right)^j D(\mathcal{T},\mathcal{H})_0 + \sum_{s=1}^{j}\left(1-\frac{\epsilon}{2}\right)^j n\\ &\leq\; \left(1-\frac{\epsilon}{2}\right)^j n^2 + O(n). && \text{[since } D(\mathcal{T},\mathcal{H}) \leq n^2\text{]}\end{aligned}$$

By choosing $z = \log_{1/(1-\epsilon/2)} n$, we have $D(\mathcal{T},\mathcal{H})_{z+1} \in O(n)$. □

Remark: We showed that for large enough j, $D(\mathcal{T},\mathcal{H})_j \in O(n)$. Can we conclude the same about $D(\mathcal{T},\mathcal{H})$ throughout the j^{th} iteration as well? In particular we look at $D(\mathcal{T},\mathcal{H})_{i,j}$. We note that in our Algorithm 1 we could have started with processing the label l_i first (instead of l_1), l_{i+1} next, and so on. Therefore for a large enough j, $D(\mathcal{T},\mathcal{H})_{i,j} \in O(n)$ as well. Since $D(\mathcal{T},\mathcal{H})_{i+1,j} \leq D(\mathcal{T},\mathcal{H})_{i,j} + O(n)$, we conclude that for a large enough passage of time $D(\mathcal{T},\mathcal{H})$ converges to $O(n)$.

In our labeled hypothesis tree $\mathcal{H}$ multiple labels could reside at a particular node. We show a simple result on the maximum number of labels that could be mapped to single vertex in T.

Corollary 8 *Let $L_{\mathcal{H},v}$ be the set of labels residing at a node v in $\mathcal{H}$, after a long enough passage of time. Then, $|L_{\mathcal{H},v}| \in O(\sqrt{n})$*

Proof. Let $|L_{\mathcal{H},v}| = w$. For l_i's, $l_i \in L_{\mathcal{H},v}$, we consider the corresponding distances D_i's. Consider that set as D_v, $D_v = \{D_i | l_i \in L_{H,v}\}$. Since the tree has degree has k, there can be at most k 1's in D_v, similarly k number of 2's, and so on. At most one member of D_v can be zero. Therefore

$$D(\mathcal{T},\mathcal{H}) \;\geq\; \sum_{x\in D_v} x \;\geq\; k(1{+}2{+}\cdots{+}(w{-}1)/k) \in \Omega(w^2).$$

Since $D(\mathcal{T},\mathcal{H}) \in O(n)$ after a long enough time from Theorem 7, we conclude $w \in O(\sqrt{n})$. □

6.2 Adversarial Evolver and Speedup > 2

We conclude with the case when the evolver is adversarial. That means we cannot rely on a result similar to Lemma 6. We show that for a speedup factor of $c > 2$, or in other words, if there exists $\delta \in \mathbb{R}$, such that $c = 2 + \delta$, we can still maintain an optimal distance.

Theorem 9 *Given a tree of size n, an adversarial evolver, there exists z (a function of n) such that for all $j > z$, Algorithm 1 achieves $D(\mathcal{T},\mathcal{H})_j \in O(n)$, with any speed-up factor $c > 2$.*

Proof. Consider the j^{th} iteration. The evolver increases $D(\mathcal{T},\mathcal{H})$ by at most $\frac{2\Delta t_j}{c}$. Therefore

$$\begin{aligned}&D(\mathcal{T},\mathcal{H})_{j+1}\\ &\leq D(\mathcal{T},\mathcal{H})_j + \frac{2\Delta t_j}{c} - \mathcal{A}_j\\ &=\; D(\mathcal{T},\mathcal{H})_j + \frac{2\Delta t_j}{c} + n - \Delta t_j && \text{[Lemma 4]}\\ &=\; D(\mathcal{T},\mathcal{H})_j + n - \left(1-\frac{2}{c}\right)\Delta t_j.\end{aligned}$$

By applying Lemma 5, we have

$$\begin{aligned}
&D(\mathcal{T},\mathcal{H})_{j+1}\\
&\leq\ D(\mathcal{T},\mathcal{H})_j + n - \frac{c-2}{2+c}\Big(D(\mathcal{T},\mathcal{H})_j + n\Big) && \text{[Lemma 5]}\\
&=\ \frac{4}{2+c}D(\mathcal{T},\mathcal{H})_j + \frac{4}{2+c}n\\
&\leq\ \left(\frac{4}{2+c}\right)^j D(\mathcal{T},\mathcal{H})_0 + \sum_{s=1}^{j}\left(\frac{4}{2+c}\right)^j n\\
&\leq\ \left(\frac{4}{2+c}\right)^j n^2 + O(n). && [c>2,\ D(\mathcal{T},\mathcal{H})\leq n^2]
\end{aligned}$$

For $c > 2$, and by choosing $j > \log_{\frac{c+2}{4}} n$, we have $D(\mathcal{T},\mathcal{H})_{j+1} = O(n)$. □

6.3 Adversarial Evolver and Speedup < 2

We adapt a construction from Biniaz *et al.* [5] to prove a lower bound on the required speed-up to ensure $D(\mathcal{T},\mathcal{H})_{(t)} \in O(n)$. Construct two configurations of a labeled tree $\mathcal{T}_0$, and $\mathcal{T}_1$ as in Figure 2. On such a tree: $D(\mathcal{T}_1,\mathcal{T}_0) \sim 2\ OPT$, where OPT is the number of optimum swaps required to go from one configuration to the other. Intuitively, an algorithm running at a speed-up factor less than 2, will fail to catch up with an adversarial evolver that takes OPT swaps to modify $\mathcal{T}_0$, to $\mathcal{T}_1$. We can show that any algorithm from class $\mathcal{C}$ running with speed-up $2-\delta$, where δ is a small positive constant, cannot achieve $D(\mathcal{T},\mathcal{H}) \in O(n)$. In fact we can prove something stronger:

Theorem 10 (Lower bounds on speed-up) *Given any time instant t_0, there exists a tree T, an adversarial evolver $\mathcal{E}$, and a time instant $t > t_0$ s.t. $D(\mathcal{T},\mathcal{H})_{(t)} \in \Omega(n^2)$, for any algorithm from class $\mathcal{C}$, which runs with a speedup $2-\delta$, where δ is a positive constant*

Proof. Suppose we have access to an algorithm $\mathcal{A}$ from the class $\mathcal{C}$, as defined in Section 2, with a speed-up factor of $c = 2-\delta$, δ is a positive real constant. We show the existence of a tree, and an adversarial evolver, where such a speed-up is not sufficient for $D(\mathcal{T},\mathcal{H}) \in O(n)$.

We adapt a construction from Biniaz *et al.* [5]. See Figure 2. Let $\mathcal{T}_0$ be a uniquely labeled tree, with β wings, α tails, and a central vertex. Each wing contains α nodes. For our purposes, we let $\alpha \in \Omega(n)$. $n = \alpha\beta + \alpha + 1$. Let $\mathcal{T}_1$ be another labeled instance of the same tree, where the labels of the wings, are cyclically permuted. The order of the labels on a wing remains the same, as do other labels of the tree. This gives us $D(\mathcal{T}_1,\mathcal{T}_0) = \beta\alpha(\alpha+1)$.

Biniaz *et al.* [5] show that the optimal number of adjacent swaps to go from $\mathcal{T}_0$ to $\mathcal{T}_1$ is $opt(\alpha,\beta) = (\beta+1)(\alpha(\alpha+1)/2 + 2\alpha)$. Consider a time t_0, where $\mathcal{T}_0$ is the labeled configuration of the tree, with our hypothesis tree $\mathcal{H}_0$ being exact, i.e., $D(\mathcal{T}_0,\mathcal{H}_0) = 0$. Next, consider an adversarial evolver $\mathcal{E}$, which performs $opt(\alpha,\beta)$ number of swaps such that at time $t_1 = t_0 + opt(\alpha,\beta)$, $\mathcal{T}_1$ is the true labeling.

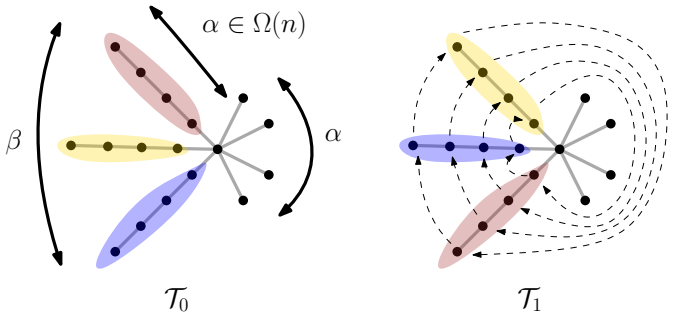


Figure 2: $\mathcal{T}_0$ is a n-node tree with β wings, α tails, and a central vertex. Each wing contains α nodes. $\mathcal{T}_1$ has the labels of the wings of $\mathcal{T}_0$ cyclically permuted. Adapted from [5].

Let $\mathcal{H}$ be the hypothesized labeling at time t_1. Since $\mathcal{A}$ has a speed-up of $2-\delta$, and can affect the distance by at most 1 every step, we have $D(\mathcal{H}_1,\mathcal{T}_0) \leq (2-\delta)\ opt(\alpha,\beta)$. Considering $\beta = 2/\delta$, and $\alpha = \Omega(n)$ we have the following:

$$\begin{aligned}
&D(\mathcal{H}_1,\mathcal{T}_1)\\
&\geq D(\mathcal{T}_1,\mathcal{T}_0) - D(\mathcal{H}_1,\mathcal{T}_0) && [D(\cdot,\cdot) \text{ is a metric}]\\
&\geq \beta\alpha(\alpha+1) - (2-\delta)\ (\beta+1)\left(\frac{\alpha(\alpha+1)}{2} + 2\alpha\right)\\
&\geq \left(\frac{2}{\delta} - \frac{(2-\delta)(2+\delta)}{2\delta}\right)\alpha(\alpha+1) - s(\delta)\alpha\\
& && \text{[Set } \beta = \tfrac{2}{\delta},\ s(\delta) \text{ is a constant]}\\
&\geq \frac{\delta}{2}\alpha(\alpha+1) - s(\delta)\alpha\ \in\ \Omega(n^2).\\
& && \text{[For } \alpha \in \Omega(n), \text{ and constant } \delta]
\end{aligned}$$

□

7 Concluding Remarks

In this paper, we have presented an efficient algorithm for tracking vertex labels in a tree in the evolving data framework. Our algorithm allows for both randomized and adversarial evolution of the data, subject to allowing a constant speedup factor over the evolver. Our analysis showed that in the limit, it is possible to maintain labels to within an average distance of $O(1)$ of their actual locations. We also presented nearly matching lower bounds, both on the distance and the speed-up factor.

This raises the question whether the evolving data framework can be fruitfully applied to tracking the movement of objects through more complex spaces and structures. Applications include real-time tracking of moving agents through GPS tracking of unmanned aerial vehicles [15] and tracking disease hot-spots that evolve over the course of time [10].

We would like to thank Michael Goodrich for introducing us to the evolving data framework and for inspiring discussions on this topic.

References

[1] Aris Anagnostopoulos, Ravi Kumar, Mohammad Mahdian, and Eli Upfal. Sorting and selection on dynamic data. *Theoretical Computer Science*, 412(24):2564–2576, 2011.

[2] Aris Anagnostopoulos, Ravi Kumar, Mohammad Mahdian, Eli Upfal, and Fabio Vandin. Algorithms on evolving graphs. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 149–160, New York, NY, USA, 2012. Association for Computing Machinery.

[3] Bahman Bahmani, Ravi Kumar, Mohammad Mahdian, and Eli Upfal. Pagerank on an evolving graph. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '12, page 24–32, New York, NY, USA, 2012. Association for Computing Machinery.

[4] Juan Jose Besa, William E. Devanny, David Eppstein, Michael T. Goodrich, and Timothy Johnson. Optimally Sorting Evolving Data. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107, pages 81:1–81:13, 2018.

[5] Ahmad Biniaz, Kshitij Jain, Anna Lubiw, Zuzana Masárová, Tillmann Miltzow, Debajyoti Mondal, Anurag Murty Naredla, Josef Tkadlec, and Alexi Turcotte. Token swapping on trees. *arXiv preprint arXiv:1903.06981*, 2019.

[6] Camil Demetrescu, David Eppstein, Zvi Galil, and Giuseppe F. Italiano. *Dynamic Graph Algorithms*, page 9. Chapman & Hall/CRC, 2 edition, 2010.

[7] Gilad Goraly and Refael Hassin. Multi-color pebble motion on graphs. *Algorithmica*, 58(3):610–636, nov 2010.

[8] Daniel Graf. How to sort by walking and swapping on paths and trees. *Algorithmica*, 78(4):1151–1181, aug 2017.

[9] D. Kornhauser, G. Miller, and P. Spirakis. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *25th Annual Symposium on Foundations of Computer Science*, pages 241–250, 1984.

[10] Kanglin Liu, Changchun Liu, Xi Xiang, and Zhili Tian. Testing facility location and dynamic capacity planning for pandemics with demand uncertainty. *European Journal of Operational Research*, 2021.

[11] Tillmann Miltzow, Lothar Narins, Yoshio Okamoto, Günter Rote, Antonis Thomas, and Takeaki Uno. Approximation and Hardness of Token Swapping. In *24th Annual European Symposium on Algorithms (ESA 2016)*, volume 57, pages 66:1–66:15, 2016.

[12] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis. Cambridge University Press, 2017.

[13] Daniel Ratner and Manfred Warmuth. The $(n^2 - 1)$-puzzle and related relocation problems. *Journal of Symbolic Computation*, 10(2):111–137, 1990.

[14] Katsuhisa Yamanaka, Erik D. Demaine, Takehiro Ito, Jun Kawahara, Masashi Kiyomi, Yoshio Okamoto, Toshiki Saitoh, Akira Suzuki, Kei Uchizawa, and Takeaki Uno. Swapping labeled tokens on graphs. *Theoretical Computer Science*, 586:81–94, 2015. Fun with Algorithms.

[15] Ugur Zengin and Atilla Dogan. Real-time target tracking for autonomous UAVs in adversarial environments: A gradient search algorithm. *IEEE Transactions on Robotics*, 23(2):294–307, 2007.

[16] Yiming Zou, Gang Zeng, Yuyi Wang, Xingwu Liu, Xiaoming Sun, Jialin Zhang, and Qiang Li. Shortest paths on evolving graphs. In Hien T. Nguyen and Vaclav Snasel, editors, *Computational Social Networks*, pages 1–13, Cham, 2016. Springer International Publishing.

Discretization to Prove the Nonexistence of "Small" Common Unfoldings Between Polyhedra

Elena Arseneva* Erik D. Demaine† Tonan Kamata‡ Ryuhei Uehara‡

Abstract

We show that no < 300-gon is a common unfolding between any two doubly covered triangles whose angles are rationally independent algebraic numbers. Here an unfolding of a polyhedron is a polygon obtained by cutting anywhere on the polyhedron's surface and unfolding it.

1 Introduction

An ***unfolding*** of a polyhedron $\mathcal{Q}$ is a simple polygon obtained from $\mathcal{Q}$ by cutting anywhere on the surface and unfolding it flat. A ***common unfolding*** between two polyhedra $\mathcal{Q}^0$ and $\mathcal{Q}^1$ is a polygon that is an unfolding of $\mathcal{Q}^0$ and of $\mathcal{Q}^1$. It is open whether any pair of Platonic solids have a common unfolding [4] (though $O(1)$ "refoldings" suffice [3]). For other classes of polyhedra, there are some positive results showing common unfoldings [1, 2, 4, 5, 6]. However, there are no results proving nonexistence of common unfoldings. In other words, it is not known whether there is a pair of polyhedra having no common unfolding.

One difficulty in proving the nonexistence of common unfoldings is that we cannot check by a simple exhaustive search whether two polyhedra have a common unfolding. When we unfold a convex polyhedron $\mathcal{Q}$ to a simple polygon $\mathcal{P}$, the cutting lines on the surface form a tree structure spanning all vertices of $\mathcal{Q}$, called the ***cutting tree***. A cutting tree can have vertices and edges anywhere on the surface of $\mathcal{Q}$. Thus there are uncountably many cutting trees, and the number of obtained unfoldings is also uncountable.

We develop a new algorithmic method to prove the nonexistence of common unfoldings, when we bound the number of vertices in the unfolding, between two polyhedra in the class of doubly covered triangles whose angles are rationally independent algebraic numbers.

In Section 2, we define unfolding and the class of polyhedral which we handle in this paper.

In Section 3.1, we show necessary properties of any common unfolding $\mathcal{P}$ between polyhedra $\mathcal{Q}^0$ and $\mathcal{Q}^1$. First, we consider a correspondence between the boundary of $\mathcal{P}$ on $\mathcal{Q}$ when a polyhedron $\mathcal{Q}$ is unfolded to a polygon $\mathcal{P}$. Next, we define automorphism maps on the boundary, which are called ***gluing maps***, induced by two ways of gluing when $\mathcal{P}$ is folded into $\mathcal{Q}^0$ and $\mathcal{Q}^1$. Finally, we focus on sequences of points on the boundary of the polygon, which are called ***spreading sequences*** and have an essential role in common unfoldings.

In Section 3.2, we introduce a form form of common unfoldings. First, we define the standard-form common unfolding using the notion ad sequence. Next, we show that it is sufficient to consider only standard-form common unfoldings for checking the existence of common unfolding. Finally, we show that the number of standard-form common unfoldings is finite for a given number of vertices in the unfolding. Moreover, we give an algorithm to enumerate the candidates of standard-form common unfoldings.

In Section 3.3, we give a necessary condition and an algorithm to decide whether a candidate standard-form common unfolding represented by a sequence of angles is feasible.

We implement these algorithms and show that, for $n < 300$, there is no n-gon that is a common unfolding between any two doubly covered triangles whose angles are algebraic and rationally independent.

2 Preliminaries

We consider the common unfolding between two doubly covered triangles (DCT). DCT is a class of polyhedra made by gluing the corresponding edges of two copies of a triangle; see Figure 1. It can be regarded as a kind

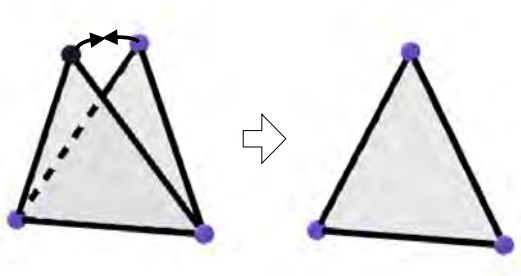


Figure 1: Doubly covered triangle.

of polyhedron whose volume is zero. Let $\mathcal{Q}^0$ and $\mathcal{Q}^1$ be

*St Petersburg State University, ea.arseneva@gmail.com
†CSAIL, MIT, USA, edemaine@mit.edu
‡Japan Advanced Institute of Science and Technology, kamata@jaist.ac.jp, uehara@jaist.ac.jp

a pair of DCTs and vertices of $\mathcal{Q}^i$ be v_0^i, v_1^i, and define the sum of angles gathering at v_i^i by θ^i. I

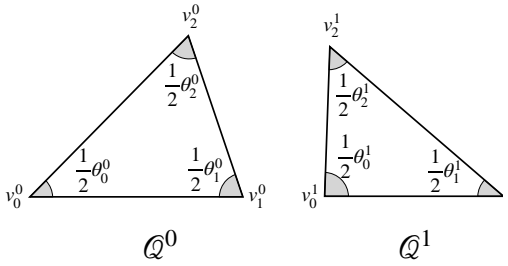


Figure 2: The interior angles of $\mathcal{Q}^0$ and $\mathcal{Q}$

Moreover, we impose the following restrictions on the angles of $\mathcal{Q}^0$ and $\mathcal{Q}^1$.

1. A DCT Q^i is ***algebraic*** if θ_0^i and $\theta_1^i \in \mathbb{Q}^*$ where $\mathbb{Q}^*$ is the algebraic closure on $\mathbb{Q}$. (Here we note that $\theta_2^i = 2\pi - (\theta_0^i + \theta_1^i)$, and $\theta_2^i \notin \mathbb{Q}^*$ if $\theta_0^i, \theta_1^i \in \mathbb{Q}^*$.)

2. A pair of DCTs $\mathcal{Q}^0$ and $\mathcal{Q}^1$ are ***(rationally) independent*** if $\forall m_i (\neq 0) \in \mathbb{Q}$, $m_0\theta_0^0 + m_1\theta_1^0 + m_3\theta_0^1 + m_4\theta_1^1 \neq 0$.

Hereafter, we assume that $\mathcal{Q}^0$ and $\mathcal{Q}^1$ are algebraic and independent. Therefore, each of $\mathcal{Q}^0$ and $\mathcal{Q}^1$ is not an isosceles triangle and has no angle that is a rational multiple of π. Here we note that we introduce these restrictions not to avoid a counterexample but to support the proof technique. We treat θ_j^i as symbols and do not care about the concrete values until Section 3.3. When we consider an assignment of the values of θ_j^i, we use map $\lambda : \{\theta_0^0, \theta_1^0, \theta_2^0, \theta_0^1, \theta_1^1, \theta_2^1\} \to \mathbb{R}_{>0}$.

Example 1 *If* $(\lambda(\theta_0^0), \lambda(\theta_1^0), \lambda(\theta_2^0), \lambda(\theta_0^1), \lambda(\theta_1^1), \lambda(\theta_2^1)) = (\sqrt{2}, \sqrt{3}, 2\pi - \sqrt{2} - \sqrt{3}, \sqrt{5}, \sqrt{7}, 2\pi - \sqrt{5} - \sqrt{7})$, $\mathcal{Q}^0$ *and* $\mathcal{Q}^1$ *are algebraic and independent.*

When we unfold a polyhedron $\mathcal{Q}$ to a polygon $\mathcal{P}$, cutting lines on the surface form a tree structure [4]. We denote it by $\mathcal{T}$. Conversely, points on the boundary of $\mathcal{P}$ are glued and make a point on $\mathcal{T}$ when we fold $\mathcal{P}$ to $\mathcal{Q}$. We call it a ***folding map*** and write it by $f : \partial\mathcal{P} \to \mathcal{T}$ where $\partial\mathcal{P}$ is the boundary of $\mathcal{P}$; see Figure 3.

Let $\mathcal{P}$ be the unfolding of a DCT $\mathcal{Q}$ by $\mathcal{T}$. The topology of $\mathcal{T}$ can be classified into two cases, as illustrated in Figure 4: a ***Y-form*** is a tree with a single point b^i of degree 3 (and with leaves at the vertices of $\mathcal{Q}$), and a ***V-form*** is just a path (through all vertices of $\mathcal{Q}$).

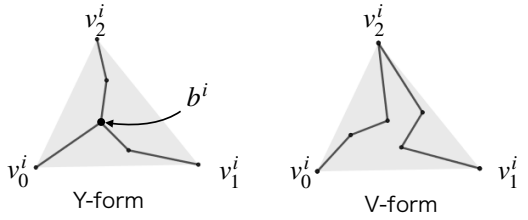


Figure 4: Topologies of cutting trees of doubly covered triangles.

3 Nonexistence of Small Common Unfoldings for $\mathcal{Q}^0$ and $\mathcal{Q}^1$

In this section, we assume there is a polygon $\mathcal{P}$ that is a common unfolding of $\mathcal{Q}^0$ and $\mathcal{Q}^1$ by $\mathcal{T}^0$ and $\mathcal{T}^1$ with folding maps f^0 and f^1.

Hereafter, we consider only the case that both $\mathcal{T}^0$ and $\mathcal{T}^1$ are Y-form. It can be shown that in other cases existence of a common unfolding would contradict our assumption that $\mathcal{Q}^0$ and $\mathcal{Q}^1$ are algebraic and independent (see the proof in Appendix A).

3.1 Gluing Map

On $\partial\mathcal{P}$, there are three points l_0^i, l_1^i, l_2^i corresponding to v_0^i, v_1^i, v_2^i, such as $f^i(l_j^i) = v_j^i$. Moreover, there are three points m_0^i, m_1^i, m_2^i corresponded to b^i, such as $b^i = f^i(m_0^i) = f^i(m_1^i) = f^i(m_2^i)$. We define $L^i := \{l_0^i, l_1^i, l_2^i\}$ and $M^i := \{m_0^i, m_1^i, m_2^i\}$; see Figure 5. Let I_j^i be the intervals on $\partial\mathcal{P}$ between m_j^i, m_{j+1}^i. The following holds.

Observation 2 *For* $p \in \partial\mathcal{P}$, *let* $\alpha(p)$ *be the interior angle at* p.

- $\alpha(l_j^i) = \theta_j^i$.
- $\alpha(m_0^i) + \alpha(m_1^i) + \alpha(m_2^i) = 2\pi$.

Without loss of generality, we can assume that l_j^i and m_j^i appear in counterclockwise order $m_0^i, l_0^i, m_1^i, l_1^i, m_2^i, l_2^i$ around $\partial\mathcal{P}$ for each $i = 0, 1$.

Definition 3 *We define a gluing map* $gl^i : \partial\mathcal{P} \to \partial\mathcal{P}$ *by the map returns the point to which is glued by the mapping as follows.*

- *If* $p \in L^i \cup M^i$, *then* $gl^i(p) := p$.

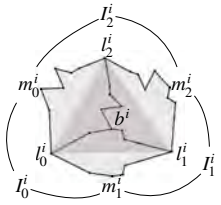


Figure 5: $L^i = \{l_0^i, l_1^i, l_2^i\}$, $M^i = \{m_0^i, m_1^i, m_2^i\}$.

- *Otherwise,* $gl^i(p) := p'$ *such that* $f^i(p) = f^i(p')$; p' *is determined uniquely.*

Observation 4 *Let* $p \notin L^i \cup M^i$. *The following holds:*

- $p \in I_j^i \Rightarrow gl^i(p) \in I_j^i$.
- $p \in I_j^i \Rightarrow l_j^i$ *is the midpoint of* $(p, gl^i(p))$ *on* $\partial\mathcal{P}$.
- $\alpha(p) + \alpha(gl^i(p)) = 2\pi$ *where* $\alpha(p)$ *is the interior angle of* $p \in \partial\mathcal{P}$.

Definition 5 *(spreading sequence* $spr(l_j^i)$*)*
For each $l_j^i \in L^i$, *we define the spreading sequence* $spr(l_j^i)$ *by the sequence of points obtained by alternative iterations of* gl^i *and* gl^{i+1},

$$(l_j^i,\ gl^{i+1}(l_j^i),\ gl^i(gl^{i+1}(gl^i)),\ gl^{i+1}(gl^i(gl^{i+1}(l_j^i))),\ \ldots)$$

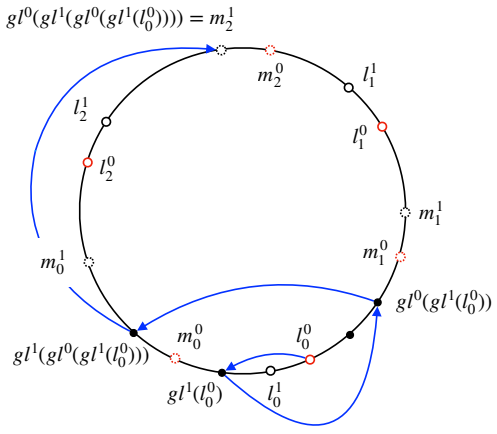


Figure 6: The spreading sequence of l_0^0.

Observation 6 *The interior angles of odd-numbered points of* $spr(l_j^i)$ *are* θ_j^i, *and even-numbered ones are* $\overline{\theta_j^i}$ *where* $\overline{\theta_j^i} := 2\pi - \theta_j^i$.

[1]The superscript indices are taken modulo 2 in this paper. Specifically, gl^{i+1} means gl^0 for $i = 1$ because gl^i defined for $i = 0, 1$.

Lemma 7 *If* $i \neq i'$ *or* $j \neq j'$, $spr(l_j^i)$ *and* $spr(l_{j'}^{i'})$ *share no point.*

Proof. If a point appears in both of $spr(l_j^i), spr(l_{j'}^{i'})$, by Observation 7, $\theta_j^i = \theta_{j'}^{i'}$, $\theta_j^i = \overline{\theta_{j'}^{i'}}$, $\overline{\theta_j^i} = \theta_{j'}^{i'}$, or $\overline{\theta_j^i} = \overline{\theta_{j'}^{i'}}$ holds. In any case, it contradicts the independence of the angles. □

Lemma 8 *For any* $l_j^i \in L^i$, *the length of* $spr(l_j^i)$ *is finite.*

Proof. Because the angles are algebraic and independent, $\theta_j^i \neq \pi$. It means that all points included in some spreading sequence are vertices of $\mathcal{P}$. By the definition of the spreading sequence, a point does not appear twice or more in a spreading sequence. Therefore if there is a spreading sequence whose length is infinite, it produces infinite vertices of $\mathcal{P}$. It is a contradiction. □

Lemma 9 *For any* $l_j^i \in L^i$, *there exists unique* $m_k^{i+1} \in M^{i+1}$ *such that* $spr(l_j^i) = (l_j^i, \ldots, m_k^{i+1})$.

Proof. The endpoint of a spreading sequence belongs to $M^0 \cup M^1 \cup L^0 \cup L^1$. If the endpoint belongs to L^0 or L^1, it contradicts the independence of the angles. Therefore, the endpoints belong to $M^0 \cup M^1$. Inversely, each of $M^0 \cup M^1$ is the endpoint of some spreading sequence because the numbers of $L^0 \cup L^1$ and $M^0 \cup M^1$ are the same. Let us consider the spreading sequences that end at m_0^0, m_1^0, or m_2^0. The sum of the angles of m_0^0, m_1^0, or m_2^0 must be 2π, and it will be realized by only $\theta_0^0 + \theta_1^0 + \theta_2^0$ and $\theta_0^1 + \theta_1^1 + \theta_2^1$ by their independence. (Note that $\overline{\theta_0^i} + \overline{\theta_1^i} + \overline{\theta_2^i} = 6\pi - (\theta_0^i + \theta_1^i + \theta_2^i) = 4\pi \neq 2\pi$.) Therefore, the length of each of the spreading sequences is odd by Observation 6. By considering the parity, we can see that these spreading sequences must start from l_0^1, l_1^1, or l_2^1. □

Lemma 10 *Let* $S_j^i := \{p : p \in spr(l_j^i)\}$.
Then $\bigcup_{i,j} S_j^i$ *divides into* $\partial\mathcal{P}$ *equilateral intervals.*

Proof. Let $d_+(p)$ and $d_-(p)$ be the distance between p and its counterclockwise and clockwise nearest point of $\bigcup_{i,j} S_j^i$ respectively. We prove that $d_+(p)$ and $d_-(p)$ are uniform for any p in $\bigcup_{i,j} S_j^i$. Let $s \in \bigcup_{i,j} S_j^i$ be the clockwise nearest point of m_0^0, and $c := d_-(m_0^0)$; see Figure 7. Let take $l_j^1 \in L^1$ such that $spr(l_j^1) = (l_j^1, \ldots, m_0^0)$. If there is a point $p' \in S_j^1$ such that $d_+(p') = c' < c$ or $d_-(p') = c' < c$, by using Observation 4 inductively, there is a point p'' such that the distance between p'', m_1^0 is c'; see Figure 8. It contradicts that s is the nearest. Therefore, $c = d_+(p) = d_-(p)$ for any point $p \in S_j^1$. Especially, $d_+(m_0^0) = c$. Next, we focus on $d_+(m_1^0), d_-(m_1^0), d_+(m_2^1)$, and $d_-(m_2^1)$. It is easy to see that $d_+(m_0^0) = d_-(m_1^0), d_+(m_1^0) = d_-(m_2^0), d_+(m_2^0) = d_-(m_0^0)$; see Figure 7. Thus, we can check that $c =$

$d_+(p) = d_-(p)$ for any point $p \in \bigcup_j S_j^0$ by repeating the same discussion for m_1^0 and m_2^0. There exist

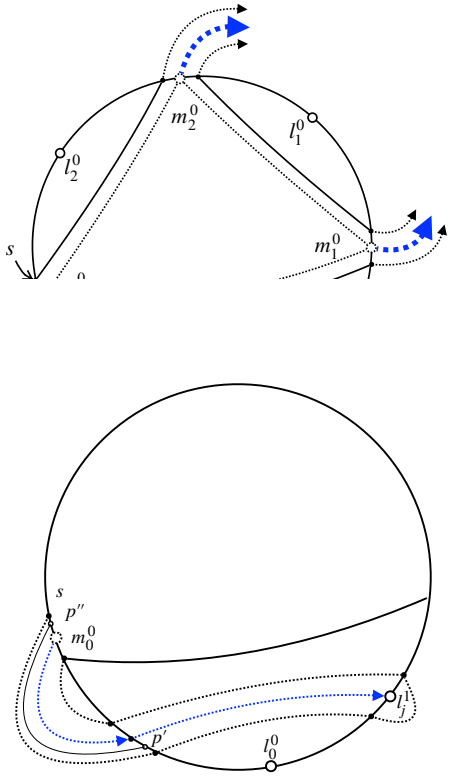


Figure 8: $c = d_+(p) = d_-(p)$ for any point $p \in$

3.2 standard

Definition 11 *If all vertices of $\mathcal{P}$ are included in $\bigcup_{i,j} S_j^i$, we call $\mathcal{P}$ is a standard-form common unfolding.*

Lemma 12 *If $\mathcal{Q}^0$ and $\mathcal{Q}^1$ have a common unfolding, $\mathcal{Q}^0$ and $\mathcal{Q}^1$ have a standard-form common unfolding.*

Proof. By Lemma 10, the points of $\bigcup_j S_j^0$ and $\bigcup_j S_j^1$ are lined up alternately on $\partial\mathcal{P}$. Let take a pair of adjacent points and m be the interval between them. Let $(p_0, p_1, p_2, \ldots, p_k)$ be the vertices of $\mathcal{P}$ on m'. Because m is glued to another interval m', $(p_0, p_1, p_2, \ldots, p_k)$ make vertices $(p'_0, p'_1, p'_2, \ldots, p'_k)$ such that $\alpha(p_i) = 2\pi - \alpha(p'_i)$. In the same way as the proof of Lemma 10, it spreads into all intervals. On the boundary of $\mathcal{P}$ except $\bigcup_{i,j} S_j^i$, the interior angles are $\alpha(p_0), \ldots, \alpha(p_k)$ and $2\pi - \alpha(p_k), \ldots, 2\pi - \alpha(p_0)$ alternately; see Figure 9. We focus on the cutting tree $\mathcal{T}$ into one side polyhedron. Let $\mathcal{T}'$ be the cutting tree replacing each interval of $\mathcal{T}$ with a straight line segment. $\mathcal{T}'$ is kept the interior angles at $\bigcup_{i,j} S_j^i$; see Figure 10. Let $\mathcal{P}'$ be the unfolding by $\mathcal{T}'$. Then $\mathcal{P}'$ is a standard-form common unfolding of $\mathcal{Q}^0$ and $\mathcal{Q}^1$. □

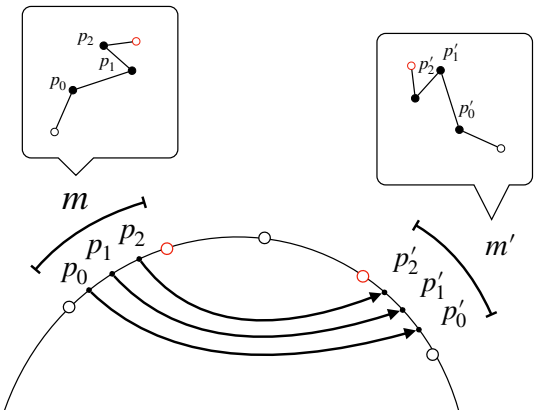


Figure 9: $(p_0, p_1, \ldots, p_m)$ on the interval m.

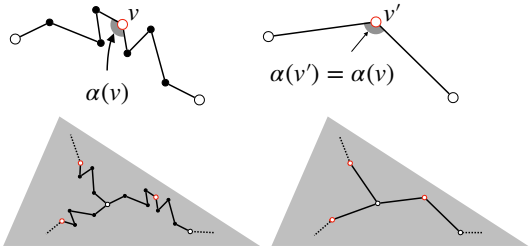


Figure 10: The reduction of a common unfolding into a standard-form common unfolding.

By Lemma 12, if there is no standard-form common unfolding between two polyhedra, there is no common unfolding.Therefore, we can search the common unfolding in the standard-form common unfoldings, whose edges are isometric and vertices are included in $\bigcup_{i,j} S_j^i$. The standard-form common unfoldings are represented by a sequence of interior angles. By fixing n, we can enumerate the sequences of interior angles of length n to be candidates of standard-form common unfolding. Details of the algorithm are given in Algorithm 1. Because the length of each spreading sequence is odd, n should be an integer that is not a multiple of 4 but even. First, we prepare a cyclic array of length n to store the interior angles. Next, we choose six array positions to store the interior angles of l_j^i. It causes $O(n^5)$ combinations. Next, we compute the spreading sequences and determine the interior angles. If distinct angles are assigned to one point, we return to the step of choosing positions of l_j^i. After the placement of l_j^i is determined, the construction of the spreading sequences takes $O(n)$

time because the length of each spreading sequence is at most n. If we obtain a feasible array, we output this one as the candidate of a standard.

Algorithm 1: Enumerating candidate angle squares for standard-form common unfoldings

input : The number of vertices n
output: Sequences of interior angles

Let C be a cyclic array of length n.
$m_0^0 := 0$
forall $m_1^0, m_2^0, m_0^1, m_1^1, m_2^1$ such that $0 = m_0^0 < m_1^0 < m_2^0 < n, 0 < m_0^1 < m_1^1 < m_2^1 < n$ **do**
  for $i = 0, 1$ and $j = 0, 1, 2$ **do**
    if $m_{j+1}^i - m_j^i$ are odd **then**
      Return to line 3.
    end
    $l_j^i := m_j^i + \frac{1}{2}(m_{j+1}^i - m_j^i) \bmod n$
  end
  Define gl^0, gl^1 by Definition 3.
  for i = 0,1 and j = 0, 1, 2 **do**
    $p := l_j^i$
    $k := (j + 1) \bmod 2$
    $C[l_j^i] := \theta_j^i$.
    while $p \neq gl^k(p)$ **do**
      $p := gl^k(p)$
      if $C[p]$ is not yet defined **then**
        if $k = 1$ **then**
          $C[p] := \theta_j^i$
        else
          $C[p] := \overline{\theta_j^i}$
        end
      else
        Return to line 3.
      end
      $k := (k + 1) \bmod 2$
    end
  end
  if $\{C[m_0^i], C[m_1^i], C[m_2^i]\} = \{\theta_0^{i+1}, \theta_1^{i+1}, \theta_2^{i+1}\}$ for each i **then**
    output: C
  end
end

3.3 Checking Polygon Closure

For example, Algorithm 1 outputs the following sequence (see Figure 11):

$$\phi = (\theta_2^1, \theta_2^0, \overline{\theta_2^1}, \theta_2^0, \theta_1^1, \theta_1^0, \theta_0^1, \theta_0^0, \theta_2^1, \overline{\theta_2^0}).$$

It remains to check whether the sequence of interior angles corresponds to a simple polygon. First, we fix the values of θ_j^i by λ like Example 1. We view the polygonal line as lying in the complex plane $\mathbb{C}$. We define an equilateral polygonal line $\mathrm{Poly}_{\phi,\lambda} = (p_0, p_1, \ldots, p_n)$ by the following:

$$p_0 = 1, p_1 = 0 \in \mathbb{C},$$
$$p_{i+1} - p_i = (p_{i-1} - p_i)e^{\sqrt{-1}\,\phi_i}.$$

Here, we remark that $e^{\sqrt{-1}\,\theta} = \cos\theta + \sqrt{-1}\sin\theta$ holds by Euler's Formula. In order to be the common unfolding, $\mathrm{Poly}_{\phi,\lambda}$ must satisfy closure $p_0 = p_n$ and not have self-intersection. We consider only the closure condition of $p_0 = p_n$ because it suffices here to prove the nonexistence of common unfoldings. We can check whether the polygon is closed using the following lemma:

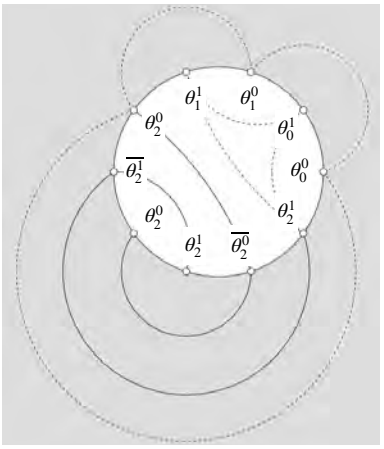


Figure 11: $\phi = (\theta_2^1, \theta_2^0, \overline{\theta_2^1}, \theta_2^0, \theta_1^1, \theta_1^0, \theta_0^1, \theta_0^0, \theta_2^1, \overline{\theta_2^0})$; solid lines represent spreading sequences, and dotted lines connect m_j^i

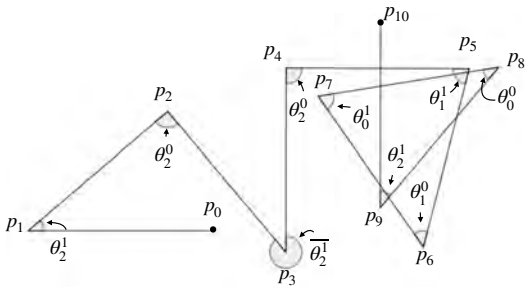


Figure 12: $\mathrm{Poly}_{\phi,\lambda}$ where $\phi = (\theta_2^1, \theta_2^0, \overline{\theta_2^1}, \theta_2^0, \theta_1^1, \theta_1^0, \theta_0^1, \theta_0^0, \theta_2^1, \overline{\theta_2^0})$ and $\lambda\{\theta_j^i\} = (\sqrt{2}, \sqrt{3}, 2\pi - \sqrt{2} - \sqrt{3}, \sqrt{5}, \sqrt{7}, 2\pi - \sqrt{5} - \sqrt{7})$.

Lemma 13 *For a sequence $\phi = (\phi_0, \phi_1, \ldots, \phi_{n-1})$ of the angles θ_j^i or $\overline{\theta_j^i}$ and an angle assignment λ, $\mathrm{Poly}_{\phi,\lambda}$ satisfies $p_0 = p_n$ if and only if the following condition holds:*

(∗) *For each* $0 \le i \le n$*, there exists* j *uniquely such that* $\phi_i + \phi_{i+1} + \cdots + \phi_{j-1} + \phi_j$ *is an integer multiple of* 2π *and* $j - i$ *is odd.*

Proof. Let $\vec{w}_i$ be the vector along the edge (p_i, p_{i+1}). Here, $p_0 = p_n$ is equivalent to $\sum_i \vec{w}_i = 0$. The slope of $\vec{w}_i$ is $\phi_0+\phi_1+\cdots+\phi_i$ or $\phi_0+\phi_1+\cdots+\phi_i-\pi$ depending on whether i is odd or even. Thus, the difference between the slopes of two vectors $\vec{w}_i$ and $\vec{w}_j$ is $\phi_i+\phi_{i+1}+\cdots+\phi_j+\pi$ or $\phi_i + \phi_{i+1} + \cdots + \phi_j$ depending on whether i is odd or even. By the independence of the angles, $\vec{w}_i = -\vec{w}_j$ holds if and only if $j-i$ is odd and $\phi_i+\phi_{i+1}+\cdots+\phi_{j-1}+\phi_j$ is an integer multiple of 2π. It is easy to see that $p_0 = p_n$ if the condition (∗) holds because all vectors are canceled with these inverses. We show $p_0 = p_n$ only if the condition (∗) holds. Let $\vec{w}'_0, \vec{w}'_1, \ldots, \vec{w}'_k$ be the subset of $\vec{w}_0, \vec{w}_1, \ldots, \vec{w}_{n-1}$ choosing without the same or inverse ones. It is sufficient to show $\vec{w}'_0, \vec{w}'_1, \ldots, \vec{w}'_k$ are linearly independent on $\mathbb{Z}$.

We use a classical result on transcendental numbers:

Theorem 14 (Lindemann's Theorem) *For any distinct algebraic numbers* $a_0, a_1, \ldots, a_m$*, the numbers* $e^{a_0}, e^{a_1}, \ldots, e^{a_m}$ *are linearly independent on* $\mathbb{Q}^*$*, where* $\mathbb{Q}^*$ *is the algebraic closure on* $\mathbb{Q}$*.*

Let ψ_i be the slope of $\vec{w}'_i$; $\vec{w}'_i$ is represented by $e^{\sqrt{-1}\,\psi_i}$. Because we choose $\vec{w}'_0, \vec{w}'_1, \ldots, \vec{w}'_k$ without the same or inverse ones, $\psi_0, \ldots, \psi_k$ are distinct algebraic numbers. Similarly, $\sqrt{-1}\,\psi_0, \ldots, \sqrt{-1}\,\psi_k$ are distinct algebraic numbers. By Lindemann's Theorem, $e^{\sqrt{-1}\,\psi_0}, \ldots, e^{\sqrt{-1}\,\psi_k}$ are linearly independent on $\mathbb{Q}^*$. On $\mathbb{Z}$, they are also linearly independent. Therefore, $e^{\sqrt{-1}\,\psi_0} + e^{\sqrt{-1}\,\psi_1} + \cdots + e^{\sqrt{-1}\,\psi_n} = 0$ only when the condition (∗) holds. □

Lemma 15 *Whether the condition* (∗) *holds does not depend on* λ*.*

Proof. From the independence, the sum of angles is an integer multiple of π only if $(\theta_0^0+\theta_1^0+\theta_2^0), (\theta_0^1+\theta_1^1+\theta_2^1)$, or $(\theta_j^i+\overline{\theta_j^i})$. Therefore, whether $\phi_i+\phi_{i+1}+\cdots+\phi_j$ is an integer multiple of 2π or not depends on only whether they can be divided into the above pairs or not. □

For a given ϕ, we check that there exists j such that the condition (∗) is satisfied for each i one by one. It can be done in $O(n^2)$ time.

4 Computational Experiment

By combining Algorithm 1 and the Lemma 15 technique, we can check that, for given n, there is no n-gon that is a common unfolding between any two doubly covered triangles whose angles are algebraic and rationally independent. It requires $O(n^7)$ time theoretically. We implemented them and checked that in a range $n < 300$. It takes 1.5 hours in a normal laptop environment (CPU: 1.4GHz Intel Quad-Core i5, OS: macOS 12.4, Memory: 16GB, compiler: GCC $11.3.0_2$, optimize: -O3).

5 Conclusion

In this paper, we proved the nonexistence of common unfoldings limited in the number of vertices between two elements in a restricted polyhedral class. The main next step is to remove the limitation on the number of vertices. As you can see from the computational experiments, Lemma 13 requires a strong condition to have a common unfolding. This condition seems not to be satisfied by any sequence obtained by Algorithm 1. If we can prove this conjecture, then we will obtain nonexistence without the limitation on the number of vertices. The extension to polyhedra with more than three vertices would also be interesting. In these cases, there are more possible cutting trees to consider, and we would have to consider how to relate restrictions of the interior angles through the spreading sequences.

Acknowledgments

A part of this research is supported by JSPS KAKENHI Grant Numbers JP18H04091, JP20H05961, JP20H05964, JP20K11673, JP22J10261.

References

[1] Yoshiaki Araki, Takashi Horiyama, and Ryuhei Uehara. Common unfolding of regular tetrahedron and Johnson-Zalgaller solid. *Journal of Graph Algorithms and Applications*, Vol. 20, No. 1, pp. 101–114, 2016.

[2] Amartya Shankha Biswas and Erik D. Demaine. Common development of prisms, anti-prisms, tetrahedra, and wedges. pp. 202–207, 2017.

[3] Erik D. Demaine, Martin L. Demaine, Yevhenii Diomidov, Tonan Kamata, Ryuhei Uehara, and Hanyu Alice Zhang. Any regular polyhedron can transform to another by $O(1)$ refoldings. *Proceedings of the 33rd Canadian Conference in Computational Geometry (CCCG 2021)*, Halifax, August 2021.

[4] Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, 2007. OCLC: ocm76416607.

[5] Toshihiro Shirakawa, Takashi Horiyama, and Ryuhei Uehara. On common unfolding of a regular tetrahedron and a cube (in Japanese). *Journal of Science Origami*, Vol. 4, No. 1, pp. 45–54, 2015.

[6] Dawei Xu, Takashi Horiyama, Toshihiro Shirakawa, and Ryuhei Uehara. Common developments of three incongruent boxes of area 30. *Computational Geometry*, Vol. 64, pp. 1–12, August 2017.

A Appendix

Here we consider the case that either or both cutting trees are V-form. We assume that at least $\mathcal{T}^0$ is V-form, and that $\mathcal{T}^0$ cuts v_0^0, v_1^0 by leaves and spans v_3^0 without loss of generality. There are two points d_0^0, d_1^0 in the boundary of $\mathcal{P}$ such that $f^0(d_0^0) = f^0(d_1^0) = v_3^0$. Let $L^0 := \{l_0^0, l_1^0\}, M^0 := \emptyset, D^0 := \{d_0^0, d_1^0\}$. If $\mathcal{T}^1$ is also V-form, we define L^1, M^1, D^1 in the same manner. Otherwise, we let $L^1 := \{l_0^1, l_1^1, l_2^1\}, M^1 := \{m_0^1, m_1^1, m_2^1\}, D^1 := \emptyset$. We modify the definition of the gluing map.

Definition 16 *We define $gl^i : \partial\mathcal{P} \to \partial\mathcal{P}$ as follows.*

- *If $p \in L^i \cup M^i \cup D^i$, $gl^i(p) := p$*
- *Otherwise, $gl^i(p) := p'$ such that $f^i(p) = f^i(p')$; p' is determined uniquely.*

We consider the spreading sequences of each $L^0 \cup L^1$. The endpoints belong to $M^i \cup D^i$ by the definition. In both cases, $|L^0 \cup L^1| = |M^0 \cup M^1 \cup D^0 \cup D^1|$. Thus, each of $M^0 \cup M^1 \cup D^0 \cup D^1$ is the endpoint of some spreading sequence. Therefore, v_3^0 is made by gluing two points that are the endpoints of some spreading sequences. It means that θ_3^0 is represented by $\theta_j^i + \theta_{j'}^{i'}$, $\theta_j^i + \overline{\theta_{j'}^{i'}}$, or $\overline{\theta_j^i} + \overline{\theta_{j'}^{i'}}$. It contradicts the independence of the angles. Therefore, it is sufficient to consider only the case that both are Y-form.

Diamonds are Forever in the Blockchain: Geometric Polyhedral Point-Set Pattern Matching

Gill Barequet* Shion Fukuzawa† Michael T. Goodrich†
David M. Mount‡ Martha C. Osegueda† Evrim Ozel†

Abstract

Motivated by blockchain technology for supply-chain tracing of ethically sourced diamonds, we study geometric polyhedral point-set pattern matching as minimum-width polyhedral annulus problems under translations and rotations. We provide two $(1+\varepsilon)$-approximation schemes under translations with $O(\varepsilon^{-d}n)$-time for d dimensions and $O(n\log\varepsilon^{-1}+\varepsilon^{-2})$-time for two dimensions, and we give an $O(f^{d-1}\varepsilon^{1-2d}n)$-time algorithm when also allowing for rotations, parameterized on f, which we define as the slimness of the point set.

1 Introduction

A notable recent computational geometry application is for tracking supply chains for natural diamonds, for which the industry and customers are strongly motivated to prefer ethically-sourced provenance (*e.g.*, to avoid so-called "blood diamonds"). For example, the ***Tracr*** system employs a blockchain for tracing the supply chain for a diamond from its being mined as a rough diamond to a customer purchasing a polished

Figure 1: Blockchain transactions in a diamond supply chain, providing provenance, traceability, and authenticity of an ethically-sourced diamond.

Essential steps in the Tracr blockchain supply-chain process require methods to match point sets against geometric shapes, *e.g.*, to guarantee that a diamond has not been replaced with one of questionable provenance [23]. Currently, the Tracr system uses standard machine-learning techniques to perform the shape matching steps. We believe, however, that better accuracy can be achieved by using computational geometry approaches. In particular, motivated by the Tracr application, we are interested in this paper in efficient methods for matching point sets against geometric shapes, such as polyhedra. Formalizing this problem, we study the problem of finding the best translation and/or rotation of the boundary of a convex polytope, P (*e.g.*, defining a polished diamond shape), to match a set of n points in a d-dimensional ($d \geq 3$) space, where the point set is a "good" sample of the boundary of a polytope that is purported to be P. Since there may be small inaccuracies in the sampling process, our aim is to compute a ***minimum width*** polyhedral annulus determined by P that contains the sampled points. In the interest of optimizing running time, rather than seeking an exact solution, we seek an approximate solution that deviates from the real solution by a predefined quantity $\varepsilon > 0$.

Related Work. We are not familiar with any previous work on the problems we study in this paper. Nevertheless, there is considerable prior work on the general area of matching a geometric shape to a set of points, especially in the plane. For example, Barequet, Bose, Dickerson, and Goodrich [12] give solutions to several constrained polygon annulus placement problems for offset and scaled polygons including an algorithm for finding the translation for the minimum offset of an m-vertex polygon that contains a set of n points in $O(n\log^2 n + m)$ time. Barequet, Dickerson, and Scharf [13] study the problem of covering a maximum number of n points with an m-vertex polygon (not just its boundary) under translations, rotations, and/or scaling, giving, *e.g.*, an algorithm running in time $O(n^3m^4\log(nm))$ for the general problem. There has also been work on finding a minimum-width annulus for rectangles and squares, *e.g.*, see [9, 11, 19, 21].

Chan [15] presents a $(1+\varepsilon)$-approximation method that finds a minimum-width spherical annulus of n points in d dimensions in $O(n\log(1/\varepsilon)+\varepsilon^{O(1)})$ time, and

*Dept. of Computer Science, Technion—Israel Inst. of Technology, barequet@cs.technion.ac.il
†Dept. of Computer Science, University of California, Irvine, {fukuzaws,goodrich,mosegued,eozel}@uci.edu
‡Dept. of Computer Science, University of Maryland, College Park, mount@umd.edu

Agarwal, Har-Peled, and Varadarajan [1] improve this to $O(n + 1/\varepsilon^{O(d^2)})$ time via coresets [2,3,22,24]. A line of work has considered computing the spherical annulus under stronger assumptions on the points samples. Most notably Devillers and Ramos [17] combine various definitions for "minimum quality assumptions" by Melhorn, Shermer and Yap [20] and Bose and Morin [14] and show that under this assumption the spherical annulus can be computed in linear time for $d = 2$ and present empirical evidence for higher dimensions. Arya, da Fonseca, and Mount [6] show how to find an ε-approximation of the width of n points in $O(n \log(1/\varepsilon) + 1/\varepsilon^{(d-1)/2+\alpha})$ time, for a constant $\alpha > 0$. Bae [10] shows how to find a min-width d-dimensional hypercubic shell in $O(n^{\lfloor d/2 \rfloor} \log^{d-1} n)$ expected time.

Our Results. Given a set of n points in $\mathbf{R}^d$, we provide an $O(\varepsilon^{-d}n)$-time $(1+\varepsilon)$-approximate polytope-matching algorithm under translations, for $d \geq 3$, and $O(n \log \varepsilon^{-1} + \varepsilon^{-2})$ time for $d = 2$, and we provide an $O(f^{d-1}\varepsilon^{1-2d}n)$-time algorithm when also allowing for rotations, where the complexity of the polytope is constant and for rotations is parameterized by f, which we define as the ***slimness*** of the point set.

The paper is organized as follows. In Section 2, we set the ground for this work by providing some necessary definitions. In Section 3, we approximate the MWA under only translations. In this section, we provide a constant factor approximation scheme, a $(1 + \varepsilon)$-approximation scheme and describe how to improve the running time in two dimensions. In Section 4, we consider the MWA under rotations.

2 Preliminaries

Following previous convention [4,5,7,8,18], we say that a point set S is a δ-***uniform sample*** of a surface $\Sigma \subset \mathbb{R}^d$ if for every point $p \in \Sigma$, there exists a point $q \in S$ such that $d(p, q) \leq \delta$. Let $C \subset \mathbb{R}^d$ be a closed, convex polyhedron containing the origin in its interior. Given C, and $x \in \mathbb{R}^d$, define $x + C = \{x + y : y \in C\}$ (the translation of C by x), and for $r \in \mathbb{R}$, define $rC = \{ry : y \in C\}$. A ***placement*** of C is a pair (x, r), where $x \in \mathbb{R}^d$ and $r \in \mathbb{R}^{\geq 0}$, representing the translated and scaled copy $x + rC$. We refer to x and r as the ***center*** and ***radius*** of the placement, respectively. Two placements are ***concentric*** if they share the same center.

Let C be any closed convex body in $\mathbb{R}^d$ containing the origin in its interior. The convex distance function induced by C is the function $d_C : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^{\geq 0}$, where

$$d_C(p, q) = \min\{r : r \geq 0 \text{ and } q \in p + rC\}$$

Thus, the convex distance between p and q is determined by the minimum radius placement of C centered at p that contains q (see Figure 2). When C is centrally symmetric, this defines a metric, but for general C, the function d_C may not be symmetric. We call the original shape C the ***unit ball*** U_C under the distance function d_C. Note that $d_C(a, c) = d_C(a, b) + d_C(b, c)$ when a, b and c are collinear and appear in that order.

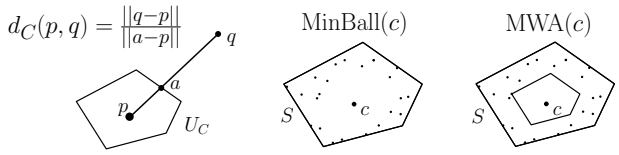


Figure 2: **Left:** a visual representation of a polyhedral distance function and the distance between two points. **Center:** The MinBall under d_C containing all points in S, centered at c. **Right:** The MWA of S with all points within MinBall(c)\MaxBall(c).

Define an ***annulus*** for C to be the set-theoretic difference of two concentric placements $(p + RC) \setminus (p + rC)$, for $0 \leq r \leq R$. The ***width*** of the annulus is $R - r$. Given a δ-uniform sample of points, S, there are three placements of C we are interested in:

- **Minimum enclosing ball (MinBall)**: A placement of C of the smallest radius that contains all of the points in S.
- **Maximum enclosed ball (MaxBall)**: A placement of C of the largest radius, centered within the convex hull of S, that contains no points in S.
- **Minimum width annulus (MWA)**: A placement of an annulus for C of minimum width, that contains all of the points in S.

Note that, following the definition of the MaxBall, we require that the center of the MWA must also lie within the convex hull of S. For each of the above placements, we also refer to parameterized versions, for example MinBall(p), MaxBall(p), or MWA(p). These respectively refer to the minimum enclosing ball, maximum enclosed ball, or minimum width annulus centered at the point p.

Further, we use |MinBall(p)| and |MaxBall(p)| to denote the radius of MinBall(p) and MaxBall(p), respectively, and we use |MWA(p)| to denote the width of MWA(p).

The ratio, F, of the MinBall over the MaxBall of $S \subset \mathbb{R}^d$ under distance function d_C defines the ***fatness*** of S under d_C, such that $F :=$ |MinBall|/|MaxBall|. Also, we define the ***concentric fatness*** as the ratio of the MinBall and MaxBall centered at the MWA, such that $F_c :=$ |MinBall(c_{opt})|/|MaxBall(c_{opt})| where c_{opt} is the center of the MWA. Conversely, we define the ***slimness*** to be $f^{-1} = 1 - F_c^{-1}$, which corresponds to the ratio of the MinBall(c_{opt}) over the MWA, ***i.e.***, $f :=$ |MinBall(c_{opt})|/|MWA|.

Remark 1 *In order for a δ-uniform sample to represent the surface, Σ, with sufficient accuracy for a*

meaningful MWA*, we assume that the sample must contain at least one point between corresponding facets of the* MWA*. Where corresponding facets refer to facets of the* MinBall *and* MaxBall *representing the same facet of U_C. Therefore, in the remainder of the paper, we assume we have a δ-uniform sample and that δ is small enough to guarantee this condition for even the smallest facets.*

In practice, it would be easy to determine a small enough δ before sampling Σ, since only sufficiently slim surfaces would benefit from finding the MWA, and very fat surfaces would yield increasingly noisy MaxBall. One easy approach would be setting δ to the smallest facet of the MinBall and scaling down by an arbitrary constant larger than the maximum expected fatness, such as 100. This example imposes a very generous bound on fatness since it would allow the inner shell to be 1% of the size of the outer shell, practically a single digit constant would often suffice.

Also, note that, for a given center point c, MWA(c) is uniquely defined as the annulus centered at c with inner radius $\min_{p\in S} d_C(c,p)$ and outer radius $\max_{p\in S} d_C(c,p)$. Further, let us assume that the reference polytope defining our polyhedral distance function has m facets, where m is a fixed constant, since the sample size is expected to be much larger than m. Thus, d_C can be calculated in $O(m)$ time; hence, MWA(c) can be found in $O(mn)$ time, which is $O(n)$ under our assumption.

3 Approximating the Minimum Width Annulus

Let us first describe how to find a constant factor approximation of MWA under translations. Note that, by assumption, the center c of our approximation lies within the convex hull of S. Let us denote the center, outer radius, inner radius, and width of the optimal MWA as c_{opt}, R_{opt}, r_{opt}, and w_{opt}.

We begin with Lemma 1, where we prove c_{opt} is within a certain distance from the center of the MinBall c, providing a search region for c_{opt}. In Lemma 2, we bound the width achieved by a center-point that is sufficiently close to c_{opt}. We then use this in Lemma 3 to prove that |MWA(c)| achieves a constant factor approximation.

Lemma 1 *The center of the* MWA*, c_{opt}, is within distance w_{opt} of the center of the* MinBall*, c. That is, $d_C(c, c_{opt}) \le w_{opt}$.*

Proof. Recall our assumption from Remark 1. By our assumption that at least one sample point lies on each facet, MinBall cannot shrink past any facets of MaxBall(c_{opt}).

Suppose for contradiction that $d_C(c, c_{opt}) > w_{opt}$. Let s be the point where a ray projected from c through c_{opt} intersects the boundary of MaxBall(c_{opt}), and let R denote the radius of the MinBall. Observe that R must be large enough for MinBall to contain s and therefore $R \ge d_C(c,s)$.

$$\begin{aligned} R &\ge d_C(c, c_{opt}) + d_C(c_{opt}, s) && \text{by collinearity} \\ &> w_{opt} + d_C(c_{opt}, s) && \text{by assumption} \\ &= w_{opt} + r_{opt} && \text{by MaxBall}(c_{opt}). \end{aligned}$$

Thus, since $w_{opt} + r_{opt} = R_{opt}$, we find $R > R_{opt}$, which is a contradiction since R must be the smallest radius of the MinBall across all possible centers. Therefore, we have that $d_C(c, c_{opt})$ cannot be larger than w_{opt}. □

Lemma 1 helps us constrain the region within which c must be contained. Let us now reason about how a given center point, c, would serve as an approximation. For convenience, let us define $R := |\text{MinBall}(c)|$ and $r := |\text{MaxBall}(c)|$ as the radii of the MinBall and MaxBall centered at c, respectively.

Lemma 2 *Suppose c is an arbitrary center-point in our search region, and the two directed distances between c and c_{opt} are at most t, i.e., $t \ge \max\{d_C(c, c_{opt}), d_C(c_{opt}, c)\}$. Then, we have that $|\text{MWA}(c)| \le w_{opt} + 2t$.*

Proof. Knowing that all sample points must be contained within the MWA, the MWA(c) cannot expand past the furthest or closest point in MWA from c under d_C. Let us now define these two points and use them to bound the radii for MinBall(c) and MaxBall(c).

Let p be the point where the ray from c through c_{opt} intersects the boundary of MinBall(c_{opt}). MinBall(c) cannot extend further than p.

$$\begin{aligned} d_C(c,p) = d_C(c, c_{opt}) + d_C(c_{opt}, p) &\le t + d_C(c_{opt}, p) \\ R &\le R_{opt} + t. \end{aligned}$$

Conversely, let q be the intersection point where the ray projected from c_{opt} through c intersects the boundary of MaxBall(c_{opt}), in which case MaxBall(c) cannot collapse further than q.

$$\begin{aligned} d_C(c,q) = d_C(c_{opt}, q) - d_C(c_{opt}, c) &\ge d_C(c_{opt}, q) - t \\ r &\ge r_{opt} - t. \end{aligned}$$

Combining these bounds with the fact that $|MWA(c)| = R - r$ we find that $|\text{MWA}(c)| \le w_{opt} + 2t$. □

For simplicity, let us consider two points a, b to be ***t-close*** (under C) whenever $t \ge \max\{d_C(a,b), d_C(b,a)\}$.

Lemma 3 *If c is the center of* MinBall*, then* MWA(c) *is a constant factor approximation of the* MWA*, that is, $|\text{MWA}(c)| \le b|\text{MWA}|$, for some constant $b \ge 1$, under translations.*

Proof. From Lemma 1, we have that $d_C(c, c_{opt}) \leq w_{opt}$. If c and c_{opt} are w_{opt}-close, then we can directly apply the second part of Lemma 2 to find $r \geq r_{opt} - w_{opt}$ and $R \leq R_{opt}$, such that $|\text{MWA}(c)| \leq R_{opt} - (r_{opt} - w_{opt})$, thus proving that this is a 2-approximation. If d_C is a metric, then $d_C(c_{opt}, c) = d_C(c, c_{opt})$ and this must always be the case. However, if $d_C(c_{opt}, c) > w_{opt}$, then we must use the Euclidean distance to find $d_C(c_{opt}, c)$. Let vector $u := c - c_{opt}$, and let us define unit vectors with respect to d_C and $d_{\overline{C}}$, such that

$$\widehat{u}_C = \frac{u}{d_C(c_{opt}, c)}, \quad \widehat{u}_{\overline{C}} = \frac{\overline{u}}{d_C(c, c_{opt})}$$
$$||\widehat{u}_C|| \, d_C(c_{opt}, c) \;=\; ||u|| \;=\; ||\widehat{u}_{\overline{C}}|| \, d_C(c, c_{opt})$$
$$d_C(c_{opt}, c) \leq \frac{||\widehat{u}_{\overline{C}}||}{||\widehat{u}_C||} w_{opt} \qquad \text{from Lemma 1.}$$

Under any convex distance function, $\frac{||\widehat{u}_{\overline{C}}||}{||\widehat{u}_C||}$ is bounded from above by $A = \max_{v \in \mathbb{R}^d} \frac{||\widehat{v}_{\overline{C}}||}{||\widehat{v}_C||}$, which corresponds to finding the direction, v, of the largest asymmetry in U_C. Thus, by Lemma 2, $|\text{MWA}(c)| \leq (A+1) w_{opt}$. Under our (fixed) polyhedral distance function, A is constant; hence, MWA(c) is a constant-factor approximation. □

$(1+\varepsilon)$-approximation. Let us now describe how to compute a $(1+\varepsilon)$-approximation of MWA. We begin with Lemma 4, which defines how close to c_{opt} is sufficient for a $(1+\varepsilon)$-approximation. In Theorem 5, we define a grid of candidate center-points so that any point in the search region has a gridpoint sufficiently close to it.

Lemma 4 *Suppose c_{opt} and c are $(\varepsilon w/(2b))$-close, where $w = |\text{MWA}(c_M)|$, c_M is the center of* MinBall*, and b is the constant from Lemma 3. Then,* MWA(c) *is a $(1+\varepsilon)$-approximation of* MWA *under translations.*

Proof. It suffices to show that the width of our approximation only exceeds the optimal width by a factor of at most $(1+\varepsilon)$. Assuming c and c_{opt} are t-close, and using Lemma 2, we require that $w_{opt} + 2t \leq (1+\varepsilon) w_{opt}$, *i.e.*, $t \leq \varepsilon w_{opt}/2$. Let us then choose $t \leq \varepsilon w/(2b)$, knowing that $w \leq b w_{opt}$ from Lemma 3, which is sufficient for achieving a $(1+\varepsilon)$-approximation. □

Knowing how close our approximation's center must be, we can now present a $(1+\varepsilon)$-approximation algorithm to find a center satisfying this condition.

Theorem 5 *One can achieve a $(1+\varepsilon)$-approximation of the* MWA *under translations in $O(\varepsilon^{-d} n)$ time.*

Proof. The MinBall can be computed in $O(n)$ time [16]. By Lemma 1, we have that $d_C(c, c_{opt}) \leq w_{opt}$, where c is the MinBall center. This implies that c_{opt} must lie within the placement $c + w_{opt} C$ or more generously in P, defined as $c + wC$. Furthermore, from Lemma 4, we know that being $(\varepsilon w/(2b))$-close to c_{opt} suffices for an $(1+\varepsilon)$-approximation. Therefore, overlaying a grid G that covers P, such that any point in $p \in P$ is $(\varepsilon w/(2b))$-close to a gridpoint, guarantees the existence of a point $g \in G$ for which MWA(g) is a $(1+\varepsilon)$-approximation.

Since P and $(\varepsilon w/(2b))$-closeness are both defined under d_C, we translate this to a cubic grid for simplicity. Let Q be the smallest cube enclosing P and q be the largest cube enclosed by $(\varepsilon w/(2b))C$. Let us now define a grid, G, to span over Q with cells the size of q. This grid, G, has $O(Fb/\varepsilon)$ gridpoints per direction and $O(F^d b^d \varepsilon^{-d})$ gridpoints in total, where F corresponds to the fatness of C under the cubic distance function.

Let us define the cubic distance function, d_q, with unit cube $U_q = q \cdot (2b)/(\varepsilon w)$, such that U_q is the largest cube enclosed by C. The grid G guarantees that for every point p, there exists a gridpoint $g \in G$ such that $d_q(p, g) \leq \varepsilon w/(2b)$. Since the unit cube is contained within the unit polyhedron, we have that $d_C(a, b) \leq d_q(a, b)\ \forall a, b$; and since d_q defines a metric, p must also be $(\varepsilon w/(2b))$-close under d_C. Finding the gridpoint providing the $(1+\varepsilon)$-approximation takes $O(F^d b^d \varepsilon^{-d} n)$ time,[1] which, under a fixed d_C, is $O(\varepsilon^{-d} n)$ time. □

Faster grid-search in two dimensions. The algorithm of Theorem 5 recalculates the MWA at every gridpoint. However, small movements along the grid should not affect the MWA much. We use this insight to speed up MWA recalculations for two dimensions.

Let us first define the ***contributing edge*** of a sample point, $p \in S$, as the edge of $C + g$ intersected by the ray emanating from a gridpoint, g, towards p. Under this center-point, p will only directly affect the placement of the contributing edge. Observe that given vectors $\overrightarrow{v} \in C$, defined as the vectors directed from the center towards each vertex, the planar subdivision, created by rays for each $\overrightarrow{v}$ originating from g, separates points by their contributing edge. For any two gridpoints, g_1 and g_2, and rays projected from them parallel to $\overrightarrow{v}$, any points within these two rays will contribute to different edges under g_1 and g_2. We denote this region as the ***vertex slab*** of vertex v, and the regions outside of this as ***edge slabs***. Points within an edge slab contribute to the same edge under both gridpoints, maintaining the constraints this imposes on the MWA, can therefore be achieved with the two extreme points per edge slab. If we consider vertex slabs for all $g \in G$, we must be able to quickly calculate the strictest constraints imposed by points in a subset of vertex slabs. An example of the planar subdivision for two points is shown in Figure 3.

[1] For metrics, MinBall provides a 2-approximation, thus b=2. For non-metrics, we can remove this constant by first using this algorithm with ε=1 in order to find a 2-approximation in linear-

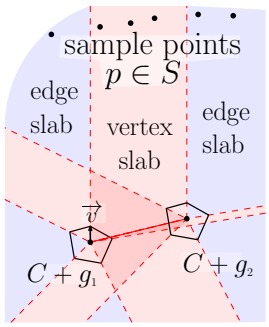


Figure 3: Planar subdivision defining vertex slabs (red) and edge slabs (blue) for two candidate center-points, and showing membership of some sample points.

Given a grid G, we write $g_{i,j} \in G$ to be the gridpoint at index (i, j). Consider the set of all grid lines L_v defined by rays parallel to $\overrightarrow{v}$ starting at each gridpoint. L_v defines a planar subdivision corresponding to the edge slabs between gridpoints. Before attempting to identify the extreme points for each edge slab, we first need to find a quick way to identify the slab in L_v that contains a given sample-point, p.

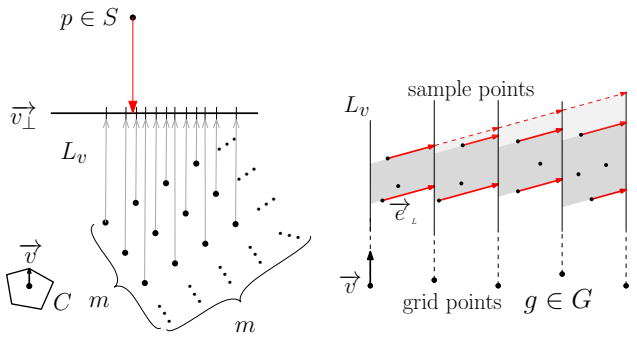
Lemma 6 *For a specific vector $\overrightarrow{v}$ and an $m \times m$ grid, we can identify which slab contains a sample point, p, in $O(\log m)$ time with $O(m^2)$-time preprocessing.*

Proof. Consider the orthogonal projection of grid lines in L_v onto a line $\overrightarrow{v_\perp}$ perpendicular to $\overrightarrow{v}$, the order in which these lines appear in $\overrightarrow{v_\perp}$ defines the possible slabs that could contain p (see Figure 4a). We can project a given grid line $l \in L_v$ onto $\overrightarrow{v_\perp}$ in constant time. With the grid lines in sorted order, we can perform a binary search through the m^2 points in $O(\log m)$ time to identify the slab containing p.

Using general sorting algorithms, we could sort the grid lines in $O(m^2 \log m)$ time. However, since these lines belong to a grid, we can exploit the uniformity to sort them in only $O(m^2)$ time. Consider the two basis vectors defining gridpoint positions $\hat{\imath} = g_{(1,0)} - g_{(0,0)}$ and $\hat{\jmath} = g_{(0,1)} - g_{(0,0)}$, and their sizes after orthogonal projection onto $\overrightarrow{v_\perp}$, $|\hat{\imath}_\perp|$, and $|\hat{\jmath}_\perp|$. Without loss of generality, assume that $|\hat{\imath}_\perp| \geq |\hat{\jmath}_\perp|$, in which case grid lines originating from adjacent gridpoints in the same row must be exactly $|\hat{\imath}_\perp|$ away. In addition, any region $|\hat{\imath}_\perp|$-wide, that does not start at a grid line, must contain at most a single point from each row. Furthermore, since points in the same row are always $|\hat{\imath}_\perp|$ away, they must appear in the same order in each region.

We can therefore initially split $\overrightarrow{v_\perp}$ into regions $|\hat{\imath}_\perp|$ wide. Sorting the grid lines $l \in L_v$ into their region can therefore be calculated in $O(m^2)$ time. Now we can sort the m points in the region containing points from every row in $O(m \log m)$ time. Since each region has the same order, we can place points in other regions by following the order found in our sorted region, thus taking $O(m^2)$ preprocessing time for sorting the points. □

time, and using this approximation for gridding in the main step.



(a) A demonstration of the point location problem with the subdivision, L_v, and a visualization of the gridpoints and sample point projections onto $\overrightarrow{v_\perp}$.

(b) Finding the extreme points (red) under $\overrightarrow{e}_L$ in subdivision L_v for each region (solid) and for all regions to its left (dashed).

Figure 4: A visual representation of the projections involved while point locating within the vertex slabs and while finding the extreme points in each slab.

Recall that points to the left of a given line $l \in L_v$ contribute to the edge to the left of v, *i.e.*, all points belonging to slabs to the left of l. We can therefore isolate the points in these slabs causing the largest potential change in MWA.

Lemma 7 *For a vertex $v \in C$ and grid line $l \in L_v$ through gridpoint g, let l_L and l_R refer to the slabs on the subdivision imposed by L_v immediately to the left and right of l, respectively. Assuming l_L maintains the points to the left of l imposing the strictest constraints on* MWA(g)*, and l_R to the right, one can calculate* MWA(g) *in $O(1)$ time.*

Proof. Finding $\min_{p \in S} d_C(g, p)$ and $\max_{p \in S} d_C(g, p)$ can now be achieved by optimizing only over the set of points in $\{l_L \cup l_R, \forall v \in C\}$ and all points in edge slabs. This set would contain two points per vertex and two points per edge, yielding a constant number of points. Thus, MWA(g) can be found in constant time. □

Theorem 8 *A $(1 + \varepsilon)$-approximation of the* MWA *in two dimensions can be found in $O(n \log \varepsilon^{-1} + \varepsilon^{-2})$ time under translations.*

Proof. For each vertex, v, we use Lemma 6 to identify the slab for every sample point. For each slab, we maintain only the two extreme points for each of the edges incident on $\overrightarrow{v}$. Let $\overrightarrow{e}_L \in C$ denote the vector

describing the edge incident on $\overrightarrow{v}$ from the left, and vice versa for $\overrightarrow{e}_R \in C$ incident from the right. For each slab, we maintain only points which when projected in the relevant direction, $\overrightarrow{e}$, cause the furthest and closest intersections with the boundary (shown for $\overrightarrow{e}_L$ in Figure 4b). With a left-to-right pass, we update a slab's extreme points relative to $\overrightarrow{e}_L$ to maintain the extreme points for itself and slabs to its left. With a right-to-left pass, we do the same for $\overrightarrow{e}_R$ and maintain points in its slab and slabs to its right.

Thus, for each vertex, we create the slabs in $O(\varepsilon^{-2})$ time, place every sample point in its slab in $O(n \log \varepsilon^{-1})$ time, and maintain only the extreme points per slab in constant time per sample point. With $O(\varepsilon^{-2})$ time to update each slab after processing the sample points, we can update the slabs such that they hold the extreme points across all slabs to their left or right (relative to $\overrightarrow{e}_L$ and $\overrightarrow{e}_R$, respectively).

For each edge slab, finding the extreme points is much simpler since finding $\min d_C(g,p)$ and $\max d_C(g,p)$ will always be based on the same contributing facet for all points within the same edge slab .

Thus, after finding the extreme points in both vertex slabs, we can calculate $\mathrm{MWA}(g)$ in constant time as described in Lemma 7. Taking $O(\varepsilon^{-2})$ time to find $\min_{g \in G} \mathrm{MWA}(g)$, which by Theorem 5 provides a $(1+\varepsilon)$-approximation of the minimum width annulus, and considering the $O(n \log \varepsilon^{-1})$ pre-processing time completes the proof of the claimed time bound. □

4 Approximating MWA allowing rotations

In this section we consider rotations. As with Lemma 4, our goal is to find the maximum tolerable rotation sufficient for a $(1+\varepsilon)$-approximation. Observe that when centered about the global optimum, the solution found under both rotation and translation is at least as good as the solution found solely through rotation (*i.e.*, under a fixed center). We will therefore first prove necessary bounds for a $(1+\varepsilon)$-approximation under rotation only with the understanding that they remain when also allowing for translation.

Consider the polyhedral cone around $\overrightarrow{v}$, and define the ***bottleneck angle*** as the narrowest angle between a point on the surface of the polyhedral cone and $\overrightarrow{v}$. Let θ be the smallest bottleneck angle across all $\overrightarrow{v} \in C$. Let $\mathrm{MWA}_\alpha(c)$ denote the MWA centered at c, where C has been rotated by angle α. Let us also use similar notations for MinBall and MaxBall.

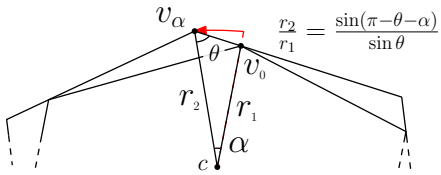
Lemma 9 *Rotating by α causes $\mathrm{MinBall}_\alpha(c)$ to grow by at most $\frac{\sin(\pi-\theta-\alpha)}{\sin\theta}$ (and the reciprocal for $\mathrm{MaxBall}_\alpha(c)$).*

Proof. Similarly to Lemma 2, all sample points must be contained within $\mathrm{MinBall}(c)$. $\mathrm{MinBall}_\alpha(c)$ can only expand to the furthest point within $\mathrm{MinBall}(c)$ under the new rotated distance function. Let us now consider the triangle formed between c, the vertex v of the original MinBall, v_0, and the rotated vertex v_α (shown in Figure 5a). Since our calculations focus towards the same vertex, we can work with Euclidean distances. The quantity $|v_0 - c|$ defines the radius r_1 of the original polyhedron, and $r_2 = |v_\alpha - c|$ the radius of the rotated one. With $\gamma = \pi - \theta - \alpha$ as the remaining angle in our triangle and using the sine rule, we find that

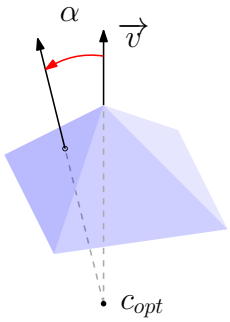
$$\frac{r_2}{r_1} = \frac{\sin\gamma}{\sin\theta} = \frac{\sin(\pi-\theta-\alpha)}{\sin\theta}.$$

Observe that θ is the angle maximizing this scale difference. This applies to rotating by α in any direction about $\overrightarrow{v}$ (as shown in Figure 5b), and since this direction need not coincide with θ, the scaled polyhedron might not touch the original. For $\mathrm{MaxBall}_\alpha(c)$ to be contained within $\mathrm{MaxBall}(c)$, the same example holds after switching references to the scaled and original. In this case, θ minimizes r_1/r_2. □

Let us now determine the rotation from the optimal orientation that achieves a $(1+\varepsilon)$-approximation.



(a) A demonstration of the scale increase necessary for a polyhedron rotated by α to contain the original.



(b) A rotation by α in an arbitrary direction about $\overrightarrow{v}$.

Figure 5: Visual representations for the effect of rotating by α, demonstrating the scale increase and demonstrating how a rotation by α is defined for higher dimensions.

Lemma 10 *Given a center c, we have that* $\mathrm{MWA}_\alpha(c)$ *is a* $(1+\varepsilon)$*-approximation when* α *is smaller than*

$$\arcsin\left(\frac{\sin\theta}{2f}\left(1{+}\varepsilon \pm \sqrt{(1{+}\varepsilon)^2+4f(f-1)}\right)\right)-\theta.$$

Proof. Define f as the ratio of the radius of $\mathrm{MinBall}(c_{opt})$ to w_{opt} (*i.e.*, $fw_{opt} = |\mathrm{MinBall}(c_{opt})|$). Note that f corresponds to the slimness of S under d_C over all rotations of C. Using Lemma 9, we know that

$$|\mathrm{MWA}_\alpha(c)| \le \frac{\sin\gamma}{\sin\theta}|\mathrm{MinBall}(c_{opt})| - \frac{\sin\theta}{\sin\gamma}|\mathrm{MaxBall}(c_{opt})|$$

$$\frac{\sin\gamma}{\sin\theta}fw_{opt} - \frac{\sin\theta}{\sin\gamma}(f{-}1)w_{opt} \le (1{+}\varepsilon)w_{opt} \quad (1)$$

$$\frac{\sin\gamma}{\sin\theta}f - \frac{\sin\theta}{\sin\gamma}(f{-}1) \le (1{+}\varepsilon). \quad (2)$$

For a $(1+\varepsilon)$-approximation, $|\mathrm{MWA}_\alpha(c)| \le (1{+}\varepsilon)w_{opt}$ imposing the right side of Relation (1), its left side follows by definition of f, and Relation (2) by cancellation of w_{opt}. Since θ is constant, we can rearrange the above into a quadratic equation and solve for $\sin\gamma$.

$$\sin\gamma = \frac{\sin\theta}{2f}\left(1{+}\varepsilon \pm \sqrt{(1{+}\varepsilon)^2+4f(f{-}1)}\right). \quad (3)$$

However, arcsin will find $\gamma \le \pi$, whereas we need the obtuse angle $\pi-\gamma$. Thus, proving this lemma's titular bound, and achieving a $(1+\varepsilon)$-approximation. □

Let us now establish a more generous lower-bound that will prove helpful when developing algorithms.

Lemma 11 *The angular deflection required for a* $(1+\varepsilon)$*-approximation is larger than* $\theta\varepsilon/(2f)$*.*

Proof. Observe that γ is of the form $\arcsin(k\sin\theta)$ and thus, in order for $\alpha=\gamma{-}\theta$ to be positive, we must have $\theta < \pi/2$ and $k > 1$. We will prove this is the case.

$$k = \frac{1{+}\varepsilon}{2f} + \sqrt{\left(\frac{1{+}\varepsilon}{2f}\right)^2 - \frac{1}{f} + 1} \quad (4)$$

$$\sqrt{\frac{1}{4f^2} - \frac{1}{f} + 1} = \left|1 - \frac{1}{2f}\right| \quad (5)$$

$$k > \frac{1{+}\varepsilon}{2f} + \left|1 - \frac{1}{2f}\right| = 1 + \frac{\varepsilon}{2f}. \quad (6)$$

Equation (4) follows from Equation (3) after expanding. Relation (6) follows after using Equation (5) as a lower bound for the square root term in Equation (4) since $\varepsilon > 0$ and $f > 1$. This allows us to bound $\arcsin\left(\left(1+\frac{\varepsilon}{2f}\right)\sin\theta\right)$ by using a Taylor's series expansion to find $(1+k)\cdot\theta \le \arcsin((1+k)\sin\theta)$, thus proving that the bound from Lemma 10 is greater than $\frac{\theta\varepsilon}{2f}$. □

Lemma 12 *For fixed rotation of* C*, assume we have an* $O(g(n))$*-time algorithm for the optimal minimum-width annulus under translation. We can find a* $(1+\varepsilon)$*-approximation of the* MWA *under rotations and translations in* $O(f^{d-1}\varepsilon^{1-d}g(n))$ *time.*

Proof. A d-dimensional shape has a $(d{-}1)$-dimensional axis of rotation. Let us evenly divide the unit circle into k directions. Let us also define a collection of all possible direction combinations as a grid of directions. For each grid direction, rotate C by the defined direction and calculate the MWA in $O(g(n))$ time. The optimal orientation must lie between the $(d{-}1)$-dimensional cube formed by 2^{d-1} grid directions. Therefore, as long as the diagonal is smaller than $\frac{\theta\varepsilon}{f}$, there exists a grid direction within $\frac{\theta\varepsilon}{2f}$ of the optimal orientation, which implies a $(1+\varepsilon)$-approximation by Lemma 11. Thus, we can achieve a $(1+\varepsilon)$-approximation in time $O\left(g(n)\cdot\left(\frac{2\pi f\sqrt{d-1}}{\theta\varepsilon}\right)^{d-1}\right)$, where d and θ are constant under a fixed distance function d_C. □

With a fixed center, Lemma 12 can be used to approximate MWA under rotations in $O(nf^{d-1}\varepsilon^{1-d})$ time.

Theorem 13 *One can find a* $(1+\varepsilon)$*-approximation of* MWA *under rotations and translations in* $O(f^{d-1}\varepsilon^{1-2d}n)$ *time for* $d{\ge}3$*, and* $O(fn\varepsilon^{-1}\log\varepsilon^{-1} + f\varepsilon^{-3})$ *time for* $d{=}2$*.*

Proof. Consider using an approximation algorithm (from Theorems 5 or 8) instead of an exact algorithm as in Lemma 12. Let us define $(1{+}\xi)$ as the approximation ratio necessary from the subroutines in order to achieve an overall approximation ratio of $(1+\varepsilon)$, such that $(1{+}\xi)^2 = 1{+}\varepsilon$. Since $\xi = \sqrt{1+\varepsilon}-1$ and $0<\varepsilon<1$, ξ must be larger than $(\sqrt{2}{-}1)\cdot\varepsilon$, and thus, we can always pick a value for ξ which is $O(\varepsilon)$ and achieves the desired approximation. Thus, by following Lemma 12, we can find a $(1+(\sqrt{2}-1)\cdot\varepsilon)$-approximation using the $(1+(\sqrt{2}-1)\cdot\varepsilon)$-approximation algorithm from Theorem 5 to find a $(1+\varepsilon)$-approximation in $O(f^{d-1}\varepsilon^{1-d}\cdot\varepsilon^{-d}n)$ time. Alternatively, for two dimensions, we can instead use the algorithm from Theorem 8 to find a $(1+\varepsilon)$-approximation in $O(fn\varepsilon^{-1}\log\varepsilon^{-1}+f\varepsilon^{-3})$ time. □

References

[1] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51(4):606–635, 2004.

[2] P. K. Agarwal, S. Har-Peled, K. R. Varadarajan, et al. Geometric approximation via coresets. *Combinatorial and Computational Geometry*, 52(1), 2005.

[3] P. K. Agarwal, S. Har-Peled, and H. Yu. Robust shape fitting via peeling and grating coresets. *Discrete & Computational Geometry*, 39(1):38–58, 2008.

[4] N. Amenta, D. Attali, and O. Devillers. Size of Delaunay triangulation for points distributed over lower-dimensional polyhedra: a tight bound. *Neural Information Processing Systems (NeurIPS): Topological Learning*, 2007.

[5] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. *Discrete & Computational Geometry*, 22(4):481–504, 1999.

[6] S. Arya, G. D. da Fonseca, and D. M. Mount. Approximate convex intersection detection with applications to width and Minkowski sums. In *26th European Symposium on Algorithms (ESA)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[7] D. Attali and J.-D. Boissonnat. Complexity of the delaunay triangulation of points on polyhedral surfaces. *Discrete & Computational Geometry*, 30(3):437–452, 2003.

[8] D. Attali and J.-D. Boissonnat. A linear bound on the complexity of the Delaunay triangulation of points on polyhedral surfaces. *Discrete & Computational Geometry*, 31(3):369–384, 2004.

[9] S. W. Bae. Computing a minimum-width square annulus in arbitrary orientation. *Theoretical Computer Science*, 718:2–13, 2018.

[10] S. W. Bae. Computing a minimum-width cubic and hypercubic shell. *Operations Research Letters*, 47(5):398–405, 2019.

[11] S. W. Bae. On the minimum-area rectangular and square annulus problem. *Computational Geometry*, 92:101697, 2021.

[12] G. Barequet, P. Bose, M. T. Dickerson, and M. T. Goodrich. Optimizing a constrained convex polygonal annulus. *J. Discrete Algorithms*, 3(1):1–26, 2005.

[13] G. Barequet, M. T. Dickerson, and Y. Scharf. Covering points with a polygon. *Computational Geometry*, 39(3):143–162, 2008.

[14] P. Bose and P. Morin. Testing the Quality of Manufactured Disks and Cylinders. In G. Goos, J. Hartmanis, J. van Leeuwen, K.-Y. Chwa, and O. H. Ibarra, editors, *Algorithms and Computation*, volume 1533, pages 130–138. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998.

[15] T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. In *16th Symposium on Computational Geometry (SoCG)*, pages 300–309, 2000.

[16] S. Das, A. Nandy, and S. Sarvottamananda. Linear time algorithm for 1-Center in $\Re^d$ under convex polyhedral distance function. In D. Zhu and S. Bereg, editors, *Frontiers in Algorithmics*, volume 9711, pages 41–52. Springer, 2016.

[17] O. Devillers and P. A. Ramos. Computing roundness is easy if the set is almost round. *International Journal of Computational Geometry & Applications*, 12(3):229–248, 2002.

[18] J. Erickson. Nice point sets can have nasty Delaunay triangulations. In *17th Symposium on Computational Geometry (SoCG)*, pages 96–105, 2001.

[19] O. N. Gluchshenko, H. W. Hamacher, and A. Tamir. An optimal $O(n \log n)$ algorithm for finding an enclosing planar rectilinear annulus of minimum width. *Operations Research Letters*, 37(3):168–170, 2009.

[20] K. Mehlhorn, T. C. Shermer, and C. K. Yap. A complete roundness classification procedure. In *Proceedings of the Thirteenth Annual Symposium on Computational Geometry - SCG '97*, pages 129–138, Nice, France, 1997. ACM Press.

[21] J. Mukherjee, P. R. Sinha Mahapatra, A. Karmakar, and S. Das. Minimum-width rectangular annulus. *Theoretical Computer Science*, 508:74–80, 2013.

[22] J. M. Phillips. Coresets and sketches. In *Handbook of Discrete and Computational Geometry*, pages 1269–1288. Chapman and Hall/CRC, 2017.

[23] U. Thakker, R. Patel, S. Tanwar, N. Kumar, and H. Song. Blockchain for diamond industry: Opportunities and challenges. *IEEE Internet of Things Journal*, pages 1–1, 2020.

[24] H. Yu, P. K. Agarwal, R. Poreddy, and K. R. Varadarajan. Practical methods for shape fitting and kinetic data structures using coresets. *Algorithmica*, 52(3):378–402, 2008.

Optimally Confining Lattice Polymers

Robert D. Barish* Tetsuo Shibuya†

Abstract

We introduce the Lattice Polymer Confinement Problem (LPCP), where provided a graph G corresponding to a solid or hole-containing finite lattice, and provided a finite set of vertex-wise lengths $\mathcal{L} \subset \mathbb{N}$ of lattice polymers modeled as Self-Avoiding Walks (SAWs), the objective is to delete the fewest possible number of vertices in G to satisfy a bound $\mathcal{S}_{(G,\mathcal{L})} \leq \Omega$ on a sum over the configuration entropies of each polymer. In this context, we use Boltzmann's expression $\mathcal{S}_{(G,\mathcal{L})} = k_B \cdot \ln(\mathcal{W}+1)$ for the system configuration entropy, where $k_B \approx 1.380649 \cdot 10^{-23}\ J \cdot K^{-1}$ is Boltzmann's constant, and $\mathcal{W}$ corresponds to a sum over the number of SAWs modeling lattice polymers in a specified host graph. We also propose a novel Self-Avoiding Walk (SAW) centrality measure, $\mathcal{C}_{SAW}(\mathcal{L}, v_i)$, for a vertex v_i in a lattice or graph as a variation on the standard notion of betweenness centrality, which for a specified finite set $\mathcal{L} \subset \mathbb{N}$, corresponds to the fraction of length $l_i \in \mathcal{L}$ SAWs that cover v_i.

Letting G be an input lattice or graph for LPCP with vertex set V_G and edge set E_G, we show that LPCP is NP-hard as well as APX-hard $\forall \Omega \geq 0$ and for all finite $\mathcal{L} \subset \mathbb{N}_{\geq 2}$. On the other hand, letting $tr(G)$ be the treewidth of G, letting $\zeta_{tw} = f(tr(G)) \cdot \mathcal{O}(|V_G| + |E_G|)$ for some computable function f, and letting $\mathcal{Y}_{init}$ be the initial system configuration entropy, we prove the existence of an $\mathcal{O}\left(\zeta_{tw} \cdot |V_G|^3 \cdot \ln(|V_G|)\right)$ time $\left(\ln\left(e^{(\mathcal{Y}_{init})} - e^{\Omega}\right)\right)$-approximation algorithm for LPCP. We moreover establish that an $\mathcal{O}(\Psi)$ deterministic algorithm for SAW centrality with multiplicative error $1 \pm \epsilon$, which we remark can be derived from existing PTAS algorithms for counting bounded-length SAWs in graphs, correspondingly implies the existence of an $\mathcal{O}\left(\Psi \cdot |V_G|^3 \cdot \ln(|V_G|)\right)$ time $\left(\frac{\ln\left(e^{(\mathcal{Y}_{init})} - e^{\Omega}\right)}{1-2\epsilon}\right)$-approximation algorithm for LPCP.

Finally, we briefly analyze variations on LPCP, including a variant where we delete edges in lieu of vertices, and variant with "rigid" lattice polymers (e.g., lattice proteins) where every embedding must satisfy a set of consecutive dihedral angles for adjacent bonds.

1 Introduction

We introduce and analyze what we denote the Lattice Polymer Confinement Problem (LPCP), which concerns minimally modifying a solid or hole-containing finite lattice G such that, provided a finite set of vertex-wise lengths $\mathcal{L} \subset \mathbb{N}$ of lattice polymers modeled as Self-Avoiding Walks (SAWs), the system configuration entropy $\mathcal{S}_{(G,\mathcal{L})} = k_B \cdot \ln(\mathcal{W}+1)$ falls below a specified threshold $\Omega \geq 0$. In this context, $k_B \approx 1.380649 \cdot 10^{-23}\ J \cdot K^{-1}$ is Boltzmann's constant, $\mathcal{W}$ corresponds to a sum over the number of SAWs (modeling lattice polymers) of length $l_i \in \mathcal{L}$ in a specified host graph, and Ω should everywhere be assumed to have units $J \cdot K^{-1}$. Briefly, we can observe that the system configuration entropy is equivalently expressed as $\mathcal{S}_{(G,\mathcal{L})} = -k_B \cdot \sum_{i=1}^{\mathcal{W}} \left(\left(\frac{1}{\mathcal{W}+1}\right) \cdot \ln\left(\frac{1}{\mathcal{W}+1}\right)\right)$, allowing us to obtain the expression for Shannon entropy by substituting k_B with the reciprocal of the logarithm of the number of observed events for a discrete random process and $\left(\frac{1}{\mathcal{W}+1}\right)$ with the probability of a specific event.

We remark that such lattice polymer models have extensive precedence in the field of protein structure prediction and engineering [14, 22]. For illustrative examples, we refer the reader to Fig. 1 and Fig. 2, where we show LatFit [23, 24] generated (semi-rigid) embeddings of the peptide backbones for the NMR solution structure of an ShK potassium channel inhibitor toxin from sea anemone (PDB ID: 1ROO) on a $\mathbb{Z}^2$ integer lattice, and the crystal structure of an antifreeze protein from notched-fin eelpout (PDB ID: 5XQN) on a 210 "knight's tour" lattice, respectively.

Our inspiration for LPCP is a visually stunning experimental demonstration by Turner et. al. [26] of how entropy gradients can invoke forces on polymers. To briefly describe their experiment, Turner et. al. [26] began by manufacturing a microfluidic cell with two adjacent quasi-two-dimensional volumes, which we will refer to as J_{open} and $J_{pillars}$, where J_{open} is an otherwise open volume and $J_{pillars}$ is populated with $\approx 35\ nm$ diameter pillars with a $\approx 160\ nm$ center-to-center spacing. The authors then used an electric field to drag double-stranded T2-phage genomic DNA (having a contour length of $\approx 51\ \mu m$) from J_{open} to $J_{pillars}$, signifi-

*Division of Medical Data Informatics, Human Genome Center, Institute of Medical Science, University of Tokyo, 4-6-1 Shirokanedai, Minato-ku, Tokyo 108-8639, Japan, rbarish@ims.u-tokyo.ac.jp

†Division of Medical Data Informatics, Human Genome Center, Institute of Medical Science, University of Tokyo, 4-6-1 Shirokanedai, Minato-ku, Tokyo 108-8639, Japan, tshibuya@hgc.jp

cantly restricting the polymer's configuration freedom. Once the electric field was lifted and one end of a given polymer diffused into J_{open}, a CCD camera was used to observe the remainder of the molecule rapidly "recoiling" out of $J_{pillars}$, acting against a hydrodynamic drag with a force of $\approx 5.7\ fN$. The authors then determined that this force was driven almost entirely by a configuration entropy gradient, noted that it was within an order of magnitude of the $\approx 40\ fN$ force expected by a $\approx \Delta 1\ kB$ change in configuration entropy per polymer Kuhn length (i.e., length units of a semi-rigid polymer that can be approximated as segments of a freely-jointed chain [16]) moving from $J_{pillars}$ to J_{open}.

We now ask the question: if we treat the configuration entropy $\mathcal{S}_{(G,\mathcal{L})}$ for one or more lattice polymers embedded in a solid lattice as roughly equivalent to the embedding of real polymers in an open volume akin to J_{open}, how can we minimally modify the lattice (e.g., by deleting vertices) to create a volume akin to $J_{pillars}$? Here, the aforementioned LPCP problem, which we formally define below, represents our attempt to formalize and generalize this problem.

Definition 1 *Lattice Polymer Confinement Problem,* $LPCP\left(G, \mathcal{L}, \Omega\right)$

Input: A graph G with vertex set V_G, corresponding to a solid or hole-containing finite lattice, a finite set of vertex-wise lengths $\mathcal{L} \subset \mathbb{N}$ of lattice polymers modeled as Self-Avoiding Walks (SAWs), and an upperbound Ω for the configuration entropy $\mathcal{S}_{(G,\mathcal{L})}$ of the system. Here, $\mathcal{S}_{(G,\mathcal{L})} = k_B \cdot \ln\left(\mathcal{W}+1\right)$, where $k_B \approx 1.380649 \cdot 10^{-23} J \cdot K^{-1}$ is Boltzmann's constant, and $\mathcal{W}$ corresponds to a sum over the number of embeddings in G of each lattice polymer corresponding to a SAW of length $l_i \in \mathcal{L}$.
Objective: Return a minimum cardinality set of vertices $\mathcal{Q} \subseteq V_G$ whose deletion converts G into a graph G' where we have that $\mathcal{S}_{(G',\mathcal{L})} \leq \Omega$.

For illustrative examples of $LPCP\left(G, \mathcal{L}, \Omega\right)$ and what (approximate) witnesses look like, we refer the reader to Fig. 3, where we show instances of input graphs G corresponding to: (a) a 6×6 induced subgraph of a $\mathbb{Z}^2$ integer lattice; (b) a $3 \times 3 \times 3$ induced subgraph of a $\mathbb{Z}^3$ integer lattice; (c) an induced subgraph of a triangular lattice; and (d) an induced subgraph of a honeycomb lattice. In each of the examples from Fig. 3(a–d), we also show a set of (white) vertices that would be selected in the specified order (first 1, then 2, etc.) for deletion by a greedy algorithm attempting to minimize $\mathcal{S}_{(G,\mathcal{L})}$. In the Fig. 3(e) table, we show the approximate configuration entropy for the examples in Fig. 3(a–d) (recall that $k_B \approx 1.380649 \cdot 10^{-23} J \cdot K^{-1}$), as well as the configuration entropies following each vertex deletion.

As a subroutine of our greedy algorithms for the LPCP problem, we also introduce a novel Self-Avoiding Walk (SAW) vertex centrality measure as a variation on *betweenness centrality*. This measure assigns a score to the vertices of a simple undirected graph based on the fraction of all possible SAW embeddings of specified lengths $l_i \in \mathcal{L}$ they are covered by, and accordingly allows one to rank vertices in a graph according to the effect of their deletion on the system configuration entropy. More specifically, letting G be a simple graph with vertex set V_G, and letting $f_{(SP,all)}\left(G, v_a, v_b\right)$ and $f_{(SP,v_i)}\left(G, v_a, v_b\right)$ be functions which return the number of shortest paths from a vertex $v_a \in V_G$ to a vertex $v_b \in V_G$ and the number of such paths traversing the vertex $v_i \notin \{v_a, v_b\}$, respectively, we can recall that the *betweenness centrality* [17, 18, 27] for a vertex $v_i \in V_G$ is given by $\mathcal{C}_{Betweenness}\left(G, v_i\right) = \sum_{(a,b \in [1,|V_G|] \wedge a<b \wedge a \neq i \wedge b \neq i)} \left\{ \begin{pmatrix} \frac{f_{(SP,v_i)}(G,v_a,v_b)}{f_{(SP,all)}(G,v_a,v_b)} \end{pmatrix}, \quad f_{(SP,all)}\left(G, v_a, v_b\right) \neq 0 \\ 0, \quad f_{(SP,all)}\left(G, v_a, v_b\right) = 0 \right\}$.
Now, letting G and V_G be defined as before, and letting $f_{(SAW,all)}\left(G, \mathcal{L}\right)$ be a function which returns the number of all simple paths (equiv. SAWs) of all possible vertex-wise lengths $l_i \in \mathcal{L}$ in G, we can define the *SAW centrality* for a vertex $v_i \in V_G$ as $\mathcal{C}_{SAW}\left(G, \mathcal{L}, v_i\right) = \left\{ \begin{pmatrix} \frac{f_{(SAW,all)}(G-v_i,\mathcal{L})}{f_{(SAW,all)}(G,\mathcal{L})} \end{pmatrix}, \quad \text{for } f_{(SAW,all)}\left(G, \mathcal{L}\right) \neq 0 \\ 0, \quad \text{for } f_{(SAW,all)}\left(G, \mathcal{L}\right) = 0 \right\}$.

To begin our analysis of LPCP, we first establish hardness results. In particular, we show that LPCP is NP-hard even if G is a subgraph of a $\mathbb{Z}^2$ integer lattice and we have either the constraint that $|\mathcal{L}| = 1$ or the constraint that $\mathcal{L} = \{1, 2, \ldots, |V_G|\}$ (Proposition 1). If G is allowed to be an arbitrary simple undirected graph, we moreover show that LPCP is NP-hard as well as APX-hard $\forall \Omega \geq 0$ and for all finite $\mathcal{L} \subset \mathbb{N}_{\geq 2}$ (Proposition 2).

We next detail approximation algorithms for LPCP. In particular, letting $tr\left(G\right)$ be the treewidth of G, letting $\zeta_{tw} = f\left(tr\left(G\right)\right) \cdot \mathcal{O}\left(|V_G| + |E_G|\right)$ for some computable function f, and letting $\mathcal{Y}_{init}$ be the initial system configuration entropy, we prove the existence of an $\mathcal{O}\left(\zeta_{tw} \cdot |V_G|^3 \cdot \ln\left(|V_G|\right)\right)$ time $\left(\ln\left(e^{(\mathcal{Y}_{init})} - e^{\Omega}\right)\right)$-approximation algorithm (Theorem 3). We additionally show that an $\mathcal{O}\left(\Psi\right)$ deterministic algorithm for SAW centrality with multiplicative error $1 \pm \epsilon$ correspondingly implies the existence of an $\mathcal{O}\left(\Psi \cdot |V_G|^3 \cdot \ln\left(|V_G|\right)\right)$ time $\left(\frac{\ln\left(e^{(\mathcal{Y}_{init})} - e^{\Omega}\right)}{1-2\epsilon}\right)$-approximation algorithm (Theorem 5).

Finally, we show how the aforementioned approximation algorithms extend to variations on LPCP where we delete edges in lieu of vertices (Corollary 8), as well as a variant where we consider the configuration entropies of "rigid" lattice polymers (e.g., lattice proteins) akin to those shown in Fig. 1 and Fig. 2 (Remark 1).

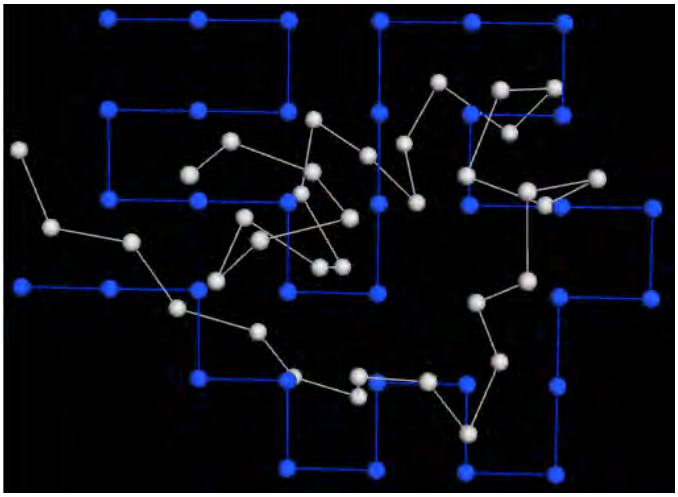


Figure 1: LatFit [23, 24] generated $\mathbb{Z}^2$ integer lattice embedding of the backbone for the NMR solution structure of an ShK potassium channel inhibitor toxin from sea anemone (PDB ID: 1ROO); the lattice embedding of the protein backbone is illustrated with (blue) vertices and edges, and the original structure of the protein backbone is illustrated with (white) vertices and edges.

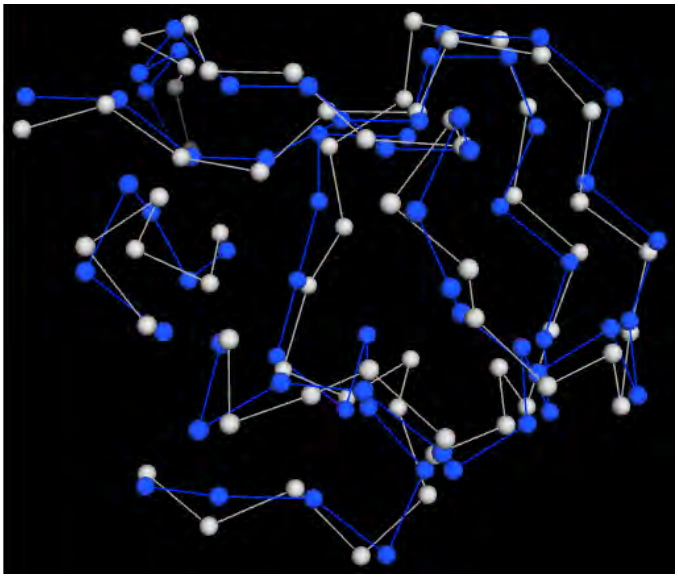
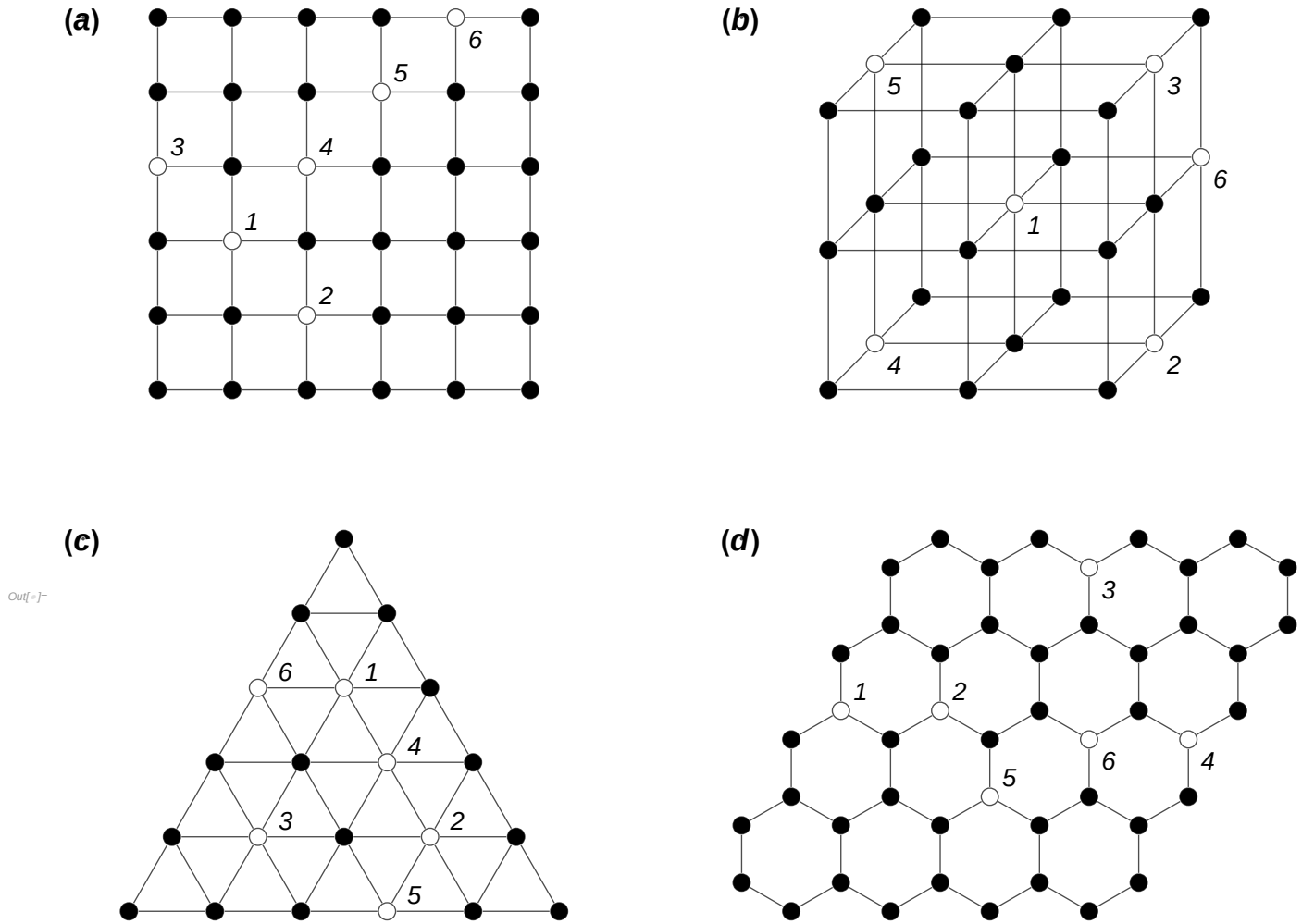


Figure 2: LatFit [23, 24] generated 210 "knight's tour" lattice embedding of the backbone for the crystal structure (obtained via X-ray diffraction techniques) of an antifreeze protein from notched-fin eelpout (PDB ID: 5XQN); the lattice embedding of the protein backbone is illustrated with (blue) vertices and edges, and the original structure of the protein backbone is illustrated with (white) vertices and edges.

2 Preliminaries

2.1 Graph theoretic terminology

We will generally follow definitions that are more-or-less standard (see, e.g., Diestel [11]). However, for some brief clarifications, when we use the term *graph* we are everywhere referring to simple undirected and unweighted graphs. We call a graph *cubic* if and only if all of its vertex degrees are uniformly equal to 3, and *subcubic* if and only if it has maximum vertex degree 3. Concerning paths and cycles in graphs, a path or cycle is called *simple*, or a Self-Avoiding Walk (SAW) in the case of paths, if it does not revisit either edges or vertices, called *Hamiltonian* if it is simple and covers all vertices, and called *induced* if it is also an induced subgraph. Here, the Hamiltonian cycle problem is the problem of deciding the existence of a Hamiltonian cycle in a graph, and the *st*-path problem and *st*-Hamiltonian



(e)

	Deleted Vertices:	None	{1}	{1,2}	{1,2,3}	{1,2,3,4}	{1,2,3,4,5}	{1,2,3,4,5,6}
(a)	6×6 Integer Lattice, $\mathcal{S}_{(G,\mathcal{L})}(J\cdot K^{-1})\cdot(k_B)^{-1}$:	19.8946	18.2877	16.5245	14.7857	13.0960	11.2828	9.36888
(b)	3×3×3 Integer Lattice, $\mathcal{S}_{(G,\mathcal{L})}(J\cdot K^{-1})\cdot(k_B)^{-1}$:	20.1267	18.0250	16.3276	14.5983	12.8702	11.1030	9.48562
(c)	Triangular Lattice, $\mathcal{S}_{(G,\mathcal{L})}(J\cdot K^{-1})\cdot(k_B)^{-1}$:	16.3800	14.2945	12.1922	10.0761	8.51759	6.90174	4.95583
(d)	Honeycomb Lattice, $\mathcal{S}_{(G,\mathcal{L})}(J\cdot K^{-1})\cdot(k_B)^{-1}$:	16.5431	15.0823	13.5212	11.9914	10.5050	9.06439	7.23778

Figure 3: Illustrative examples of $LPCP\,(G, \mathcal{L}, \Omega)$ problem instances, where letting $\mathcal{L}$ be the set of all possible SAW lengths, for each graph G in **(a–d)** the order of the first 6 vertices (colored white) selected by a naive greedy algorithm minimizing the system configuration entropy is shown (with the labels "1" for the first selected vertex, "2" for the second selected vertex, etc.). The approximate system configuration entropies before and after each successive vertex deletion event, divided by k_B, are given in the table shown in **(e)**. Here, **(a)** corresponds to a 6×6 induced subgraph of a $\mathbb{Z}^2$ integer lattice (36 vertices and 60 edges), **(b)** corresponds to a $3 \times 3 \times 3$ induced subgraph of a $\mathbb{Z}^3$ integer lattice (27 vertices and 54 edges), **(c)** corresponds to an induced subgraph of a triangular lattice (21 vertices and 45 edges), and **(d)** corresponds to an induced subgraph of a honeycomb lattice (48 vertices and 63 edges).

path problem is the problem of deciding the existence of a simple path (equiv., SAW) and Hamiltonian path, respectively, between a pair of vertices v_s and v_t.

2.2 Fixed-parameter tractability and intractability

A problem can be denoted *Fixed-Parameter Tractable* (FPT) if, letting x be a string encoding a given prob-

lem instance and $f(k)$ be any computable function, its time complexity can be written as $f(k) \cdot |x|^{\mathcal{O}(1)}$. With regard to parameterized hardness, we concern ourselves with completeness for the class $W[1]$ of all parameterized languages that can be encoded as Boolean decision circuits with *weft* at most 1 (see, e.g., ref. [13]). Here, a circuit with weft k can have at most k large gates (i.e. degree ≥ 3 vertices in the finite directed acyclic graph representation of the circuit) along any given path from an input node to an output node.

2.3 Approximation tractability and intractability

Concerning approximation tractability, we concern ourselves with the notion of a Polynomial-Time Approximation Scheme (PTAS) and Fully Polynomial-Time Approximation Scheme (FPTAS). Here, for some error parameter $\epsilon > 0$, a PTAS is a deterministic algorithm which produces a solution for a given optimization problem with a multiplicative error of $1 \pm \epsilon$ (typically $1-\epsilon$ and $1+\epsilon$ for maximization and minimization problems, respectively), with a running time polynomial in length of an input string specifying the optimization problem. If a PTAS also has a running time polynomial in $\frac{1}{\epsilon}$, then we refer to the PTAS as a FPTAS. With regard to approximation hardness, we concern ourselves with hardness for the class APX of problems admitting a constant-ratio approximation algorithm. As there are problems in the class APX that do not admit a PTAS unless $NP = RP$, including a number of interesting special cases of the geometric set cover problem [5], this correspondingly implies that an APX-hard problem cannot admit a PTAS unless $P = NP$.

3 Hardness results

Proposition 1 *For a subgraph G of a $\mathbb{Z}^2$ integer lattice with vertex set V_G, we have that $LPCP(G, \mathcal{L}, \Omega)$ is NP-hard under both the constraint that $|\mathcal{L}| = 1$ and the constraint that $\mathcal{L} = \{1, 2, \ldots, |V_G|\}$.*

Proof. Letting G be a subgraph of a $\mathbb{Z}^2$ integer lattice with vertex set V_G, by the proof argument for "Theorem 7" of Liśkiewicz et. al. [20] we have that there is an efficient polynomial time counting reduction (more specifically, a polynomial time many-one counting "weakly parsimonious" reduction) from counting (case 1) SAWs of a specific length $l_r \in \mathbb{N}$ in G, and (case 2) SAWs of all possible lengths in G, to counting st-Hamiltonian paths in a subcubic planar graph H. The aforementioned proof argument also gives specific polynomial-time computable formula for the number of SAWs that must exist in (case 1) and (case 2), which we will denote T_1 and T_2, respectively, for there to exist at least one st-Hamiltonian path in H. We can also observe, as detailed in "Section 3" of Liśkiewicz et. al. [20], that H is constructed via a polynomial time many-one counting reduction from an instance of $\#3SAT$ to the problem of counting st-Hamiltonian paths in a subcubic planar graph.

Now, let G_1 and G_2 correspond to subgraphs of a $\mathbb{Z}^2$ integer lattice constructed from a subcubic planar graph H for (case 1) and (case 2), respectively, in the proof argument for "Theorem 7" of Liśkiewicz et. al. [20]. Observe that by specifying parameters $\mathcal{L} = \{l_r\}$ and $\Omega = k_B \cdot \ln(T_1 + 1)$, a witness for $LPCP(G_1, \mathcal{L}, \Omega)$ will be the null set if and only if H possesses an st-Hamiltonian path. Similarly, observe that by specifying parameters $\mathcal{L} = \{1, 2, \ldots, |V_G|\}$ and $\Omega = k_B \cdot \ln(T_2 + 1)$, a witness for $LPCP(G_2, \mathcal{L}, \Omega)$ will be the null set if and only if H possesses an st-Hamiltonian path. As the st-Hamiltonian path problem is NP-complete for arbitrary instances of the graph H due to the manner in which the graph is constructed, this yields the proposition. □

Proposition 2 *$LPCP(G, \mathcal{L}, \Omega)$ is NP-hard and APX-hard $\forall \Omega \geq 0$ and for all finite $\mathcal{L} \subset \mathbb{N}_{\geq 2}$.*

Proof. By metatheorems of Yannakakis & Lewis [28, 19] and Lund & Yannakakis [21], we have that the problem of deleting a minimum set of vertices in a simple undirected graph G to satisfy a property Π is NP-hard and APX-hard, respectively, if Π is a *nontrivial* and *hereditary* property. Here, a property being nontrivial means that it both holds and fails to hold for infinitely many graphs, and a property being *hereditary* means that it is satisfied for a graph if and only if it is satisfied for all of the graph's induced subgraphs.

Now, letting $\mathcal{L} \subset \mathbb{N}_{\geq 2}$ be some finite set of vertex-wise lengths for SAWs, observe that there are infinitely many independent sets having no embeddings of SAWs of length $l_i \in \mathcal{L}$, and infinitely many finite undirected graphs having more than an arbitrary number of embeddings of SAWs of length $l_i \in \mathcal{L}$. Accordingly, in the context of the LPCP problem where we model lattice polymers as SAWs having vertex-wise lengths from a set $\mathcal{L}$, $\forall \Omega \geq 0$ we have that there are infinitely many graphs failing to satisfy and satisfying a property Π that $\mathcal{S}_{(G,\mathcal{L})} \leq \Omega$. This implies that the aforementioned property Π is *nontrivial*. We also trivially have that Π is hereditary, as deleting vertices in a graph will cause the number of embeddings of SAWs with lengths in $\mathcal{L}$ to weakly monotonically decrease.

Putting everything together, and recalling that a witness for an instance of $LPCP(G, \mathcal{L}, \Omega)$ is a minimum set of vertices in a simple undirected graph G whose deletion yields a graph G' satisfying the property $\mathcal{S}_{(G',\mathcal{L})} \leq \Omega$, we have that $LPCP(G, \mathcal{L}, \Omega)$ is NP-hard and APX-hard $\forall \Omega \geq 0$ and for all finite $\mathcal{L} \subset \mathbb{N}_{\geq 2}$. □

4 Approximation algorithms for LPCP

Theorem 3 *Letting G is a simple undirected graph with vertex set V_G, edge set E_G, and treewidth $tr(G)$, letting $\zeta_{tw} = f(tr(G)) \cdot \mathcal{O}(|V_G| + |E_G|)$ for some computable function f, and letting $\mathcal{Y}_{init}$ be an initial system configuration entropy, we have that $LPCP(G, \mathcal{L}, \Omega)$ admits an $\mathcal{O}\left(\zeta_{tw} \cdot |V_G|^3 \cdot \ln(|V_G|)\right)$ time $\left(\ln\left(e^{(\mathcal{Y}_{init})} - e^{\Omega}\right)\right)$-approximation algorithm.*

Proof. Interpreting SAWs in G with lengths drawn from the set $\mathcal{L}$ as a universe of elements, and treating each vertex in V_G as the set of SAWs it is covered by, observe that we can correspondingly interpret $LPCP(G, \mathcal{L}, \Omega)$ as a partial set cover problem wherein the objective is to cover at least $\approx \left(e^{(\mathcal{Y}_{init})} - e^{\Omega}\right)$ such elements (i.e., SAW embeddings) with the minimum possible number of sets (i.e., vertices). Accordingly, we immediately have a $\left(\ln\left(e^{(\mathcal{Y}_{init})} - e^{\Omega}\right)\right)$-approximation algorithm as a consequence of the harmonic approximation guarantee for the greedy algorithm for partial set cover [15] (see also Slavík [25] for a detailed performance analysis of the greedy algorithm for the original set cover problem).

In the current context, we can observe that: (obs. 1) there will be an $\mathcal{O}\left(|V_G|^2\right)$ overhead for the subroutines of the greedy algorithm, where for at most $|V_G|$ iterations, we scan at most $|V_G|$ vertices to find the ones whose deletion will maximize coverage of the elements corresponding to SAW embeddings in G; (obs. 2) the selected vertex for each iteration will necessarily be a vertex $v_i \in V_G$ having the largest SAW centrality, $\mathcal{C}_{SAW}(G, \mathcal{L}, v_i)$ (as defined in the introduction of the current work); and (obs. 3) that there will be at most $\mathcal{O}(|V_G|!)$ SAWs of all possible lengths in G, implying that there will be at most the same number of elements to cover in the partial set covering formulation of $LPCP(G, \mathcal{L}, \Omega) \implies$ we will need to read at most the first $\mathcal{O}(|V_G| \cdot \ln(|V_G|))$ bits of each vertex SAW centrality $\mathcal{C}_{SAW}(G, \mathcal{L}, v_i)$ to determine the largest values. Letting Ψ be the cost of computing the SAW centrality for a vertex $v_i \in V_G$, (obs. 1) through (obs. 3) imply that the aforementioned approximation algorithm will have a time complexity of $\mathcal{O}\left(\Phi \cdot |V_G|^3 \cdot \ln(|V_G|)\right)$.

We can now observe the following lemma concerning the treewidth fixed-parameter tractability of computing $\mathcal{C}_{SAW}(G, \mathcal{L}, v_i)$:

Lemma 4 *For a simple undirected graph G with vertex set V_G and edge set E_G, the problem of determining the SAW centrality values for a vertex $v_i \in V_G$, $\mathcal{C}_{SAW}(G, \mathcal{L}, v_i)$, is treewidth FPT, and can be calculated in $\zeta_{tw} = \mathcal{O}(|V_G| + |E_G|)$ time if G has bounded treewidth.*

Proof. It suffices to show there exists a linear time treewidth FPT algorithm for counting the number of SAWs between an arbitrary pair of vertices v_s and v_t in a graph. Observe that we can simply run this procedure for an instance of a graph with or without a specified vertex to determine $\mathcal{C}_{SAW}(G, \mathcal{L}, v_i)$.

We proceed by appealing to an extension of Courcelle's well-known algorithmic metatheorem [6, 7, 8, 9] to counting and optimization problems [1, 10]. In particular, we appeal to "Theorem 32" of Courcelle, Makowsky, & Rotics [9], which states in part that if we can express the existence of a graph property ϕ in the fragment of second order logic denoted "extended" Monodic Second Order (MS_2) (see, e.g., Downey & Fellows [13] for an elaboration), then we are guaranteed an algorithm for this problem having time complexity $c \cdot \mathcal{O}(|V| + |E|)$, where c is a constant that depends only on ϕ and the graph treewidth $tw(G)$. Here, this time complexity is a consequence of the proof being based on the bottom-up traversal of a tree decomposition for a finite simple undirected graph G, which has time complexity linear in the size of the tree, and the existence of an $\mathcal{O}(|V| + |E|)$ algorithm due to Bodlaender [3] for computing a tree decomposition of G having width at most $tw(G)$.

To establish the lemma at hand, it now suffices to note that the existence of a path between an arbitrary pair of vertices v_s and v_t in a graph is expressible in first-order (FO_1) logic. In particular, we refer the reader to "pg. 4" of [7], where Courcelle discusses the use of an FO_1 auxiliary predicate " QuasiPath' " for expressing reachability between a pair of vertices in an undirected graph. □

Putting everything together, we can set $\Phi = \zeta_{tw}$ in the earlier asymptotic time analysis of the $\left(\ln\left(e^{(\mathcal{Y}_{init})} - e^{\Omega}\right)\right)$-approximation algorithm for $LPCP(G, \mathcal{L}, \Omega)$ to yield the time complexity in the statement of the current theorem. □

Theorem 5 *Letting G be a simple undirected graph with vertex set V_G and letting $\mathcal{Y}_{init}$ be an initial system configuration entropy, if an $\mathcal{O}(\Psi)$ deterministic algorithm exists for computing the SAW centrality of a vertex $v_i \in V_G$, $\mathcal{C}_{SAW}(G, \mathcal{L}, v_i)$ with multiplicative error $1 \pm \epsilon$, then we correspondingly have that $LPCP(G, \mathcal{L}, \Omega)$ admits an $\mathcal{O}\left(\Psi \cdot |V_G|^3 \cdot \ln(|V_G|)\right)$ time $\left(\frac{\ln\left(e^{(\mathcal{Y}_{init})} - e^{\Omega}\right)}{1-2\epsilon}\right)$-approximation algorithm.*

Proof. Recalling our earlier reformation of $LPCP(G, \mathcal{L}, \Omega)$ as a partial set cover problem in the proof argument for Theorem 3, we begin by observing the following lemma:

Lemma 6 *Letting $\mathcal{P}$ be an instance of the partial set cover problem, where $\mathcal{U}$ is the universe of elements, X is a collection of sets of elements from $\mathcal{U}$, and $0 \leq p \leq 1$*

is the fraction of elements that must be covered, and letting $f_{\epsilon-greedy}$ be an instance of the greedy algorithm, which in each iteration selects a set uniformly at random from all sets in X covering a fraction $(1-2\epsilon)$ of the maximum possible number of elements that can covered in the iteration, we have that $f_{\epsilon-greedy}$ will be a $\left(\frac{\ln(p)}{2\epsilon-1}\right)$-approximation algorithm for $\mathcal{P}$.

Proof. Letting α be the size of the minimum partial set cover for $\mathcal{P}$, observe that the kth iteration of $f_{\epsilon-greedy}$ will, in the worst case, reduce the number of uncovered elements in $\mathcal{U}$ by a fraction $\left(1-\frac{1-2\epsilon}{\alpha}\right)$. Accordingly, we can express the number of uncovered elements in $\mathcal{U}$ after r iterations of $f_{\epsilon-greedy}$ as $|\mathcal{U}| \cdot \left(1-\frac{1-2\epsilon}{\alpha}\right)^r$, or equivalently, as $|\mathcal{U}| \cdot \left(\left(1-\frac{1-2\epsilon}{\alpha}\right)^{\alpha}\right)^{\frac{r}{\alpha}}$.

We next establish that $\left(1-\frac{1-2\epsilon}{\alpha}\right)^{\alpha}$ will weakly monotonically increase with α for $0 \leq \epsilon \leq 1$ and $\alpha \geq 1$. To begin, we can note that:

$$\frac{\partial}{\partial\alpha}\left[\left(1-\frac{1-2\epsilon}{\alpha}\right)^{\alpha}\right] \geq 0$$

$$\Longleftarrow \left(\frac{1}{\alpha}\right) \cdot \left(\frac{\alpha+2\epsilon-1}{\alpha}\right)^{(\alpha-1)} \cdot$$

$$\left((1-2\epsilon)+(\alpha+2\epsilon-1)\cdot\ln\left(\frac{\alpha+2\epsilon-1}{\alpha}\right)\right) \geq 0$$

$$\Longleftarrow \left((1-2\epsilon)+(\alpha+2\epsilon-1)\cdot\ln\left(\frac{\alpha+2\epsilon-1}{\alpha}\right)\right) \geq 0$$

Now let $\omega = \left((1-2\epsilon)+(\alpha+2\epsilon-1)\cdot\ln\left(\frac{\alpha+2\epsilon-1}{\alpha}\right)\right)$. Here, we can observe that $\frac{\partial}{\partial\epsilon}(\omega) = 2\ln\left(\frac{\alpha+2\epsilon-1}{\alpha}\right)$, and accordingly, that for fixed $\alpha \geq 1$, the expression ω will be minimized for $\epsilon = \frac{1}{2}$. As $\epsilon = \frac{1}{2} \implies \omega = 0$, we therefore have that ω is non-negative whenever $\alpha \geq 1$ and $0 \leq \epsilon \leq 1$, and therefore that $\frac{\partial}{\partial\alpha}\left[\left(1-\frac{1-2\epsilon}{\alpha}\right)^{\alpha}\right]$ will be non-negative $\forall\alpha \geq 1$. It now suffices to note that $\alpha = 1$ and $0 \leq \epsilon \leq 1 \implies \left(1-\frac{1-2\epsilon}{\alpha}\right)^{\alpha} \geq 0$.

Putting everything together, we can use the approximation $\lim_{\alpha\to\infty}\left(1-\frac{1-2\epsilon}{\alpha}\right)^{\alpha} = e^{(2\epsilon-1)}$ to express the fraction of covered elements after r iterations of $f_{\epsilon-greedy}$ as $|\mathcal{U}| \cdot \left(e^{(2\epsilon-1)}\right)^{\frac{r}{\alpha}}$. Thus, $|\mathcal{U}| \cdot \left(e^{(2\epsilon-1)}\right)^{\frac{r}{\alpha}} = p \cdot |\mathcal{U}| \implies r = \left(\frac{\alpha\cdot\ln(p)}{2\epsilon-1}\right)$, yielding the lemma. □

To establish the theorem at hand, following the proof argument for Theorem 3, it now suffices to observe that $p \cdot |\mathcal{U}|$ from Lemma 6 can be understood to correspond to $\left(e^{(\mathcal{Y}_{init})} - e^{\Omega}\right)$, and that $f_{\epsilon-greedy}$ from Lemma 6 can be understood to correspond to the $\mathcal{O}(\Psi)$ deterministic algorithm for computing $\mathcal{C}_{SAW}(G, \mathcal{L}, v_i)$ with multiplicative error $1 \pm \epsilon$. □

Corollary 7 *There exists an instance of the $\mathcal{O}\left(\Psi \cdot |V_G|^3 \cdot \ln(|V_G|)\right)$ time $\left(\frac{\ln\left(e^{(\mathcal{Y}_{init})}-e^{\Omega}\right)}{1-2\epsilon}\right)$-approximation algorithm for $LPCP(G, \mathcal{L}, \Omega)$ from Theorem 5, where letting G be a simple undirected graph with vertex set V_G and edge set E_G, we have that $\Psi \in \mathcal{O}\left(\sum_{i=1}^{|\mathcal{L}|}\left(4^{l_i+\mathcal{O}\left(\sqrt{l_i}\cdot\left(\ln^2(l_i)+\ln^2\left(\frac{1}{\epsilon}\right)\right)\right)}\right)\cdot|E_G|\cdot\ln(|V_G|)\right)$.*

Proof. This result follows directly from a recent result of Björklund et. al. [2] that an $\mathcal{O}\left(\left(4^{k+\mathcal{O}\left(\sqrt{k}\cdot\left(\ln^2(k)+\ln^2\left(\frac{1}{\epsilon}\right)\right)\right)}\right)\cdot|E_G|\cdot\ln(|V_G|)\right)$ time and polynomial-space deterministic PTAS exists for counting the number of length k SAWs in a simple undirected graph G with vertex set V_G and edge set E_G. □

Corollary 8 *For a variant of $LPCP(G, \mathcal{L}, \Omega)$ where we delete edges in lieu of vertices, the time complexities of the Theorem 3 and Theorem 5 approximation algorithms become $\mathcal{O}\left(\zeta_{tw}\cdot|E_G|\cdot|V_G|^2\cdot\ln(|V_G|)\right)$ and $\mathcal{O}\left(\Psi\cdot|E_G|\cdot|V_G|^2\cdot\ln(|V_G|)\right)$, respectively.*

Proof. Observe that we can measure the SAW centrality of edges in a lattice or graph G in exactly the same manner (and with the same time complexity) as we computed the SAW centrality of vertices – e.g., by simply computing the change in the number of relevant SAW embeddings with and without a given edge being present. Therefore, the only change in the time complexity for the Theorem 3 and Theorem 5 approximation algorithms comes from having to compute the SAW centralities of $|E_G|$ edges instead of $|V_G|$ vertices. □

Remark 1 *For a variant of $LPCP(G, \mathcal{L}, \Omega)$ where we consider the configuration entropies of "rigid" lattice polymers (e.g., lattice proteins) where every embedding must satisfy a set of consecutive dihedral angles for bond edges, for an interpretation of "rigid" lattice polymers as SAWs required to have a specific geometry when embedded in a lattice or graph, the time complexities of the Theorem 3 and Theorem 5 approximation algorithms becomes $\mathcal{O}\left(|E_G|\cdot|V_G|^3\cdot\ln(|V_G|)\right)$ and $\mathcal{O}\left(|E_G|\cdot|V_G|^3\cdot\ln(|V_G|)\right)$, respectively.*

Proof. It suffices to observe that if we require SAWs to have a specific geometry, we can trivially enumerate the number of embeddings of such SAWs in $\mathcal{O}(|E_G|)$ time, as any edge of a specific SAW will fix the remaining edges. The stated changes in the time complexities for the Theorem 3 and Theorem 5 approximation algorithms then follow as a consequence of removing the cost of computing SAW centralities. □

5 Concluding Remarks

For a universe of elements $\mathcal{U}$, the general set cover problem is known not to be approximable within a factor of $(1-o(1))\cdot\ln(|\mathcal{U}|)$ unless $P = NP$ [12]. Accordingly, as

we establish the Theorem 3 and Theorem 5 approximation algorithms via reduction to equivalent partial set cover problems, it is unlikely that we can significantly improve the current approximation guarantees in either case. However, concerning a future research direction, we remark that much better performance guarantees can be achieved for the geometric set cover problem (see, e.g., Brönnimann & Goodrich [4]). Here, it should be possible to take advantage of a particular embedding of a lattice or graph to treat sets of vertices or SAWs (e.g., in a geometric hitting set formulation) as polygons or other shapes, and in some cases achieve better approximation guarantees or time complexities.

References

[1] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

[2] A. Björklund, D. Lokshtanov, S. Saurabh, and M. Zehavi. Approximate counting of k-paths: simpler, deterministic, and in polynomial space. *ACM Trans. Algorithms*, 17(3):26:1–26:44, 2021.

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

[4] H. Brönnimann and M. T. Goodrich. Almost optimal set covers in finite VC-dimension. *Discrete Comput. Geom.*, 14(1):463–479, 1995.

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

[6] B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inform. Comput.*, 85(1):12–75, 1990.

[7] B. Courcelle. On the expression of graph properties in some fragments of monadic second-order logic. *In: (N. Immerman and P. G. Kolaitis, eds.) Descriptive Complexity and Finite Models, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 31, 1997.

[8] B. Courcelle. Graph structure and monadic second-order logic: language theoretical aspects. *Proc. of the 35th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 1–13, 2008.

[9] B. Courcelle, J. A. Makowsky, and U. Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discrete Appl. Math.*, 108(1-2):23–52, 2001.

[10] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized algorithms, 1st edition*. Springer: Switzerland, 2015.

[11] R. Diestel. *Graph theory, 5th edition*. Springer-Verlag: Heidelberg, 2017.

[12] I. Dinur and D. Steurer. Analytical approach to parallel repetition. *Proc. 46th annual ACM Symposium on Theory of Computing (STOC)*, pages 624–633, 2014.

[13] R. G. Downey and M. R. Fellows. *Fundamentals of parameterized complexity, 1st edition*. Springer: New York, NY, 2013.

[14] S. P. N. Dubey, N. G. Kini, S. Balaji, and M. S. Kumar. A review of protein structure prediction using lattice model. *Crit. Rev. Biomed. Eng.*, 46(2):147–162, 2018.

[15] T. Elomaa and J. Kujala. Covering analysis of the greedy algorithm for partial cover. *Algorithms and Applications. Lecture Notes in Computer Science (LNCS)*, 6060:102–113, 2010.

[16] P. J. Flory. *Principles of polymer chemistry (1st edition)*. Cornell University Press: Ithaca, NY, 1953.

[17] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40(1):35–41, 1977.

[18] A. Landherr, B. Friedl, and J. Heidemann. A critical review of centrality measures in social networks. *Bus. Inf. Syst. Eng.*, 2:371–385, 2010.

[19] J. M. Lewis and M. Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *J. Comput. Syst. Sci.*, 20(2):219–230, 1980.

[20] M. Liśkiewicz, M. Ogihara, and S. Toda. The complexity of counting self-avoiding walks in subgraphs of two-dimensional grids and hypercubes. *Theoret. Comput. Sci.*, 304(1-3):129–156, 2003.

[21] C. Lund and M. Yannakakis. The approximation of maximum subgraph problems. *Proc. 20th International Colloquium on Automata, Languages, and Programming (ICALP)*, 700:40–51, 1993.

[22] N. Madras and G. Slade. *The self-avoiding walk, 1st edition*. Birkhäuser: Boston, MA, 1996.

[23] M. Mann, R. Saunders, C. Smith, R. Backofen, and C. M. Deane. Producing high-accuracy lattice models from protein atomic co-ordinates including side chains. *Adv. Bioinform.*, 2012(Article ID 148045):1–6, 2012.

[24] M. Mann, C. Smith, M. Rabbath, M. Edwards, S. Will, and R. Backofen. CPSP-web-tools: a server for 3D lattice protein studies. *Bioinformatics*, 25(5):676–677, 2009.

[25] P. Slavík. A tight analysis of the greedy algorithm for set cover. *J. Algorithms*, 25(2):237–254, 1997.

[26] S. W. P. Turner, M. Cabodi, and H. G. Craighead. Confinement-induced entropic recoil of single DNA molecules in a nanofluidic structure. *Phys. Rev. Lett.*, 88(12):128103:1–128103:4, 2002.

[27] D. R. White and S. P. Borgatti. Betweenness centrality measures for directed graphs. *Social Networks*, 16(4):335–346, 1994.

[28] M. Yannakakis. Node- and edge-deletion NP-complete problems. *Proc. 10th annual ACM Symposium on Theory of Computing (STOC)*, pages 253–264, 1978.

Efficiently Enumerating Scaled Copies of Point Set Patterns

Aya Bernstine* and Yehonatan Mizrahi†

Abstract

Problems on repeated geometric patterns in finite point sets in Euclidean space are extensively studied in the literature of combinatorial and computational geometry. Such problems trace their inspiration back to Erdős' original work on this topic. In this paper, we investigate the problem of finding scaled copies of any pattern within a set of n points, that is, the algorithmic task of efficiently enumerating all such copies. We initially focus on one particularly simple pattern of axis-parallel squares, and present an algorithm with an $O(n\sqrt{n})$ running time and $O(n)$ space for this task, involving various bucket-based and sweep-line techniques. Our algorithm's running time is worst-case optimal, as it matches the known lower bound of $\Omega(n\sqrt{n})$ on the maximum number of axis-parallel squares determined by n points in the plane, thereby solving an open question for more than three decades of realizing that bound for this pattern. We extend our result to an algorithm that enumerates all copies, up to scaling, of any full-dimensional fixed set of points in d-dimensional Euclidean space, that runs in time $O(n^{1+1/d})$ and space $O(n)$, matching the more general lower bound due to Elekes and Erdős.

1 Introduction

The problems of geometric point pattern matching and the identification of repeated geometric patterns are fundamental computational problems with a myriad of applications, ranging from computer vision [12, 10], image and video compression [1], model-based object recognition [15], structural biology [11] and even computational chemistry [9]. Such problems were motivated in part by questions regarding the maximal number of occurrences of a given pattern determined by a set of points, a field historically inspired by Erdős' well-known Unit Distance Problem (1946) regarding the maximal number of unit distance pairs induced by such sets [8]. Our paper approaches the computational problems of identifying patterns using tools and techniques encountered in the framework of computational geometry, ensuring exact, provably correct and efficient solutions.

In this paper, we analyze the problem of identifying and listing all translated and scaled copies of *any* point set pattern in Euclidean space, termed homothetic copies of this set, where the scaling is applied identically in all axes. We begin with focusing on the problem of repeated patterns of squares having axis-parallel edges in the plane, where a square is defined by a subset of four points that constitute its vertices. As articulated in 1990 by van Kreveld and de Berg [13], the maximum possible number of axis-parallel squares determined by n points in the plane is $\Theta(n\sqrt{n})$ (attained, for example, in a regular $\sqrt{n}\times\sqrt{n}$ grid), and those can be enumerated in time $O(n\sqrt{n}\log n)$[1] and space $O(n)$ by an algorithm whose extension also treats the enumeration of all full-dimensional axis-parallel d-dimensional hypercubes in d-dimensional Euclidean space in time $O(n^{1+1/d}\log n)$. This exhibits a logarithmic-factor gap separating this computational result from the lower bound of a maximum of $\Theta(n^{1+1/d})$ possible hypercubes, raising the challenge of overcoming this gap as an open question. We remark that in [14], a later journal version of [13], an algorithm for the planar case that works in time $O(n\sqrt{n\log n})$ is presented. However, as the authors point out, its approach does not generalize to higher dimensions.

The combinatorial result from [13] was further extended by Elekes and Erdős [7], establishing a bound of $\Theta(n^{1+1/d})$ on the maximum number of copies of *any* full-dimensional pattern (i.e., a set of points that generates the vector space) in $\mathbb{Q}^d$. The computational aspect of it occurs in [4], providing an algorithm that works in time $O(n^{1+1/d}\log n)$, assuming that the pattern and d are constant, for the task of enumerating all such copies, exhibiting the same logarithmic-factor gap between the two results.

1.1 Our Results

Our main result of this paper is an efficient deterministic algorithm that enumerates all scaled copies of any fixed d-dimensional pattern, for any constant d. The treatment of general patterns appeared, e.g., in [4], but [13] were the first to raise the question of whether it is computationally feasible to realize the combinatorial

*School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel, aya.bernstine@mail.huji.ac.il

†School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel, yehonatan.mizrahi@mail.huji.ac.il

[1]The analysis given throughout this paper of time and space complexities is based on the relatively non-restrictive Pointer Machine model of computation [3], as mentioned later in this paper.

bound of $\Theta(n\sqrt{n})$ possible axis-parallel *squares*, thereby improving their algorithmic result. Our algorithm fully answers this question which was open for more than three decades. To this end, we use in our algorithm a reduction from arbitrary input points to points having "compressed" coordinates, that is, we relabel the coordinates, allowing the use of linear-time sorting methods. Second, we deploy a sweep-line scanning sub-procedure that marks points forming a square, instead of searching those in a set, avoiding the logarithmic cost of searching a point in a set. Third, we relabel the sum and the difference of the input coordinates, in addition to the relabeling of the coordinates themselves. We show why the last step is crucial for the algorithm to succeed in Section 2.

Theorem: *Given a planar set P of points of size n, all axis-parallel squares defined by points from P can be enumerated in time $O(n\sqrt{n})$ and $O(n)$ space.*

Our main result for general patterns relies on the ideas from the previous theorem. Specifically, we relabel some affine transformations of the input coordinates, a relabeling that creates a representation of the points for the purpose of sweep-line scanning them.

Theorem: *Given a fixed set Q of points of full dimension in the d-dimensional Euclidean space, and a set P of points of size n, all scaled copies of Q determined by subsets of P can be enumerated in time $O(n^{1+1/d})$ and $O(n)$ space.*

The running time in this theorem matches the corresponding lower bound of the same magnitude, and improves the best known running time of $O(n^{1+1/d}\log n)$ for the specific case of d-dimensional hypercubes [13], extended later for general arbitrary patterns [4]. Note that although the improvement suggested is by a logarithmic factor, the upshot is an asymptotically *worst-case optimal* algorithm[2] in terms of running time analysis, even for the most general case of arbitrary patterns. This can be compared with [6], where the authors studied the problem of enumerating all *rotated* copies of a given pattern, improving the running time of the trivial algorithm for this companion task by a logarithmic factor as well. An excellent survey that covers this variant of our problem can be found in [2].

Aside from the worst-case optimality of our results, the techniques deployed form a rather general scheme, and may therefore be potentially useful to treat other variants of the problem studied.

[2]For the task of outputting an explicit representation of all copies of the pattern, rather than some other representation of this set of copies, that later needs to be further parsed.

2 Axis-Parallel Squares

In this section, we present an efficient algorithm that reports all axis-parallel squares defined by a planar set of n points. A relatively efficient algorithm, devised by van Kreveld and de Berg [13], works as follows (Note that we refer, for any x_0, to the set of all points whose x coordinate is x_0, as the "column" corresponding to x_0. Moreover, we refer to columns with at most $\sqrt{n}$ points as "short columns").

Squares-Listing$(p_1,\ldots,p_n)$:

1. Build a balanced search tree T and an array A on the input, sorted by the x coordinate.
2. For every pair of points p and q in A residing in a short column, search in T whether they can be complemented to a square from the right or from the left. Report each square found unless the other two vertices defining it are on a short column to the left of p and q.
3. Delete all short columns from T and A, and convert each remaining point (x,y) to (y,x).
4. Apply step 2 on the remaining converted points.

It operates correctly with a running time of $O(n\sqrt{n}\log n)$ and $O(n)$ space, in essence, since the total number of searched points defined in each of the two iterations of step 2 is

$$O\left(\sum_i s_i^2\right) \le O\left(\sum_i s_i\sqrt{n}\right) = \\ = O\left(\sqrt{n}\cdot\sum_i s_i\right) \le O\left(n\sqrt{n}\right)$$

where s_i denotes the length of the i'th column scanned. Every pair is scanned during its course, since there are at most $\frac{n}{\sqrt{n}}$ original long columns (otherwise there are more than n points), so the length of each column in step 4 is at most $\frac{n}{\sqrt{n}} = \sqrt{n}$. We strive for an algorithm with a running time of $O(n\sqrt{n})$ and space $O(n)$. As shown in [13]:

Theorem 1 *(van Kreveld, de Berg) For a set P of n points in d-dimensional space, the maximal number of 2^d points that are subsets of P and that form the vertices of an axis-parallel hypercube is $\Theta(n^{1+1/d})$.*

This theorem induces a lower bound on the running time of the optimal relevant algorithm. Our result bridges the gap between this bound, and the previously best known upper bound.

2.1 Main Ideas Towards an Improvement

Assume that all input points have coordinates in $\{1, \ldots, n\}$. Instead of searching in a set for the *query points* that complement the pair $(x, y), (x, y + \delta)$ to a square, i.e., the points in the pair $(x+\delta, y), (x+\delta, y+\delta)$ and those in the pair $(x - \delta, y), (x - \delta, y + \delta)$, we apply the following procedure: We put all query points along with the original points in an array, apply radix sort on it, treating each point as a two-digit number in base n corresponding to its two coordinates, and then scan and mark all positive query points. That is, we mark each query point adjacent to an existing input point sharing the same coordinates, or to an already marked identical query point. The resulting marked query points define the existing squares.

However, we cannot generally assume that all coordinates are taken from $\{1, \ldots, n\}$. We address this issue by "shrinking" the coordinates of all input points by relabeling their coordinates to values in $\{1, \ldots, n\}$. The main caveat, though, is that arithmetical considerations regarding these labels are invalid, as the proportions are not necessarily preserved after relabeling.

So, we avoid using arithmetic considerations when defining the query points q_1, q_2 that complement the pair $p_1 = (x, y), p_2 = (x, y + \delta)$ to a square (from the right, assuming $\delta > 0$). Instead of using the invalid label $x + \delta$ as a coordinate, we make use of the diagonals by replacing each point (x, y) with $(x, y, x + y, x - y)$ and relabel each of those four coordinates for all points to values in $\{1, \ldots, n\}$. We call the points after this relabeling the *post-labeled points*. Then, the pair q_1, q_2 (with q_2 above q_1) is defined using *identical* labels as those of p_1, p_2. The query point q_1 is defined having the same horizontal y label as p_1 and the same diagonal $x + y$ label as p_2. The point q_2 is treated similarly, only with the second diagonal. Searching in this manner, we can use two out of the four coordinates for each point we search, leaving the other two as wildcards.

Another related observation is that the linear transformation that rotates a vector (x, y) in the plane by 45^o and stretches it by $\sqrt{2}$ yields the vector $(x + y, y - x)$, as illustrated in Figure 1. So, this process is in fact a labeling of the post-rotated points.

2.2 The Efficient Solution

The ideas from the previous subsection lead to our main theorem of this section. We will first describe our algorithm in full detail, and then analyze its correctness and its complexity.

Theorem 2 *Given a planar set P of points of size n, all axis-parallel squares defined by points from P can be enumerated in time $O(n\sqrt{n})$ and $O(n)$ space.*

Proof. The following algorithm is considered:

Amplified-Squares-Listing$(p_1, \ldots, p_n)$:

1. Change the representation of each point $p = (x, y)$ to the representation $(x, y, x + y, y - x)$. Map each x coordinate in the input to a value in $\{1, \ldots, n\}$ according to its ranking, using a sorting algorithm. Perform a similar procedure for the y coordinates, the $x + y$ coordinates and the $y - x$ coordinates. Apply this mapping on the input, to obtain the post-labeled points.

2. Build an array A on the input points, sorted by the x coordinate.

3. For each pair of post-labeled points $p_1 = (x, y_1, w_1, z_1)$ and $p_2 = (x, y_2, w_2, z_2)$ with $y_2 > y_1$, out of the first n pairs of points in A that reside in a short column – construct the query points $q_1 = (*, y_1, w_2, *), q_2 = (*, y_2, *, z_1)$ that complement p_1, p_2 to a square from the right, both pointed by the same pointer R, and the query points $q_1' = (*, y_1, *, z_2), q_2' = (*, y_2, w_1, *)$ that complement to a square from the left, both pointed by the same pointer L. The wildcards replace the unknown coordinates.

4. Place each query point defined by its y and w coordinates in an array B_1 along with all input points, and apply radix sort on B_1 based on those two coordinates. Perform a similar procedure for points of the form of q_2 and q_1' from step 3 in another array B_2.

5. Scan B_1 and mark each query point adjacent to an input point sharing the same coordinates, or to an already marked identical query point. Act similarly on B_2. Scan the list of pointers defined in step 3, and report each square found (a pointer with both points marked), unless the other two vertices defining it are on a short column and complement to a square from the left, to avoid reporting the same square more than once.

6. Perform steps 3-5 iteratively on each subsequent n pairs of points in A in a short column.

7. Delete all points that are on short columns from A. Convert each remaining point (x, y) to (y, x). Apply steps 1-6 on the remaining converted points.

Algorithm's Correctness: Most of the main ideas behind the algorithm's correctness were described in Subsection 2.1. Some other details: All squares having at least one edge on a short column are reported in step 5 of the algorithm, before applying step 7. The rest have both edges on long columns, and so they are reported in step 7.

Given a pair $p_1 = (x_1, y_1)$ and $p_2 = (x_1, y_2)$ with $y_2 > y_1$, the pair $q_1 = (x_2, y_1)$ and $q_2 = (x_2, y_2)$ that

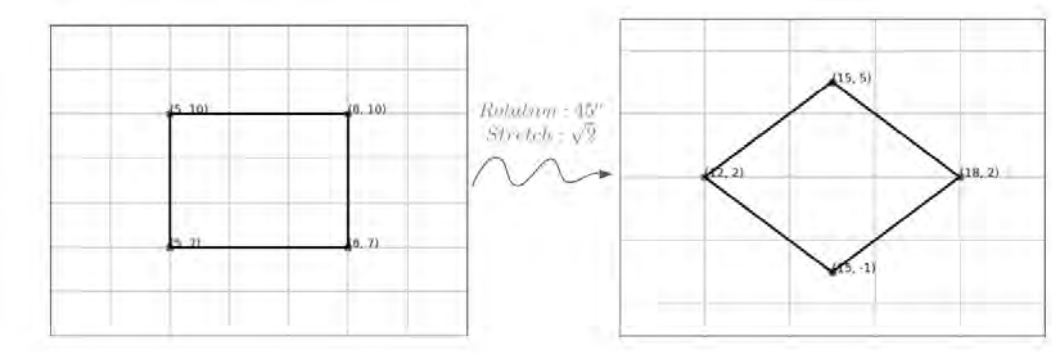


Figure 1: Illustrating the rotation by 45^o and the stretch by a factor of $\sqrt{2}$ applied on four points in the plane. Each point (x, y) was converted to the point $(x + y, y - x)$ as a result.

complements to a square from the right (i.e., $x_2 > x_1$) maintains that $x_2 + y_1 = x_1 + y_2$ since

$$x_2 = x_1 + |y_2 - y_1| \Rightarrow x_2 + y_1 = x_1 + (y_2 - y_1) + y_1 = x_1 + y_2$$

As for q_2, the latter equality also shows that $x_2 - y_2 = x_1 - y_1$ by subtracting $y_1 + y_2$ from both sides. These are exactly the query points defined by the algorithm, up to the labeling that maintains those properties. The analysis for the pair that complements to a square from the left is symmetric.

Algorithm's Complexity: As for the running time, the first two steps of the algorithm cost $O(n \log n)$ using some standard sorting algorithm, e.g., merging sort. Each time step 3 is performed, at most $2n$ query points are constructed in $O(n)$ time. Each time steps 4-5 are performed, two arrays, each of size at most n, are sorted and then scanned in a linear time. Marking the obtained squares in step 5, based on the marked queries, is also carried out in $O(n)$ time by scanning the constructed pointers from step 3. The total number of query points constructed after finishing step 6 is

$$O\left(\sum_i s_i^2\right) \leq O\left(\sum_i s_i \sqrt{n}\right) = \\ = O\left(\sqrt{n} \cdot \sum_i s_i\right) \leq O\left(n\sqrt{n}\right)$$

where s_i denotes the length of the i'th short column, i.e., the number of points in it. As mentioned, each batch of $O(n)$ queries is handled in $O(n)$ time, so the total running time analysis for steps 1-6 of this algorithm is $O(n\sqrt{n})$. The analysis for the converted points in step 7 is symmetric. It only remains to notice that the number of pairs, this time, is

$$O\left(\sum_i d_i^2\right) \leq O\left(\sum_i d_i \sqrt{n}\right) = \\ = O\left(\sqrt{n} \cdot \sum_i d_i\right) \leq O\left(n\sqrt{n}\right)$$

where d_i denotes the length of the i'th row out of the remaining rows, after deleting the short columns. We used the fact that there are at most $\frac{n}{\sqrt{n}}$ long columns, as otherwise there are more than n input points. Therefore, the length of each remaining row, after deletion, is at most $\frac{n}{\sqrt{n}} = \sqrt{n}$, and all points are treated.

As for the space complexity, note that each of the data structures defined in the above algorithm is of size $O(n)$, and that each step involving those structures does not cost more than $O(n)$ space. □

Note that both of these algorithms need not rely on any random-access operation, as no pointer arithmetic or tests on pointers other than equality tests need to be performed. Dereferencing of pointers, along with arithmetic operations on data and comparisons on data are performed, but those are allowed in the Pointer Machine model [3]. The only step which classically involves random access to array cells is the one in which radix sort is used, but even this can be be adjusted to work in the mentioned model ([5]). This statement is true also for the algorithms given in the following sections.

3 Axis-Parallel Hypercubes

This subsection discusses the particular case of axis-parallel hypercubes in d-dimensional Euclidean space. Although following immediately from Theorem 3 given in the following section, it presents some of the ideas behind it in a clearer manner, and can serve as a warm-up for that theorem. We provide an algorithm with a

running time complexity of $O(n^{1+1/d})$ and with a linear space complexity, addressing the open question of matching the lower bound from Theorem 1. Our algorithm builds on the techniques and the observations from Section 2, with some additions and adjustments so it complies with the properties of the d-dimensional space.

One observation that is true for the d-dimensional case, is that any two points with all but one equal coordinate, determine 2^{d-1} full-dimensional possible hypercubes. Denote such a pair of points by t and r. Each of these hypercubes is uniquely associated with a vector $e \in \{-1,1\}^{d-1}$, in which the j'th coordinate determines the direction of progress from t and r along the j'th axis, where j is any coordinate except the one in which they differ. In a similar fashion to the planar case, we would like to relabel the input coordinates, their sums and their differences, place them in an array, radix sort it and mark the correct vertices that complement to a hypercube in an efficient manner, while scanning this array. Moreover, we scan only pairs of points lying on short axis-parallel lines, similarly to the planar case, only that this time, by "short" we mean having not more than $n^{1/d}$ point on it.

Proposition: Given a set P of points of size n in d-dimensional Euclidean space, all axis-parallel full-dimensional hypercubes defined by points from P can be enumerated in time $O(n^{1+1/d})$ and $O(n)$ space.

Proof. The following algorithm establishes the proposition's statement:

Amplified-Hypercubes-Listing$(p_1, \ldots, p_n)$:

1. For each input point $p = (x_1, x_2, \ldots, x_d)$, add the following additional list of coordinates:

   $$((x_i - x_j), (x_i + x_j) \;\mid\; \forall 1 \le i < j \le d)$$

   Map each of those augmented coordinates, including the original ones, to a label in $\{1, \ldots, n\}$.

2. Build an array A on the input points, sorted by each of their coordinates based on the coordinates' order, except for the last *original* coordinate (i.e., x_d).

3. For each pair of points t, r that lie in the same short axis-parallel line, having the same coordinates except for the last, out of the first n pairs with this property, add $2^d - 2$ query points which define together a hypercube. Do this for all 2^{d-1} possible hypercubes in the following manner. First, any axis-parallel hypercube having t and r as its vertices, is defined using one additional vertex

   $$r' = (r_1 + e_1 \cdot \delta, \ldots, r_{d-1} + e_{d-1} \cdot \delta, t_d)$$

   where r_i is the i'th coordinate in r (and similarly for t), $\delta = r_d - t_d$ and $e \in \{-1,1\}^{d-1}$. The rest of the vertices in each such hypercube are defined similarly, except for replacing all subsets of the coordinates in the vector e by zeros, and using t instead of r.

   Now, define the coordinates that are to be searched – *not* in the aforementioned arithmetic manner, but using the labels from step 1 instead. That is, translate $r_i + e_i \cdot \delta$ to the label of $r_i + r_d$ if $e_i = 1$, and to that of $r_i - r_d$ otherwise. Fill in the unknown coordinates using wildcards, as those are uniquely defined anyway, given the others.

4. Place all query points defined by the same coordinates in an array along with all input points. Apply radix sort on each of those arrays, according to the known coordinates in it.

5. Scan each array from step 4, and mark each query point adjacent to an input point sharing the same coordinates, or to an already marked identical query point. Report all hypercubes that were found (by checking that all vertices are present for each hypercube), except for hypercubes that have two vertices on a short axis-parallel line of the same type, only with a smaller index.

6. Perform steps 3-5 on each subsequent n pairs of points in A on a short axis-parallel line of that type.

7. Delete all points that are on a short axis-parallel line of a currently analyzed type from A, and convert each remaining point $(x_1, x_2, \ldots, x_d)$ to $(x_d, x_1, \ldots, x_{d-1})$. Apply steps 1-6 on the remaining converted points. This step is carried out $d-1$ times.

Algorithm's Correctness: Almost all details regarding the analysis of the correctness of this algorithm already appeared in that of Amplified-Squares-Listing$(p_1, \ldots, p_n)$. As for the phase of searching by labels, note that if r'_j is some unknown coordinate, for which we only have an undesired arithmetic definition based on the coordinate r_j and on δ, then it holds that

$$r'_j = r_j + \delta = r_j + (r_d - t_d) \Longrightarrow r'_j + t_d = r_j + r_d$$

which exactly corresponds to the labels that the algorithm searches (the treatment of positive or negative values of δ is symmetric), and similarly for subtraction.

Algorithm's Complexity: The running time analysis is similar to the running time analysis of Amplified-Squares-Listing$(p_1, \ldots, p_n)$ with the following differences. The relabeling in step 1 and the sorting of A in step 2; the definitions of 2^{d-1} hypercubes, each consisting of additional $2^d - 2$ points in step 3; applying

radix sort on $2^{O(d)}$ arrays in step 4; the linear scanning of those arrays in step 5 and applying step 7 for $d-1$ times – all of those involve multiplying the complexity of the planar case by at most a constant factor of $2^{O(d)}$. The only major difference concerns the total number of query points defined each time step 7 in the algorithm is invoked. In the treatment of the first $d-1$ axis-parallel lines, only short lines are considered, so the cost for each line is

$$O\left(\sum_i s_i^2\right) \leq O\left(\sum_i s_i \cdot n^{1/d}\right) = \\ = O\left(n^{1/d} \cdot \sum_i s_i\right) \leq O\left(n^{1+1/d}\right)$$

where s_i denotes the length of the i'th short axis-parallel line of that form. Regarding the treatment of the last axis-parallel line, we note that all remaining such lines are short. Assume that there exists a long remaining line. Then all points on it are on long axis-parallel lines, with respect to some axis, which induces more points that are on long lines with respect to another axis, yielding that there are more than $\left(n^{1/d}\right)^d$ input points in total, which is obviously a contradiction. Thus, all input points are treated with the mentioned running time, so the total running time analysis is indeed $O(n^{1+1/d})$. The analysis of the space complexity is similar, with a constant multiplicative factor of $2^{O(d)}$ compared to the analysis of the planar case. Note that although being exponential in d, as also occurs in the solution from [13], the running time can be regarded as polynomial in d and the size of the pattern, as presented in Section 4. □

4 The General Case

In this section, we describe an algorithm that enumerates all scaled copies of any fixed arbitrary full-dimensional pattern in d-dimensions. For general fixed patterns, where d is fixed, our algorithm works in time $O(n^{1+1/d})$ and $O(n)$ space. This answers the open question of realizing the lower bound of [7].

Theorem 3 *Given a fixed set Q of points of full dimension in the d-dimensional Euclidean space, and a set P of points of size n, all scaled copies of Q determined by subsets of P can be enumerated in time $O(n^{1+1/d})$ and $O(n)$ space.*

Proof. We first assume that no three points in Q are on the same line, and present an appropriate algorithm for this case. Then we describe how this algorithm can be adjusted to handle the more general case.

Amplified-Patterns-Listing$(p_1, \ldots, p_n)$:

1. Rotate the pattern points such that two of them, p and q, share afterwards all coordinates except the last one. Apply this rotation on the input points.
2. For each point $r \neq p, q$ in Q, compute $d-1$ hyperplanes of dimension $d-1$ that include p and r, and an additional hyperplane including q and r, altogether defining r uniquely. Apply each of the $d \cdot |Q|$ transformations corresponding to those hyperplanes on each input point, attach those values to the original points' list of coordinates, and label the resulting values – the augmented coordinates (original coordinates along with those corresponding to the transformations) using $\{1, \ldots, n\}$.
3. Build an array A on the input points, sorted by all original coordinates by their order.
4. Scan A. For each pair r and t on an axis-parallel line that corresponds to step 1 that also has at most $n^{1/d}$ points ("short" line), construct the rest of the $|Q|-2$ points that complement to a pattern using the labels obtained from step 2, until constructing n such sets of queries. Point each such set that corresponds to a single copy by the same pointer.
5. Place all query points that are defined by the same augmented coordinates in an array with all input points, forming several such arrays. Apply radix sort on each such array.
6. Scan each array from step 5, and mark each query point adjacent to an identical input point or an already marked query point. Scan the list of pointers defined in step 4, and report each copy found (a pointer with all points marked), only after applying on it the rotation which is inverse to that of step 1.
7. Perform steps 4-6 on each subsequent n pairs of points in A of the form of step 4.
8. Apply steps 2-7 for each pair among the pattern points that determines a line parallel to that through p and q, excluding enumeration of duplicate copies (similarly to the identification of duplicate squares, i.e., using an appropriate ordering). Delete all points on those "short" lines from step 4. Apply steps 1-7 on the remainder, for a different pair of points from the pattern, and perform this $d-1$ times.

As for the case where at least three points from Q are collinear: If $d \geq 2$, then in step 2 of the above algorithm, if the point r lies on the line that goes through p and q, it is not uniquely defined as the intersection of a line that goes through p and a hyperplane of dimension $d-1$ that goes through q. However, since Q is full-dimensional, there is another point r' which is not on

that line. So, instead, define r as the intersection of a line that goes through p and a hyperplane of dimension $d-1$ that goes through r', and label the additional extra coordinate that corresponds to r' in the augmented form. However, note that in step 4 of the above algorithm we scan input points that correspond to p and q, and not to r', yet for each such a pair (that corresponds to p and q) we still need to know the value of the corresponding extra coordinate of r' which is associated with them, and is unknown at that moment. This is bypassed by first defining the query points that corresponds to r' for each such a pair, then marking all of the positive r' query points that also exist in the input, and only then defining the query points that corresponds to r based on the r' coordinate fetched during the scanning process.

If $d = 1$, one can use a completely different approach than that described earlier. We sort the input points on the line, and then place $|Q|$ pointers on the $|Q|$ leftmost input points. If the point which is pointed by the third pointer is too close to the second one in terms of the proportions from Q, increment the third pointer. If it is too far, increment to second pointer. If they align correctly, increment the rest of the pointers until either a copy of the pattern is found, or until one of the pointers is too distant. Then continue and advance the second pointer, and proceed similarly. In this manner, for each leftmost point, the rest of the pointers only advance forward with a cost of $O((|Q|-1)\cdot n)$. Since this is performed n times, the desired running time of $O(n^2)$ is obtained.

Analysis: The main ideas behind the correctness of Amplified-Patterns-Listing$(p_1,\ldots,p_n)$ already appeared in Section 2. Aside from those ideas, note that step 2, in fact, defines each point as the intersection of a line and a $(d-1)$-dimensional hyperplane, and under the assumption that no three points are on the same line, the points are uniquely defined in that manner.

As for the running time, note that there is no remaining long line analyzed at the ultimate iteration. Otherwise, all points on it are on another long line defined by a linearly independent vector. This induces more points on a different long line, and so forth, yielding that there are more than $\left(n^{1/d}\right)^d = n$ input points, a contradiction. Other than that, we did not need the lines to be axis-parallel, but rather merely that the corresponding vectors form an independent set.

Compared to the squares or the hypercubes case, steps 1-3 cost $O(poly(d\cdot|Q|)\cdot n\log n)$ using a sorting algorithm; constructing $O(n)$ sets of queries in step 4 costs $O(poly(d\cdot|Q|)\cdot n)$ and constructing $O(|Q|)$ arrays of size $O(n)$ in step 5 has a similar running time; applying radix sort and then scanning and marking those arrays also cost $O(poly(d\cdot|Q|)\cdot n)$ (this is multiplied by $|Q|$ due to pairs among the pattern points which are parallel). As mentioned, only short lines are scanned during the algorithm's course, and since each batch of size n costs $O(poly(d\cdot|Q|)\cdot n)$, then each dimension costs $O(poly(d\cdot|Q|))$ multiplied by

$$O\left(\sum_i s_i^2\right) \le O\left(\sum_i s_i\cdot n^{1/d}\right) = \\ = O\left(n^{1/d}\cdot\sum_i s_i\right) \le O\left(n^{1+1/d}\right)$$

where s_i denotes the length of the i'th short line of that form. This is multiplied by d iterations, and results in a running time of $O(poly(d\cdot|Q|)\cdot n^{1+1/d})$, which is $O(n^{1+1/d})$ under our assumptions. Space complexity is linear due to similar arguments to those above, and to those presented in Section 2. □

5 Conclusion and Further Work

In this paper, we analyzed the problem of enumerating all scaled copies of a pattern in a set of n points in time $O(n^{1+1/d})$, answering open questions from [13] and [4] by realizing the lower bound due to Elekes and Erdős [7]. We relied on some existing ideas, amplified using bucket-based methods, sweep-line scanning and more. As far as we are aware of, the combinations of these techniques this way was not noted in the literature so far for similar tasks. One open question is whether these techniques can be adjusted for different pattern matching problems. Other questions include comparing the task of finding one copy of a pattern with the task of enumerating all copies of it ([13] show a separation between those for d-dimensional boxes), and similarly for the task of counting the number of copies instead of outputting them. In addition, the existence of an output-sensitive algorithm for our problem, and the existence of an efficient enumeration algorithm for patterns not of a constant size, form another two open questions for further research.

References

[1] M. Alzina, W. Szpankowski, and A. Grama. 2d-pattern matching image and video compression: theory, algorithms, and experiments. *IEEE Transactions on Image Processing*, 11(3):318–331, 2002.

[2] D. Avis, A. Hertz, and O. Marcotte. Graph theory and combinatorial optimization. *Springer Science & Business Media*, 2005.

[3] A. Ben-Amram. What is a "pointer machine"? *ACM SIGACT News*, 26(2):88-95, 1995.

[4] P. Braß. Combinatorial geometry problems in pattern recognition. *Discrete & Computational Geometry*, 28:495–510, 2002.

[5] A. Buchsbaum, H. Kaplan, A. Rogers, and J. Westbrook. Linear-time pointer-machine algorithms for least

common ancestors, mst verification, and dominators. *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, pages 279–288, 1998.

[6] P. de Rezende and D. Lee. Point set pattern matching in d-dimensions. *Algorithmica*, 13:387–404, 1995.

[7] G. Elekes and P. Erdős. Similar configurations and pseudo grids. *Intuitive geometry*, 63:85–104, 1994.

[8] P. Erdős. On sets of distances of n points. *American Mathematical Monthly*, 53:248–250, 1946.

[9] P. Finn, L. Kavraki, J. Latombe, R. Motwani, C. Shelton, S. Venkatasubramanian, and A. Yao. Rapid: Randomized pharmacophore identification for drug design. *Computational Geometry: Theory and Applications*, 10:324–333, 1997.

[10] D. Mount, N. Netanyahu, and J. L. Moigne. Efficient algorithms for robust feature matching. *Pattern Recognition*, 32(1):17–38, 1999.

[11] R. Norel, D. Fischer, H. Wolfson, and R. Nussinov. Molecular surface-recognition by a computer vision-based technique. *Protein engineering*, 7:39–46, 1994.

[12] G. Schindler, P. Krishnamurthy, R. Lublinerman, Y. Liu, and F. Dellaert. Detecting and matching repeated patterns for automatic geo-tagging in urban environments. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7, 2008.

[13] M. van Kreveld and M. de Berg. Finding squares and rectangles in sets of points. In *Graph-Theoretic Concepts in Computer Science*, 1990.

[14] M. van Kreveld and M. de Berg. Finding squares and rectangles in sets of points. In *BIT Numerical Mathematics*, 31(2):202–219, 1991.

[15] V. Zografos and B. Buxton. Affine invariant, model-based object recognition using robust metrics and bayesian statistics. In *Kamel M., Campilho A. (eds) Image Analysis and Recognition. ICIAR 2005. Lecture Notes in Computer Science*, volume 3656, 2005.

A Sub-quadratic Time Algorithm for the Proximity Connected k-center Problem on Paths via Modular Arithmetic

Binay Bhattacharya* Tsunehiko Kameda* Amirhossein Mozafari*†

Abstract

The k-center problem is one of the most well-known problems in combinatorial optimization which has been extensively studied in the past. In this paper, we introduce a generalized version of the k-center problem called *proximity connected k-center (PCkC)* problem. In this problem, we are given a set of demand points in a metric space and a parameter $\delta > 0$. We are going to locate k center points such that the maximum distance of a demand point to its nearest center is minimized and each pair of centers can communicate with each other either directly or via other centers assuming that each center can directly communicate with any other center within the range of δ of itself. Note that when δ is large enough, the problem turns to the k-center problem and when δ tends to zero, the problem turns to the 1-center problem. We consider the PCkC problem when the underlying space is a path and present a sub-quadratic time algorithm for both the unweighted and the weighted demand points cases.

1 Introduction

The k-center problem is one of the most important facility location problems which has been extensively studied in the past [4, 6, 10, 11, 16]. In this problem, we are given a set of n demand points $U = \{v_1 \ldots, v_n\}$ in a metric space such that each demand point $v_i \in U$ has a non-negative weight w_i. The objective is to find a k-center (a set of k points in the space) C such that $cost(C) := \max_{v_i \in U}\{w_i d(v_i, C)\}$ is minimized, where $d(v_i, C) := min_{c \in C} d(v_i, c)$ (here $d(v_i, c)$ is the distance between v_i and c in the space). We call this minimum cost the *optimal cost* for the problem. If we have unit weights on all demand points, the problem is called *unweighted*. We say that a k-center C satisfies the *proximity connectedness condition (PCC)* with respect to a parameter $\delta > 0$ if the δ-distance graph of C is connected (the δ-distance graph of C is a graph with the vertex set C such that there is an edge between c_1 an c_2 in C if and only if $d(c_1, c_2) \leq \delta$). In the *proximity connected k-center (PCkC)* problem, in addition to U, we are also given a parameter $\delta > 0$ and we are going to find a k-center with the minimum cost that satisfies the PCC.

In practice, if we consider the centers as facility locations, the parameter δ can represent the range for which, each facility can directly communicate with any other facility within the range δ of itself. So, if the centers satisfy the PCC, each pair of facilities can communicate with each other directly or via other facilities. For example, suppose that we need to locate k communication/control equipment to observe n sensors while the equipment need to send/receive messages between themselves (directly or via other equipment). Also, each equipment can safely send/receive data with any other equipment within the range δ of itself. The problem of locating the equipment as close as possible to the sensors can be modeled as PCkC problem in the plane.

Note that if δ is sufficiently large, the problem reduces to the k-center problem which is known to be NP-hard in both the plane and metric graphs [6, 13] (a metric graph is a graph for which each of its edges has a length and the lengths satisfy the triangular inequality). This implies that the PCkC problem is also NP-hard in the plane and metric graphs and so it is not possible to solve it efficiently. In [6], Kariv and Hakimi showed that the k-center problem can be solved in polynomial time when the underlying space is a metric tree and gave an $O(n^2 \log n)$ time algorithm for the problem. In 1991, Frederickson [4, 5] showed that the unweighted k-center problem can be solved in linear time in trees. Finally, in 2018, Wang and Zhang [16] provided an $O(n \log n)$ time algorithm for the k-center problem in trees. The PCC condition first appeared in the context of wireless networks in 1992 [7]. Later, Huang and Tsai studied the 2-center problem in the plane, considering the proximity condition between the centers [8, 9]. As another work, in 2022, Bhattacharya et al. [2] presented an $O(n^2 \log n)$ time algorithm to solve the proximity connected 2-center problem in the plane improving the previous algorithm for the problem with $O(n^5)$ time complexity [7]. Although there are some related works in the context of theory of wireless sensor networks [1, 14], the k-center problem has not been studied when we have the proximity condition between the centers. In this paper, we address this problem by providing a sub-quadratic time algorithm for the k-center problem on paths having the PCC.

2 PCkC Problem for Unweighted Paths

Let $P = (v_1, \ldots, v_n)$ be the given unweighted path (consisting of both the vertices and the edges between them) such that the vertices lie on the x-axis from left to right and v_1 lies on the origin. Without loss of generality, we assume that n is a power of 2. Also, we use the notation v_i $(1 \leq i \leq n)$ for both the vertex itself and the

*School of Computing Science, Simon Fraser University
†amozafar@sfu.ca, corresponding author

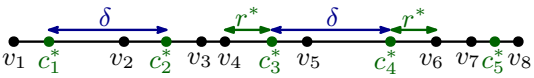


Figure 1: An example of the PCkC problem on a path with 8 vertices.

x-coordinate of the vertex. Thus, we have an order on the vertices based on their x-coordinates. Also, if $v_i < v_j$, we denote the interval between v_i and v_j on the x-axis by $[v_i, v_j]$. In this section, we are going to find a k-center C^* for P such that C^* satisfies PCC and,

$$cost(C^*) = min\{cost(C) : C \text{ is a } k\text{-center for } P \text{ and satisfies PCC}\}$$

We call C^* an *optimal solution* and its cost the *optimal cost*. We denote the optimal cost by r^* and the centers in C^* by $(c_1^*, \ldots, c_k^*)$ from left to right on the x-axis. Figure 1 shows an example of the PCkC problem on a path and its corresponding optimal solution.

The idea of obtaining an optimal solution for the problem is first computing r^* and then, using it to build an optimal solution. In order to do that, we first design a feasibility test for the problem which gets a value $r \geq 0$ and determines whether it is feasible ($r \geq r^*$) or infeasible ($r < r^*$). Algorithm UPATH-FT presents such a feasibility test for the unweighted PCkC problem on a path. Note that if $r \geq r^*$, UPATH-FT(P,r) also gives us a k-center with a cost at most r. Using the feasibility test, we can check whether $r^* = 0$. In this case the trivial solution is putting a center at each vertex. So henceforth, we assume that $r^* > 0$. Note that in Algorithm 1, the vertices in V are

Algorithm 1 UPATH-FT(P, r)

```
1: Set Counter = 1 and V = (v_2, ..., v_n).
2: Put a center at x_c = r.
3: while there is an element in V do
4:     Eliminate all vertices v ∈ V with d(x_c, v) ≤ r.
5:     Put a center at x_c = min{x_c + δ, V[1] + r}.
6:     Counter = Counter + 1.
7:     if Counter > k then
8:         return infeasible.
9:     end if
10: end while
11: return feasible.
```

eliminated in order and so the time complexity of UPATH-FT would be $O(n + k)$. It is important to mention that we might have more than one optimal solution for a given problem instance but, having r^* (which is unique), the algorithm UPATH-FT gives us a unique optimal solution. In order to avoid confusion, henceforth we exclusively use the notation C^* for this optimal solution. We say that a vertex v is *covered* by a center $c_i^* \in C^*$ if $d(v, c_i^*) = d(v, C^*)$. Also, $d(v, c_i^*)$ is called the cost that c_i^* induces on v. We say that a sequence of t points $(c_1, \ldots, c_t)$ (the order is left to right on the x-axis) is a *t-train* if $\forall\ 1 \leq i < t,\ d(c_i, c_{i+1}) = \delta$.

Proposition 1 *There exists a pair of vertices (v_i, v_j) such that the subset $C' \subseteq C^*$ of centers in $[v_i, v_j]$ is a t-train (for some t) and $d(v_i, C') = d(v_j, C') = r^*$.*

The reason of the above proposition is that if such a pair does not exist, for any vertex v with $d(v, C^*) = r^*$, we can move the covering center of v (and possibly other centers to ensure the PCC) towards v to get a solution with a cost smaller than r^*, which contradicts the optimality of r^*. We call any pair (v_i, v_j) satisfying the condition of Proposition 1, a *determining pair* for the problem.

Proposition 2 *If $d(v_1, v_n) \geq k\delta$, then (v_1, v_n) is a determining pair for the problem.*

Proof. For any vertex v in $[c_1^*, c_k^*]$, $d(v, C^*)$ should be at most $\delta/2$ because of the PCC. So, if $d(v_1, v_n) \geq k\delta$, the cost of C^* should be greater than or equal to $\delta/2$ which means that (v_1, v_n) is a determining pair. □

Based on the above proposition, if $d(v_1, v_n) \geq k\delta$, we have $d(c_1^*, c_k^*) = (k-1)\delta$ and $d(v_1, c_1^*) = d(c_k^*, v_n)$. Therefore, $r^* = (d(v_1, v_n) - k\delta)/2$. Now, UPATH-FT($P, r^*$) will give us C^*. Henceforth in this section, we assume that $d(v_1, v_n) < k\delta$ and so $0 < r^* < \delta/2$ (because of the PCC). In order to find r^*, we build a set of candidate values $\mathcal{C}$ and iteratively use the feasibility test to discard its values until r^* becomes clear. Consider a pair of vertices (v_i, v_j) and a t-train T such that $d(v_i, v_j) > (t-1)\delta$. We say that T is *fitted* in $[v_i, v_j]$ if $d(v_i, T) = d(v_j, T)$. Note that if T is fitted in $[v_i, v_j]$, the *induced cost* of T on v_i and v_j would be $(d(v_i, v_j) - (t-1)\delta)/2$ and is denoted by $IC_t(v_i, v_j)$. If $d(v_i, v_j) \leq (t-1)\delta$, we say that (v_i, v_j) does not accept a t-train. Note that any pair of vertices accepts 1-train which is indeed the mid-point of the connecting segment of v_i and v_j. Based on Proposition 1, the set of candidate values $\mathcal{C}$ can be considered as follows:

$$\mathcal{C} = \{IC_t(v_i, v_j)\ :\ (v_i, v_j) \text{ accepts a } t\text{-train}\}$$

Because each pair of vertices can *generate* up to $O(k)$ candidate values, the size of $\mathcal{C}$ would be $O(n^2k)$. A naive algorithm to find r^* is computing the entire $\mathcal{C}$, then sort it and perform binary search using the feasibility test to find r^*. It is easy to see that the time complexity of this approach is $O(n^2k \log(n + k))$. In the rest, we show that how we can reduce this bound and get a sub-quadratic algorithm but before, it is useful to discuss about the geometric interpretation of the candidate values.

Geometric View: Let L_i and R_i be two half-lines from v_i with angles $\pi/4$ and $3\pi/4$ with the positive direction of the x-axis respectively. Note that the y-coordinate of the intersection of a vertical line at point x with $L_i \cup R_i$ is the cost that a center at x will induce on v_i (this is because we assumed that the vertices are unweighted). Based on this observation, for a pair (v_i, v_j), $IC_1(v_i, v_j)$ is the y-coordinate of the intersection point of R_i and L_j. Furthermore, if (v_i, v_j) accepts a t-train, $IC_t(v_i, v_j)$ would

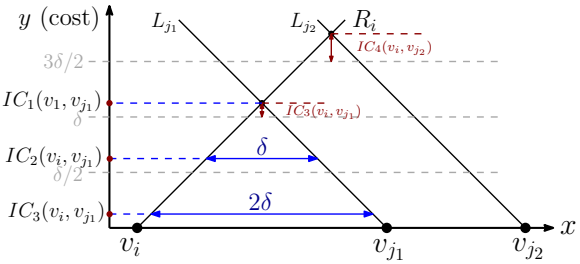


Figure 2: The geometric view of the candidate values generates by (v_i, v_{j_1}) and the effective candidate value generated by (v_i, v_{j_2}).

be the y-coordinate of the horizontal segment with length $(t-1)\delta$ with sides on R_i and L_j (see Figure 2). Based on this geometric view, the following observation can be concluded:

Observation 1 *If (v_i, v_j) accepts a t-train ($t > 1$) then $IC_t(v_i, v_j) = IC_{t-1}(v_i, v_j) - \delta/2$.*

Consider a pair (v_i, v_j) and the non-zero candidate value $IC_{k'}(v_i, v_j)$ such that either $k' = k$ or (v_i, v_j) does not accept a $(k'+1)$-train (equivalently, k'-train is the longest train that can be fitted in (v_i, v_j)). According to Observation 1, $IC_{k'}(v_i, v_j)$ is the only candidate value that (v_i, v_j) can generate in $(0, \delta/2)$. If (v_i, v_j) generates a candidate value in $(0, \delta/2)$, we call this candidate value an *effective candidate value*. Because $r^* \in (0, \delta/2)$, we only need to search the effective candidates generated by the pairs in P in order to find r^*. Let us gather all the effective candidates into an $n \times n$ matrix M such that $M[i, j]$ is the effective candidate value generated by (v_i, v_j) if $i < j$ and zero otherwise. We can see that M is not a sorted matrix because for a fixed i, by increasing j, the number of centers in the train that induces $M[i, j]$ might change. Indeed, this is the main obstacle to get a linear time algorithm like [4, 5] for the unweighted PCkC problem. Precisely, the k-center problem is equivalent to the PCkC problem when $\delta = \infty$. In this case, all the effective candidates are generated by 1-trains. The key point here is that the effective cost generated by a 1-train on a pair (v_i, v_i) is an increasing function of $d(v_i, v_j)$. This monotonicity makes the matrix M sorted which plays a pivotal role in obtaining a linear time algorithm.

In order to search M in a sub-quadratic time, we define an auxiliary matrix $\bar{M}$ such that applying the feasibility test on its elements enables us to discard the elements of M in an efficient way. We define $\bar{M}$ as an $n \times n$ matrix such that:

$$\bar{M}[i, j] = max\{M[i, j'] : i < j' \leq j\}$$

Note that $\bar{M}$ is a row sorted (increasing) matrix but may not be sorted column-wise. We define the remainder function $rem_\delta(x)$ as follows:

$$rem_\delta(x) = x - \left\lfloor \frac{x}{\delta} \right\rfloor \times \delta$$

Observation 2 *If $i < j$, then we would have $M[i, j] = rem_\delta(d(v_i, v_j))/2$.*

This is from the fact that the size of the portion of $[v_i, v_j]$ not covered by the longest train in the interval is $rem_\delta(d(v_i, v_j))$.

Proposition 3 *If $M[i, j] = r^*$ then for all $i < j' < j$, $M[i, j'] \leq r^*$.*

Proof: We proceed by contradiction. Suppose that $M[i, j] = r^*$ and $\exists j' : i < j' < j$ such that $M[i, j'] > r^*$. Let $C' = (c^*_{h_1}, \ldots, c^*_{h_2}) \subseteq C^*$ be the train in $[v_i, v_j]$ that induces r^* on v_i and v_j. Also, let $C = (c_1, \ldots, c_q)$ be the longest train that can be fitted in $[v_i, v_{j'}]$ that induces the cost $M[i, j']$. Note that $|C| < |C'|$, otherwise because $v'_j < v_j$, $M[i, j']$ could not be greater than $M[i, j]$. Note that $c^*_{h_1} < c_1$ because we assumed $M[i, j'] > r^*$. Now, if $c^*_{h_1+q} < v_{j'}$, we can fit a $(q+1)$-train in $[v_i, v_{j'}]$, which contradicts the way we chose C. So, let us assume that $c^*_{h_1+q} > v_{j'}$ (see Figure 3).

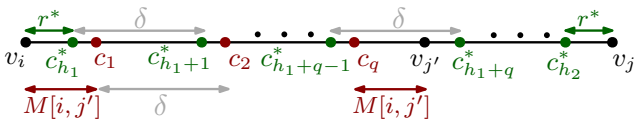


Figure 3: Proof of Proposition 3.

Here, $c^*_{h_1+q}$ is the center that covers $v_{j'}$ in C^*. If $d(v_{j'}, c^*_{h_1+q}) = r^*$, $d(v_i, v_{j'})$ would be a multiple of δ and so $M[i, j'] = 0$ which is against our assumption that $M[i, j'] > r^*$. Thus, we have $d(v_{j'}, c^*_{h_1+q}) < r^*$ but in this case we can fit a $(q+1)$-train in $[v_i, v_{j'}]$ which is a contradiction. □

Example: In Figure 4, the fitted 4-train $(c^*_1, \ldots, c^*_4)$ between v_1 and v_j induces the optimal cost r^* for the problem. In order to have $M[i, j'] > r^*$ for some $1 < j' < j$, $v_{j'}$ should lie on a *forbidden region*, which are the set of points with distances greater than r^* to their closest center (these regions are specified in red in Figure 4).

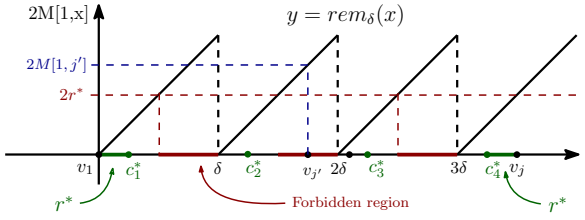


Figure 4: An example for Proposition 3.

Observation 3 *By applying the feasibility test on $\bar{M}[i, j]$, one of the following cases will happen:*

1. *$\bar{M}[i,j]$ is feasible. In this case, we can discard all $M[i,j']$ with $j' > j$ (based on Proposition 3).*

2. *$\bar{M}[i,j]$ is infeasible. In this case, we can discard all $M[i,j']$ with $j' \leq j$ (based on the definition of $\bar{M}$).*

Note that in the part 1 of the above observation, when $\bar{M}[i,j]$ is feasible, then either $\bar{M}[i,j] > r^*$ or $\bar{M}[i,j] = r^*$. For the former case, if $M[i,j'] = r^*$ for some $j' > j$, it contradicts Proposition 3 and for the later case we still have r^* in our undiscarded values. According to the above observation, we can find r^* by iteratively applying the feasibility test on the elements of $\bar{M}$ and discard the elements of M until r^* becomes clear. Algorithm DISC-ROUND(M) shows how we can discard $1/4^{th}$ of the undiscarded elements in M at each iteration. We can see that at the beginning of each iteration the undiscarded elements of each row make a connected region. We call this region the *undiscarded region*. Because $\bar{M}$ is row sorted, if d_1 and d_2 are the first and the last indices of the undiscarded region of an i^{th}-row in M, if we know whether $\bar{M}[i, d_1 + \lfloor (d_1+d_2)/2 \rfloor]$ is feasible, we can discard half of the elements in the region. Note that in Algorithm 2, the variables d_1, d_2 and w_i can be

Algorithm 2 DISC-ROUND(M)

for i from 1 to n **do**
  Set d_1, d_2 and n_i as the first index, the last index and the number of elements in the undiscarded region of the i^{th}-*row* of M respectively.
  Set m_i as $\bar{M}[i, d_1 + \lfloor (d_1 + d_2)/2 \rfloor]$.
end for
Compute the weighted median m of $\{m_i : 1 \leq i \leq n\}$ where m_i has weight n_i.
Run UPATH-FT(P,m).
if m is feasible **then**
  For each i with $m_i \geq m$, discard $M[i,j'] : j' > m_i$.
else
  For each i with $m_i \leq m$, discard $M[i,j'] : j' \leq m_i$.
end if

updated after the discarding phase of the previous iteration (so we don't need to search the entire matrix to compute them at the beginning of the current iteration). Also, we compute the weighted median of the mid-indexes of the undiscarded region of the rows because at the beginning of an iteration, the undiscarded region of the rows in M may not have the same size. We can see that in each iteration, we need to compute the median of $O(n)$ values in $\bar{M}$. The bottleneck of the time complexity of DISC-ROUND is the cost of obtaining an element of $\bar{M}$. Precisely, if the time complexity of computing an element of $\bar{M}$ is $O(g(n))$, then the total time complexity of DISC-ROUND would be $O(ng(n)+k)$ and so the overall time complexity of our algorithm for the unweighted PCkC problem on paths would be $O((ng(n)+k)\log n)$ (because we have $O(\log n)$ iterations). In the next subsection, we discuss how we can compute an element of $\bar{M}$ efficiently.

2.1 Computing an Element of $\bar{M}$

In this subsection, we provide a preprocessing phase that enables us to compute $\bar{M}[i,j]$ in sub-linear time. Let $\mathcal{M}_{i,j} = \{M[i,i+1], \ldots, M[i,j]\}$ and so, $\bar{M}[i,j] = max\ \mathcal{M}_{i,j}$. We first build a balanced binary tree $\mathcal{T}$ on top of the vertices in P (we assumed that n is a power of 2). Thus, each leave of $\mathcal{T}$ corresponds to a single vertex. For a node $\nu \in \mathcal{T}$, $span(\nu)$ is defined as the set of vertices that have ν as a common ancestor. Note that the root of $\mathcal{T}$ spans the entire P. Also, we denote the first and the last indexes of the vertices in $span(\nu)$ by $left(\nu)$ and $right(\nu)$ respectively. In each node $\nu \in \mathcal{T}$, we store the sequence $\sigma(\nu)$ obtained from sorting $\{2M[v_1, v] : v \in span(\nu)\}$ increasingly. It is easy to see that the time complexity of building $\mathcal{T}$ and the sequences in its nodes is $O(n \log n)$.

Observation 4 *For any two numbers a and b, we have:*

$$rem_\delta(a+b) = rem_\delta(rem_\delta(a) + rem_\delta(b))$$

Based on the above observation and Observation 2, for any $j' \geq i$ we can write $M[i,j']$ as:

$$\begin{aligned} &M[i,j'] = rem_\delta(d(v_i, v_{j'}))/2 = \\ &rem_\delta(d(v_1, v_{j'}) - d(v_1, v_i))/2 = \\ &rem_\delta(rem_\delta(d(v_1, v_{j'})) - rem_\delta(d(v_1, v_i)))/2 = \\ &rem_\delta(2M[v_1, v_{j'}] - 2M[v_1, v_i])/2 \end{aligned}$$

Now, for each vertex ν with $\sigma(\nu) = (s_1, \ldots, s_t)$ and $i \leq left(\nu)$, we define $\sigma_i(\nu)$ as:

$$\sigma_i(\nu) = \big(rem_\delta(s_1 - 2M[v_1, v_i]), \ldots, rem_\delta(s_t - 2M[v_1, v_i])\big)$$

Let $\mu_i(\nu)$ be the maximum of $\sigma_i(\nu)$. Based on the above argument, we can see $max\{M[left(\nu), left(\nu)+1], \ldots, M[left(\nu), right(\nu)]\}$ is indeed $\mu_i(\nu)/2$. An important observation here is that because the elements of M are at most $\delta/2$, $\sigma_i(\nu)$ is a concatenation of two sorted sequences namely $\sigma_i^1(\nu)$ and $\sigma_i^2(\nu)$ (note that one of these sequences might be empty). So, in order to find $\mu_i(\nu)$, we need to compare the last elements of $\sigma_i^1(\nu)$ and $\sigma_i^2(\nu)$ (if they exist) and pick the greater value. Precisely, if $s_{j'} - 2M[v_1, v_i]$ is negative (resp. positive) for some $s_{j'} \in \sigma(\nu)$, $rem_\delta(s_{j'} - 2M[v_1, v_i])$ belongs to $\sigma_i^1(\nu)$ (resp. $\sigma_i^2(\nu)$). Thus, we can do binary search to obtain the index of the last element of $\sigma_i^1(\nu)$ and so $\mu_i(\nu)$ in $O(\log |span(\nu)|)$ time.

We can use the above data structure to find $\bar{M}[i,j]$ as follows: we first obtain two paths π_i and π_j and their split vertex ν_{split} from the root of $\mathcal{T}$ to v_i and v_j respectively. Let $\mathcal{V}_{i,j}$ be the set of right (resp. left) children of π_i (resp. π_j) from ν_{split} to its leaf (including v_j). Now, $\mathcal{M}_{i,j} = 1/2 \cup_{\nu \in \mathcal{V}_{i,j}} \sigma_i(\nu)$ where the multiplication is done element-wise. Therefore,

$$\bar{M}[i,j] = max\ \mathcal{M}_{i,j} = max\{\mu_i(\nu) : \nu \in \mathcal{V}_{i,j}\} \qquad (1)$$

because $|\mathcal{V}_{i,j}| = O(\log n)$ and computing each $\mu_i(\nu)$ in (1) also costs $O(\log n)$, the total complexity of computing $\bar{M}[i,j]$ would be $O(\log^2 n)$ which leads to an overall

$O\big((n\log^2 n + k)\log n\big)$ time complexity for the PCkC problem in unweighted paths.

Further improvements: First, we observe that if for two nodes $\nu, \nu' \in \mathcal{T}$, ν' is a parent of ν then $\sigma(\nu)$ is a sub-sequence of $\sigma(\nu')$. This property enables us to use a technique called *fractional cascading* [3] to avoid doing binary search on each of the nodes in $\mathcal{V}_{i,j}$ to find their maximum. Precisely, we equip each element s of $\sigma(\nu')$ with a pointer that points to the smallest element in ν larger than or equal to s. This structure can be constructed in $O(n\log n)$ time [3]. So, in order to obtain all $\{\mu_i(\nu) : \nu \in \mathcal{V}_{i,j}\}$, we only perform one binary search on $\sigma_i(root(\mathcal{T}))$ with cost $O(\log n)$ and follow the pointers along the paths to obtain each $\mu_i(\nu) : \nu \in \mathcal{V}_{i,j}$ in a constant time. So, the total complexity of computing $\bar{M}[i,j]$ would be $O(\log n)$ and so, the total running time would be $O\big((n\log n + k)\log n\big)$.

As another improvement, note that we only need to do binary search on $\sigma_i(root(\mathcal{T}))$ once for each row i in the entire algorithm. Also, by spending $O(n\log n)$ time, for each root-leaf path π_i and each $\nu' \in \pi_i$, we can store $max\{\mu_i(\nu) : \nu$ is right child of a node in $\pi_i[\nu', v_i]\}$ in ν' ($\pi_i[\nu', v_i]$ is the portion of π_i from ν' to v_i) by walking along π_i twice. So, having ν_{split}, we only need to take care about computing $max\{\mu_i(\nu) : \nu \in \mathcal{V}_{i,j}$ and hanging from $\pi_j\}$. To address this problem, consider a fixed i^{th}-row. Based on Algorithm 2, at each iteration r, the undiscarded region of the i^{th}-row corresponds to $span(\nu^r)$ for some $\nu^r \in \mathcal{T}$. Let ν^r_m be the left child of ν^r (if we are not at the last iteration) with $m^r = right(\nu^r_m)$. We can see that m^r is the median of the undiscarded region. Now, ν^{r+1}_m is either the left child of ν^r_m or the left child of the right neighbor of ν^r_m. Let r_0 be the last iteration for which $\nu^{r_0}_m$ is on π_i. For iterations $r \le r_0$, we only need to consider the maximum of the values in $\sigma_i(\nu')$ where ν' the first right child on π_i after ν^r_m. Also, for iterations $r > r_0$, we only need to have the set of maximum values in the left hanging nodes of $\pi_{m^r}[\nu_{split}, \nu^r_m]$ and ν^r_m itself. Now, it is easy to see that as r increases to $r+1$, these set of values can be updated in a constant time. Thus, we can conclude that computing $\bar{M}[i, m^r]$ for all iterations r only takes $O(\log n)$ time and because we have linear number of rows, we would have the following theorem:

Theorem 1 *The unweighted PCkC problem can be solved in $O((n+k)\log n)$ time.*

3 PCkC Problem for Weighted Paths

Let $P = (v_1, \ldots, v_n)$ be the given weighted path such that w_i is the weight of v_i. For a point x on P, we define $wd(v_i, x) = w_i d(v_i, x)$. Again each pair of vertices (v_i, v_j) generates $O(k)$ candidate values which corresponds to the trains that can be fitted in $[v_i, v_j]$. Here, because the weights of v_i and v_j might be different, a train may not be required to have the same distance from v_i and v_j in order to induce the same cost on them. Again, we denote the cost that a fitted t-train in $[v_i, v_j]$ induces on v_i and v_j by $IC_t(v_i, v_j)$. Suppose that $d(v_i, v_j) > t\delta$ for some $t > 1$. We define the *width* of (v_i, v_j) as $IC_{t-1}(v_i, v_j) - IC_t(v_i, v_j)$ and denote it by $W(v_i, v_j)$. Note that this value is independent of t and only depends on w_i and w_j and so, we can compute it in a constant time (in the unweighted case, the width of all pairs in P are $\delta/2$). Because here the widths of the pairs in P might not be equal, we first need to find an interval I^* such that each pair of vertices can generate at most one cost in I^*. But before going into that, we need to update our feasibility test to support weighted vertices. Algorithm 3 presents the feasibility test procedure WPATH-FT(P,r) which gets a weighted path P and a test value r and determines whether $r \ge r^*$ or $r < r^*$.

Algorithm 3 WPATH-FT(P, r)

```
1: Set Counter = 1
2: for i=1 to n do
3:     Let x_i be the point on the right side v_i such that wd(v_i, x_i) = r.
4: end for
5: Let X = (x_1, ..., x_n).
6: Let x_c = x_1.
7: while There is an element left in X do
8:     Eliminate x_i s from X corresponding to the vertices for which wd(v_i, x_c) ≤ r.
9:     Put a center at x_c = min{x_c + δ, X[1]}.
10:    Counter = Counter + 1.
11:    if Counter > k then
12:        return infeasible.
13:    end if
14: end while
15: return feasible.
```

Note that in the while loop of Algorithm 3, we eliminate x_is according to the order in the sequence X and so, the running time of the above feasibility test is $O(n+k)$. The geometric view for the weighted case is similar to the unweighted case but here, for each vertex v_i, the magnitude of the slopes of R_i and L_i is w_i. For each pair (v_i, v_j), the y-coordinate of the intersection point of R_i and L_j is the cost that a fitted 1-train (single point) in $[v_i, v_j]$ induces on v_i and v_j which is denoted by $IC_1(v_i, v_j)$. Similarly, if $d(v_i, v_j) > (t-1)\delta$, $IC_t(v_i, v_j)$ would be the y-coordinate of the horizontal segment with length $(t-1)\delta$ and endpoints on R_i and L_j (see Figure 5).

3.1 Matrix Search for Weighted Paths

First, we need to build an interval $I_1 = [a, b]$ such that $r^* \in I_1$ and it's interior does not contain any $IC_1(v_i, v_j)$ for any $i < j$ (note that $IC_1(v_i, v_j)$ is indeed the y-coordinate of the intersection point of R_i and L_j). If we use Lemma 2.5 [16] on all R_i and L_j ($1 \le i, j \le n$), we can get I_1 in $O((n+k)\log n)$ time. Let us define W^* as follows:

$$W^* = min\{W(v_i, v_j) : i < j \text{ and } IC_1(v_i, v_j) \ge b\}$$

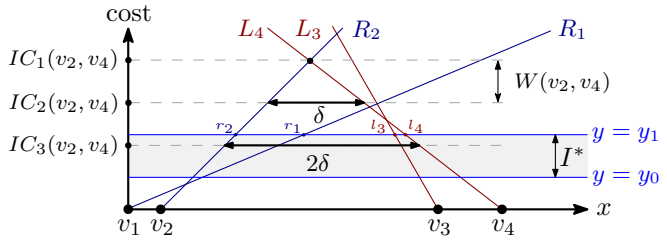


Figure 5: A weighted path (v_1, v_2, v_3, v_4), the width of (v_2, v_4) and three costs generated by the pair. Note that only one of them lies inside I^*.

We can see that $r_* \in I_2 := [b - kW^*, b] \cap I_1$. This is because r^* can't be smaller than the cost that a fitted k-train induces on the generating pair of W^*.

Proposition 4 *W^* can be computed in $O(n \log n)$ time.*

Proof. We first compute the intersection points of all L_i and R_j for $1 \leq i, j \leq n$ with the horizontal line $y = b$. Then, we sort these intersections on the line from left to right in $O(n \log n)$ time. So, each of these intersections corresponds to a line with a positive or a negative slope. We traverse these intersections from left to right and store the minimum positive slope and the minimum width we have seen in variables *min_slope* and *min_width* respectively. Finally, we set *min_width* as W^*. Precisely, when we visit an intersection point, if it came from a line with a positive slope, we update *min_slope* if necessary and if it came from a line with a negative slope, we compute the width it creates with the line that generated *min_slope* and update *min_width* if necessary. □

We can see that the length of I_2 is at most kW^*. This implies that by applying the feasibility test $O(\log k)$ times at the costs $b - iW^*$ $(0 \leq i \leq k)$ we get an interval $I_3 \subseteq I_2$ with length at most W^* containing r^*. Because W^* is the minimum width, each pair (v_i, v_j) with $IC_1(v_i, v_j) \geq b$ can generate at most one candidate value in I_3.

Consider the set of half-lines $\{R_1, \ldots, R_{n-1}\}$ (all with positive slopes) and their upper-envelope polygonal chain as a function $f_{UE}(x)$. We can see that f_{UE} is a piece-wise linear and an increasing function. Also, $f_{UE}(x)$ is the cost of covering all the vertices on the left side of x if we put a center at x. We can compute f_{UE} in linear time as follows: suppose that we have already computed the upper-envelope of $\{R_1, \ldots, R_{j-1}\}$ consisting of it's lines and break points. Now, when we add R_j and update our envelope, if R_j is below the last break points, we consider R_j and the last line of the envelope for a possible new break point. Otherwise, we find the first break point below the line (be checking the break points one by one from the last) and consider the line next to it (on its left) for a break point. Note that when we check a break point and it turns out it is below R_j, the line next to it (on its right) can never be a part of the envelope. Because we have linear number of lines, the time complexity of computing f_{UE} is linear.

Let $(x_1, \ldots, x_s)$ be the x-coordinates of the break points of f_{UE} where s is the number of break points. Then, we can use our feasibility test to do binary search on $\{f_{UE}(x_i) : 1 \leq i \leq s\}$ to find an interval $[x_q, x_{q+1}]$ such that $r^* \in [f_{UE}(x_q), f_{UE}(x_{q+1})]$. Let R_q (generated by v_q) be the line corresponding to the portion of f_{UE} in $[x_q, x_{q+1}]$. Then we have the following observation:

Observation 5 *If c_1^* induces r^*, then v_q is the first vertex of a determining pair.*

Based on the above observation, we can consider all pairs $\{(v_q, v_{q+1}), \ldots, (v_q, v_n)\}$, obtain the candidate value that each generates, sort them and do binary search (using our feasibility test) to get an interval $I^{(1)}$. Now, c_1^* can't generate any candidate value in the interior of $I^{(1)}$. Similarly, we can do the above process on $\{L_2, \ldots, L_n\}$ to get an interval $I^{(2)}$ such that c_k^* can't generate any candidate value in the interior of $I^{(2)}$. Let $I^* = I_3 \cap I^{(1)} \cap I^{(2)}$. So, it is only left to resolve the candidates in the interior of I^*.

Observation 6 *If (v_i, v_j) is a determining pair and a train $(c_{h_1}^*, \ldots, c_{h_2}^*)$ in $[v_i, v_j]$ induces r^* on the interior of I^*, then*

1. $0 < d(v_i, c_{h_1}^*), d(v_j, c_{h_2}^*) < \delta/2$.
2. *$(c_{h_1}^*, \ldots, c_{h_2}^*)$ is the longest train that can be fitted in $[v_i, v_j]$.*

The first part of the above observation comes from the fact that if r^* lies on the interior of I^*, then $h_1 \neq 1$ and $h_2 \neq k$. So, if for example $d(v_i, c_{h_1}^*) \geq \delta/2$ then because of the PCC, $c_{h_1-1}^*$ can cover v_i in the optimal solution. For the second part, note that if we are able to fit a longer train in $[v_i, v_j]$ then either $d(v_i, c_{h_1}^*)$ or $d(v_j, c_{h_2}^*)$ should be greater than $\delta/2$ which contradicts the first part.

Based on Observation 6, for any pair of vertices (v_i, v_j), we define our matrix M for the weighted case such that $M[i, j]$ is the cost r induced by the longest train $(c_1, \ldots, c_q)$ that can be fitted in $[v_i, v_j]$ if $r \in I^*$ and $0 < d(v_i, c_1), d(v_j, c_q) < \delta/2$. If we didn't have either of these two conditions, we assign $M[i, j] = 0$. It is clear that r^* is an element of M. Similar to the unweighted case, we define $\bar{M}[i, j]$ as $\max\{M[i, i+1], \ldots, M[i, j]\}$. Again, we can see that $\bar{M}$ is a row sorted matrix but may not be sorted column-wise. Next, we show Proposition 3 is still valid for our new definition of M and $\bar{M}$ in the weighted case.

Proposition 5 *If $M[i, j] = r^*$, then for all $i < j' < j$, $M[i, j'] \leq r^*$.*

Proof. We proceed by contradiction. Suppose that (v_i, v_j) induces r^* and $\exists i < j' < j$ such that $M[i, j'] > r^*$. Let $C = (c_1, \ldots, c_q)$ be the longest train that can be fitted in $[v_i, v_{j'}]$ and induces the cost $M[i, j']$ on v_i and $v_{j'}$. Also,

let $C' = (c^*_{h_1}, \ldots, c^*_{h_2}) \subseteq C^*$ be the train that induces r^* in $[v_i, v_j]$. Now, $c^*_{h_1+q-1} < c_q$ (because we assumed that $M[i,j'] > r^*$) and $|C| < |C'|$ (because $v_j > v_{j'}$). We consider two cases:

case 1: $\boldsymbol{c^*_{h_1+q} \leq v_{j'}}$: In this case, we could fit a $(q+1)$-train namely $C'' = (c'_1, \ldots, c'_{q+1})$ in $[v_i, v'_j]$ which contradicts the fact that C was the longest train in $[v_i, v_{j'}]$.

case 2: $\boldsymbol{c^*_{h_1+q} > v_{j'}}$: In this case, $v_{j'}$ should be covered from its right in C^* (because $c^*_{h_1+q-1} < c_q$ and we assumed $M[i,j'] > r^*$). Also, the cost of covering $v_{j'}$ in C^* should be no more than r^*. So, if $w_i \leq w_{j'}$, $d(v_i, c^*_{h_1}) \geq d(v_{j'}, c^*_{h_1+q})$ and thus, we can fit a $(q+1)$-train in $[v_i, v_{j'}]$ which is a contradiction.

Now, assume that $w_i > w_{j'}$. Let t_1 and t_2 be the points on the right side of $v_{j'}$ such that $wd(v_{j'}, t_1) = r^*$ and $wd(v_{j'}, t_2) = M[i,j']$. Note that $t_2 > t_1$ and t_2 is the mirror image of c_q with respect to $v_{j'}$. Now, $d(c^*_{h_1}, c_1) < d(t_1, t_2)$ (because $w_i > w_{j'}$ and the cost that v_i induces on $c^*_{h_1}$ and c_1 are r^* and $M[i,j']$ respectively). Also, because $M[i,j'] \neq 0$, $d(c_q, v_{j'}) < \delta/2$ (based on the definition of M), $c_q + \delta > v_{j'} + \delta/2$ which implies that $[t_1, t_2] \subseteq [c^*_{h_1+q}, c_q + \delta]$. This contradicts the fact that $d(c^*_{h_1+q}, c_q + \delta) = d(c^*_{h_1}, c_1) < d(t_1, t_2)$ (see Figure 6). □

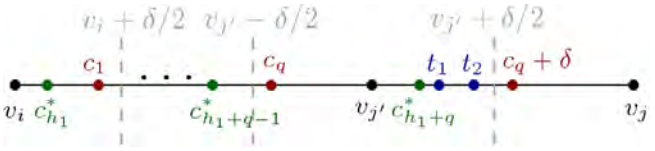


Figure 6: Proof of Proposition 5

The above proposition implies that Observation 3 is valid for M and $\bar{M}$ in the weighted case and so we can use Algorithm 2 to find r^* and get C^*. Based on Algorithm 2, the time complexity of finding r^* would be $O((ng(n)+k)\log n)$ where $g(n)$ is the time complexity for computing an element of $\bar{M}$. In the Appendix, we show how we can compute an element of $\bar{M}$ in $O(\log^3 n)$ time by spending $O(n \log^3 n)$ time for preprocessing. This gives us the following theorem:

Theorem 2 *The PCkC problem can be solved in* $O((n \log^3 n + k) \log n)$ *time.*

References

[1] Ahlberg M, Vlassov V, Yasui T. Router placement in wireless sensor networks. In *2006 IEEE International Conference on Mobile Ad Hoc and Sensor Systems* 2006 Oct 9 (pp. 538-541). IEEE.

[2] Bhattacharya B, Mozafari A, Shermer TC. An efficient algorithm for the proximity connected two center problem. In *International Workshop on Combinatorial Algorithms* 2022 (pp. 199-213). Springer, Cham.

[3] De Berg M, Cheong O, Van Kreveld M, Overmars M. Computational geometry: introduction. Computational geometry: algorithms and applications. 2008:1-7.

[4] Frederickson GN. Parametric search and locating supply centers in trees. In *Workshop on Algorithms and Data Structures* 1991 Aug 14 (pp. 299-319). Springer, Berlin, Heidelberg.

[5] Frederickson GN. Optimal algorithms for tree partitioning. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms* 1991 Mar 1 (pp. 168-177).

[6] Hakimi SL. Optimum distribution of switching centers in a communication network and some related graph theoretic problems. *Operations research.* 1965 Jun;13(3):462-75.

[7] Huang CH. Some problems on radius-weighted model of packet radio network, Doctoral dissertation, Ph. D. Dissertation, Dept. of Comput. Sci., Tsing Hua Univ., Hsinchu, Taiwan, 1992.

[8] Huang PH, Tsai YT, Tang CY. A near-quadratic algorithm for the alpha-connected two-center problem. *Journal of information science and engineering.* 2006 Nov 1;22(6):1317.

[9] Huang PH, Te Tsai Y, Tang CY. A fast algorithm for the alpha-connected two-center decision problem. *Information Processing Letters.* 2003 Feb 28;85(4):205-10.

[10] Jeger M, Kariv O. Algorithms for finding P-centers on a weighted tree (for relatively small P). *Networks.* 1985 Sep;15(3):381-9.

[11] Kariv O, Hakimi SL. An algorithmic approach to network location problems. I: The p-centers. *SIAM Journal on Applied Mathematics.* 1979 Dec;37(3):513-38.

[12] Megiddo N. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM journal on computing.* 1983 Nov;12(4):759-76.

[13] Megiddo N, Supowit KJ. On the complexity of some common geometric location problems. *SIAM journal on computing.* 1984 Feb;13(1):182-96.

[14] Patel M, Chandrasekaran R, Venkatesan S. Energy efficient sensor, relay and base station placements for coverage, connectivity and routing. In *PCCC 2005. 24th IEEE International Performance, Computing, and Communications Conference*, 2005. 2005 Apr 7 (pp. 581-586). IEEE.

[15] Toth CD, O'Rourke J, Goodman JE, editors. Handbook of discrete and computational geometry. CRC press; 2017 Nov 22.

[16] Wang H, Zhang J. An $O(n \log n)$-Time Algorithm for the k-Center Problem in Trees. *SIAM Journal on Computing.* 2021;50(2):602-35.

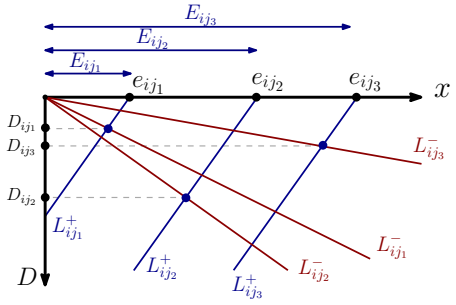


Figure 7: Three points e_{ij_1}, e_{ij_2} and e_{ij_3} located at distances E_{ij_1}, E_{ij_2} and E_{ij_3} respectively and their generating points. In this example, (v_i, v_{j_1}) generates the maximum of $\{M[i,j_1], M[i,j_2], M[i,j_3]\}$.

Appendix: Computing an Element of $\bar{M}$ for Weighted Paths

In this section, we build a data structure such that for any query pair (i,j) $(i<j)$, it enables us to compute $\bar{M}[i,j] = max\{M[i,j'] : i < j' \leq j\}$ in a sub-linear time. Suppose that $I^* = [y_0, y_1]$. We denote the x-coordinates of the intersection points of L_i and R_i $(1<i<n)$ with line $y = y_1$ by l_i and r_i, respectively (see Figure 5). Note that if for a pair $(v_i, v_{j'})$, $l_{j'} < r_i$, it can not generate any candidate value in I^* (because of the way we built I^*) and so, $M[i,j'] = 0$. Thus, we only consider the pairs $(v_i, v_{j'})$ for which $l_{j'} \geq r_i$. Let us define the complement function with respect to δ as:

$$comp_\delta(x) = \left\lceil \frac{x}{\delta} \right\rceil \times \delta - x$$

We also denote $comp_\delta(l_i)$ and $comp_\delta(r_i)$ by $\hat{l}_i$ and $\hat{r}_i$ respectively where $1<i<n$. For any pair (i,j') with $j' > i$ and $l_{j'} > r_i$, let $E_{ij'} = comp_\delta(l_{j'} - r_i) = rem_\delta(\hat{l}_{j'} - \hat{r}_i)$ and $D_{ij'} = E_{ij'}/(w_i^{-1} + w_{j'}^{-1})$ (note that w_i and $w_{j'}$ are the magnitudes of the slopes of R_i and $L_{j'}$ respectively). Based on the geometric view, it is easy to see that $M[i,j'] = y_1 - D_{ij'}$ if $D_{ij'} \leq |I^*|$ and zero otherwise. So, the problem of finding $\bar{M}[i,j]$ is equivalent to find $D_{min} = min\{D_{ij'} : i < j' \leq j\}$. It is convenient to visualize this set as follows: for each $i < j' \leq j$, we consider $e_{ij'}$ as the point located at $(E_{ij'}, 0)$ on the x-axis. Each $e_{ij'}$ has a half-line $L^+_{ij'}$ attached to it with slope w_i and a half-line $L^-_{ij'}$ from the origin with slope $-w_{j'}$ (see Figure 7). We can see that the distance between the intersection point of L^+_{ij} and L^-_{ij} from the x-axis is indeed D_{ij}. We call this distance the *D-coordinate* of the intersection (when a point moves downward, its D-coordinate increases). So, each value $E_{ij'}$ generates exactly one value $D_{ij'}$ call it the *D-value* of $E_{ij'}$. Like the unweighted case, we build a balanced binary tree $\mathcal{T}$ on top of the vertices and in each node $\nu \in \mathcal{T}$ we store $\{\hat{l}_h : v_h \in span(\nu)\}$ as an increasingly sorted sequence $\sigma(\nu)$. So, if we preprocess each $\nu \in \mathcal{T}$ such that for a given vertex v_i, we can quickly compute $\mu_i(\nu) = min\{D_{ih} : v_h \in span(\nu)\}$, we can decompose the set $\{v_j : i < j' \leq j\}$ into $\cup_{\nu \in \mathcal{V}_{i,j}} span(\nu)$ (as we did in Section 2.1) and set $D_{min} = min\{\mu_i(\nu) : \nu \in \mathcal{V}_{i,j}\}$

Let $\nu \in \mathcal{T}$ be a fixed node. In the rest, we show how we can preprocess ν such that given a query vertex v_i, we can efficiently compute $\mu_i(\nu)$. First, note that the set of half-lines $\{L^-_{ih} : v_h \in span(\nu)\}$ is independent of i. Also, for each i, $\{E_{ih} : v_h \in span(\nu)\}$ is the union of two sorted sequences $\sigma^1_i(\nu)$ and $\sigma^2_i(\nu)$, where $\sigma^1_i(\nu)$ (resp. $\sigma^2_i(\nu)$) is obtained by a shift (adding a constant value) of the elements in $\sigma(\nu)$ smaller than (resp. greater than or equal to) $\hat{r}_i$. Consider the lines $L^+_{ij'}(x) = w_i(x - e_{ij'})$ and $L^-_{ij'}(x) = -w_{j'}x$, where $e_{ij'}$ is a variable (see Figure 7). When $e_{ij'}$ increases, the D-value of $e_{ij'}$ (the intersection of $L^+_{ij'}$ and $L^-_{ij'}$) increases linearly. Let $f(x)$ be the minimum D-value generated by $\{e_{ij'} = \hat{l}_{j'} + x : j' \in span(\nu)\}$. We can see that $f(x)$ is the lower-envelope of a set of lines which can be computed in $O(|\nu| \log |\nu|)$ time ($|\nu|$ is the number of vertices in $span(\nu)$) using the divide-and-conquer algorithm (use the order in $\sigma(\nu)$ for breaking the vertices). Because we need to work with the sub-sequences of $\sigma(\nu)$, we store the entire *recursion tree* [15] (with the solutions of its subproblems) of the divide-and-conquer algorithm and denote this tree by $\mathcal{R}_i(\nu)$. Based on the above discussion, one way to preprocess ν is that for each $1<i<n$, we compute and store $\mathcal{R}_i(\nu)$. Now, when we are given a vertex v_i, we first use binary search to get $\sigma^1_i(\nu)$ and $\sigma^2_i(\nu)$. Next, we use $\mathcal{R}_i(\nu)$ to get $\mu_i(\nu)$. Note that this process costs $O(\log^2 |\nu|)$ time (one $O(\log |\nu|)$ factor because of the height of $\mathcal{R}_i(\nu)$ and the other for binary search to get the minimum point of the envelopes in the nodes of $\mathcal{R}_i(\nu)$ needed to construct $\sigma^1_i(\nu)$(resp. $\sigma^2_i(\nu)$) at an specific x-coordinate determined by the shift in $\sigma^1_i(\nu)$(resp. $\sigma^2_i(\nu)$). Because the height of $\mathcal{T}$ is $O(\log n)$, the total time complexity of computing $\bar{M}[i,j]$ would be $O(\log^3 n)$. Note that the values $\sigma(\nu)$ of any (non root) node $\nu \in \mathcal{T}$ is a subset of the values of its parent node. So, using the fractional cascading technique, this cost can be reduced to $O(\log^2 n)$ time.

The problem here is that if we build $\mathcal{R}_i(\nu)$ for all $1<i<n$ and all $\nu \in \mathcal{T}$, the time complexity of the preprocessing phase would be $O(n^2 \log^2 n)$. In order to make the preprocessing cost sub-quadratic, consider an internal node ω of the recursion tree of ν (note that the vertices in ω are independent of i). Let $\tau_i(\omega)$ be the sequence of points in ω who generate a line in its corresponding lower-envelope in $\mathcal{R}_i(\nu)$ where the order is according to the appearance of the lines in the envelope.

Proposition 6 *If for two indices i_1 and i_2, $w_{i_1} < w_{i_2}$, then $\tau_{i_2}(\omega) \subseteq \tau_{i_1}(\omega)$*

The proof of the above proposition is straightforward using elementary geometry. In order to use the above proposition, we first sort all the slopes increasingly into a sequence $(w_{i_1}, \ldots, w_{i_n})$. Now, when we preprocess ω, instead of building $\mathcal{R}_i(\omega)$ for all $1<i<n$, we can use a binary tree structure for the slopes which leads to $O(|\omega| \log n)$ time complexity for preprocessing ω. This, reduces the overall preprocessing time to $O(n \log^3 n)$ and increases the time complexity of computing $M[i,j]$ to $O(\log^3 n)$.

Drawing complete outer-1-planar graphs in linear area

Therese Biedl*

Abstract

A *complete outer-1-planar graph* is a graph that can be drawn such that every edge has at most one crossing, all vertices are on the infinite face, and the so-called dual tree is a complete ternary tree. We show that every complete outer-1-planar graph has a straight-line grid-drawing that has area $O(n)$.

1 Introduction

In this paper we consider the question of how to create a *straight-line grid-drawing* of a graph, i.e., we want to map the vertices to grid points, and draw edges as straight-line segments between their endpoints such that vertex-points are distinct and no edge-segment contains a vertex-point except at its endpoints. If the input graph is *planar* (it has a *planar drawing* without crossing), then we further require that the drawing is likewise planar. Generally, whenever the given graph comes with a drawing (not necessarily using straight lines), then we expect the created straight-line grid-drawing to reflect the given drawing of the graph.

The objective is usually to achieve small *area of the drawing* (i.e., the area of the minimum enclosing axis-aligned bounding box of the drawing). Let n be the number of vertices. Any graph can be drawn with area $O(n^3)$ by placing the vertices on the moment-curve. For planar graphs, it has long been known that $O(n^2)$ is always sufficient [15, 16], and for some planar graphs $\Omega(n^2)$ area is required in a planar drawing [14]. For some subclasses of planar graphs, sub-quadratic area can be achieved. Of particular relevance to this paper are the results for *outer-planar graphs*, i.e., graphs that have a planar drawing where all vertices are incident with the unbounded region (the *outer-face*). Such graphs have straight-line grid-drawings in sub-quadratic area [9], and very recently the area has been reduced to $O(n^{1+\varepsilon})$ [13].

We are interested here in drawing *1-planar graphs*, i.e., graphs that have a drawing that is not necessarily planar but every edge is crossed at most once. Such graphs do not always have a straight-line grid-drawing [10] but if they are 3-connected then there is a straight-line drawing after deleting at most one edge [2] and the area is quadratic. Clearly some 1-planar graphs require $\Omega(n^2)$ area since all planar graphs are also 1-planar.

The natural question is now whether there are subclasses of 1-planar graphs that have straight-line grid-drawings in sub-quadratic area? The most obvious class to consider are *outer-1-planar graphs*, which are 1-planar graphs with a 1-planar drawing where all vertices are on the outer-face. It is known that outer-1-planar graphs can be drawn in sub-quadratic area in the drawing style of "visibility representations" (not reviewed here) [4]. Straight-line drawings of outer-1-planar graphs appear to have studied only a little bit. Dekhordi and Eades showed that they have so-called RAC-drawings [8] but they did not analyze the area. Auer et al. [3] showed that they have a straight-line grid-drawings in quadratic area. Bulatovic [5] achieved sub-quadratic area in some special situations.

In the pursuit of sub-quadratic-area drawings for outer-planar graphs [9, 13], one helpful ingredient was to first study a *complete outer-planar graph*, i.e., an outer-planar graph for which the dual graph (minus the outer-face vertex) is a complete binary tree when rooting it suitably. By exploiting its recursive structure, Di Battista and Frati showed that a complete outer-planar graph has a straight-line grid-drawing in $O(n)$ area [9].

In the same spirit, we ask here whether we can create small straight-line grid-drawings of *complete outer-1-planar graphs* (defined formally below). Bulatovic [5] showed that these have a grid-drawing of area $O(n^{2\cdot\log_3 2}) = O(n^{1.26})$. In this paper, we improve on this result and show that all complete outer-1-planar graphs have a straight-line grid-drawing of area $O(n)$. This fits into a long line of research of achieved optimal $O(n)$ area for straight-line grid-drawings of special graphs, see e.g. [1, 6, 7, 9].

2 Preliminaries

We assume familiarity with graph theory and planar graphs, see for example [11]. Assume throughout that G is an outer-1-planar graph with n vertices that is maximal in the sense that no edges can be added while maintaining simplicity and outer-1-planarity. Then G consists of an n-cycle as the outer-face and chords of the n-cycle. The *skeleton* G^s of G is the subgraph of G formed by the *uncrossed* edges, i.e., edges without crossing. The *inner faces* of G^s are the maximal bounded regions that contain no edges of G^s; it is known that

*David R. Cheriton School of Computer Science, University of Waterloo, biedl@uwaterloo.ca

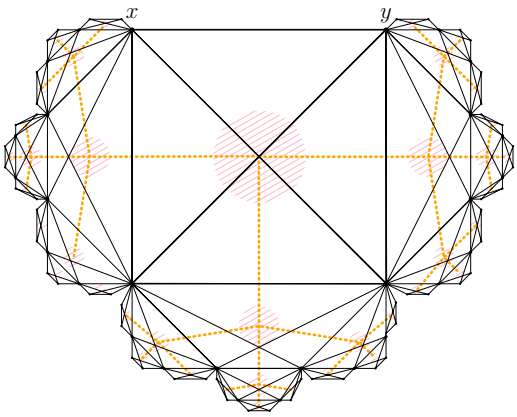


Figure 1: The complete outer-1-planar graph of depth 4. The dual tree is orange (striped/dotted).

all inner faces of G^s are triangles or quadrangles if G is maximal outer-1-planar [8]. The *dual tree* of G is obtained by creating a vertex for every inner face of G^s and making them adjacent if the corresponding faces share an edge. The dual tree of an outer-1-planar graph is (as the name suggests) a tree and all vertices have degree at most 4.

We call G a *complete outer-1-planar graph* if the dual tree $\mathcal{T}$ is a complete ternary tree after rooting it suitably. See Figure 1. The *depth* D of G is the number of vertices on the path in $\mathcal{T}$ from the root to the leaves. If $D \geq 2$, then G consists of K_4 (drawn with one crossing and corresponding to the root of the dual tree) with three copies of a complete outer-1-planar graph of depth $D-1$ attached at three of the four uncrossed edges of K_4. The *poles* of G are the endpoints of the uncrossed edge (x, y) of K_4 that is on the outer-face of G.

For an uncrossed edge (a, b) not on the outer-face, the *hanging subgraph* H_{ab} *at* (a, b) is the maximal subgraph that has (a, b) on the outer-face and does not contain both poles of G. The poles of H_{ab} are a and b.

The complete outer-1-planar graph of G depth D has $\Theta(3^D)$ vertices, hence $D \in \Theta(\log n)$. It is very easy to draw G in a grid of width $O(n)$ and height $O(D)$ [5], so with area $O(n \log n)$. But achieving linear area with this approach seems hopeless since even the skeleton of G requires $\Omega(\log n)$ width and height in any drawing. (This follows from [12] since its so-called pathwidth is logarithmic.) Instead for a linear-area drawing we construct a drawing of width and height $O(\sqrt{n})$.

Triangular grids. One ingredient for drawing complete outer-1-planar graph in linear area will be to use the grid points of a *triangular grid* (with grid-lines of slope $\sqrt{3}, 0, -\sqrt{3}$), rather than the standard (orthogonal) grid. This makes no difference overall, since the triangular grid can be mapped to an orthogonal grid with a shear, but allows us to treat hanging subgraphs symmetrically.

The following shortcuts will be useful. We use arrows such as $\nearrow$ and $\nwarrow$ for grid-lines of slope $\sqrt{3}$ and $-\sqrt{3}$, and so for example speak of a $\nearrow$-ray or the distance in $\nwarrow$-direction. An *axis-aligned equilateral triangle* is a triangle with three equal sides that all lie along grid-lines. An *axis-aligned isosceles triangle* is a triangle where two equal-length sides lie along grid-lines while the third side connects two grid points and has angle 30° on both ends. We will usually drop "axis-aligned" as we study no other equilateral or isosceles triangles. A triangle is called *upward* if it has a unique *top corner*, i.e., point with maximum y-coordinate. We use terms such as top/bottom/left/right side/corner only when this uniquely identifies the feature.

3 Drawing types

Let G be the complete outer-1-planar graph of depth D, and let x, y be its poles. We will need three kinds of drawings of G that will be combined recursively:

A *type-A* drawing $\mathcal{A}$ of G is contained within an equilateral upward triangle T. Vertices x and y are placed on the left and right side of T, respectively, with distance exactly D from the top corner. Drawing $\mathcal{A}$ occupies no points on the right side of T except for y. See Figure 2.

Furthermore, $\mathcal{A}$ must have the flexibility to move x as follows. Let the *wedge of* $\mathcal{A}$ be the smaller wedge between the $\nearrow$-ray and the $\nwarrow$-ray emanating from x. We require that for any position x' within the wedge, moving x to x' gives a drawing of G for which all edges are either within T or within the triangle spanned by x', y and the left corner of T.

A *type-B* drawing $\mathcal{B}$ of G is contained within an equilateral upward triangle T. Vertices x and y are placed at the top and right corner of T, respectively, and the left corner is empty. See Figure 2.

Furthermore, $\mathcal{B}$ must have the flexibility to move y as follows. Let z be the point on the bottom side of T that has distance exactly D to y (we call this the *attachment point* of $\mathcal{B}$). Let the *wedge of* $\mathcal{B}$ be the smaller wedge between the $\searrow$-ray and the $\rightarrow$-ray emanating from y. We require that for any position y' within the wedge, moving y to y' gives a drawing of G. Furthermore, the drawing is contained within T and the triangle spanned by x, y', z.

We call a type-B drawing a *type-B^+-drawing* if additionally no point other than x is on the left side of triangle T. With the exception of $D = 1$ all type-B drawings that we construct are actually type-B^+-drawings.

A *type-C* drawing $\mathcal{C}$ of G is contained within an isosceles upward triangle T where the left and bottom side have the same length. Vertices x and y are placed at the top and right corner of T, respectively. Drawing

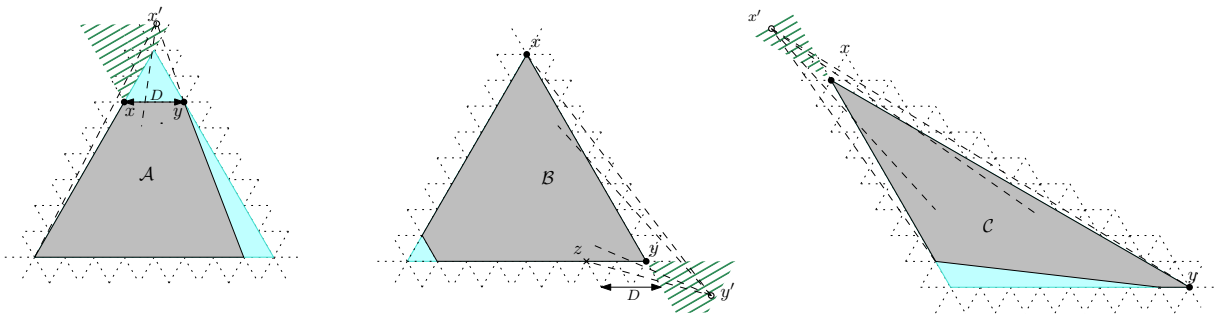


Figure 2: Drawings of type A, B and C. The wedge is green (striped).

$\mathcal{C}$ occupies no points on bottom side of T except for y and (possibly) points within unit distance from y. See Figure 2.

Furthermore, $\mathcal{C}$ must have the flexibility to move x as follows. Let the *wedge of* $\mathcal{C}$ be the smaller wedge between the $\nwarrow$-ray and a ray with slope $-\sqrt{3}/2$ (i.e., extending $\overline{xy}$) emanating from x. For any position x' within the wedge, moving x to x' gives a drawing of G.

Define the following function $w(\cdot)$ on positive integers:

$$w(D) := \begin{cases} 2 & \text{if } D = 1 \\ 6 & \text{if } D = 2 \\ 3w(D-2) + 4(D-2) + 6 & \text{if } D \geq 3 \end{cases}$$

A simple proof by induction shows that

$$w(D) \leq 16 \cdot 3^{D/2-1} - 2D - 5 \in O(3^{D/2}).$$

We will show the following by induction on D:

Lemma 1 *The complete outer-1-plane graph of depth D has drawings of type A, B and C where the shortest side of the bounding triangle T has length exactly $w(D)$. It also has a type-B^+ drawing where the side-length of T is at most $w(D) + 1$.*

In the base case (where $D = 1$ or 2) these drawings are easily created, see Figure 3 for some cases and Figure 10 in the appendix for all remaining ones.

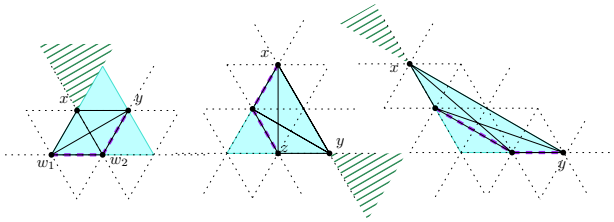


Figure 3: The drawings for $D = 1$ for type A, B, C.

4 The inductive step

Assume that the dual tree $\mathcal{T}$ of G has depth $D + 2$ where $D \geq 1$. We can hence split the graph into the subgraph Q corresponding to the root of $\mathcal{T}$ and and its three children, and the hanging subgraphs that are attached at the uncrossed edges that bound Q. (Each hanging subgraph is a complete outer-1-plane graph of depth D.) Enumerate the outer-face of Q as $\langle x, a, b, c, d, e, f, g, h, y\rangle$ in ccw order where x, y are the poles of G. See Figure 4.

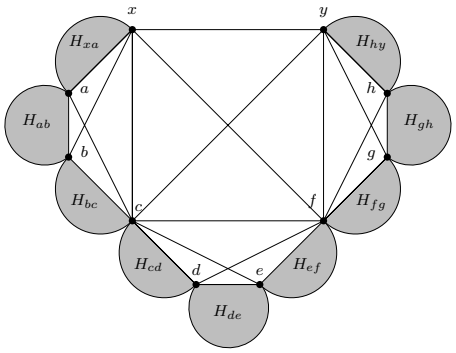


Figure 4: Splitting the graph into Q and nine hanging subgraphs.

The idea. Building a drawing of G uses the obvious recursive approach: create drawings of the nine hanging subgraphs of Q, combine them, and add the edges of Q. However, there are some intricate details with regards to placement of poles and spacing of subgraphs. We therefore first give a rough idea.

Observe that both an equilateral and an isosceles triangle T can be split into 9 equal-area triangles that are either equilateral or isosceles, see Figure 5. We assign the hanging subgraphs to these triangles as indicated in the figure, and plan to draw Q within the thick black lines (after expanding a bit).

Note that in our plan to place the vertices, some poles (e.g. vertex c for subgraph H_{bc}) are far away from the corresponding triangle; here the flexibility to move one pole within the wedge of the drawing will be crucial. However, this comes with the price that we must keep line segment $\overline{cz}$ free of other drawings, where z is the

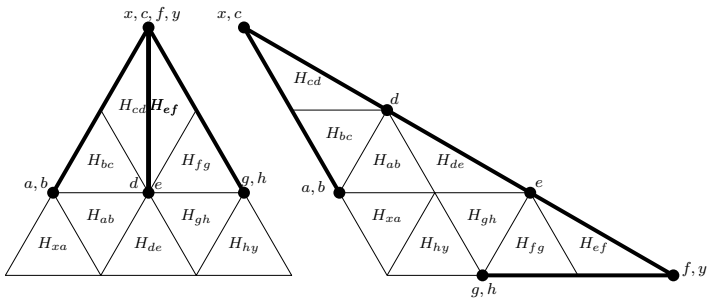


Figure 5: The idea of combining subgraphs. Locations for the vertices of Q are approximate.

attachment point of the drawing of H_{bc}. Therefore subgraphs cannot be placed exactly edge-to-edge as Figure 5 suggests and we must be more careful in spacing them.

Placing four subgraphs. We first explain how to place drawings of $H_{xa}, H_{ab}, H_{bc}, H_{cd}$; this will be the same for all three constructions below. Consult Figure 6. For any hanging subgraph H_{uv}, let Γ_{uv} be a (recursively obtained) drawing of H_{uv}—the text below will specify its type. Sometimes we will rotate Γ_{uv}; we use T_{uv} (drawn in cyan/light gray) for the bounding triangle of Γ_{uv} after such a rotation has been applied.

- Let Γ_{xa} be a type-A drawing for H_{xa}. The white circle in Figure 6 shows where pole x would be within Γ_{xa}, but it will actually be placed later somewhere within the wedge of Γ_{xa}.
- Let Γ_{ab} be a type-A drawing for H_{ab}, rotated by $+60°$. Place the left corner of T_{ab} one unit in $\searrow$-direction from the top corner of T_{xa}. This puts pole a within the wedge of Γ_{ab} as required.
- Let Γ_{bc} be a type-B drawing for H_{bc}, rotated by $+120°$ and placed such that the two locations of b coincide. Pole c will be placed somewhere within the wedge of Γ_{bc}.
- Let Γ_{cd} be a type-C drawing for H_{cd}, rotated by $-60°$ and placed such that the left corner of T_{cd} coincides with the attachment point z of T_{bc}. Pole c will be placed somewhere within the wedge of Γ_{cd}.
- Consider the point where the $\nearrow$-ray from b intersects the $\uparrow$-ray from d, and let r_c be the $\nearrow$-ray emanating from here. We will later place c somewhere on ray r_c, which keeps it within both wedges of Γ_{bc} and Γ_{cd}, and keeps line segment $\overline{cz}$ outside all other drawings.

Observe that all drawings are disjoint except where they share a vertex. This holds because in a type-A drawing the right side only contains the pole, and in Γ_{cd} the shorter side at d contains points only within distance 1 from d, but these points are not used by Γ_{bc}. Also observe that for any placement of x within the wedge of Γ_{xa}, line segment $\overline{ax}$ will be outside all other drawings. Finally observe that the path a-b-c (shown thick dashed) is drawn with slopes alternating between $[0, \sqrt{3})$ and $\sqrt{3}$; this will be crucial below.

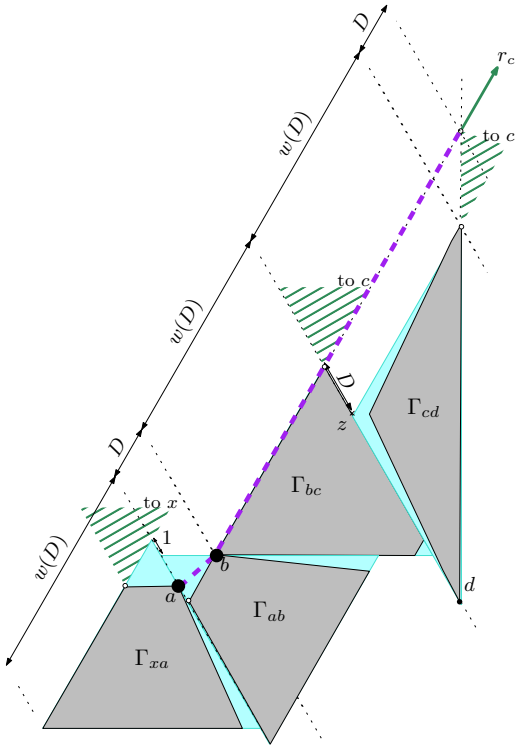


Figure 6: Placing $H_{xa}, \ldots, H_{cd}$.

Completing a type-A drawing. To complete the drawing to a type-A drawing, we copy and flip the existing drawing along a vertical line. See also Figure 7(a). More precisely, let ℓ_v be a vertical line that has $\rightarrow$-distance $D/2$ from d. Mirror $\Gamma_{xa}, \ldots, \Gamma_{cd}$ along this line to get $\Gamma_{ef}, \ldots, \Gamma_{hy}$. The only subgraph missing is H_{de}, for which we use a type-A drawing that fits exactly with the existing points for d and e. One verifies that all drawings are disjoint except at common poles.

We define the bounding triangle T of the drawing to be the upward equilateral triangle that touches the left side of T_{xa}, has $\nearrow$-distance one to the bottom side of T_{de}, and has $\nearrow$-distance three from the right side of T_{hy}. (This is slightly asymmetric; the line ℓ_v does *not* go through the top corner of T.) Elementary computation shows that T has side-length $3w(D)+4D+6 = w(D+2)$ as desired. Place x and y (as required for a type-A drawing) at distance $D+2$ from the top corner of T; this puts x within the wedge of Γ_{xa}.

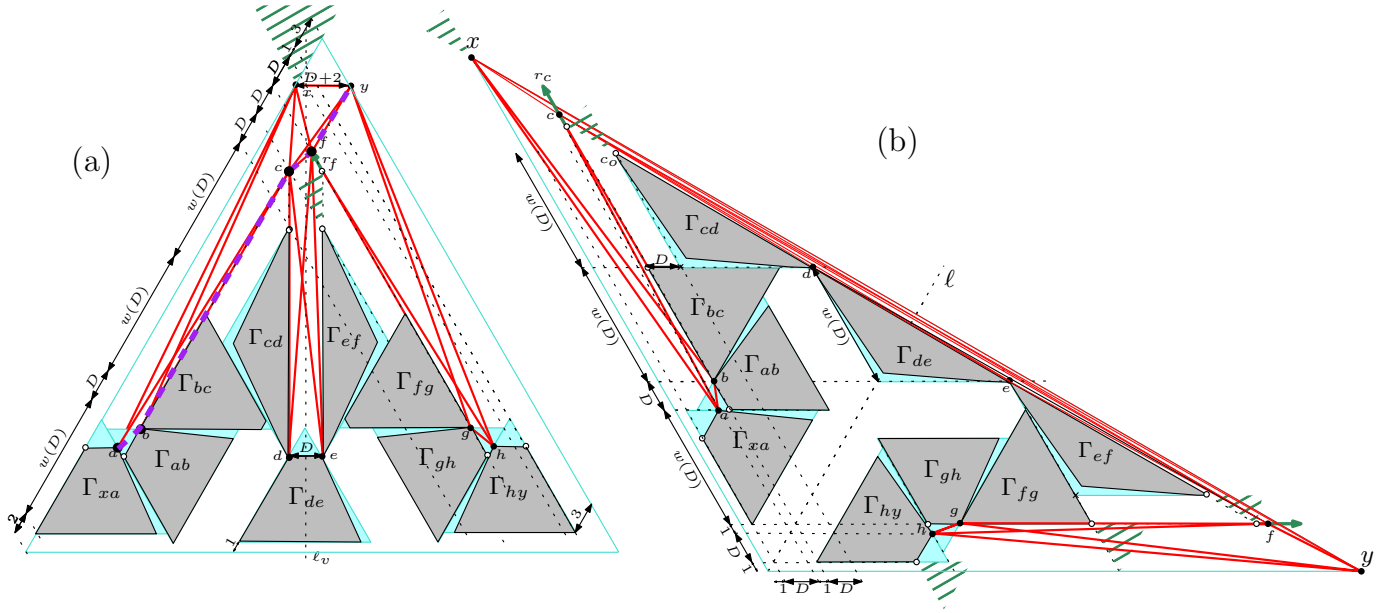


Figure 7: Creating (a) a type-A drawing and (b) a type-C drawing.

We place c at the start-point of ray r_c, which has $\nwarrow$-distance $D{+}1$ from the left side of T. Let r_f be the copy of ray r_c on the right side; we place vertex f on this ray with $\nwarrow$-distance $D{+}2$ from the left side of T. With this, $\overline{fy}$ has slope $\sqrt{3}$ while $\overline{cf}$ has slightly smaller slope.

We must argue that we have the flexibility to move x within the wedge $\mathcal{W}$ of the drawing. Consider the path $\pi = \langle w_1, w_2, \ldots, w_{2D+1}\rangle$ of neighbours of x. [The last five vertices on π are a, b, c, f, y, and this part is shown purple/dotted in Figures 3, 7, 10.] Path π connects the left side of T with the right side, and hence separates vertex x from all other vertices of the drawing. Also (as argued directly above or known by induction for the part of π in Γ_{xa}) the slopes along π alternate between a value in $[0, \sqrt{3})$ and exactly $\sqrt{3}$. For $1 \le i \le D$, let $\mathcal{W}_i$ be the smaller wedge between the two rays emanating from w_{2i} through w_{2i-1} and w_{2i+1}. By the slopes of the edges, $\mathcal{W}$ is strictly inside $\mathcal{W}_i$. Therefore $\{w_{2i-1}, w_{2i}, w_{2i+1}, x'\}$ forms a strictly convex quadrangle for any location of $x' \in \mathcal{W}$, and the K_4 formed by these four vertices is drawn with a crossing as required. Also, the quadrangles for different values of i are disjoint. So moving x' within $\mathcal{W}$ gives a drawing of G.

Creating a type-B drawing. To create a type-B drawing, we place all hanging subgraphs except H_{hy} exactly as in construction for the type-A drawing. Vertex h is placed as dictated by Γ_{gh}. For H_{hy} we use a type-B^+ drawing Γ_{hy} that we place such that the two drawings of h coincide. See Figure 8. One verifies that all drawings are disjoint except where they have common poles (this holds for Γ_{hy} since we use a type-B^+ drawing).

We define the bounding triangle T of the drawing to be the upward equilateral triangle that has $\nwarrow$-distance one from the left side of T_{xa}, $\nearrow$-distance two from the line through $\overline{gh}$ and has side-length $3w(D) + 4D + 6 = w(D{+}2)$. Elementary computation shows that this triangle then includes Γ_{hy} since T_{hy} has side-length at most $w(D) + 1$. The left side of T is empty, so the created type-B drawing is automatically a type-B^+ drawing. We place x and y as required at the top and the right corner of T.

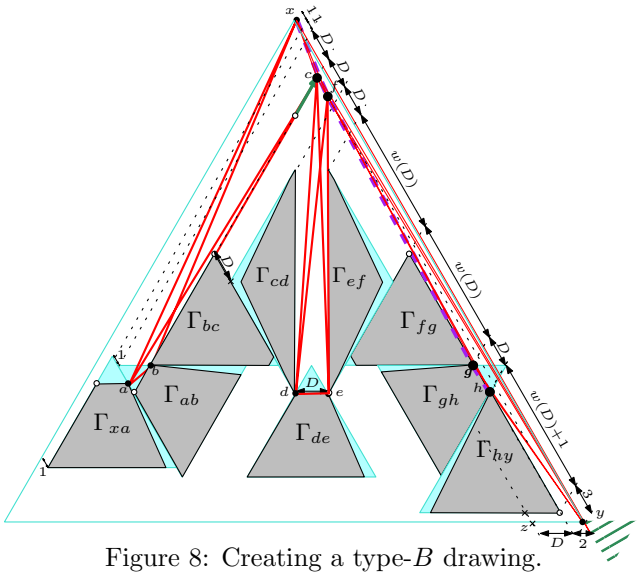


Figure 8: Creating a type-B drawing.

Let R be the right side of T. Place vertex c on r_c and vertex f on the $\uparrow$-ray from e, both with $\nearrow$-distance one to R. In particular $\overline{xy}$ is on R, $\overline{cf}$ has slope $-\sqrt{3}$ and $\nearrow$-distance one to R, and $\overline{gh}$ has slope $-\sqrt{3}$ and

↗-distance two to R; with this the complete graphs $\{x, y, c, f\}$ and $\{f, y, g, h\}$ of Q are drawn correctly (albeit with very small angles). All other edges of Q are also drawn correctly, see Figure 8. Also f is above e and to the right of the ↖-ray from g, hence within the wedges of Γ_{ef} and Γ_{fg} as required.

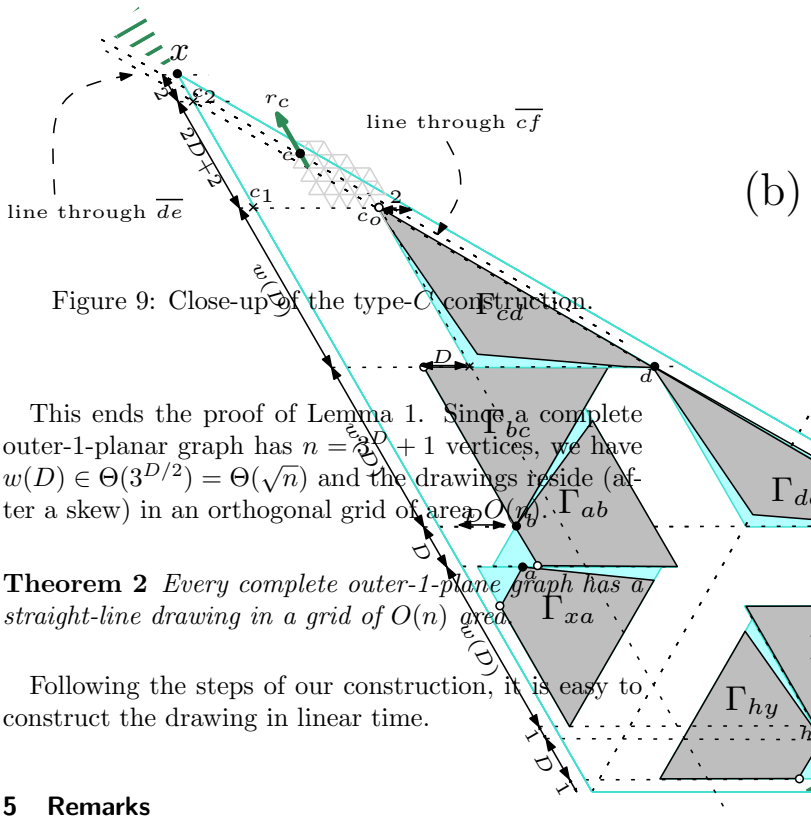
Also note that the attachment point of Γ_{hy} is the lowest point of the drawing, and its ↘-projection onto the bottom side of T has distance $D+2$ from y. Finally path x-c-f-g-h is drawn alternating between slopes in $[-\infty, -\sqrt{3})$ and $-\sqrt{3}$. Therefore as for type-A drawings one argues that y has the flexibility to move within its wedge, as long as nothing is placed between the attachment point z of T and the new location of y.

Creating a type-C drawing. Start with $\Gamma_{xa}, \ldots, \Gamma_{cd}$, placed as described above, but rotate everything by 60°. Let ℓ be the ↗-line that has ↘-distance $w(D)$ from d. Copy and mirror $\Gamma_{xa}, \ldots, \Gamma_{cd}$ along line ℓ to get $\Gamma_{ef}, \ldots, \Gamma_{hy}$. The only subgraph missing is then H_{de}, for which we use a type-C drawing that fits exactly with the existing points for d and e. See Figure 7(b). One verifies that all drawings are disjoint except where they have common poles.

We define the bounding triangle T to be the upward isosceles triangle where the left side is parallel to the left side of T_{xa} and at ↗-distance 1, the bottom side is parallel to the bottom side of T_{hy} and at ↗-distance 1, and the right side is parallel to the top side of T_{de} and at →-distance 2. (Line ℓ is the axis of symmetry for T.) Place x and y (as required for a type-C drawing) at the top and right corner of T. We place c and f on the rays r_c and r_f, with distance one from the start-point of the ray. This places the line through $\overline{cf}$ halfway between the line through $\overline{de}$ and the line through $\overline{xy}$. With this the complete graphs $\{x, y, c, f\}$ and $\{c, d, e, f\}$ of Q are drawn correctly (albeit with very small angles). All other edges of Q can clearly be added.

As for the flexibility of moving x, the same argument as for the type-A drawing applies with respect to the complete graph formed by $\{x, a, b, c\}$. For the complete graph formed by $\{x, c, f, y\}$, observe that $\overline{xy}$ and $\overline{cf}$ are parallel and therefore moving x to some point x' in the wedge (hence strictly above the line through $\overline{cf}$ keeps $\{x', c, f, y\}$ as a strictly convex quadrilateral.

To analyze the length of the shorter sides of T, let c_0 be the top corner of T_{cd}. Observe first (see also Figure 7(b)) that c_0 has ←-distance $2D+2$ to the left side of T and ↘-distance $3w(D)+2D+2$ to the bottom side of T. Now consider the close-up in Figure 9, let c_1 be the ←-projection of c_0 onto the left side of T, and let c_2 be the place where the line through $\overline{de}$ intersects the left side of T. Since $\overline{de}$ has slope $-\sqrt{3}/2$ while $\overline{c_0c_1}$ has slope 0 and $\overline{c_1c_2}$ has slope $-\sqrt{3}$, the triangle $\{c_0, c_1, c_2\}$ is isosceles, and therefore $d(c_1, c_2) = 2D+2$. The ↖-distance from c_2 to x is 2 by definition of T. Therefore the left side of T has length $3w(D)+4D+6 = w(D+2)$.



(b)

Figure 9: Close-up of the type-C construction.

This ends the proof of Lemma 1. Since a complete outer-1-planar graph has $n = 3^D + 1$ vertices, we have $w(D) \in \Theta(3^{D/2}) = \Theta(\sqrt{n})$ and the drawings reside (after a skew) in an orthogonal grid of area $O(n)$.

Theorem 2 *Every complete outer-1-plane graph has a straight-line drawing in a grid of $O(n)$ area.*

Following the steps of our construction, it is easy to construct the drawing in linear time.

5 Remarks

Our result is easily stated, but its proof is annoyingly complicated. The corresponding result for complete outer-planar graphs by Di Battista and Frati [9] has a very elegant proof: Draw a complete binary tree with a special property called "star-shaped", and one can derive a drawing of the balanced outer-planar graph from it. This does not translate to outer-1-planar graphs for multiple reasons. First, any complete outer-planar graph contains a complete binary tree (of roughly the same depth) as a subtree, so after drawing the complete binary tree one "only" has to add some edges. Attempts to generalize this for drawing a complete outer-1-planar graph G led to super-linear area [5]. The dual tree T of G is a complete ternary tree, but it does not map naturally to a subtree of G, and it would not be clear how to expand a drawing of T to one of G. Is there a simpler way to prove Theorem 2?

Also, in the paper by Di Battista and Frati [9] drawing the complete outer-planar graph was really just a warm-up to get results for all outer-planar graphs via star-shaped drawings of trees, useful also for [13]. We studied drawings of complete outer-1-planar graphs in the hopes that it would lead to sub-quadratic area-bounds for drawing *all* outer-1-planar graphs. But this seems significantly harder and obtaining area-bounds that are sub-quadratic (and ideally $O(n^{1+\varepsilon})$) remains open.

References

[1] H. Akitaya, M. Löffler, and I. Parada. How to fit a tree in a box. In *Graph Drawing and Network Visualization (GD'18)*, volume 11282 of *LNCS*, pages 361–367. Springer, 2018.

[2] M. J. Alam, F. J. Brandenburg, and S. G. Kobourov. Straight-line grid drawings of 3-connected 1-planar graphs. In *Graph Drawing (GD 2013)*, volume 8242 of *Lecture Notes in Computer Science*, pages 83–94. Springer, 2013.

[3] C. Auer, C. Bachmaier, F. Brandenburg, A. Gleißner, K. Hanauer, D. Neuwirth, and J. Reislhuber. Outer 1-planar graphs. *Algorithmica*, 74(4):1293–1320, 2016.

[4] T. Biedl, G. Liotta, J. Lynch, and F. Montecchiani. Optimal-area visibility representations of outer-1-planar graphs. In *Graph Drawing and Network Visualization (GD 2021)*, volume 12868 of *Lecture Notes in Computer Science*, pages 287–303. Springer, 2021.

[5] P. Bulatovic. Area-efficient drawings of outer-1-planar graphs. Master's thesis, University of Waterloo, September 2020.

[6] P. Crescenzi, G. D. Battista, and A. Piperno. A note on optimal area algorithms for upward drawings of binary trees. *Comput. Geom.*, 2:187–200, 1992.

[7] P. Crescenzi, P. Penna, and A. Piperno. Linear area upward drawings of AVL trees. *Comput. Geom.*, 9(1-2):25–42, 1998.

[8] H. R. Dehkordi and P. Eades. Every outer-1-plane graph has a right angle crossing drawing. *Int. J. Comput. Geom. Appl.*, 22(6):543–558, 2012.

[9] G. Di Battista and F. Frati. Small area drawings of outerplanar graphs. *Algorithmica*, 54(1):25–53, 2009.

[10] W. Didimo. Density of straight-line 1-planar graph drawings. *Inf. Process. Lett.*, 113(7):236–240, 2013.

[11] R. Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[12] S. Felsner, G. Liotta, and S. Wismath. Straight-line drawings on restricted integer grids in two and three dimensions. *J. Graph Alg. Appl*, 7(4):335–362, 2003.

[13] F. Frati, M. Patrignani, and V. Roselli. LR-drawings of ordered rooted binary trees and near-linear area drawings of outerplanar graphs. *J. Comput. Syst. Sci.*, 107:28–53, 2020.

[14] H. d. Fraysseix, J. Pach, and R. Pollack. Small sets supporting Fary embeddings of planar graphs. In *ACM Symposium on Theory of Computing (STOC '88)*, pages 426–433, 1988.

[15] H. d. Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.

[16] W. Schnyder. Embedding planar graphs on the grid. In *ACM-SIAM Symposium on Discrete Algorithms (SODA '90)*, pages 138–148, 1990.

Appendix

In Figure 10 we show the drawings for the base case in the other situations.

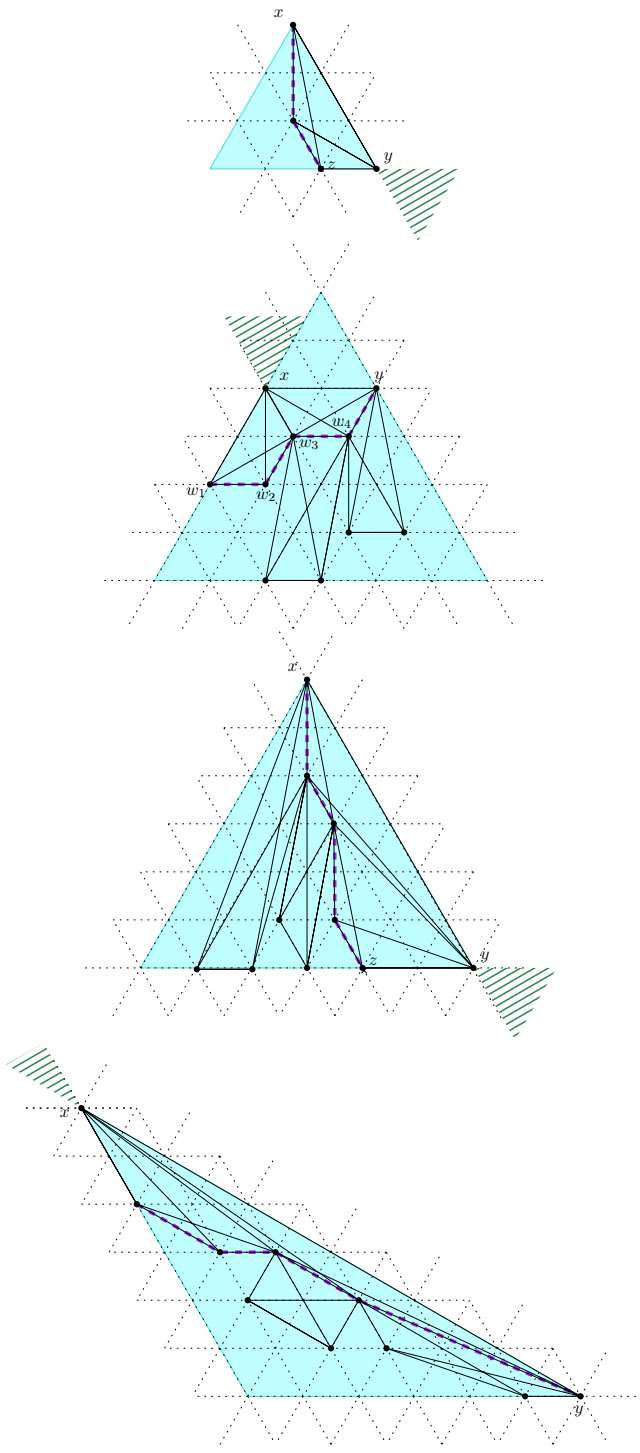


Figure 10: The type-B^+ drawing for $D = 1$ and the drawings (of type A, $B = B^+$ and C) for $D = 2$.

A $\frac{13}{9}$-approximation of the average-$\frac{2\pi}{3}$-MST

Ahmad Biniaz* Prosenjit Bose† Patrick Devaney‡

Abstract

Motivated by the problem of orienting directional antennas in wireless communication networks, we study average bounded-angle minimum spanning trees. Let P be a set of points in the plane and let α be an angle. An α-spanning tree (α-ST) of P is a spanning tree of the complete Euclidean graph induced by P with the restriction that all edges incident to each point $p \in P$ lie in a wedge of angle α with apex p. An α-minimum spanning tree (α-MST) of P is an α-ST with minimum total edge length.

An average-α-spanning tree (denoted by $\overline{\alpha}$-ST) is a spanning tree with the relaxed condition that incident edges to all points lie in wedges with average angle α. An average-α-minimum spanning tree ($\overline{\alpha}$-MST) is an $\overline{\alpha}$-ST with minimum total edge length. In this paper, we focus on $\alpha = \frac{2\pi}{3}$. Let $A\left(\frac{2\pi}{3}\right)$ be the smallest ratio of the length of the $\overline{\frac{2\pi}{3}}$-MST to the length of the standard MST, over all sets of points in the plane. Biniaz, Bose, Lubiw, and Maheshwari (Algorithmica 2022) showed that $\frac{4}{3} \leq A\left(\frac{2\pi}{3}\right) \leq \frac{3}{2}$. In this paper we improve the upper bound and show that $A\left(\frac{2\pi}{3}\right) \leq \frac{13}{9}$.

1 Introduction

A wireless communication network can be represented as a geometric graph in the plane. Each antenna is represented by a point p, its transmission range is represented by a disk with radius r centered at p, and there is an edge between two points if they are within each other's transmission ranges. The problems of assigning transmission ranges to antennas to achieve networks possessing certain properties has been widely studied [3, 5, 9, 12, 14, 15, 16, 17].

In recent years, there has been considerable research on the problem of replacing omni-directional antennas with directional antennas [1, 2, 4, 6, 8, 10, 11, 13, 14, 18]. Here, the transmission range of each point p is an oriented wedge with apex p and angle α. Directional antennas provide several advantages over omni-directional antennas, including less potential for interference, lower power consumption, and reduced area where communications could be maliciously intercepted [3, 18].

Motivated by this problem, Aschner and Katz [2] introduced the α-*Spanning Tree* (α-ST): a spanning tree of the complete Euclidean graph in the plane where all incident edges of each point p lie in a wedge of angle α with apex p. They also presented approximation algorithms for the cases where $\alpha = \frac{\pi}{2}, \frac{2\pi}{3}$, and π, with approximation factors of 16, 6, and 2, respectively, with respect to the MST. For $\alpha = \frac{2\pi}{3}$ and $\alpha = \frac{\pi}{2}$, the approximation ratios have been improved to $\frac{16}{3}$ [6] and 10 [7], respectively. Aschner and Katz further proved the NP-hardness of the problem of computing the α-MST for the $\alpha = \frac{2\pi}{3}$ and $\alpha = \pi$ cases.

Most previous research in this context has been done on the case where α is one fixed value for all antennas [6]. Biniaz et al. [6] extended this concept to an average-α-minimum spanning tree ($\overline{\alpha}$-MST): an α-MST with the relaxed restriction that the average angle of all the wedges is at most α. More formally, a total angle of αn must be allocated among n points p so that each point has a sufficient allowed angle to cover all incident edges. In the case where $\overline{\alpha} = \frac{2\pi}{3}$, they presented an algorithm that achieves an $\overline{\alpha}$-ST of length at most $\frac{3}{2}$ times the length of the MST. They also proved a lower bound of $\frac{4}{3}$ on the approximation factor with respect to the MST.

In this paper, we improve the upper bound on $A\left(\frac{2\pi}{3}\right)$ from $\frac{3}{2}$ to $\frac{13}{9}$. In fact we modify the algorithm of [6] and obtain an $\overline{\alpha}$-ST of length at most $\frac{13}{9}$ times the length of the MST. Our algorithm involves a stronger exploitation of the Euclidean metric than the previous work.

Our improved upper bound immediately gives an approximation algorithm with ratio $\frac{13}{9}$ (with respect to the MST) for the $\overline{\alpha}$-MST problem for any $\alpha \geq \frac{2\pi}{3}$. Similar to that of [6], our algorithm runs in linear time after computing the MST.

1.1 Notation

We use the terms point and vertex interchangeably depending on the context.

To facilitate comparison, we borrow the following notation from [6]. A *maximal path* in a tree is a path with at least two edges where all internal vertex degrees are 2, and the end vertex degrees are not 2. To *contract* a

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

†School of Computer Science, Carleton University, jit@scs.carleton.ca

‡School of Computer Science, University of Windsor, devaneyp@uwindsor.ca

maximal path is to remove all vertices of degree 2 on the path and the edges between them, and add an edge connecting the end vertices. The angle that the incident edges of a vertex in an $\overline{\alpha}$-MST are allowed to fall within is called its *charge*. Charges can be redistributed between vertices. We denote the total length of edges of a geometric graph G by $w(G)$.

As the length of the optimal solution is not known, we use the underlying MST of the points as a lower bound in our analysis. We denote the smallest ratio of the length of the $\overline{\frac{2\pi}{3}}$-MST to the length of the standard MST over all points in the plane as $A\left(\frac{2\pi}{3}\right)$. In [6], it was shown that $\frac{4}{3} \leq A\left(\frac{2\pi}{3}\right) \leq \frac{3}{2}$.

1.2 Outline

The approximation algorithm of [6] for the $\overline{\frac{2\pi}{3}}$-MST starts with a standard MST that has maximum degree 5 (which always exists). Then it re-assigns angle charges from leaves to inner vertices. Their approach first considers the MST with all maximal paths contracted, and then introduces edge shortcuts in each contracted path.

By exploiting additional geometric properties we ensure the connectivity of path vertices with less total charge. This enables us to save some charges. The saved charges allow us to introduce fewer shortcuts than the original algorithm, resulting in a shorter $\overline{\frac{2\pi}{3}}$-ST.

2 The Algorithm of Biniaz et al.

In this section we briefly describe the algorithm of Biniaz et al. [6], which we refer to by "Algorithm 1".

The algorithm starts by computing a degree-5 minimum spanning tree T of the point set, where each vertex holds a charge of $\frac{2\pi}{3}$. Then the algorithm goes through two phases that redistribute the charges and also modify the tree. In the first phase, all maximal paths of T are contracted (to edges), resulting in a tree with no vertices of degree 2, and all other vertices having the same degree as in T. The charge from the leaves are then redistributed among the internal vertices so that each vertex of degree 3, 4, and 5 has a charge of $\frac{4\pi}{3}$, 2π, and $\frac{8\pi}{3}$, respectively. Since the charge of each internal vertex with degree n is at least $\left(1-\frac{1}{n}\right)2\pi$, which covers any set of n edges, all vertices can cover their incident edges. After redistribution, degree-1 vertices have 0 charge and each degree-2 vertex holds its original $\frac{2\pi}{3}$ charge. This redistribution retains a pool of $\frac{4\pi}{3}$ charge that can be split among all leaves in the tree at the end of the algorithm.

In the second phase, the edges of each path $p_1, p_2, \ldots, p_m$ that was contracted in phase 1 are split into two matchings, M_1 and M_2 with equal number of edges (if the path has odd number of edges then the last edge is not in either matching). The edges of the matching with the larger weight are removed, and a set $S = \{(p_1, p_3), (p_3, p_5), ...\}$ of new edges called *shortcuts* are introduced (see Figure 15 of [6], which we include here as Figure 1). By this process, the charge of every new degree-1 vertex is redistributed among other vertices so that each new degree-2 and degree-3 vertex along the path has a charge of π and $\frac{4\pi}{3}$, respectively; this is handled in four cases based on which matching is heavier and whether the path length is even or odd, as shown in Figure 1. Note that the charge given to vertices assigned degree 2 and 3 allows them to cover any set of 2 and 3 edges, respectively.

Let M_1' and M_2' be the union of the edges in the smaller and larger-weight matchings of all contracted paths, respectively. Let T' be the final tree obtained by the above algorithm, and let E be the set of edges of T not in $M_1' \cup M_2'$. Then $w(T) = w(E) + w(M_1') + w(M_2')$. By the triangle equality we have $w(S) \leq w(M_1') + w(M_2')$. Since $w(M_2') \geq w(M_1')$ we get

$$\begin{aligned} w(T') &= w(E) + w(M_1') + w(S) \\ &\leq w(E) + w(M_1') + w(M_1') + w(M_2') \\ &= w(T) + w(M_1') \leq \frac{3}{2} w(T). \end{aligned}$$

3 The Improved Algorithm

We begin by modifying the charge-redistribution of phase 2 of Algorithm 1 with a more careful charge redistribution. In particular we show that the 3 edges, that are incident to new degree-3 vertices, can be covered by $\frac{4\pi}{3} - \frac{\pi}{12}$ charge (meaning that we can *save* the $\frac{\pi}{12}$ charge). We then use the saved charge of $\frac{\pi}{12}$ to achieve a better approximation with respect to the original MST. The following lemma, although very simple, plays an important role in the design of the modified algorithm.

Lemma 1 *It is possible to save at least $\frac{\pi}{12}$ charge from every shortcut performed by phase 2 of Algorithm 1.*

Proof. Consider a shortcut ac between two consecutive edges ab and bc of a contracted path as depicted in Figure 2. Up to symmetry we may assume that ab is in M_2 and thus it has been removed in phase 2 of Algorithm 1. Denote the angle $\angle bca$ by β. Since the path (a, b, c) is part of the MST, ac is the largest edge of the triangle $\triangle abc$, and thus $\angle abc$ is its largest angle. Therefore $\beta \leq \frac{\pi}{2}$.

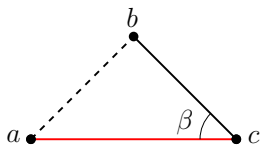


Figure 2: illustration of the proof of Lemma 1.

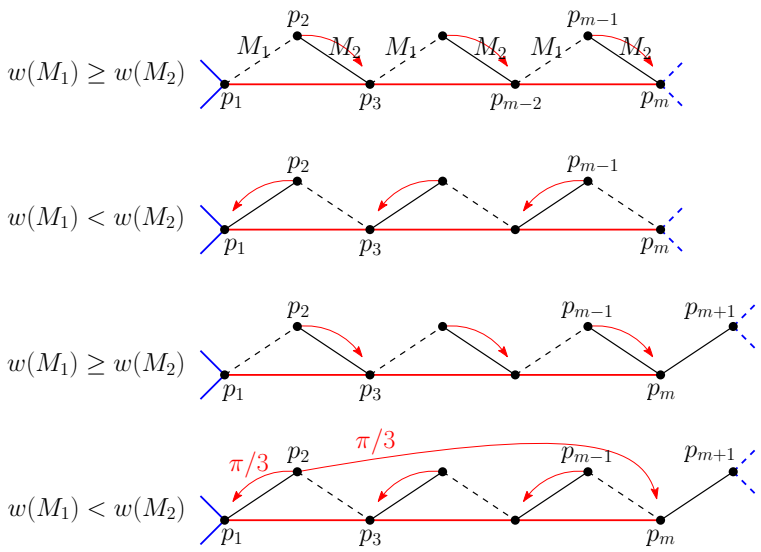


Figure 1: (borrowed from [6]) The contracted path is shown by black segments. The dashed-black edges belong to M_2 and the red edges belong to S.

The replacement of ab by the shortcut ac has not changed the degree of a, has decreased the degree of b by 1, and has increased the degree of c by 1. Thus the charge assigned to a by Algorithm 1 remains enough to cover its incident edges. Since b has degree 1, its $\frac{2\pi}{3}$ charge is free. Algorithm 1 transfers this free charge to c to cover its new edge. We show how to cover all edges incident to c while saving $\frac{\pi}{12}$ charge. If c's original degree (i.e. after phase 1 and before phase 2) was 4 or 5 then it carries at least 2π charge which is sufficient to cover its edges. We may assume that the original degree of c is 1, 2, or 3, in which case it holds a charge of 0, $\frac{2\pi}{3}$, or $\frac{4\pi}{3}$, respectively. Thus the new degree of c (after phase 2) is 2, 3, or 4. Based on this we distinguish three cases.

- If $\deg(c) = 2$ then the two incident edges of c are ac and bc. We can cover these edges by a charge of β ($\leq \frac{\pi}{2}$). Thus we transfer $\frac{\pi}{2}$ charge from b to c and we save $\frac{\pi}{6}$.
- If $\deg(c) = 3$ then we cover β and the smaller of the other two angles at c. Thus the three incident edges to c can be covered by charge of

$$\beta + \left(\frac{2\pi - \beta}{2}\right) = \frac{2\pi + \beta}{2} \leq \frac{2\pi + \frac{\pi}{2}}{2} = \frac{5\pi}{4}.$$

  Thus by transferring $\frac{7\pi}{12}$ from b to c it will have charge of $\frac{5\pi}{4}$ (including its original $\frac{2\pi}{3}$ charge). Thus we save charge of $\frac{2\pi}{3} - \frac{7\pi}{12} = \frac{\pi}{12}$ from b.
- If $\deg(c) = 4$ then we transfer $\frac{\pi}{6}$ charge from b to c and save the remaining $\frac{\pi}{2}$ charge of b. The vertex c now holds $\frac{3\pi}{2}$ charge (including its charge $\frac{4\pi}{3}$ after phase 1) which covers its four incident edges.

□

The following is a direct implication of Lemma 1.

Corollary 2 *It is possible to save $\frac{\pi}{3}$ charge from every four shortcuts that are performed by Algorithm 1.*

3.1 Reversing Shortcuts

In this section, we present an approximation algorithm that uses fewer shortcuts than Algorithm 1. In fact the new algorithm reverses a constant fraction of the shortcuts performed by Algorithm 1.

Theorem 3 *Given a set of n points in the plane and an angle $\alpha \geqslant \frac{2\pi}{3}$, there is an $\overline{\alpha}$-spanning tree of length at most $\frac{13}{9}$ times the length of the MST. Furthermore, there is an algorithm to find such an $\overline{\alpha}$-ST that runs in linear time after computing the MST.*

Proof. Let T be a degree-5 minimum spanning tree of the point set, and T' be the $\overline{\frac{2\pi}{3}}$-spanning tree obtained from T by Algorithm 1.

Consider the sequence of shortcuts introduced by Algorithm 1 along each contracted path. Let $s_1, s_2, ..., s_m$ be the concatenation of the sequences for all contracted paths. We split these shortcuts into nine sets $S_0, \ldots, S_8$ such that $s_i \in S_{(i \mod 9)}$ for each $i \in \{1, \ldots, m\}$. Note that no two adjacent shortcuts in the same contracted path will be in the same set S_i. Moreover the number of shortcuts in any two sets S_i and S_j differ by at most 1. Recall that the edges of each contracted path in Algorithm 1 are split into two matchings M_1 and M_2. Let M_1' be the set of edges that are kept in the tree (i.e. M_1' is the union of the smaller-weight matchings from each

contracted path), and let the set of edges in the heavier matchings be M_2'. Among $S_0, \ldots, S_8$, let S_8 be the one whose corresponding edges in M_1' have the largest total weight.

Our plan now is to reverse the shortcuts in S_8, i.e., to replace them by their corresponding edges in M_2'. Let S' be the union of $S_0, \ldots, S_7$. Notice that $|S'| \geq 8 \cdot (|S_8| - 1)$. Let C denote the pool of charges that is obtained after phase 1 of Algorithm 1, and recall that it contains $\frac{4\pi}{3}$ charge. For each shortcut in S' we reassign the charges between its corresponding points to save at least $\frac{\pi}{12}$ charge (as shown in Lemma 1), and add this charge to C. Thus the total charge of C is at least

$$\frac{4\pi}{3} + 8 \cdot (|S_8| - 1) \cdot \frac{\pi}{12} = (|S_8| + 1) \cdot \frac{2\pi}{3}.$$

We will show that to reverse each shortcut from S_8 it suffices to take $\frac{2\pi}{3}$ charge from C.

Consider any shortcut ac from S_8 between two consecutive edges ab and bc of a contracted path as depicted in Figure 3. We *reverse* this shortcut by replacing ac with the removed edge ab. We also reclaim any portion of b's charge that was transferred to c. Thus the reverse operation brings the charges of b and c back to what it was after phase 1 and before phase 2; in particular it brings the charge of b back to $\frac{2\pi}{3}$. There is one exceptional case where $w(M_1) < w(M_2)$ and the path has odd number of edges (the last case in Figure 1 where p_3, p_2, p_1 play the roles of a, b, c, respectively). In this case the charge of b (i.e. p_2) would be $\frac{\pi}{3}$ as p_m holds the other $\frac{\pi}{3}$ portion. (Since no two shortcuts in S_8 are adjacent in the same contracted path, we can analyze a reverse operation independently of others. Notice, however, that it is possible that two or more shortcuts of S_8 are adjacent at a vertex that has degree at least 3 after phase 1. In this case, the charge of such a vertex suffices to cover its edges after reversing the shortcuts since it will have at least $\frac{\pi}{6}$ charge added for each new edge introduced by the process described in Lemma 1.) The reverse operation does not change the degree of a and thus its charge remains sufficient to cover its edges. The reverse operation makes b of degree 2 and decreases the degree of c by 1.

We take $\frac{\pi}{3}$ charge from C for b to bring it to a charge of π, which covers its two incident edges. If $\deg(c) = 1$ or $\deg(c) \geq 3$, its charge is sufficient to cover its edges. If $\deg(c) = 2$ then we take an additional charge of $\frac{\pi}{3}$ from C for c to cover its two incident edges. In the exceptional where $w(M_1) < w(M_2)$ and the path has odd number of edges (the last case in Figure 1), $p_2 = b$ holds $\frac{\pi}{3}$ charge, so we take $\frac{2\pi}{3}$ from C for p_2 to cover its two incident edges. Since $p_1 = c$ is of degree 1 or at least 3 (as the contracted path is maximal), its charge (acquired after phase 1) is sufficient to cover its edges. Thus, in the worst case we take $\frac{2\pi}{3}$ from C to reverse every shortcut.

After reversing all shortcuts in S_8, the pool C is left with at least $\frac{2\pi}{3}$ charge which can be distributed among the leaves of the resulting tree.

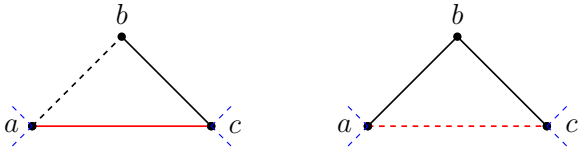


Figure 3: Left: The tree T' before reversing shortcut ac. Right: The tree T'' after reversing ac.

Let T'' be the $\overline{\frac{2\pi}{3}}$-ST tree obtained from T' after reversing all shortcuts in S_8. Let E be the set of edges of T'' not in $M_1' \cup M_2'$. Let E' be the set of all edges of $M_1' \cup M_2'$ that correspond to the shortcuts in S_8. Let $M_1'' = M_1' \setminus E'$ and $M_2' = M_2' \setminus E'$ (i.e. all edges in M_1' and M_2', respectively, with a shortcut between their endpoints in T''). Then,

$$\begin{aligned} w(T'') &= w(E) + w(E') + w(S') + w(M_1'') \\ &\leq w(E) + w(E') + w(M_1'') + w(M_2'') + w(M_1'') \\ &= w(T) + w(M_1''). \end{aligned}$$

Since S_8 has the largest corresponding M_1' weight, $w(M_1'') \leq \frac{8}{9} w(M_1') \leq \frac{8}{9} \cdot \frac{1}{2} w(T) = \frac{4}{9} w(T)$. Thus,

$$w(T'') \leq w(T) + \frac{4}{9} w(T) = \frac{13}{9} w(T).$$

□

With Theorem 3 in hand, we report the following bound for $A\left(\frac{2\pi}{3}\right)$.

Corollary 4 $\frac{4}{3} \leq A\left(\frac{2\pi}{3}\right) \leq \frac{13}{9}$.

4 Conclusions

An obvious open problem is to further tighten the gap between the upper bound of $\frac{13}{9}$ and lower bound of $\frac{4}{3}$ for $\mathrm{A}\left(\frac{2\pi}{3}\right)$. This could be done by either introducing a new algorithm with a better approximation factor, or by finding a new set of points whose $\overline{\frac{2\pi}{3}}$-MST must have a weight of more than $\frac{4}{3}$ times that of the MST.

References

[1] E. Ackerman, T. Gelander, and R. Pinchasi. Ice-creams and wedge graphs. *Computational Geometry: Theory and Applications*, 46(3):213–218, 2013.

[2] R. Aschner and M. J. Katz. Bounded-angle spanning tree: Modeling networks with angular constraints. *Algorithmica*, 77(2):349–373, 2017.

[3] R. Aschner, M. J. Katz, and G. Morgenstern. Do directional antennas facilitate in reducing interferences? In *Proceedings of the 13th Scandinavian Symposium and Workshops on Algorithm Theory* (*SWAT*), pages 201–212, 2012.

[4] S. Ashur and M. J. Katz. A 4-approximation of the $\frac{2\pi}{3}$-MST. In *Proceedings of the 17th International Symposium on Algorithms and Data Structures (WADS)*, pages 129–143, 2021.

[5] A. Biniaz. Euclidean bottleneck bounded-degree spanning tree ratios. In *ACM-SIAM Symposium on Discrete Algorithms* (*SODA*), 2020.

[6] A. Biniaz, P. Bose, A. Lubiw, and A. Maheshwari. Bounded-angle minimum spanning trees. *Algorithmica*, 84(1):150–175, 2022.

[7] A. Biniaz, M. Daliri, and A. H. Moradpour. A 10-Approximation of the $\frac{\pi}{2}$-MST. In *Proceedings of the 39th International Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 13:1–13:15, 2022.

[8] P. Bose, P. Carmi, M. Damian, R. Y. Flatland, M. J. Katz, and A. Maheshwari. Switching to directional antennas with constant increase in radius and hop distance. *Algorithmica*, 69(2):397–409, 2014.

[9] P. M. Camerini. The min-max spanning tree problem and some extensions. *Information Processing Letters*, 7(1):10–14, 1978.

[10] I. Caragiannis, C. Kaklamanis, E. Kranakis, D. Krizanc, and A. Wiese. Communication in wireless networks with directional antennas. In *Proceedings of the 20th Annual ACM Symposium on Parallelism in Algorithms and Architectures* (*SPAA*), pages 344–351, 2008.

[11] P. Carmi, M. J. Katz, Z. Lotker, and A. Rosén. Connectivity guarantees for wireless networks with directional antennas. *Computational Geometry: Theory and Applications*, 44(9):477–485, 2011.

[12] T. M. Chan. Euclidean bounded-degree spanning tree ratios. *Discrete & Computational Geometry*, 32(2):177–194, 2004.

[13] M. Damian and R. Y. Flatland. Spanning properties of graphs induced by directional antennas. *Discrete Mathematics, Algorithms and Applications*, 5(3), 2013.

[14] S. Dobrev, E. Kranakis, D. Krizanc, J. Opatrny, O. M. Ponce, and L. Stacho. Strong connectivity in sensor networks with given number of directional antennae of bounded angle. *Discrete Mathematics, Algorithms and Applications*, 4(3), 2012.

[15] S. Dobrev, E. Kranakis, O. M. Ponce, and M. Plzík. Robust sensor range for constructing strongly connected spanning digraphs in UDGs. In *Proceedings of the 7th International Computer Science Symposium in Russia* (*CSR*), pages 112–124, 2012.

[16] M. M. Halldórsson and T. Tokuyama. Minimizing interference of a wireless ad-hoc network in a plane. *Theoretical Computer Science*, 402(1):29–42, 2008.

[17] S. Khuller, B. Raghavachari, and N. E. Young. Low-degree spanning trees of small weight. *SIAM Journal on Computing*, 25(2):355–368, 1996.

[18] E. Kranakis, F. MacQuarrie, and O. M. Ponce. Connectivity and stretch factor trade-offs in wireless sensor networks with directional antennae. *Theoretical Computer Science*, 590:55–72, 2015.

Weighted shortest path in equilateral triangular meshes*

Prosenjit Bose† Guillermo Esteban‡ § Anil Maheshwari¶

Abstract

Let $\mathcal{T}$ be a tessellation composed of equilateral triangular regions, in which each region has an associated positive weight. We present two approximation algorithms for solving the Weighted Region Problem. Our algorithms are based on the method of discretizing the space by placing points on the cells of the tessellation and using Dijkstra's algorithm for computing the weighted shortest path in the geometric graph obtained by such a discretization. For a given parameter $\varepsilon \in [0, 1]$, the weight of our paths are $\left(1 + \frac{14(4\sqrt{2}\sqrt{\sqrt{6\varepsilon}+8}-\sqrt{6\varepsilon}-8)-16(7+\varepsilon)}{(-4\sqrt{2}\sqrt{\sqrt{6\varepsilon}+8}+\sqrt{6\varepsilon}+8)(7-\varepsilon)}\right) \leq 1 + 0.39\varepsilon$, and $1 + \varepsilon$ (using fewer points) times the cost of the actual shortest path.

1 Introduction

In this paper, we study optimal obstacle-avoiding paths from a starting point s to an ending point t in the 2-dimensional plane. Shortest path problems are among the most studied problems in computational geometry. These problems have applications in several areas such as robotics [16], video-games [11, 17], and geographical information systems (GIS) [7], among others.

Mitchell and Papadimitriou [13, 12] examined a generalization of the shortest path problem, called the Weighted Region Problem (WRP). They allowed the two-dimensional space to be subdivided into regions, each of which has a (non-negative) weight associated to it, representing the cost per unit distance of traveling in that region. They provided an approximating algorithm which computes a $(1 + \varepsilon)$-approximation path in $O(n^8 \log \frac{nNW}{w\varepsilon})$ time, where N is the maximum integer coordinate of any vertex of the subdivision, W (respectively, w) is the maximum finite (respectively, minimum non-zero) integer weight assigned to faces of the subdivision.

Motivated by this result, several authors proposed algorithms for computing approximated paths, reducing the running time and producing geometric problem instances with fewer "bad" configurations (e.g., the Delaunay triangulation is used to maximize the minimum angle).

The most common scheme followed in the literature is to position Steiner points, and then build a graph by connecting pairs of Steiner points. An approximate solution is constructed by finding a shortest path in this graph, by using well-known combinatorial algorithms (e.g., Dijkstra's algorithm).

Aleksandrov et al. [3, 4] proposed placing Steiner points on edges of an appropriate mesh, and then, interconnecting the Steiner points within each face. Since an infinite number of Steiner points would be required for the approximation, they constructed a star shaped polygon around each vertex of the mesh; ensuring that Steiner points are placed outside these regions. They also deal with the problem of large sized graphs. By deriving geometric properties of Snell's law of refraction for a discrete domain, they reduced the search space. They employed a pruned Dijskstra's algorithm where the execution is restricted to a sparse set of potential edges, given that the preceding edge on a path is known. Employing all these steps together and using geometric spanners they obtained a $(1 + \varepsilon)$-approximation path.

Reif and Sun [18] used the same discretization approach as in [4]. They employed an algorithm called BUSHWHACK to compute an optimal path in the discrete graph by dynamically adding edges.

In addition, Aleksandrov et al. [5] used a similar approach as in [4], but placing, for the first time, the Steiner points on the bisectors of the angles, and not on the face boundaries. However, this complicates computation of the discrete path because now the edges join Steiner points that belong to neighboring faces.

See Table 1 for the time complexity of the approximation algorithms designed following these schemes.

Recently, it has been shown that the WRP cannot be solved exactly within the Algebraic Computation Model over the Rational Numbers (ACM$\mathbb{Q}$) [6], i.e., a solution to an instance of the WRP cannot be expressed as a closed formula in ACM$\mathbb{Q}$. This emphasizes the need for high-quality approximate paths instead of optimal paths. So, in practice, the geometric space is discretized

*Partially supported by NSERC, project PID2019-104129GB-I00/MCIN/AEI/ 10.13039/501100011033 of the Spanish Ministry of Science and Innovation, and H2020-MSCA-RISE project 734922 - CONNECT.

†School of Computer Science, Carleton University, Canada, jit@scs.carleton.ca

‡Departamento de Física y Matemáticas, Universidad de Alcalá, Spain,

§School of Computer Science, Carleton University, Canada g.esteban@uah.es

¶School of Computer Science, Carleton University, Canada, anil@scs.carleton.ca

Time complexity	Reference
$O(n^8 \log \frac{nNW}{w\varepsilon})$	[13]
$O(N^4 \log \left(\frac{NW}{w\varepsilon}\right) \frac{n}{\varepsilon^2} \log \frac{nN}{\varepsilon})$	[3]
$O(N^2 \log \left(\frac{NW}{w}\right) \frac{n}{\varepsilon} \log \frac{1}{\varepsilon}(\frac{1}{\sqrt{\varepsilon}} + \log n))$	[4]
$O(N^2 \log \left(\frac{NW}{w}\right) \frac{n}{\varepsilon} \log \frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$	[18]
$O(N^2 \log \left(\frac{NW}{w}\right) \frac{n}{\sqrt{\varepsilon}} \log \frac{n}{\varepsilon} \log \frac{1}{\varepsilon})$	[5]

Table 1: ε-approximation algorithms for the Weighted Region Problem.

using grids. The concept of grid is essential and heavily used in digital elevation models (DEMs) [2, 10] and in video games [8]. Because of their symmetry and natural neighborhood structure, regular triangle meshes are preferred over square and hexagonal.

Although it is the most complex among the three regular tessellations (it has the largest number of vertices), it has various advantages in applications, e.g., the distance between the vertices of adjacent cells is always the same, which simplifies distance calculations. Triangles can represent complex shapes, and they can include hexagonal grids. Although they are built with triangles in two different orientations, each pixel has 12 neighbors sharing at least a corner, which gives a valid alternative for applications in image processing. Recently, various image processing algorithms have been defined and implemented for the triangular grid, such as discrete tomography [14, 15], thinning [9], and mathematical morphology [1].

1.1 Our results

In this paper, we present algorithms for computing approximate shortest paths between two vertices s and t on a triangular tessellation. We work with the particular case in which every cell of the mesh is an equilateral triangle. In addition, each cell has a positive real weight associated to it.

Our results are based on a previous work of Aleksandrov et al. [5]. With a finer analysis, we improve the results in two different ways:

1. If we use the same number of Steiner points as in [5], we prove that the approximation factor is minimized when placing the Steiner points on the edges of the cells. In addition, we provide an upper bound on the quality of the approximation path with respect to the actual shortest path. Our result gives an approximation factor which is at least $\frac{5(1+\varepsilon)(7-\varepsilon)}{7(5+\varepsilon)} \geq 1 + 0.428\varepsilon$ times better than the previous result.

2. If we decide to maintain the approximation factor in each of the cells, we provide a discretization using fewer Steiner points than in [5]. We increase the distance between Steiner points in each segment by about a factor of $\frac{7-\sqrt{\varepsilon}}{40} \approx 0.175 - 0.025\sqrt{\varepsilon}$, which decreases the running time of the algorithm to determine the approximation path.

To solve these problems, we use the traditional technique of partitioning the continuous 2D space into a discrete space by designing an appropriate graph. Differently from the previous work of Aleksandrov et al. [5], the discretization is done by placing Steiner points along a segment from each vertex of the mesh inclined by α radians. Then, we minimize the number of Steiner points to be added by optimization over the angle $\alpha \in [0, \frac{\pi}{3}]$. All these improvements were obtained by taking into account trigonometric properties from the points of entry of the paths into the cells and carrying out a thorough analysis when optimizing the results.

The paper is organized as follows: we start Section 2 by introducing the definitions that are needed for the forthcoming calculations. We also provide Lemma 2, where two properties about the entry and leaving points of the shortest path are calculated. Then, in Section 2.1, with the same number of Steiner points proposed by Aleksandrov et al. [5] we improve their results on the approximation factor of the whole path. Similarly, in Section 2.2, we fix the approximation factor of $1 + \frac{\varepsilon}{2}$ for each segment joining two points on the edges of a cell, and we optimize the number of Steiner points to be placed in each triangular cell. Finally, in Section 3 we compare the results that we obtain with the previous results from [5].

2 Equilateral triangle mesh

Let $\mathcal{T}$ be a triangular tessellation in the 2-dimensional space. We will suppose that $\mathcal{T}$ is a connected union of a finite number of equilateral triangles, denoted by $T_1, \ldots, T_n$. Two triangles of the set can share a vertex, an edge, or not being adjacent. Each face T_i, $i \in \{1, \ldots, n\}$, of the tessellation has a positive weight ω_i associated to it. This weight represents the cost of traveling through a face per unit of Euclidean distance.

Any continuous (rectifiable) curve lying in $\mathcal{T}$ is called a path. Every path in $\mathcal{T}$ consists of a sequence of segments, whose endpoints are on the edges of $\mathcal{T}$. Each of these segments is of one of the following two types:

- face-crossing: the endpoints belong to adjacent edges;
- edge-using: the endpoints belong to the same edge of a face.

The cost of a path π is given by $\|\pi\| = \sum_{i=1}^{n} \omega_i \|\pi_i\|$, where $\|\pi_i\|$ denotes the Euclidean length of the intersection between π and a triangle T_i. In case π_i is an edge-using segment, then the cost of traveling along that

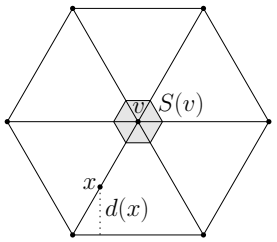


Figure 1: $d(x)$ is the length of the dotted segment. The vertex vicinity of v is depicted in gray.

edge is the minimum of the weights of the triangles incident to the edge. Given two distinct points s and t in $\mathcal{T}$, a shortest path $\pi(s,t)$ is a path that minimizes the weighted distance between s and t. Without loss of generality, we may assume that s and t are vertices of the tessellation.

A path $\pi(s,t)$ is represented by a sequence of points $s = a_0, \ldots, a_\ell = t$ lying on the edges. The points a_i, $i \in \{1, \ldots, \ell - 1\}$, that are not vertices of the tessellation are called bending points of the path.

Following notation from [5], the function $d(x)$ is defined as the minimum Euclidean distance from a point x on a side of a triangle to the boundary of the union of the faces containing x, see Figure 1.

For each vertex v of the tessellation $\mathcal{T}$, let $\omega_{max}(v)$ and $\omega_{min}(v)$ be, respectively, the maximum (finite) weight and the minimum weight of the faces adjacent to v. Let $r(v)$ be the weighted radius of the vertex v defined as follows:

$$r(v) = \frac{\omega_{min}}{7\omega_{max}} d(v)$$

Then, for each face adjacent to v, an equilateral triangle with side length $\varepsilon r(v)$ is defined. Around v, a regular hexagon $S(v)$, called the vertex-vicinity of v, is obtained, see Figure 1. Let e_1 be the edge of T_j that is encountered first when traversing the edges of T_j from v in counterclockwise order. We also define $\ell_v(j,\alpha)$ as the segment in T_j from v inclined by α radians with respect to e_1, see Figure 2.

Definition 1 *Let T_j be an equilateral triangle of the tessellation $\mathcal{T}$, and let v be a vertex of T_j. We define a set of Steiner points $p_0, p_1, \ldots, p_k$ on $\ell_v(j,\alpha)$ by:*

$$|p_{i-1}p_i| = a(\varepsilon)\sin\alpha |vp_{i-1}|, \text{ for } i \in \{1, \ldots, k\}, \quad (1)$$

where $a : (0,1] \to \mathbb{R}$, p_0 is the intersection point between $\ell_v(j,\alpha)$ and the boundary of $S(v)$, and k is the largest integer such that $|vp_k| \leq |\ell_v(j,\alpha)|$.

Lemma 2 *Let e_1, e_2 be two edges adjacent to v in T_j, and let x_1, x_2 be two points in e_1 and e_2, respectively. Let p' be the intersection point between $|x_1x_2|$ and $\ell_v(j,\alpha)$. Let p be the closest Steiner point to p'.*

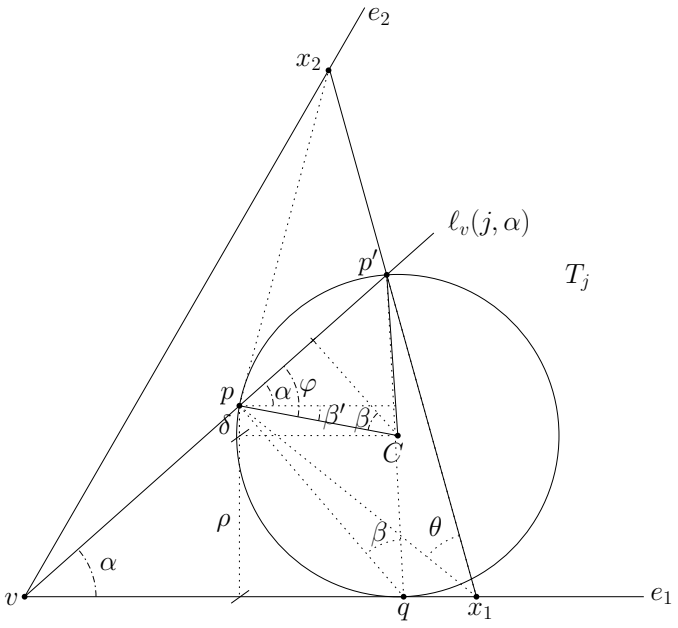


Figure 2: Illustration of Lemma 2.

- *Let θ be the angle $\angle px_1p'$, then*

$$\tan\frac{\theta}{2} \leq \frac{2\sqrt{2}\sqrt{a\sin\alpha + 2} - a\sin\alpha - 4}{a(\cos\alpha - 1)}. \quad (2)$$

- *Let $A = \frac{2\sqrt{2}\sqrt{a\sin\alpha+2}-a\sin\alpha-4}{a(\cos\alpha-1)}$ and $B = \frac{2\sqrt{2}\sqrt{a\sin(\frac{\pi}{3}-\alpha)+2}-a\sin(\frac{\pi}{3}-\alpha)-4}{a(\cos(\frac{\pi}{3}-\alpha)-1)}$, then*

$$|x_1p| + |px_2| \leq \left(1 + \frac{AB}{1-AB}\right)|x_1x_2|. \quad (3)$$

Proof. Let p' belong to the segment $[p_i, p_{i+1}]$. We want to calculate an upper bound on the value of the angle θ, for any x_1 and x_2. It is well known, that θ is maximum when the circle through p, p' and x_1 is tangent to e_1. So, let q be this point of tangency, and let C be the center of the circle. Then, an upper bound on the angle θ is given by the angle $\beta = \angle pqp'$, i.e., $\theta \leq \beta$. Let ρ be the radius of the circle through pqp'. Let $\rho + \delta$ be the distance from p to e_1, see Figure 2. We define the angle $\angle p'pC$ as φ. So, considering the triangle formed by p, C and the midpoint of the segment $\overline{pp'}$, we have that $\pi = \frac{\pi}{2} + \beta + \varphi \Rightarrow \varphi = \frac{\pi}{2} - \beta$. We also define the angle β' as $\beta' = \varphi - \alpha = \frac{\pi}{2} - \beta - \alpha$. Hence,

$$\begin{cases} \sin\beta' = \frac{\delta}{\rho} \Rightarrow \delta = \rho\sin(\frac{\pi}{2} - \beta - \alpha) \\ \sin\alpha = \frac{\rho+\delta}{|vp|} = \frac{\rho(1+\sin(\frac{\pi}{2}-\beta-\alpha))}{|vp|}. \end{cases}$$

Note that the angle β is maximum when p' is the midpoint of the segment $\overline{p_ip_{i+1}}$. Thus, if $p = p_i$, and using equation (1) with the triangle $\triangle qp_ip'$, we get that

$$\begin{aligned}
\sin\beta &= \frac{\frac{|p_ip_{i+1}|}{2}}{2\rho} = \frac{\frac{a}{2}\frac{\rho(1+\sin(\frac{\pi}{2}-\beta-\alpha))}{|vp|}|vp_i|}{2\rho} \\
&= \frac{\frac{a}{2}(1+\sin(\frac{\pi}{2}-\beta-\alpha))}{2} = \frac{\frac{a}{2}(1+\cos\beta\cos\alpha-\sin\beta\sin\alpha)}{2} \\
&\Leftrightarrow \sin\beta + \frac{\frac{a}{2}\sin\beta\sin\alpha}{2} = \frac{\frac{a}{2}(1+\cos\beta\cos\alpha)}{2} \\
&\Leftrightarrow \sin\beta\left(\frac{2+\frac{a}{2}\sin\alpha}{2}\right) = \frac{\frac{a}{2}(1+\cos\beta\cos\alpha)}{2} \\
&\Leftrightarrow \sin\beta = \frac{\frac{a}{2}(1+\cos\beta\cos\alpha)}{2+\frac{a}{2}\sin\alpha} \\
&\Leftrightarrow \tan\frac{\beta}{2} = \frac{2\sqrt{\frac{a}{2}\sin\alpha+1}-\frac{a}{2}\sin\alpha-2}{\frac{a}{2}(\cos\alpha-1)} \\
&= \frac{2\sqrt{2}\sqrt{a\sin\alpha+2}-a\sin\alpha-4}{a(\cos\alpha-1)}.
\end{aligned}$$

Now, suppose that $p = p_{i+1}$. Then, following an analogous reasoning as before, and using the fact that $|vp_i| < |vp'|$ for triangle $\triangle qp'p_{i+1}$, we get that

$$\begin{aligned}
\sin\beta &= \frac{\frac{|p_ip_{i+1}|}{2}}{2\rho} = \frac{\frac{a}{2}\frac{\rho(1+\sin(\frac{\pi}{2}-\beta-\alpha))}{|vp|}|vp_i|}{2\rho} \\
&< \frac{\frac{a}{2}(1+\sin(\frac{\pi}{2}-\beta-\alpha))}{2} \\
&\Leftrightarrow \tan\frac{\beta}{2} < \frac{2\sqrt{2}\sqrt{a\sin\alpha+2}-a\sin\alpha-4}{a(\cos\alpha-1)}.
\end{aligned}$$

From the results above, we get that β is maximized when $p = p_i$, hence equation (2) is proved.

Finally, we prove equation (3). Let θ, θ_1, and θ_2 be the angles of the triangle px_1x_2 at p, x_1, and x_2, respectively, see Figure 3.

Since $\theta_1 + \theta_2 + \theta = \pi$, we known that

$$\begin{aligned}
|x_1p| + |px_2| &\leq \left(1 + \frac{2\sin\frac{\theta_1}{2}\sin\frac{\theta_2}{2}}{\sin\frac{\theta}{2}}\right)|x_1x_2| \\
&= \left(1 + \frac{2\tan\frac{\theta_1}{2}\tan\frac{\theta_2}{2}}{1-\tan\frac{\theta_1}{2}\tan\frac{\theta_2}{2}}\right)|x_1x_2|.
\end{aligned}$$

Hence, using equation (2) for θ_1 and θ_2, we get the desired formula. □

The results in Lemma 2 depend on the value of a function $a(\varepsilon)$. In order to improve the results in [5] for equilateral meshes, we need to give a value for this function.

2.1 Fixing the number of Steiner points

We first fix the distance between consecutive Steiner points, which implies fixing the total number of Steiner points in each triangular face. In this way, we are improving the upper bound on the distance from x_1 to x_2 through a Steiner point p. In [5], the distance between Steiner points on a segment was defined as $|p_{i-1}p_i| = \sqrt{\frac{\varepsilon}{2}}\sin\frac{\beta}{2}|vp_{i-1}|$, where $\beta = 2\alpha$. So, if we substitute $a(\varepsilon) = \sqrt{\frac{\varepsilon}{2}}$ in equation (2), we get that:

$$\begin{aligned}
\tan\frac{\theta_1}{2} &\leq \frac{2\sqrt{2}\sqrt{\sqrt{\frac{\varepsilon}{2}}\sin\alpha+2}-\sqrt{\frac{\varepsilon}{2}}\sin\alpha-4}{\sqrt{\frac{\varepsilon}{2}}(\cos\alpha-1)} \\
\tan\frac{\theta_2}{2} &\leq \frac{2\sqrt{2}\sqrt{\sqrt{\frac{\varepsilon}{2}}\sin(\frac{\pi}{3}-\alpha)+2}-\sqrt{\frac{\varepsilon}{2}}\sin(\frac{\pi}{3}-\alpha)-4}{\sqrt{\frac{\varepsilon}{2}}(\cos(\frac{\pi}{3}-\alpha)-1)},
\end{aligned}$$

where θ_1, and θ_2 are the angles of the triangle $\triangle px_1x_2$ at x_1, and x_2, respectively, see Figure 3.

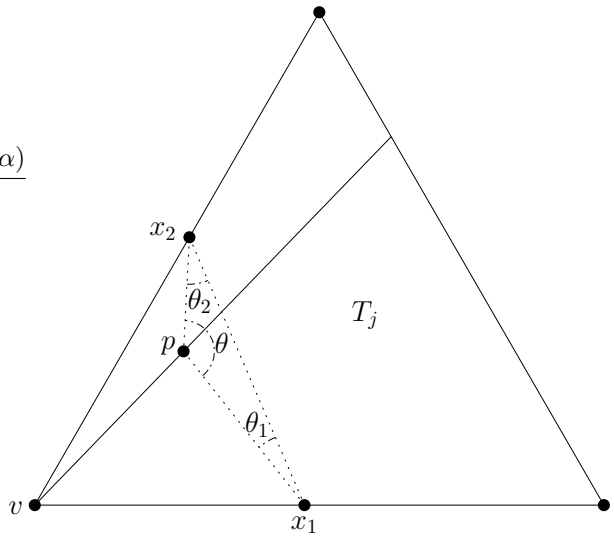


Figure 3: Illustration of Lemma 2.

We want to minimize the upper bound on equation (3) when $a(\varepsilon) = \sqrt{\frac{\varepsilon}{2}}$. Thus, we get that this value is maximized when $\alpha = \frac{\pi}{6}$, i.e., when the Steiner points are placed at the bisectors of the triangles, and minimized when $\alpha = \frac{\pi}{3}$, i.e., when the Steiner points are placed on the sides of the triangles. Hence, using Lemma 2 when placing the Steiner points on the sides of the triangles gives us the following result:

Proposition 3 *Let x_1 and x_2 be two points on two edges e_1 and e_2 of a triangle T_j, and outside the vertex vicinity of the vertex v incident to e_1 and e_2. If p is the Steiner point closest to the intersection between segment x_1x_2 and the segment $\ell_v(j, \frac{\pi}{3})$, then*

$$\begin{aligned}
|x_1p| + |px_2| &= \left(\frac{4\sqrt{2}\sqrt{\sqrt{6\varepsilon}+8}-\sqrt{6\varepsilon}-24}{-4\sqrt{2}\sqrt{\sqrt{6\varepsilon}+8}+\sqrt{6\varepsilon}+8}\right)|x_1x_2| \\
&\lessapprox 1.042\cdot|x_1x_2| \qquad (4)
\end{aligned}$$

Once we have the approximation factor in each of the cells, we need to calculate the approximation factor of

the whole path. We first define a graph G_ε that consists of a set of vertices V_ε, and a set of edges E_ε. Using the corners of the triangles and the set of Steiner points introduced in Definition 1, we create the set of vertices V_ε. For the set of edges we need the notion of neighbor bisectors introduced by Aleksandrov et al. [5]. A bisector is a neighbor to itself. Two different bisectors are neighbors if they belong to the same face of $\mathcal{T}$. Now, consider a pair (ℓ_1, ℓ_2) of neighbor bisectors. We join any pair of nodes p and q lying on ℓ_1 and ℓ_2, respectively. The set of all pairs (p, q) is the set E_ε of edges. Once we have the graph G_ε associated to the discretization, we proceed to compare the weighted length of the approximation path and the actual shortest path in Theorem 4.

Theorem 4 *Let $\pi(s,t)$ be a weighted shortest path between two different vertices s and t on $\mathcal{T}$. There exists a path $\tilde{\pi}(s,t)$ in G_ε such that $\|\tilde{\pi}(s,t)\| \leq \left(1 + \frac{14(4\sqrt{2}\sqrt{\sqrt{6\varepsilon}+8} - \sqrt{6\varepsilon} - 8) - 16(7+\varepsilon)}{(-4\sqrt{2}\sqrt{\sqrt{6\varepsilon}+8} + \sqrt{6\varepsilon}+8)(7-\varepsilon)}\right) \cdot \|\pi(s,t)\|$.*

Proof. Let $(s = v_0, v_1, \ldots, v_n = t)$ be the ordered set of vertices of $\mathcal{T}$ such that $\pi(s,t)$ intersects their vertex vicinities. Let $a_i, b_i,\ i \in \{0, \ldots, n\}$, be, respectively, the last and first bending point on $\pi(s,t)$ that is in the vertex vicinity of v_i. Thus, we obtain a sequence of bending points $s = b_0, a_0, b_1, a_1, \ldots, a_{n-1}, b_n, a_n = t$ on $\pi(s,t)$ such that segments of $\pi(s,t)$ between a_i and b_i are not contained in the vertex vicinities.

Consider the subsegment $\pi(a_i, b_{i+1})$, for some $0 \leq i < n$. A subpath $\pi'(v_i, v_{i+1})$ is defined [5] as the path from v_i to v_{i+1} along the sequence of bending points of $\pi(a_i, b_{i+1})$. Using the triangle inequality, the fact that $a_i \in S(v_i)$, $b_{i+1} \in S(v_{i+1})$, and the definition of weighted radius, we get that

$$\begin{aligned}\|\pi'(v_i, v_{i+1})\| &\leq \|v_i a_i\| + \|\pi(a_i, b_{i+1})\| + \|b_{i+1} v_{i+1}\| \\ &\leq \frac{\varepsilon}{7}\omega_{min}(v_i)d(v_i) + \|\pi(a_i, b_{i+1})\| \\ &\quad + \frac{\varepsilon}{7}\omega_{min}(v_{i+1})d(v_{i+1}).\end{aligned}$$

Therefore, we obtain the path $\pi'(s,t) = \pi'(s, v_1) \cup \pi'(v_1, v_2) \cup \ldots \cup \pi'(v_{n-1}, t)$. Let $x_j^i,\ j = 1, \ldots m$, be the inner bending points of the subpath $\pi(a_i = x_0^i, b_{i+1} = x_{m+1}^i)$. For each $j = 0, \ldots, m$, we define the point p_j^i to be the closest Steiner point to the intersection between $[x_j^i, x_{j+1}^i]$ and $\ell_v(j, \frac{\pi}{3})$, where v is the common endpoint of the edges containing x_j^i and x_{j+1}^i. Now, we create the path $\pi''(s,t) = \pi''(s, v_1) \cup \pi''(v_1, v_2) \cup \ldots \cup \pi''(v_{n-1}, t)$, where

$$\pi''(v_i, v_{i+1}) = (v_i, p_0^i, x_1^i, p_1^i, x_2^i, \ldots, x_m^i, p_m^i, v_{i+1}).$$

Let $A = \frac{8\sqrt{2}\sqrt{\sqrt{6\varepsilon}+8} - 2\sqrt{6\varepsilon} - 32}{-4\sqrt{2}\sqrt{\sqrt{6\varepsilon}+8} + \sqrt{6\varepsilon}+8}$. It follows from (4) that $\|\pi''(v_i, v_{i+1})\| \leq (1+A)\|\pi'(v_i, v_{i+1})\|$. Thus,

$$\begin{aligned}\|\pi''(s,t)\| &= \sum_{i=0}^{n-1}\|\pi''(v_i, v_{i+1})\| \leq (1+A)\sum_{i=0}^{n-1}\|\pi'(v_i, v_{i+1})\| \\ &\leq (1+A)\sum_{i=0}^{n-1}\left(\|\pi(a_i, b_{i+1})\| + \frac{\varepsilon\kappa_i}{7}\right), \qquad (5)\end{aligned}$$

where $\kappa_i = \omega_{min}(v_i)d(v_i) + \omega_{min}(v_{i+1})d(v_{i+1})$, so it remains to determine an upper bound for the sum $\sum_{i=0}^{n-1}\kappa_i$. So, using the definition of $d(\cdot)$ it follows that

$$\begin{aligned}\kappa_i &\leq \|v_i a_i\| + 2\|\pi(a_i, b_{i+1})\| + \|b_{i+1} v_{i+1}\| \\ &\leq 2\|\pi(a_i, b_{i+1})\| + \frac{\varepsilon\kappa_i}{7} \Longrightarrow \kappa_i \leq \frac{14}{7-\varepsilon}\|\pi(a_i, b_{i+1})\|.\end{aligned}$$

This, when substituted in equation (5) implies that

$$\begin{aligned}&(1+A)\sum_{i=0}^{n-1}\left(\|\pi(a_i, b_{i+1})\| + \frac{\varepsilon\kappa_i}{7}\right) \\ &\leq (1+A)\sum_{i=0}^{n-1}\left(\|\pi(a_i, b_{i+1})\| + \frac{\varepsilon}{7}\cdot\frac{14}{7-\varepsilon}\|\pi(a_i, b_{i+1})\|\right) \\ &= (1+A)\frac{7+\varepsilon}{7-\varepsilon}\sum_{i=0}^{n-1}\|\pi(a_i, b_{i+1})\| \leq (1+A)\frac{7+\varepsilon}{7-\varepsilon}\|\pi(s,t)\|. \qquad (6)\end{aligned}$$

Finally, two consecutive Steiner points p_j^i and p_{j+1}^i lie on neighbor sides, and v_i belongs to the same edge as p_0^i and v_{i+1} belongs to the same edge as p_m^i. Therefore, the sequence of points $(v_i = p_0^i, p_1^i, \ldots, p_m^i = v_{i+1})$ defines a path $\tilde{\pi}(v_i, v_{i+1})$, such that $\|\tilde{\pi}(v_i, v_{i+1})\| \leq \|\pi''(v_i, v_{i+1})\|$. If we combine all the paths $\tilde{\pi}(s, v_1), \ldots, \tilde{\pi}(v_{n-1}, t)$, we get that $\|\tilde{\pi}(s,t)\| \leq \|\pi''(s,t)\| \leq (1+A)\frac{7+\varepsilon}{7-\varepsilon}\|\pi(s,t)\|$. And the result is proved. □

2.2 Fixing the approximation factor of segment joining two points

The other parameter that we can fix is the approximation factor in each of the triangular cells. By doing this, we are optimizing the number of Steiner points placed in the faces. Using the approximation factor given by Aleksandrov et al. [5], we prove that the distance between consecutive Steiner points can be decreased, see Lemma 5. Due to space limitations, we defer the proof to the full version of the paper.

Lemma 5 *Let e_1, e_2 be two edges adjacent to v in T_i, and let x_1, x_2 be two points in e_1 and e_2, respectively. Let $|p_{i-1}p_i| = \frac{4(\varepsilon+2\sqrt{\varepsilon}\sqrt{\varepsilon+4})}{\sqrt{3}(\varepsilon+4)}\sin\frac{\pi}{3}|vp_{i-1}|,\ i \in \{1, \ldots, k\}$, be the distance between two consecutive Steiner points in side e_2. Let p be the closest Steiner point to x_2, then*

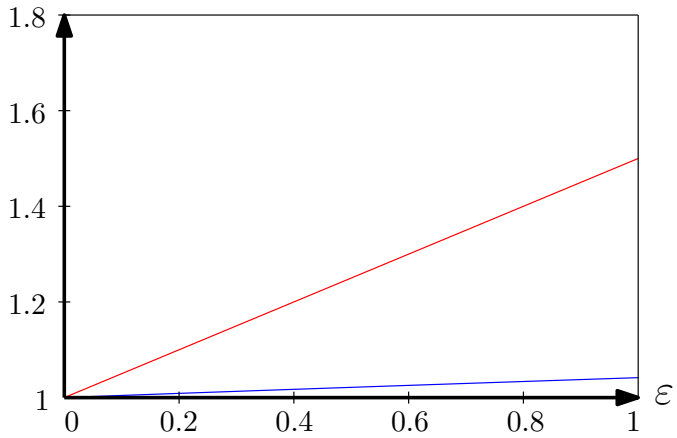


Figure 4: Comparison of approximation factor on a cell. The red function represents $y = 1 + \frac{\varepsilon}{2}$ from [5], and the blue function represents $y = \left(\frac{4\sqrt{2}\sqrt{\sqrt{6\varepsilon+8}}-\sqrt{6\varepsilon}-24}{-4\sqrt{2}\sqrt{\sqrt{6\varepsilon+8}}+\sqrt{6\varepsilon}+8}\right)$ from Proposition 3.

$$|x_1 p| + |p x_2| \leq \left(1 + \frac{\varepsilon}{2}\right) |x_1 x_2|.$$

The next lemma gives us an estimation on the number of Steiner points inserted on a particular side of a triangle and on their total number. The result is obtained by using Lemma 5, and the proof can be found in the full version of the paper.

Lemma 6 *1. The number of Steiner points inserted on a side of a triangle T_i is upper bounded by* $\frac{\log_2 \frac{2|\ell|}{r(v)}}{\log_2 e} \frac{3(\varepsilon+4)^3}{(2\varepsilon+4\sqrt{\varepsilon}\sqrt{\varepsilon+4})(20\varepsilon^2+76\varepsilon-(2\varepsilon+24)\sqrt{\varepsilon}\sqrt{\varepsilon+4}+48)} \log_2 \frac{2}{\varepsilon}$.

2. The total number of Steiner points on $\mathcal{T}$ is less than $C(\mathcal{T}) \frac{9n(\varepsilon+4)^3}{(2\varepsilon+4\sqrt{\varepsilon}\sqrt{\varepsilon+4})(20\varepsilon^2+76\varepsilon-(2\varepsilon+24)\sqrt{\varepsilon}\sqrt{\varepsilon+4}+48)} \log_2 \frac{2}{\varepsilon}$, *where* $C(\mathcal{T}) = \frac{\log_2 \frac{2|\ell|}{\min_{v \in \mathcal{T}} r(v)}}{\log_2 e}$

3 Discussion and future work

We provide some figures where we compare our results with the ones given by Aleksandrov et al. [5]. First, Figure 4 shows the error we commit when the segment between two points on the boundary of two adjacent edges of the tessellation is approximated by the subpath through a Steiner point. The red function represents the error obtained in [5], while the blue function represents the error obtained in Proposition 3, for values of ε in $[0, 1]$. Looking at Table 2, we notice that the error committed by our approach in each cell is about 70% less than using results in [5].

Secondly, in Figure 5 we depict the error obtained when the actual shortest path is approximated by the approach in [5] (see red function) and our result in Theorem 4 (see blue function). The error is shown for values of ε in the interval $[0, 1]$. These two functions show that our result is at least $\frac{5(1+\varepsilon)(7-\varepsilon)}{7(5+\varepsilon)}$ times better than the one provided in [5], i.e., about 150%. See also Table 3 for certain values of ε.

Recall that, in Figures 4 and 5, the approximation factors are obtained by using the same number of Steiner points in our result and in [5].

Finally, let p_i, p_{i+1} be two consecutive Steiner points on a segment from a vertex v on a triangular cell inclined by α radians. Then, Figure 6 represents the distance between p_i and p_{i+1}, divided by the distance $|vp_i|$. The function in red shows the results from [5] when placing the points on the bisector from v, while the function in blue shows our results when placing the points at the

ε	$1 + \frac{\varepsilon}{2}$	$\frac{4\sqrt{2}\sqrt{\sqrt{6\varepsilon+8}}-\sqrt{6\varepsilon}-24}{-4\sqrt{2}\sqrt{\sqrt{6\varepsilon+8}}+\sqrt{6\varepsilon}+8}$
0	1	1
0.1	1.05	1.0044
0.2	1.1	1.0088
0.4	1.2	1.017
0.6	1.3	1.025
0.8	1.4	1.033
0.9	1.45	1.037
1	1.5	1.041

Table 2: Comparison of approximation factor on a cell.

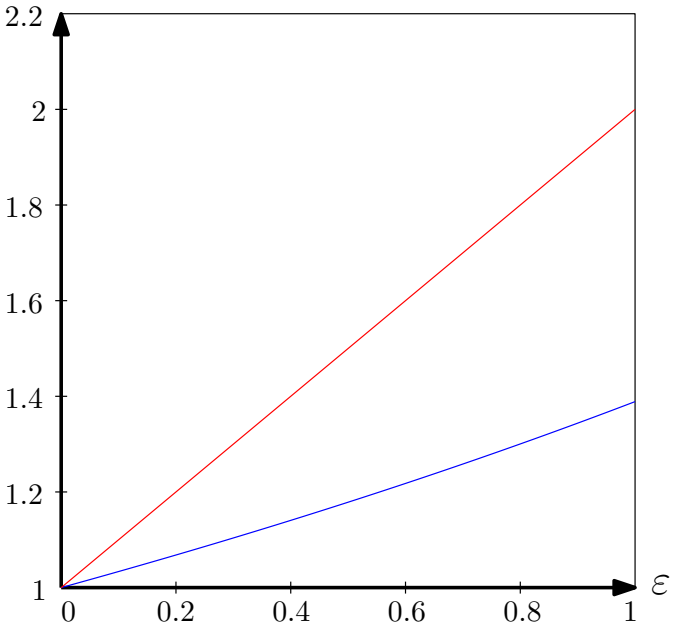


Figure 5: Comparison of approximation factor of paths. The red function represents $y = 1 + \varepsilon$ from [5], and the blue function represents $y = \left(1 + \frac{14(4\sqrt{2}\sqrt{\sqrt{6\varepsilon+8}}-\sqrt{6\varepsilon}-8)-16(7+\varepsilon)}{(-4\sqrt{2}\sqrt{\sqrt{6\varepsilon+8}}+\sqrt{6\varepsilon}+8)(7-\varepsilon)}\right)$ from Theorem 4.

ε	$1+\varepsilon$	$1+\frac{14(4\sqrt{2}\sqrt{\sqrt{6\varepsilon}+8}-\sqrt{6\varepsilon}-8)-16(7+\varepsilon)}{(-4\sqrt{2}\sqrt{\sqrt{6\varepsilon}+8}+\sqrt{6\varepsilon}+8)(7-\varepsilon)}$
0	1	1
0.1	1.1	1.033
0.2	1.2	1.068
0.4	1.4	1.14
0.6	1.6	1.217
0.8	1.8	1.3
0.9	1.9	1.343
1	2	1.3889

Table 3: Comparison of approximation factor of paths.

ε	$\frac{1}{2}\sqrt{\frac{\varepsilon}{2}}$	$\frac{2(\varepsilon+2\sqrt{\varepsilon}\sqrt{\varepsilon+4})}{\varepsilon+4}$
0	0	0
0.1	0.111	0.673
0.2	0.158	0.968
0.4	0.223	1.387
0.6	0.273	1.705
0.8	0.316	1.966
0.9	0.335	2.081
1	0.353	2.188

Table 4: Comparison of distance between consecutive Steiner points on the same cell.

sides of the cells. This value, using our result, is about $\frac{7-\sqrt{\varepsilon}}{40}$ times larger than the value given by Lemma 5, which is an improvement of the bound of about 500%. See also Table 4 for some values of $\varepsilon \in [0,1]$. For this result, we are using the same approximation factor for the segment between two bending points of the shortest path on the boundary of the same cell as in [5]. Moreover, another consequence of Lemma 5 is that the number of Steiner points that we add on each cell is less than in [5], see also Lemma 6, part 1. Hence, we decrease the total number of points that are added to the triangulation, see Lemma 6, part 2. Compared to [5], our method reduces the number of Steiner points in at least 4.5 times. Therefore, the space and time complexity of algorithms that compute weighted shortest paths (e.g., Dijkstra's algorithm) using our approach is less than the complexity of these algorithms using previous results.

As future work, it would be interesting to work with other types of regular grids, e.g., square or hexagonal, or take into account other realistic scenarios like triangulated irregular networks. Another possible extension would be to work with 3D environments.

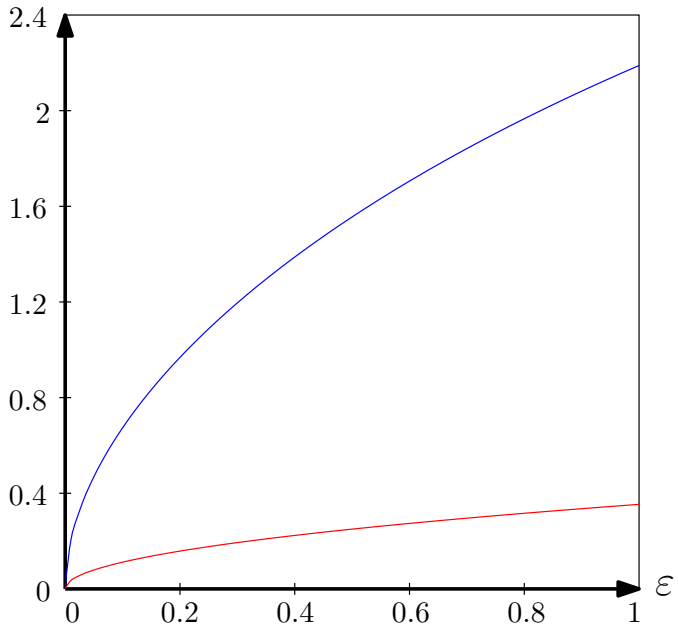


Figure 6: Comparison from Lemma 5. The red function represents $y = \frac{1}{2}\sqrt{\frac{\varepsilon}{2}}$ from [5], and the blue function represents $y = \frac{2(\varepsilon+2\sqrt{\varepsilon}\sqrt{\varepsilon+4})}{\varepsilon+4}$ from Lemma 5.

References

[1] M. Abdalla and B. Nagy. Dilation and erosion on the triangular tessellation: an independent approach. *IEEE Access*, 6:23108–23119, 2018.

[2] Advanced Spaceborne Thermal Emission and Reflection Radiometer Global Digital Elevation Model. Hadoop. `https://asterweb.jpl.nasa.gov`, 2019-08-05. V3.

[3] L. Aleksandrov, M. Lanthier, A. Maheshwari, and J.-R. Sack. An ε-approximation algorithm for weighted shortest paths on polyhedral surfaces. In *Scandinavian Workshop on Algorithm Theory*, pages 11–22. Springer, 1998.

[4] L. Aleksandrov, A. Maheshwari, and J.-R. Sack. Approximation algorithms for geometric shortest path problems. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 286–295, 2000.

[5] L. Aleksandrov, A. Maheshwari, and J.-R. Sack. Determining approximate shortest paths on weighted polyhedral surfaces. *Journal of the ACM (JACM)*, 52(1):25–53, 2005.

[6] J.-L. De Carufel, C. Grimm, A. Maheshwari, M. Owen, and M. Smid. A note on the unsolvability of the weighted region shortest path problem. *Computational Geometry*, 47(7):724–727, 2014.

[7] L. de Floriani, P. Magillo, and E. Puppo. Applications of computational geometry to geographic information systems. *Handbook of computational geometry*, 7:333–388, 2000.

[8] Firaxis Games. Civilization V. `http://civilization.com/civilization-5/`. Accessed: 2022-04-14.

[9] P. Kardos and K. Palágyi. Topology preservation on the triangular grid. *Annals of Mathematics and Artificial Intelligence*, 75(1):53–68, 2015.

[10] M. v. Kreveld. Algorithms for triangulated terrains. In *International Conference on Current Trends in Theory and Practice of Computer Science*, pages 19–36. Springer, 1997.

[11] V. Kvachev. Colossal Citadels. `http://colossalcitadels.com`. Accessed: 2022-04-05.

[12] J. S. B. Mitchell. Shortest paths and networks. In J. E. Goodman, J. O'Rourke, and C. D. Toth, editors, *Handbook of Discrete and Computational Geometry, Second Edition*, pages 811–848. Chapman and Hall/CRC, 2017.

[13] J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem: finding shortest paths through a weighted planar subdivision. *Journal of the ACM (JACM)*, 38(1):18–73, 1991.

[14] B. Nagy and T. Lukić. Dense projection tomography on the triangular tiling. *Fundamenta Informaticae*, 145(2):125–141, 2016.

[15] B. Nagy and E. V. Moisi. Memetic algorithms for reconstruction of binary images on triangular grids with 3 and 6 projections. *Applied Soft Computing*, 52:549–565, 2017.

[16] M. Sharir and S. Sifrony. Coordinated motion planning for two independent robots. *Annals of Mathematics and Artificial Intelligence*, 3(1):107–130, 1991.

[17] N. Sturtevant. A sparse grid representation for dynamic three-dimensional worlds. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 6, 2011.

[18] Z. Sun and J. Reif. Bushwhack: An approximation algorithm for minimal paths through pseudo-euclidean spaces. In *International Symposium on Algorithms and Computation*, pages 160–171. Springer, 2001.

Maximum Subbarcode Matching and Subbarcode Distance

Oliver Chubet *†

Abstract

We investigate the maximum subbarcode matching problem which arises from the study of persistent homology and introduce the subbarcode distance on barcodes. A barcode is a set of intervals which correspond to topological features in data and is the output of a persistent homology computation. A barcode $\mathbb{A}$ has a subbarcode matching to $\mathbb{B}$ if each interval in $\mathbb{A}$ matches to an interval in $\mathbb{B}$ which contains it. We present an algorithm which takes two barcodes, $\mathbb{A}$ and $\mathbb{B}$, and returns a maximum subbarcode matching. The subbarcode matching algorithm we present is a generalization of the up-right matching algorithm given by Karp et al [11]. Our algorithm also works on multiset input. It has $O(n \log n)$ runtime, where n is the number of distinct intervals in the barcodes. We show that the subbarcode relation is transitive and induces a partial order on barcodes. We introduce subbarcode distance and show that the subbarcode distance is a lower bound for bottleneck distance. We also give an algorithm to compute subbarcode distance, which has expected $O(n \log^2 n)$ runtime and uses $O(n)$ space.

1 Introduction

In persistent homology the barcode is a multiset of intervals encoding topological information. There is new interest in the implications arising when one has only partial knowledge or an approximation of the barcode. For example, in recent work, Chubet et al [3] establish that one can use subbarcodes in topological data analysis to make strong claims about an unknown function given only upper and lower bounds. Having efficient subbarcode matching algorithms allows one to implement strategies suggested by these new theoretical developements. The subbarcode matching algorithm and subbarcode distance are practical tools for comparing the topological invariants of two datasets.

2 Background

A multiset $\mathbb{A} = (A, \omega_A)$ is a pair consisting of a set A and a multiplicity function $\omega_A : A \to \mathbb{N}$. The weight of $\mathbb{A}$ is the sum of the multiplicities of the elements of A, denoted, $|\mathbb{A}| = \sum_{a \in A} \omega_A(a)$.

A matching $\mathbb{M}$ between multisets $\mathbb{A} = (A, \omega_A)$ and $\mathbb{B} = (B, \omega_B)$ is a multiset $\mathbb{M} = (M, \omega)$ where $M \subset A \times B$ with multiplicity function $\omega : M \to \mathbb{N}$ such that

$$\sum_{b \in B} \omega(a, b) \leq \omega_A(a) \text{ for all } a \in A \text{ and}$$
$$\sum_{a \in A} \omega(a, b) \leq \omega_B(b) \text{ for all } b \in B.$$

A matching $\mathbb{M}$ is a maximum matching if it has maximum weight over all valid matchings. If $|\mathbb{M}| = |\mathbb{A}| = |\mathbb{B}|$ then we call $\mathbb{M}$ a perfect matching.

An interval is a pair (a_x, a_y) for $a_x, a_y \in \mathbb{R}$. See Figure 1. Given intervals $s = (s_L, s_R)$ and $b = (b_L, b_R)$, if

$$b_L \leq s_L, \quad \text{and} \quad s_R \leq b_R.$$

then b contains s, denoted $s \preceq b$. Containment of intervals defines a partial order on intervals.

A barcode $\mathbb{B} = (B, \omega_B)$ is a multiset where B is a set of intervals. A subbarcode matching from $\mathbb{S}$ to $\mathbb{B}$ is

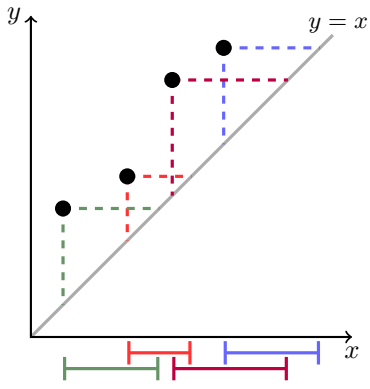


Figure 1: We may represent intervals as points in $\mathbb{R}^2$ by taking their endpoints as coordinates .

a multiset matching $\mathbb{M} = (M, \omega_M)$, of $\mathbb{S}$ and $\mathbb{B}$, where $(s, b) \in M$ implies $s \preceq b$. See Figure 2.

The maximum subbarcode matching problem is to find a subbarcode matching of maximum weight. If there exists a subbarcode matching $\mathbb{M}$ from $\mathbb{A}$ to $\mathbb{B}$ such that $|\mathbb{M}| = |\mathbb{A}|$, then we call $\mathbb{A}$ a subbarcode of $\mathbb{B}$, denoted $\mathbb{A} \sqsubseteq \mathbb{B}$.

*North Carolina State University, oliver.chubet@gmail.com

†This work was partially funded by the NSF under grant CCF-2017980.

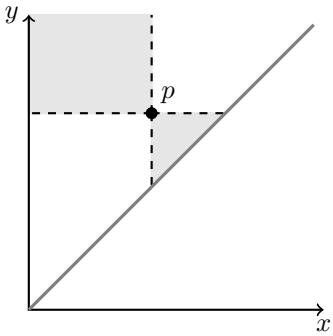


Figure 2: Any point in the upper shaded region contains p as an interval. Any point in the lower shaded region is contained in p as an interval.

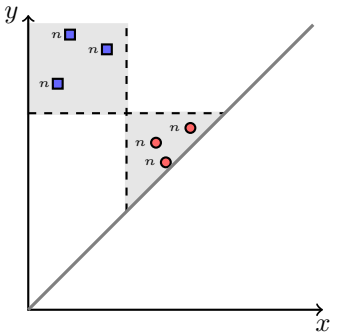


Figure 3: Two barcodes for which there exists a quadratic size subbarcode matchings.

3 Related Work

Traditionally, persistence diagrams have been compared via bottleneck distance. Bottleneck matching is an instance of the assignment problem. The traditional Hopcroft-Karp algorithm for maximum matching in bipartite graphs runs in $O(n^{\frac{5}{2}})$ [10]. However, Efrat et al [5] reduced this runtime to $O(n^{\frac{3}{2}} \log n)$ by using a geometric data structure. Kerber et al [12] also improved this algorithm for persistence diagrams, using k-d trees.

We use a sweepline approach in our subbarcode matching algorithm [1]. Our algorithm builds upon the up-right matching algorithm given by Karp et al [11]. In the case of matching finite subsets of the unit square, this algorithm has been proven to find the optimal matching. Two additional related problems include the maximum matching problem for intersecting intervals [2] and maximum matching in convex bipartite graphs [7, 13, 8]. The strategy used in these algorithms is to avoid backtracking to keep the total operations per element small.

4 Subbarcode Algorithm

We present an algorithm to compute a linear-sized maximum multiset subbarcode matching. See Figure 4.

SUBMATCH($\mathbb{A}, \mathbb{B}$):
Input Two barcodes: $\mathbb{A} = (A, \omega_A)$, $\mathbb{B} = (B, \omega_B)$
Output A subbarcode matching from $\mathbb{A}$ to $\mathbb{B}$

Sort $A \cup B$ by the x-coordinates.
Initialize $\mathtt{T}$ to be an empty balanced binary search tree to store points from B ordered by y-coordinate.
Initialize residual weights $\mathtt{r}_b = \omega_B(b)$ for each $b \in B$ and $\mathtt{r}_a = \omega_A(a)$ for each $a \in A$.
Initialize $(\mathtt{M}, \mathtt{W})$ to store the matching and multiplicities.

For each $p \in A \cup B$, where $p = (p_x, p_y)$:

If $p \in B$, insert b into $\mathtt{T}$.
Else

While $\mathtt{r}_p > 0$:
Search for $b \in \mathtt{T}$ with minimum b_y such that $b_y \geq p_y$.
If there is none, then `break`.
Let $\mathtt{r} = \min\{\mathtt{r}_p, \mathtt{r}_b\}$.
Add (p, b) to $\mathtt{M}$ and set $\mathtt{W}[(p, b)] = \mathtt{r}$,
then update the residual weights of p and b: $\mathtt{r}_p = \mathtt{r}_p - \mathtt{r}$ and $\mathtt{r}_b = \mathtt{r}_b - \mathtt{r}$.
If $\mathtt{r}_b = 0$, then remove b from $\mathtt{T}$.

Return $(\mathtt{M}, \mathtt{W})$.

When both input weight functions uniformly map all elements to 1 this algorithm reduces to the up-right matching algorithm presented by Karp et al [11]. In this case, it is clear that the output size is linear. However, in the case where we are matching multisets, it is possible for a subbarcode matching to have quadratic size.

For example, suppose there are n intervals in barcodes $\mathbb{A}$ and $\mathbb{B}$ respectively such that all intervals have multiplicity n and all intervals in $\mathbb{A}$ are subbars of all intervals in $\mathbb{B}$, as depicted in Figure 3. Then a valid matching could match each interval in A once with each of the intervals in $\mathbb{B}$. This illustrates the significance of a linear-size guarantee.

In the following lemma we prove that the output remains linear.

Lemma 1 *Let $\mathbb{A} = (A, \omega_A)$ and $\mathbb{B} = (B, \omega_B)$ be barcodes. The subbarcode matching $\mathbb{M} = \text{SUBMATCH}(\mathbb{A}, \mathbb{B})$ has size $O(n)$, where $n = \#A + \#B$.*

In particular, $\#M \leq n$.

Proof. Let $(M, \omega) = \text{SUBMATCH}(\mathbb{A}, \mathbb{B})$. Let $G = (V, M)$ be the weighted graph induced by taking M as the edge

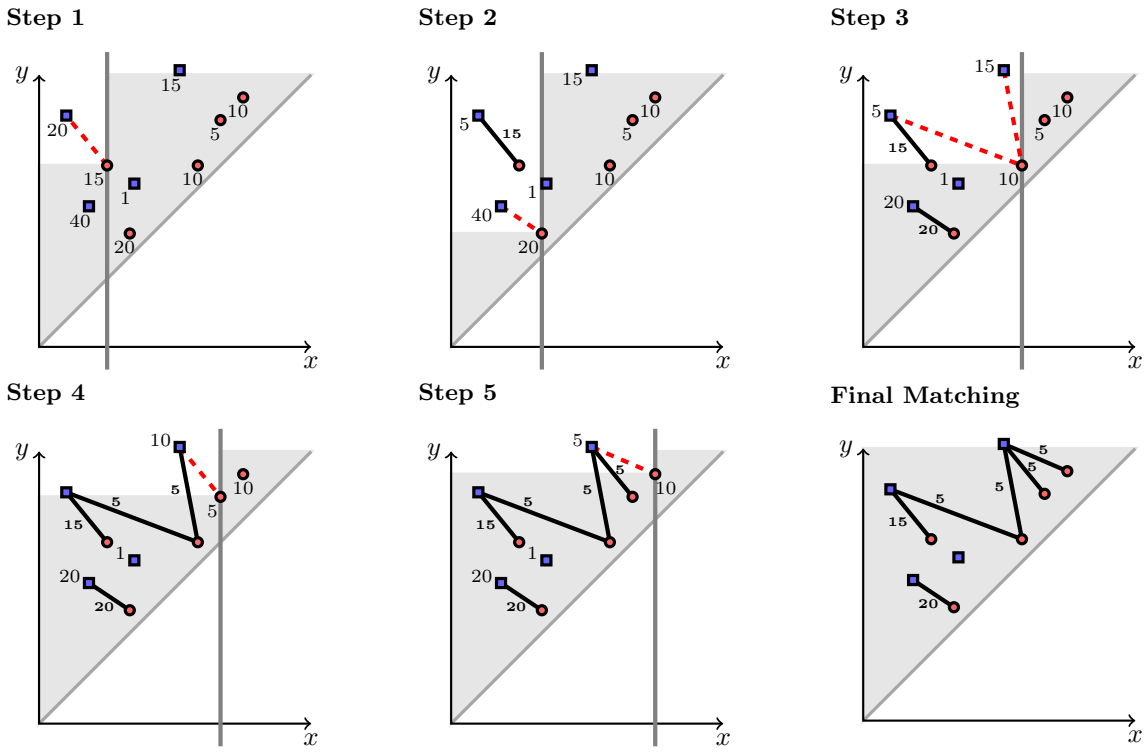


Figure 4: We find a maximum subbarcode matching from $\mathbb{A}$ to $\mathbb{B}$ (circles and squares respectively) labeled by their multiplicities. We iterate through $\mathbb{A}$ in order of x-coordinate and match to the point in $\mathbb{B}$ with lowest y-coordinate. Each edge represents the match labeled with the multiplicity, and the residual multiplicities are updated for $\mathbb{A}$ and $\mathbb{B}$ accordingly.

set with weights given by ω. All edges $(a, b) \in M \subseteq A \times B$. Let $m = \#M$, and $n = \#V$.

We know $m = \frac{1}{2}\sum_{v \in V} \deg(v)$ to be a property of all graphs. Because G is bipartite, it is sufficient to consider only the degrees of elements in A. We partition A into high and low degree nodes,

$$H = \{a \in A \mid \deg(a) \geq 2\} \text{ and } L = A \setminus H.$$

Then, $m = \sum_{a \in H} \deg(a) + \sum_{a \in L} \deg(a)$.

For $a \in H$, consider the sequence $(b_0, \ldots, b_r)$ of all points in B adjacent to a in G, where b_0 is the first point to match to a and b_r is the last point to match to a. Then for $b_i \in \{b_0, \ldots, b_{r-1}\}$ we know that a is the last point to match to b_i, because the algorithm does not proceed to matching b_{i+1} until the remaining multiplicity of b_i is matched.

Each point in B can only have one point being the last to match to it, so

$$\sum_{a \in H} \deg(a) \leq \#H + \#B \text{ and } \sum_{a \in L} \deg(a) \leq \#L.$$

Therefore, $m \leq \#H + \#L + \#B \leq n$. □

Theorem 2 SUBMATCH *uses $O(n)$ space.*

Proof. The only structures maintained during SUBMATCH are the input, the output, and the search tree. The input and search tree are linear size. By Lemma 1, the output of SUBMATCH is linear size as well. Thus total space used is $O(n)$. □

Theorem 3 *The matching from* SUBMATCH *is maximum.*

Proof. First consider the case where $\mathbb{A} = (A, \omega_A)$ and $\mathbb{B} = (B, \omega_B)$ with $\omega_A \equiv \omega_B \equiv 1$. Then SUBMATCH reduces to the up-right matching algorithm given by Karp et al [11], which has previously been shown to be optimal.

If we consider two barcodes $\mathbb{A} = (A, \omega_A)$ and $\mathbb{B} = (B, \omega_B)$, we can construct $\mathbb{A}' = (A', \omega'_A)$, $\mathbb{B}' = (B', \omega'_B)$ such that $\forall a \in A$ we have $\omega_A(a)$ distinct copies $a^{(i)}$ of a in A', for $i \in \{1, \ldots, \omega_A(a)\}$. Similarly, $b^{(j)} \in B'$ for $j \in \{1, \ldots, \omega'_B(b)\}$. Then we have reduced the input to the first case described above. □

Theorem 4 SUBMATCH *computes a linear-sized maximum subbarcode matching in $O(n \log n)$ time.*

Proof. Let $\mathbb{A} = (A, \omega_A)$ and $\mathbb{B} = (B, \omega_B)$ be barcodes. Let $\mathtt{T}$ be search tree constructed in SUBMATCH and let

$\mathbb{M} = (M, \omega)$ be the output matching. Let $G = (V, M)$ be the graph induced by taking M as the edge set. Given $a \in A$ each time we search $\mathtt{T}$ either we find a match or we don't. We find a match $\deg(a)$ times, and we don't find a match at most once. It follows, the number of searches is at most $\sum_{a \in A}(\deg(a)+1) = \#M + \#A$. In Lemma 1 we proved that $\#M$ is linear size. Thus, the number of searches is $O(n)$. Furthermore, there are $O(n)$ insertions and deletions and $\mathtt{T}$ is balanced, so each search operation takes $O(\log n)$. Therefore, the runtime is $O(n \log n)$. □

5 Subbarcode Transitivity

For intervals a and b, recall that $a \preceq b$ if b contains a.

Transitivity of set matching follows easily by composing the matchings. However, functions over multisets do not have a well-defined composition. In 1957, Ford and Fulkerson showed that Hall's Theorem for systems of representitives could equivalently be expressed in terms of flow networks [6, 9]. We use this approach to show the existence of a subbarcode matching is transitive.

Lemma 5 (Transitivity) *If* $\mathbb{A} \sqsubseteq \mathbb{B}$ *and* $\mathbb{B} \sqsubseteq \mathbb{C}$ *then* $\mathbb{A} \sqsubseteq \mathbb{C}$.

Proof. Given barcodes $\mathbb{A} = (A, \omega_A)$, $\mathbb{B} = (B, \omega_B)$, and $\mathbb{C} = (C, \omega_C)$ with subbarcode matchings (M, ω_M) for $\mathbb{A} \sqsubseteq \mathbb{B}$ and (T, ω_T) for $\mathbb{B} \sqsubseteq \mathbb{C}$, there is a corresponding network, $\mathsf{Net}(G)$, where $G = (A \sqcup B \sqcup C,\ M \sqcup T)$ is a digraph [6, 4]. See Figure 5.

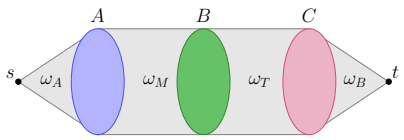


Figure 5: An (s,t)-flow f in $\mathsf{Net}(G)$ corresponds to a matching of $\mathbb{A}$ and $\mathbb{C}$.

If f is a max-flow in $\mathsf{Net}(G)$, then the corresponding matching is maximum and the value of the flow, $|f|$, is equal to the weight of the matching [4]. In Appendix A

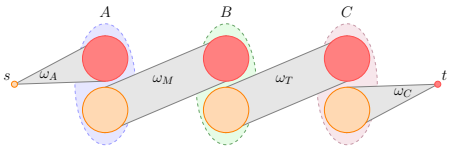


Figure 6: The the capacity of an arbitrary cut, $(L, \overline{L})$ of $\mathsf{Net}(G)$.

we show that $c(L, \overline{L}) \geq |\mathbb{A}|$ for any cut $(L, \overline{L})$ of $\mathsf{Net}(G)$. See Figure 6. Therefore $\mathbb{A} \sqsubseteq \mathbb{C}$. □

Corollary 6 *The relation* $\sqsubseteq$ *defines a partial order on barcodes.*

We call the poset of barcodes $(\mathbf{Bar}, \sqsubseteq)$.

6 Shifted Subbarcodes

There are cases where the maximum matching is not sufficient. Rather, one prefers to know "how far off" two barcodes are from having a subbarcode matching. For example, if we have only an approximation to the input, we can consider the maximum matching after shifting one set by distance δ. There are cases when only a small shift is needed to obtain a subbarcode matching.

If $\mathbb{A} \not\sqsubseteq \mathbb{B}$, we can determine the minimum shift of $\mathbb{A}$ such that the translation results in a subbarcode of $\mathbb{B}$. We use this minimum shift to define a metric on barcodes.

A δ-shift of $\mathbb{A} = (A, \omega)$ is a barcode $\mathbb{A}^\delta$ where

$$\mathbb{A}^\delta := (\delta(A), \omega \circ \delta^{-1}) \text{ and}$$
$$\delta(a) := (a_x + \delta, a_y - \delta).$$

Let $\mathbb{A}$ and $\mathbb{B}$ be barcodes such that $|\mathbb{A}| = |\mathbb{B}|$. The subbarcode distance is

$$\mathbf{d}_S(\mathbb{A}, \mathbb{B}) := \max\{\min_{\delta \geq 0} \mathbb{A}^\delta \sqsubseteq \mathbb{B}, \min_{\delta \geq 0} \mathbb{B}^\delta \sqsubseteq \mathbb{A}\}.$$

The subbarcode distance is similar to Hausdorff distance in that it is bidirectional and asymmetric in nature.

Lemma 7 (Approximation is additive.) *If* $\mathbb{A}^\delta \sqsubseteq \mathbb{B}$ *and* $\mathbb{B}^\varepsilon \sqsubseteq \mathbb{C}$ *then* $\mathbb{A}^{\delta+\varepsilon} \sqsubseteq \mathbb{C}$.

Proof. Let $\mathbb{A}$, $\mathbb{B}$, and $\mathbb{C}$ be barcodes such that $\mathbb{A}^\delta \sqsubseteq \mathbb{B}$ and $\mathbb{B}^\varepsilon \sqsubseteq \mathbb{C}$. Consider intervals, a and b.

If $a \preceq b$, then $a_x \geq b_x$ and $a_y \leq b_y$.

Then, $a_x + \delta \geq b_x + \delta$ and $a_y - \delta \leq b_y - \delta$.

Thus, $\delta(a) \preceq \delta(b)$.

By extension, if $\mathbb{A} \sqsubseteq \mathbb{B}$, then $\mathbb{A}^\delta \sqsubseteq \mathbb{B}^\delta$. By assumption, $\mathbb{A}^\delta \sqsubseteq \mathbb{B}$, so it follows, $\mathbb{A}^{\delta+\varepsilon} \sqsubseteq \mathbb{B}^\varepsilon$. Thus by transitivity of subbarcodes (Lemma 5), $\mathbb{A}^{\delta+\varepsilon} \sqsubseteq \mathbb{C}$. □

Lemma 8 (Triangle Inequality)

$$\mathbf{d}_S(\mathbb{A}, \mathbb{C}) \leq \mathbf{d}_S(\mathbb{A}, \mathbb{B}) + \mathbf{d}_S(\mathbb{B}, \mathbb{C})$$

Proof. Let $\mathbb{A}, \mathbb{B}$, and $\mathbb{C}$ be barcodes. Suppose $\mathbf{d}_S(\mathbb{A}, \mathbb{B}) = \delta$ and $\mathbf{d}_S(\mathbb{B}, \mathbb{C}) = \varepsilon$. Then by definition,

$$\mathbb{A}^\delta \sqsubseteq \mathbb{B},\ \mathbb{B}^\delta \sqsubseteq \mathbb{A},\ \mathbb{B}^\varepsilon \sqsubseteq \mathbb{C}, \text{ and } \mathbb{C}^\varepsilon \sqsubseteq \mathbb{B}.$$

By Lemma 7, it follows $\mathbb{A}^{\delta+\varepsilon} \sqsubseteq \mathbb{C}$ and $\mathbb{C}^{\delta+\varepsilon} \sqsubseteq \mathbb{A}$. Therefore $\mathbf{d}_S(\mathbb{A}, \mathbb{C}) \leq \delta + \varepsilon$. □

The remaining metric properties are easily verified, so we may conclude the following theorem.

Theorem 9 *The subbarcode distance is a metric on barcodes.*

7 Subbarcode Distance Computation

In this section we present algorithms which allow us to compute the subbarcode distance. The goal is to compute the minimum shift such that we have a subbarcode matching. To find this shift it is useful to determine cases in which we may easily recognize that we have shifted by an excesse amount.

Lemma 10 *For a subbarcode matching, (M, ω), of $\mathbb{A}^{\Delta} \sqsubseteq \mathbb{B}$, let*

$$\gamma = \min_{(\Delta(a),b)\in M} \min\{a_x + \Delta - b_x, b_y - a_y + \Delta\}.$$

Then $\mathbb{A}^{\Delta-\gamma} \sqsubseteq \mathbb{B}$.

Proof. For all $(\Delta(a), b) \in M$, $a_e + \Delta - b_x \geq \gamma$, and $b_y - a_y + \Delta \geq \gamma$. So, $a_x + (\Delta - \gamma) \geq b_x$, and $b_y \geq a_y - (\Delta - \gamma)$. Therefore $\mathbb{A}^{\Delta-\gamma} \sqsubseteq \mathbb{B}$. □

We can think of γ as an excess shift of $\mathbb{A}$. That is, we could have shifted $\mathbb{A}$ by a distance γ less than we did and the corresponding matching is still be a valid matching. So intuitively, if the shift is the subbarcode distance, then $\gamma = 0$ because there can be no excess shift.

In the next lemma we prove that the subbarcode distance, similar to Hausdorff distance and bottleneck distance, is determined by a pair from A and B. This motivates us to devise a search method to find this pair.

Lemma 11 *For some $(a, b) \in A \times B$,*

$$\mathbf{d}_S(\mathbb{A}, \mathbb{B}) = \min\{a_x - b_x, b_y - a_y\}.$$

Proof. Let $\Delta = \mathbf{d}_S(\mathbb{A}, \mathbb{B})$. Then there is a subbarcode matching (M, ω) for $\mathbb{A}^{\Delta} \sqsubseteq \mathbb{B}$. By Lemma 10, $\mathbb{A}^{\Delta-\gamma} \sqsubseteq \mathbb{B}$ for $\gamma = \min_{(\Delta(a),b)\in M} \min\{a_x - b_x, b_y - a_y\}$. It follows that $\gamma = 0$ because Δ is minimum. So $\Delta = \min\{a_x - b_x, b_y - a_y\}$ for some $(a, b) \in A \times B$. □

Lemma 11 enables us to compute $\mathbf{d}_S$ by finding the correct pair in $A \times B$. There are n^2 possibilities, however, we search these possibilities efficiently by taking a uniform sample of the endpoints for which the difference is within given upper and lower bounds.

For barcodes (A, ω_A) and (B, ω_B), define:

$$\text{UB} := \max\{(\max_{b\in B} b_x - \min_{a\in A} a_x), (\max_{a\in A} a_y - \min_{b\in B} b_y), 0\}$$
$$\text{LB} := \max\{(\max_{b\in B} b_x - \max_{a\in A} a_x), (\min_{a\in A} a_y - \min_{b\in B} b_y), 0\}.$$

Here, the upper bound UB is simply the distance between the farthest corners of the minimum bounding rectangles of A and B. The lower bound LB is the distance between the bottom right corners of the minimum bounding rectangles. These may be replaced with any suitable upper and lower bounds.

In MINSHIFT we use these bounds to perform a binary search through all pairs of coordinate differences in order to find the points that give us the exact subbarcode distance.

MINSHIFT($\mathbb{A}, \mathbb{B}$, LB, UB):
Input: Barcodes $\mathbb{A}$,$\mathbb{B}$, and upper and lower bounds $\text{LB} \leq \mathbf{d}_S(\mathbb{A}, \mathbb{B}) \leq \text{UB}$
Output: The subbarcode distance, Δ

Let X, Y be the sorted x- and y-coordinates of $A \cup B$.
Δ = SAMPLE(X, Y, LB, UB)
While Δ exists:

  (M, ω) = SUBMATCH($\mathbb{A}^{\Delta}, \mathbb{B}$)
  If (M, ω) is a perfect matching, set UB = Δ.
  Else LB = Δ
  Δ = SAMPLE(X, Y, LB, UB)

Return UB

A binary search is made possible by using SAMPLE to obtain a uniform random sample of all pairs with coordinate differences contained within the given bounds. In a linear scan of the sets of x- and y-coordinates we determine the prevalence of each coordinate in the set suitable pairs. We then sample a pair from this set and return the minimum coordinate difference. See Figure 7 and Figure 8.

SAMPLE(X, Y, LB, UB):
Input: Sorted lists X and Y, and bounds LB and UB
Output: A uniform random sample

In a linear scan of X, find indices, l_i and u_i, such that

$$\text{X}[l_i - 1] \leq \text{X}[i] + \text{LB} < \text{X}[l_i],$$
$$\text{and } \text{X}[u_i] < \text{X}[i] + \text{UB} \leq \text{X}[u_i + 1].$$

Similarly, scanning Y, find indices, l'_i and u'_i, such that

$$\text{Y}[u'_i - 1] \leq \text{Y}[i] - \text{UB} < \text{Y}[u'_i],$$
$$\text{and } \text{Y}[l'_i] < \text{Y}[i] - \text{LB} \leq \text{Y}[l'_i + 1].$$

If $l_i = u_i$ and $l'_i = u'_i$ for all i, return nothing.
Otherwise, sample an index i with probability proportional to $(u_i - l_i) + (l'_i - u'_i)$.
Sample endpoint e uniformly from $\text{X}[l_i : u_i] \sqcup \text{Y}[u'_i : l'_i]$.
If e is from X then return $e - \text{X}[i]$. Otherwise return $\text{Y}[i] - e$.

Theorem 12 MINSHIFT *computes the subbarcode distance with an expected $O(n \log^2 n)$ time.*

Proof. Using SAMPLE to get a uniform sample of all pairwise distances of endpoints, MINSHIFT reduces to a

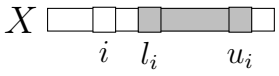


Figure 7: If $X[i]$ is an endpoint and $X[j]$ is from the range $X[l_i : u_i]$ then $\text{LB} < X[j] - X[i] < \text{UB}$.

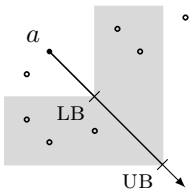


Figure 8: Depicted above are the points considered by SAMPLE for a single point $a \in A$. The points in the shaded region form a subset of B for which the minimum coordinate differences are within the bounds given.

randomized binary search over n^2 elements. Thus there is an expected $O(\log n)$ iterations, where each iteration is $O(n \log n)$. Therefore MINSHIFT has expected runtime $O(n \log^2 n)$. □

8 Persistence Diagrams

In topological data analysis it is common to compare persistence diagrams rather than barcodes. In this section we show that, with slight modification, the algorithms presented in Section 7 also apply in this setting.

The diagonal of $\mathbb{R}$ is the set $\mathcal{D} = \{(x, x) \mid x \in \mathbb{R}\}$. A persistence diagram for a barcode $\mathbb{B} = (B, \omega_B)$ is a multiset $\mathsf{PD}(\mathbb{B}) := (B \cup \mathcal{D}, \omega)$, where

$$\omega(x) = \begin{cases} \omega_B(x), & x \in B \\ \infty, & x \in \mathcal{D}. \end{cases}$$

We have added the diagonal of $\mathbb{R}$ with infinite multiplicity.

Let $\mathsf{PD}(\mathbb{A}) = (A \cup \mathcal{D}, \omega)$, be a persistence diagram. Note that shifting this diagram by δ gives us the multiset

$$\mathsf{PD}(\mathbb{A})^\delta = (\delta(A \cup \mathcal{D}),\ \omega \circ \delta^{-1}).$$

It is useful to refer to only the points above the diagonal, because points which have been shifted below $y = x$ can now match to the diagonal. We denote this as $[\mathbb{X}^\delta]$, where $\mathbb{X}$ is a barcode.

Lemma 13 *Let $\mathbb{A} = (A, \omega_A)$ and $\mathbb{B} = (B, \omega_B)$ be barcodes. Then*

$$\mathsf{PD}(\mathbb{A})^\delta \sqsubseteq \mathsf{PD}(\mathbb{B}) \quad \textit{if and only if} \quad [\mathbb{A}^\delta] \sqsubseteq \mathbb{B}.$$

Proof. Let (M, ω) be a subbarcode matching for $\mathsf{PD}(\mathbb{A})^\delta \sqsubseteq \mathsf{PD}(\mathbb{B})$. Consider $a \in [\delta(A)]$. Note that if $(a, b) \in M$, then $b \notin \mathcal{D}$,so we can restrict M to $M \cap ([\delta(A)] \times B)$ to obtain a matching for $[\mathbb{A}^\delta] \sqsubseteq \mathbb{B}$.

Now let (N, ω) be a matching for $[\mathbb{A}^\delta] \sqsubseteq \mathbb{B}$. For any $a \in \delta(A \cup \mathcal{D}) \setminus [\delta(A)]$, there is $d = (a_x, a_x) \in \mathcal{D}$ such that $a \preceq d \in \mathcal{D}$. Because d has infinite multiplicity in $\mathsf{PD}(\mathbb{B})$, we can add (a, d) to N and set $\omega(a, d) = \omega_A \circ \delta^{-1}(a)$. Thus N is a subbarcode matching. □

This result allows us to compute a subbarcode matching of persistence diagrams $\mathsf{PD}(\mathbb{A}^\delta)$ and $\mathsf{PD}(\mathbb{B})$ by computing $\text{SUBMATCH}([\mathbb{A}^\delta], \mathbb{B})$. Additionally, we can compute $\mathbf{d}_S(\mathsf{PD}(\mathbb{A}), \mathsf{PD}(\mathbb{B}))$ by modifying MINSHIFT slightly. Rather than returning the minimum Δ such that $\mathbb{A}^\Delta \sqsubseteq \mathbb{B}$, we return the minimum Δ such that $[\mathbb{A}^\Delta] \sqsubseteq \mathbb{B}$.

Note that because persistence diagrams fall under our definition of barcodes, the subbarcode distance is also a metric on persistence diagrams.

9 Subbarcode Distance and Bottleneck Distance

In this section we establish the relationship between the subbarcode distance and bottleneck distance.

Let $\mathbb{A}$ and $\mathbb{B}$ be barcodes such that $|\mathbb{A}| = |\mathbb{B}|$. Let $\mathcal{M}$ be the set of all possible perfect matchings between $\mathbb{A}$ and $\mathbb{B}$. The bottleneck distance is

$$\mathbf{d}_B(\mathbb{A}, \mathbb{B}) := \min_{(M,\omega)\in\mathcal{M}} \left\{ \max_{(a,b)\in M} ||a - b||_\infty \right\}$$

A bottleneck matching between barcodes $\mathbb{A}$ and $\mathbb{B}$ is a matching $\mathbb{M} = (M, \omega)$ where

$$\max_{(a,b)\in M} ||a - b||_\infty = \mathbf{d}_B(\mathbb{A}, \mathbb{B}).$$

Theorem 14 *For any two barcodes $\mathbb{A}$ and $\mathbb{B}$, where $|\mathbb{A}| = |\mathbb{B}|$,*

$$\mathbf{d}_S(\mathbb{A}, \mathbb{B}) \leq \mathbf{d}_B(\mathbb{A}, \mathbb{B}).$$

Proof. Let $\mathbb{M} = (M, \omega_M)$ be a bottleneck matching between $\mathbb{A}$ and $\mathbb{B}$. Let $\beta := \mathbf{d}_B(\mathbb{A}, \mathbb{B})$. Then for any edge $(a, b) \in M$, $||a - b||_\infty \leq \beta$. Moreover, $|b_x - a_x| \leq \beta$ and $|b_y - a_y| \leq \beta$. It follows that $b_x \leq a_x + \beta$ and $a_y - \beta \leq b_y$, implying $\beta(a) \preceq b$ for each $(a, b) \in M$. We can then construct a matching as follows: Let $\mathbb{T} = (T, \omega_T)$, where

$$T = \{(\beta(a), b) \mid (a, b) \in M\} \text{ and}$$

$$\omega_T(\beta(a), b) := \omega_M(a, b).$$

Then $\mathbb{T}$ is a subbarcode matching. We note that $|\mathbb{T}| = |\mathbb{M}|$ and $|\mathbb{A}| = |\mathbb{A}^\beta|$. Additionally, $\mathbb{M}$ is a perfect matching, so $|\mathbb{M}| = |\mathbb{A}| = |\mathbb{B}|$. It follows that $\mathbb{A}^\beta \sqsubseteq \mathbb{B}$. By a similar argument we may also show that $\mathbb{B}^\beta \sqsubseteq \mathbb{A}$. Therefore, $\mathbf{d}_S(\mathbb{A}, \mathbb{B}) \leq \beta = \mathbf{d}_B(\mathbb{A}, \mathbb{B})$. □

10 Conclusion

We have given an efficient method for computing maximum subbarcode matchings and subbarcode distance. We have shown that barcodes are a poset under the subbarcode relation, and that subbarcode distance is a metric on persistence diagrams. Subbarcodes present efficient methods of comparison for persistence diagrams.

Acknowledgement

Thank you to Don Sheehy, my advisor for invaluable discussions and feedback, especially regarding the sampling algorithm.

References

[1] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on computers*, 28(09):643–647, 1979.

[2] D. Z. Chen, X. S. Hu, and X. Wu. Maximum red/blue interval matching with application. In *International Computing and Combinatorics Conference*, pages 150–158. Springer, 2001.

[3] O. A. Chubet, K. P. Gardner, and D. R. Sheehy. A theory of sub-barcodes. *arXiv preprint arXiv:2206.10504*, 2022.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms*. MIT press, 2022.

[5] A. Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.

[6] L. R. Ford and D. Fulkerson. Network flow and systems of representatives. *Canadian Journal of Mathematics*, 10:78–84, 1958.

[7] G. Gallo. An o (n log n) algorithm for the convex bipartite matching problem. *Operations Research Letters*, 3(1):31–34, 1984.

[8] F. Glover. Maximum matching in a convex bipartite graph. *Naval research logistics quarterly*, 14(3):313–316, 1967.

[9] P. Hall†. *On Representatives of Subsets*, pages 58–62. Birkhäuser Boston, Boston, MA, 1987.

[10] J. E. Hopcroft and R. M. Karp. An nˆ5/2 algorithm for maximum matchings in bipartite graphs. *SIAM Journal on computing*, 2(4):225–231, 1973.

[11] R. M. Karp, M. Luby, and A. Marchetti-Spaccamela. A probabilistic analysis of multidimensional bin packing problems. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 289–298, 1984.

[12] M. Kerber, D. Morozov, and A. Nigmetov. Geometry helps to compare persistence diagrams, 2017.

[13] G. Steiner and J. S. Yeomans. A linear time algorithm for maximum matchings in convex, bipartite graphs. *Computers & Mathematics with Applications*, 31(12):91–96, 1996.

A Subbarcode Transitivity

Lemma 15 (Transitivity) *If* $\mathbb{A} \sqsubseteq \mathbb{B}$ *and* $\mathbb{B} \sqsubseteq \mathbb{C}$ *then* $\mathbb{A} \sqsubseteq \mathbb{C}$.

Proof. Let $\mathbb{A} = (A, \omega_A)$, $\mathbb{B} = (B, \omega_B)$, and $\mathbb{C} = (C, \omega_C)$ be barcodes such that $\mathbb{A} \sqsubseteq \mathbb{B}$ and $\mathbb{B} \sqsubseteq \mathbb{C}$. Then there exists subbarcode matchings, (M, ω_M) from $\mathbb{A}$ to $\mathbb{B}$ and (T, ω_T) from $\mathbb{B}$ to $\mathbb{C}$.

Let $\mathsf{Net}(G)$ be the corresponding network to find the the maximum subbarcode matching from $\mathbb{A}$ to $\mathbb{C}$, as described in Section 5. Let $(L, \overline{L})$ be a cut of $\mathsf{Net}(G)$. Then $L = X \sqcup Y \sqcup Z$ and $\overline{L} = \overline{X} \sqcup \overline{Y} \sqcup \overline{Z}$ for

$$X = A \cap L \qquad Y = B \cap L \qquad Z = C \cap L$$
$$\overline{X} = A \setminus X \qquad \overline{Y} = B \setminus Y \qquad \overline{Z} = C \setminus Z$$

We examine $c(L, \overline{L})$:

$$\begin{aligned} \mathsf{c}(L, \overline{L}) &= \mathsf{c}(X \sqcup Y \sqcup Z,\ \overline{X} \sqcup \overline{Y} \sqcup \overline{Z}) \\ &= \mathsf{c}(s, \overline{X}) + \mathsf{c}(X, \overline{Y}) + \mathsf{c}(Y, \overline{Z}) + \mathsf{c}(Z, t) \end{aligned}$$

We now evaluate each term:

$$\mathsf{c}(s, \overline{X}) = \sum_{a \in \overline{X}} \omega_A(a) \qquad \mathsf{c}(X, \overline{Y}) = \sum_{a \in X} \sum_{b \in \overline{Y}} \omega_M(a, b)$$
$$\mathsf{c}(Z, t) = \sum_{c \in Z} \omega_C(c) \qquad \mathsf{c}(Y, \overline{Z}) = \sum_{b \in Y} \sum_{c \in \overline{Z}} \omega_T(b, c)$$

Notice (T, ω_T) is a subbarcode matching, so by necessity ω_C is greater than the marginals of ω_T for each $c \in C$. Similarly, ω_B is greater than the marginals of ω_M for each $b \in B$.

$$\begin{aligned} \sum_{c \in Z} \omega_C(c) &\geq \sum_{c \in Z} \sum_{b \in B} \omega_T(b, c) \\ &= \sum_{c \in Z} \sum_{b \in Y} \omega_T(b, c) + \sum_{c \in Z} \sum_{b \in \overline{Y}} \omega_T(b, c) \end{aligned}$$

It follows,

$$\begin{aligned} \mathsf{c}(Y, \overline{Z}) + \mathsf{c}(Z, t) &\geq \sum_{y \in Y} \sum_{c \in C} \omega_T(b, c) = \sum_{b \in Y} \omega_B(b) \\ &\geq \sum_{b \in Y} \sum_{a \in X} \omega_M(a, b). \end{aligned}$$

Then,

$$\begin{aligned} &\mathsf{c}(X, \overline{Y}) + \mathsf{c}(Y, \overline{Z}) + \mathsf{c}(Z, t) \\ &\qquad \geq \sum_{a \in X} \sum_{b \in \overline{Y}} \omega_M(a, b) + \sum_{a \in X} \sum_{b \in Y} \omega_M(a, b) \\ &\qquad = \sum_{a \in X} \omega_A(a). \end{aligned}$$

Finally,

$$\begin{aligned} &\mathsf{c}(s, \overline{X}) + \mathsf{c}(X, \overline{Y}) + \mathsf{c}(Y, \overline{Z}) + \mathsf{c}(Z, t) \\ &\qquad \geq \sum_{a \in \overline{X}} \omega_A(a) + \sum_{a \in X} \omega_A(a) \\ &\qquad = \sum_{a \in A} \omega_A(a) = |\mathbb{A}|. \end{aligned}$$

Thus, $c(L, \overline{L}) \geq |\mathbb{A}|$ for any cut $(L, \overline{L})$ of $\mathsf{Net}(G)$. Therefore $\mathbb{A} \sqsubseteq \mathbb{C}$. □

Approximating Convex Polygons by Histogons*

Jaehoon Chung† Sang Won Bae‡ Chan-Su Shin§ Sang Duk Yoon¶ Hee-Kap Ahn‖

Abstract

We study the problem of finding the largest inscribed histogon and the smallest circumscribed histogon for a convex polygon. A histogon is an axis-aligned rectilinear polygon such that every horizontal edge has an integer length. Depending on whether the horizontal width of a histogon is predetermined or not, we consider four different versions of the problem and present exact algorithms.

1 Introduction

Motivated by optimization problems in shape analysis, classification, and simplification [1, 2], we consider two optimization problems of approximating a convex polygon P, one by a largest inscribed histogon in P, and the other by a smallest circumscribing histogon.

A *histogon* is an axis-aligned rectilinear polygon such that every horizontal edge has an integer length. We call a histogon of width 1 a *unit histogon* and histogon of *width* k a *k-histogon*. Thus, a unit histogon is simply an axis-aligned rectangle of horizontal width 1, and its height is the length of the vertical sides which is a positive real number. A k-histogon H for a positive integer k can be described by k interior-disjoint unit histogons whose union is H. See Figure 1 for an illustration.

In the inscribed histogon problem, we compute a histogon with maximum area that can be inscribed in P.

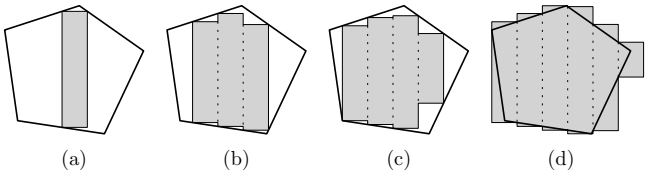


Figure 1: Histogons. (a) The largest inscribed unit histogon of P. (b) The largest inscribed histogon with width 3 of P. (c) The largest inscribed histogon of P. (d) The smallest circumscribed histogon of P

We call such a histogon a *largest* inscribed histogon of P. Depending on whether the horizontal width of a histogon is predetermined (1 or a positive integer k) or not, we consider three versions of the problem.

In the circumscribed histogon problem, we compute a histogon with minimum area that can be circumscribed to P. We call such a histogon a *smallest* circumscribed histogon of P.

We call a copy of a histogon rotated by $\theta \in [0, \pi)$ in counterclockwise direction a *histogon of orientation* θ. Our results can be applied to inscribed and circumscribed problems for histogons of orientation θ with the same time and space.

Approximation of shapes by histogons found its applications in several topics in calculus, most notably in Riemann sums and optimization. For a function graph (or a curve), the area under the graph can be approximated by a histogon: an inscribed histogon is an under-approximation of the area, called a *lower sum,* and a circumscribed histogon is an over-approximation of the area, called an *upper sum.* Many optimization problems are concerned with the largest inscribed figure and the smallest circumscribed figure of a shape. They are also closely related to real-world cost-optimization problems such as painting a piece using a spray gun, etching VLSI masks by electron beams with a fixed minimum width, and inspection.

Related Work. Extensive research has been done in past decades in computational geometry for inscribing and circumscribing polygons, and most of which handle relatively elementary shapes such as triangles, rectangles or parallelograms in a convex or a simple polygon. Alt et al. [3] gave an $O(\log n)$-time algorithm for finding a maximum-area axis-aligned rectangle that can be inscribed in a convex n-gon. Daniels et al. [7] gave an

*Work by H.-K. Ahn and J. Chung was supported by the Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. 2017-0-00905, Software Star Lab (Optimal Data Structure and Algorithmic Applications in Dynamic Geometric Environment)) and (No. 2019-0-01906, Artificial Intelligence Graduate School Program(POSTECH)). Work by S. D. Yoon was supported by "Cooperative Research Program for Agriculture Science & Technology Development (Project No. PJ015269032022)" Rural Development Administration, Republic of Korea.

†Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang, Korea. sk7755@postech.ac.kr

‡Division of Computer Science and Engineering, Kyonggi University, Suwon, Korea. swbae@kgu.ac.kr

§Division of Computer Engineering, Hankuk University of Foreign Studies, Seoul, Korea. cssin@hufs.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

$O(n \log^2 n)$-time algorithm for finding a maximum-area axis-aligned rectangle in a simple polygon with n vertices, possibly with holes. The running time was improved to $O(n \log n)$ by Boland et al. [4].

DePano et al. [8] gave an $O(n^2)$-time algorithm for finding a maximum-area equilateral triangle and square that can be inscribed in a convex n-gon and an $O(n^3)$-time algorithm for finding a maximum-area equilateral triangle that can be inscribed in a simple polygon with n vertices. Cabello et al. [5] first suggested $O(n^3)$-time exact algorithm for finding maximum-area or maximum-perimeter rectangles in a convex n-gon. Jin et al. [12] designed an $O(n^2)$-time algorithm for computing all the parallelograms with maximum area in a convex n-gon. Choi et al. [6] gave an $O(n^3 \log n)$-time algorithm for finding maximum-area rectangles in a simple polygon, possibly with holes. Lee et al. [14] studied maximum-area triangles with various restrictions in a convex or a simple polygon, possibly with holes.

Using the observation due to Freeman and Sharpia [10], Toussaint [16] gave an $O(n)$-time algorithm for finding a minimum-area rectangle enclosing a convex n-gon. The algorithm also works for finding a minimum-perimeter rectangle enclosing a convex polygon. O'Rourke et al. [15] gave an $O(n)$-time algorithm for finding a minimum-area triangle enclosing a convex n-gon.

Our Results. Our main results are efficient algorithms for computing optimal histogons (largest inscribed and smallest circumscribed histogons) for a convex polygon P with n vertices and all our algorithms use $O(n)$ space. We assume that the vertices of P are stored in an array in counterclockwise order along the boundary of P.

For the problem of inscribing a largest histogon in a convex n-gon, we present an $O(\log n)$-time algorithm for a largest unit histogon, an $O(\min\{n, k \log^2 \frac{n}{k}\})$-time algorithm for a largest histogon of width k for a fixed $k > 1$, and an $O(\min\{n, w \log^2 \frac{n}{w}\})$-time algorithm for a largest histogon. The symbol w denotes the width of a largest inscribed histogon in P, so the last algorithm is output-sensitive.

For the problem of circumscribing a smallest histogon of a fixed orientation for a convex n-gon, we present an $O(\min\{n, W \log \frac{n}{W}\})$-time algorithm. The symbol W denotes the (horizontal) width of P, so our algorithm is output-sensitive.

Sketch of Our Results. For the problem of inscribing a largest unit histogon, we define a function f that maps $t \in \mathbb{R}$ into the height of the largest inscribed unit histogon of P with the left side at $x = t$. We show that f is a concave, piecewise linear function, so we can perform a binary search to find a maximum of f, which corresponds to a largest unit histogon inscribed in P.

To find a largest k-histogon inscribed in P with $k > 1$, we present a characterization for the existence of k-histogon inscribed in P. For a k-histogon with the leftmost vertical side at $x = t$, we define a function $F(t)$ by the height of the k-histogon and show that F is a concave, piecewise linear function. We find a closed interval containing the $x = t^*$ which maximizes F and apply binary search to find t^* in the restricted domain. We present two algorithms for finding t^*, one using $O(n)$ time which is optimal for $k = \Omega(n)$ (Section 3.2.1) and the other using $O(k \log^2 \frac{n}{k})$ time for $k = O(n)$ (Section 3.2.2). When there is no restriction on the width of the histogon, we show that a largest inscribed histogon can be computed by invoking the algorithm for fixed width a constant number of times. (Section 3.3).

For the problem of circumscribing a smallest histogon, we show that the smallest circumscribed histogon H has width $\lceil W \rceil$. Moreover, either the leftmost vertical side of H contacts the leftmost vertex or the rightmost vertical side of H contacts the rightmost vertex of P. Thus, we compute histogons for two cases, and take the smaller one. See Section 4.

2 Preliminaries

Let P be a convex polygon with n vertices, stored in an array in counterclockwise order along the boundary of P. We denote by ∂P the boundary of P. For a point $p \in \mathbb{R}^2$, let $x(p)$ and $y(p)$ be the x-coordinate and the y-coordinate of p, respectively.

For a histogon H, let $w(H)$ be the horizontal width of H and let $|H|$ denote the area of H. We call a line segment connecting two distinct boundary points of P a *chord* of P.

3 Inscribed histogons

We compute a largest inscribed histogon in a convex polygon P with n vertices for three versions of the problem: a unit histogon, a histogon of width k for a given integer k, and a histogon of any integer width.

3.1 Largest inscribed unit histogon

For ease of discussion, we assume that no two edges of P are parallel to each other. The case with parallel edges can be handled with little modification. Observe that not every convex polygon contains a unit histogon. The following lemma shows the condition for P to contain a unit histogon of a positive height.

Lemma 1 *The longest horizontal chord in P has length larger than 1 if and only if there is a unit histogon of a positive height contained in P.*

Proof. Assume that the longest horizontal chord in P has length larger than 1. Let s be a horizontal unit segment contained in the interior of the longest chord. Then s can be translated vertically upward or downward while it is contained in P. Let s' be such a translated copy of s. Since P is convex, the convex hull of s and s' is a unit histogon of a positive height contained in P.

Assume that there is a unit histogon $\bar{H}$ of a positive height contained in P. Let s be a horizontal unit segment contained in $\bar{H}$, other than its top and bottom sides. Since no two edges of P are parallel to each other, one endpoint of s is in the interior of P. By extending s until both endpoints of s meet ∂P, we get a horizontal chord of length larger than 1 in P. □

By Lemma 1, we can determine the existence of a unit histogon contained in P from the length of the longest horizontal chord in P. Since P is convex and the vertices of P are stored in an array in counterclockwise order along ∂P, we apply binary search to find the longest horizontal chord in P in $O(\log n)$ time.

From now on, we assume that the length of the longest horizontal chord in P is larger than 1. Let $\bar{P}$ be the translate of P by vector $(-1, 0)$. Let $Q = P \cap \bar{P}$, which is a convex polygon. Then there is one-to-one correspondence between any vertical chord at $x = t$ of Q and the largest unit histogon with left side at $x = t$ inscribed in P. Moreover, the length of a vertical chord and the height of its corresponding histogon are the same. Thus, the height of any largest inscribed unit histogon of P is the length of a longest vertical chord in Q. See Figure 2(a).

Note that ∂P and $\partial \bar{P}$ intersect each other at most twice. If there is a horizontal edge of length larger than 1 in P, one intersection may appear as a horizontal line segment on the horizontal edge. Then each intersection corresponds to the horizontal chord of unit length in P or a horizontal edge of length larger than 1 of P. We can compute the intersections $\partial P \cap \partial \bar{P}$ in $O(\log n)$ time by binary search using the sorted array of vertices of P. The longest vertical chord in Q, and the horizontal chords of unit length of P, can be computed in $O(\log n)$ time by applying binary search on the boundary of Q using the sorted array of vertices of P.

To sum up, we can determine whether a unit histogon of a positive height inscribed in P exists in $O(\log n)$ time, and if so, we can compute the largest inscribed unit histogon in the same time.

Theorem 2 *Given a convex polygon P with n vertices stored in an array in order along its boundary, we can find in $O(\log n)$ time the largest unit histogon inscribed in P.*

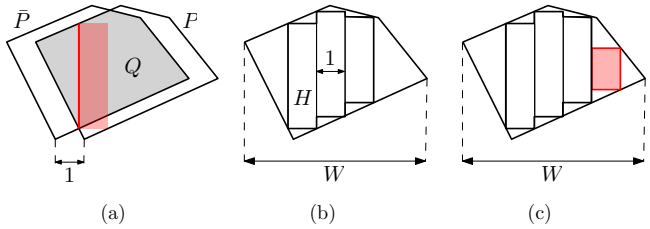


Figure 2: (a) For the translate $\bar{P}$ of P by vector $(-1, 0)$, $P \cap \bar{P}$ is also a convex polygon. (b) $\bar{w}$ is 5.2 and $w(H) = \lfloor \bar{w} \rfloor - 2 = 3$. (c) The largest histogon has width larger than $\lfloor \bar{w} \rfloor - 2$.

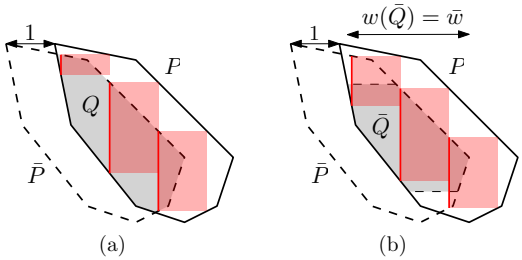


Figure 3: (a) Three unit histogons whose union is not a 3-histogon. (b) Every unit histogon intersects $\bar{Q}$, and every two consecutive unit histogons share a portion along their vertical sides. Their union is a 3-histogon.

3.2 Largest inscribed histogon of a fixed width

Given a positive integer $k > 1$, we compute a largest inscribed histogon H of P with $w(H) = k$. For $t \in \mathbb{R}$, let $H(t)$ denote the largest inscribed k-histogon in P with the leftmost vertical side at $x = t$, and let $\bar{H}(t)$ denote the largest inscribed unit histogon of P with the left side at $x = t$. Then $H(t)$ can be determined by a disjoint union of k unit histogons $\bar{H}(t), \bar{H}(t+1), \ldots, \bar{H}(t+k-1)$.

It is possible that there is no k-histogon that can be inscribed in P even if there are k interior-disjoint unit histogons inscribed in P. Figure 3(a) shows an example with three unit histogons whose union is not a 3-histogon.

Thus, to guarantee a k-histogon inscribed in P, we need the following lemma. Let $\bar{Q}$ be the union of $\ell \cap Q$ over all horizontal lines ℓ with $|\ell \cap Q| \geq 1$. Since Q is a convex polygon, $\bar{Q}$ is also a convex polygon.

Lemma 3 *Assume that $w(\bar{Q}) \neq k - 1$. Then, $w(\bar{Q}) > k - 1$ if and only if there is a k-histogon that can be contained in P.*

Proof. Assume that $w(\bar{Q}) > k - 1$. By letting $t = x(v) + \epsilon$ for the leftmost vertex v of $\bar{Q}$ and sufficiently small $\epsilon > 0$ (smaller than $w(\bar{Q}) - k + 1$), $|\ell_i \cap \bar{Q}| > 0$ for each vertical line $\ell_i : x = t + i$ and $|\bar{H}(t+i)| > 0$ for $i = 0, 1, \ldots, k-1$. Observe that the union of $\bar{H}(t+i)$ for $i = 0, 1, \ldots, k-1$ is a k-histogon $H(t)$ if and only if every two consecutive unit histogons $\bar{H}(t+i)$ and $\bar{H}(t+i+1)$

share a portion (a point or a vertical segment) along their vertical sides at $x = t+i+1$ for $i = 0, 1, \ldots, k-2$. Two consecutive unit histogons $\bar{H}(t+i)$ and $\bar{H}(t+i+1)$ share a portion along their vertical sides if and only if there is a horizontal line ℓ that intersects both $\bar{H}(t+i)$ and $\bar{H}(t+i+1)$.

Suppose that there is no horizontal line that intersects both $\bar{H}(t+i)$ and $\bar{H}(t+i+1)$. Then, we can find a horizontal line ℓ' such that $\ell_i \cap \bar{Q}$ and $\ell_{i+1} \cap \bar{Q}$ are on the opposite sides of ℓ', and the length $|\ell' \cap \bar{Q}|$ must be smaller than 1. This contradicts to the definition of $\bar{Q}$. Therefore, the union of $\bar{H}(t+i)$ for $i = 0, 1, \ldots, k-1$ is a k-histogon contained in P. See Figure 3(b) for an illustration.

Let H be a k-histogon that is inscribed in P. Then H can be partitioned into k interior-disjoint unit histogons $\bar{H}(t+i)$ of positive heights, each corresponding to a vertical line $\ell_i : x = t+i$ for $i = 0, 1, \ldots, k-1$, where t is the x-coordinate of the leftmost side of H. Since every two consecutive unit histogons $\bar{H}(t+i)$ and $\bar{H}(t+i+1)$ share a portion (a point or a vertical segment) along their vertical sides at $x = i+1$, there is a horizontal line ℓ that intersects both $\bar{H}(t+i)$ and $\bar{H}(t+i+1)$. Then $|\ell \cap Q| \geq 1$ which means both ℓ_i and ℓ_{i+1} intersects $\bar{Q}$. Thus $w(\bar{Q}) > k-1$. □

By Lemma 3, we can check whether a k-histogon exists in P by computing $w(\bar{Q})$ using binary search. First, we compute $Q = P \cap \bar{P}$ as we do in Section 3.1 in $O(\log n)$ time. Since the vertices of Q are stored in an array in order, $\bar{Q}$ can be computed in $O(\log n)$ time by locating the two horizontal chords of unit length in Q by binary search on the array.

By applying binary search on the array of $\bar{Q}$, we can compute $w(\bar{Q})$ in $O(\log n)$ time, and decide whether a k-histogon can be inscribed in P or not by Lemma 3 if $w(\bar{Q}) > k-1$. If $w(\bar{Q}) = k-1$, we have to check the existence of $\bar{H}(x(v)+i)$ in P for $i = 0, 1, \ldots, k-1$, where v is the leftmost vertex of $\bar{Q}$. From the convexity of P, the existence of $\bar{H}(x(v)+i)$ in P for $i = 0, 1, \ldots, k-1$ can be confirmed from the existence of two unit histogons $\bar{H}(x(v))$ and $\bar{H}(x(v)+k-1)$ in P, which can be done in $O(\log n)$ time by binary search on the array of Q.

Let I be the set of x-coordinates of all points in $\bar{Q}$, so I is equivalent to the projection of $\bar{Q}$ onto the x-axis. If a k-histogon $H(t)$ can be inscribed in P, there are k unit histogons $\bar{H}(t), \bar{H}(t+1), \ldots, \bar{H}(t+k-1)$ inscribed in P such that any two consecutive unit histogons share a portion along their vertical sides. This implies that $t, t+1, \ldots, t+k-1$ must be contained in the interval I.

We define a function $f\colon \mathbb{R} \to \mathbb{R}$ such that $f(t) = |\bar{H}(t)|$ for any $t \in I$ and $f(t) = -\infty$ for any $t \notin I$. Observe that f is a concave function consisting $O(n)$ linear pieces.

Let $F(x) = \sum_{0 \leq i < k} f(x+i)$. If there exists a k-histogon $H(x)$ inscribed in P, $F(x)$ is the area of $H(x)$. Otherwise, $F(x)$ is $-\infty$. Observe that F is also a concave, piecewise linear function with $O(kn)$ complexity. Our goal is to maximize the function $F(x)$ over $x \in \mathbb{R}$. Let $x^* \in \mathbb{R}$ be a value at which F attains the maximum. If there are more than one such value, we choose the least one as x^*.

Let F'_- be the left-hand derivative of F. There exists a real value $\hat{x} \in \mathbb{R}$ such that $F'_-(\hat{x}) > 0$ and $F'_-(\hat{x}+1) \leq 0$, since F is a concave function. If we restrict the domain of F to $[\hat{x}, \hat{x}+1)$, the function consists of $O(n)$ pieces and we find x^* on it.

We present two algorithms for finding x^*, one using $O(n)$ time which is optimal for $k = \Omega(n)$ (Section 3.2.1) and one using $O(k \log^2 \frac{n}{k})$ time for $k = O(n)$ (Section 3.2.2).

3.2.1 An $O(n)$-time algorithm

We present an $O(n)$-time algorithm for finding x^*, which is optimal for $k = \Omega(n)$. Recall that we can get the interval I in $O(\log n)$ time. We compute the function f that maps $t \in I$ to $|\bar{H}(t)|$ and $t \notin I$ to $-\infty$ by traversing Q in $O(n)$ time. Assume that f consists of $m+2$ linear pieces, where $m = O(n)$. $h_0, h_1, \ldots, h_m, h_{m+1}$ denote the linear functions of these pieces in the order of their domain. Let h'_i denote the derivative of h_i and $(a_i, a_{i+1}]$ denote the domain of h_i for $i = 0, 1, \ldots, m+1$. From the construction of f, $a_0 = -\infty$, $a_{m+2} = +\infty$, $h'_0 = +\infty$, and $h'_{m+1} = -\infty$.

Lemma 4 *We can find $\hat{x}$ such that $F'_-(\hat{x}) > 0$ and $F'_-(\hat{x}+1) \leq 0$ in $O(n)$ time using $O(n)$ space.*

Proof. Let t_i be the smallest integer in $(a_i, a_{i+1}]$ and let N_i be the number of integers in $(a_i, a_{i+1}]$ for $i = 1, 2, \ldots, m$. Then $F'_-(t_i) = \sum_{i \leq j < r_i} (h'_j \cdot N_j) + h'_{r_i} \cdot (k - \sum_{i \leq j < r_i} N_j)$, where r_i is the largest integer such that $\sum_{i \leq j < r_i} N_j < k$. Note that $r_i \leq r_{i+1}$ for each integer i. Then we compute $\sum_{i \leq j < r_i} (h'_j \cdot N_j)$ for all integers i in $O(n)$ time in total. Thus we compute $F'_-(t_i)$ for all integers i in $O(n)$ time. We find an index L such that $F'_-(t_L) > 0$ and $F'_-(t_{L+1}) \leq 0$ in $O(n)$ time. Similarly, we find an index R such that $F'_-(t_R - k) > 0$ and $F'_-(t_{R+1} - k) \leq 0$.

Then there exists an integer s_L with $0 \leq s_L < N_L$ such that $F'_-(t_L + s_L) > 0$ and $F'_-(t_L + s_L + 1) \leq 0$, and there exists an integer s_R with $0 \leq s_R < N_R$ such that $F'_-(t_R - k + s_R) > 0$ and $F'_-(t_R - k + s_R + 1) \leq 0$. This means that $\hat{x} = t_L + s_L = t_R - k + s_R$.

Observe that $F'_-(t_L + a) = \sum_{L \leq j < R} (h'_j \cdot N_j) - h'_L \cdot a + h'_R \cdot b$, where $b = a + k - (t_R - t_L)$. Since the first term $\sum_{L \leq j < R} (h'_j \cdot N_j)$ remains the same for varying a, we can find s_L satisfying $F'_-(t_L + s_L) > 0$ and $F'_-(t_L + s_L + 1) < 0$ in $O(n)$ time. Then $s_R = s_L + k - (t_R - t_L)$. Thus we compute $\hat{x}$ such that $F'_-(\hat{x}) > 0$ and $F'_-(\hat{x}+1) \leq 0$ in $O(n)$ time. □

Then $x^* = \max\{x \in [\hat{x}, \hat{x}+1) \mid F'_-(x) > 0\}$. For each integer i with $1 \le i \le m$, let $k_i(x)$ be a function $k_i : [0,1) \mapsto \mathbb{Z}$ that maps $x \in [0,1)$ to the number of integers j with $0 \le j < k$ satisfying $\hat{x}+x+j \in (a_i, a_{i+1}]$. Note that k_i is a step function having at most three steps in domain $[0,1)$. Let $g_i(x) = k_i(x) \cdot h'_i$ be a function $g_i : [0,1) \to \mathbb{R}$ for each integer i with $1 \le i \le m$. Then g_i is also a step function with at most three steps in domain $[0,1)$. For given an integer i, functions k_i, h'_i, and g_i can be computed in $O(1)$ time. In a step function, each step has an interval as its domain and endpoints of the interval are called *breakpoints* of the step function. Then $\sum_{1 \le i \le m} g_i(x) = F'_-(\hat{x}+x)$ and x^* is a breakpoint of g_i's. For a breakpoint s of g_i's, we can compute $F'_-(\hat{x}+s)$ in $O(m) = O(n)$ time by computing $g_i(s)$ for each i and taking the sum of them. Since F'_- is decreasing, we can find x^* in the set of breakpoints by using the median of the set.

Lemma 5 *We can find $x^* \in [\hat{x}, \hat{x}+1)$ in $O(n)$ time using $O(n)$ space.*

Proof. We can construct all g_i functions in $O(n)$ time since each g_i can be computed in $O(1)$ time. Let X be the set of breakpoints in all g_i's. Then $|X| = O(n)$

We find x^* in X iteratively by using the medians of X. The number of breakpoints of X halves over each iteration, and thus the total time spent for computing the median s and $F'_-(\hat{x}+s)$ is $O(n)$. The median s of X can be computed by a selection algorithm that takes time linear to the cardinality of X using Hoare's selection algorithm [11]. Note that $F'_-(\hat{x}+x) = \sum_{1 \le i \le m} g_i(x)$ and g_j remains constant in the rest of iterations if X contains no breakpoint of g_j. Let G be the sum of g_j's values such that X contains no breakpoint of g_j. In each iteration, we compute the sum of $g_i(s)$ if X contains a breakpoint of g_i, and compute $F'_-(\hat{x}+s)$ from the sum and G. Then we update X by removing those breakpoints larger than s if $F'_-(\hat{x}+s) \le 0$, and removing those breakpoints smaller than s if $F'_-(\hat{x}+s) > 0$. Finally, we update G. This can be done in time linear to the number of breakpoints in X. We repeat this until X consists of at most two breakpoints.

Observe that F'_- has a positive value at one of the breakpoints. We return the breakpoint as x^*. Since the size of X halves over each iteration, the total time spent over all iterations is $O(n)$. Therefore, x^* can be found in $O(n)$ time using $O(n)$ space. □

Combining Lemma 4 and Lemma 5, we have the following theorem.

Theorem 6 *Given a convex polygon P with n vertices given in order along its boundary and an integer $k > 1$, we can find the largest inscribed k-histogon H in P in $O(n)$ time using $O(n)$ space.*

3.2.2 An $O(k \log^2 \frac{n}{k})$-time algorithm

We present another algorithm for finding x^* in $O(k \log^2 \frac{n}{k})$ time for $k = O(n)$. From now on, we assume that $n \ge 4k$. If $n < 4k$, we apply the algorithm in Section 3.2.1 taking $O(k)$ time. We partition $Q = P \cap \bar{P}$ into two parts along the line ℓ through the leftmost vertex and the rightmost vertex of Q. Let Q^+ denote the upper part and let Q^- denote the lower part of it. Observe that any vertical chord of Q can be partitioned into two pieces by ℓ, one vertical chord of Q^+ and one vertical chord of Q^-.

We group the edges of Q^+ into blocks $B_1, B_2, \ldots, B_m$ of size $\lfloor \frac{n}{k} \rfloor$ consecutively in order from left to right. Similarly, we group the edges of Q^- to blocks $C_1, C_2, \ldots, C_l$ of size $\lfloor \frac{n}{k} \rfloor$ consecutively in order from left to right. Both m and l are $O(k)$. Every block has size $\lfloor \frac{n}{k} \rfloor$, except that the last blocks, B_m and C_l, may consist of less number of edges. For an edge e of P, we say e *contains* an x-coordinate t if the vertical line at $x = t$ intersects e. We say a block B *contains* an x-coordinate t if B contains an edge e and e contains an x-coordinate t.

Our algorithm works as follows. It first computes $\bar{x}$ that maximizes $f(x)$, and sets $D = f'_-(\bar{x})$. It initializes indices $i = 0$ and $j = 0$. Then it searches $\hat{x}$ linearly from $\bar{x}$ by updating D value $k-1$ times as follows. It increases i by 1 and sets $w = \bar{x} + i$ if $D > 0$, and it increases j by 1 and sets $w = \bar{x} - j$ if $D \le 0$. Then it finds blocks B_s and C_t that contain w, computes $f'_-(w)$ using the edges containing w, and updates $D = D + f'_-(w)$. After $k-1$ iterations, we have $D = F'_-(\bar{x} - j)$. The algorithm returns $\hat{x} = \bar{x} - j$ if $D > 0$, and $\hat{x} = \bar{x} - j - 1$ if $D \le 0$.

Lemma 7 *We can find $\hat{x}$ such that $F'_-(\hat{x}) > 0$ and $F'_-(\hat{x}+1) \le 0$ in $O(k \log \frac{n}{k})$ time using $O(n)$ space.*

Proof. First we compute $\bar{x}$ that maximizes $f(x)$ in $O(\log n)$ time using binary search. Since $F(x) = \sum_{0 \le i < k} f(x+i)$ and f is a concave, piecewise linear function, we can get a larger k-histogon than $H(x)$ by decreasing x if $\bar{x} < x$ or by increasing x if $x + k - 1 < \bar{x}$. Then $\bar{x} - k + 1 \le x^* \le \bar{x}$. At the end of iterations, $D = F'_-(\bar{x} - j)$. If $D > 0$, $F'_-(\bar{x} - j) > 0$ and $F'_-(\bar{x} - j + 1) \le 0$, that is $\hat{x} = \bar{x} - j$. If $D \le 0$, $F'_-(\bar{x} - j - 1) > 0$ and $F'_-(\bar{x} - j) \le 0$, that is $\hat{x} = \bar{x} - j - 1$. Note that the indices s and t of blocks B_s and C_t containing $w = \bar{x} + i$ monotonically increase while i increases. The indices s' and t' of blocks $B_{s'}$ and $C_{t'}$ containing $w = \bar{x} - j$ monotonically decrease while j increases. Then the step for finding the blocks takes $O(k)$ time in total, since the number of blocks is $O(k)$. Moreover, we can find two edges containing $w = \bar{x} + i$ or $w = \bar{x} - j$ in the blocks in $O(\log \frac{n}{k})$ time using binary search. Thus the time complexity of the algorithm is $O(k \log \frac{n}{k})$ time using $O(n)$ space. □

We define a function $q_i : [0,1) \mapsto \mathbb{R}$ for each integer i with $0 \le i < k$ by $q_i(x) = f'_-(\hat{x} + i + x)$. Then

$\sum_{0\le i<k} q_i(x) = F'_-(\hat{x}+x)$. Note that q_i is a step function on its domain and the total number of breakpoints of all q_i's is $O(n)$. Let b^* be the largest breakpoint of q_i's such that $\sum_{0\le i<k} q_i(b^*) = F'_-(\hat{x}+b^*) > 0$. Then x^* is $\hat{x}+b^*$ since $x^* = \max\{x \in [\hat{x}, \hat{x}+1) \mid F'_-(x) > 0\}$.

Note that each breakpoint is induced by a vertex of Q. Consider the sequence of the breakpoints of q_i induced by the vertices on Q^+ from left to right. Let $b_{i,j}$ denote the j-th breakpoint of q_i in the sequence. Similarly, in the sequence of the breakpoints of q_i induced from the vertices on Q^- from left to right, let $c_{i,j}$ be the j-th breakpoint of q_i in the sequence. Then there are $2k$ sequences, two for each q_i, and there are $O(n)$ breakpoints in total.

Lemma 8 *After $O(k \log \frac{n}{k})$-time preprocessing, we can get $b_{i,j}$ and $c_{i,j}$ in $O(1)$ time for any given indices i and j.*

Proof. We show how to get $b_{i,j}$. We can get $c_{i,j}$ similarly. Let $u_{i,j}$ denote the vertex corresponding to $b_{i,j}$. By the definition of q_i, $x(u_{i,j}) = \hat{x} + i + b_{i,j}$. Thus, for given indices i and j, we can get $b_{i,j}$ in $O(1)$ time if we can get $x(u_{i,j})$ in $O(1)$ time. We group the vertices of Q^+ into blocks of size $\lfloor \frac{n}{k} \rfloor$ consecutively in order from left to right. Let B and B' be the two leftmost blocks containing some $t \in [\hat{x}+i, \hat{x}+i+1)$. Then $u_{i,1}$ is the leftmost vertex on edges of B and B' satisfying $x(u_{i,1}) \in [\hat{x}+i, \hat{x}+i+1)$. We search for B and B' for every i linearly in $O(k)$ time. For each i, we find $u_{i,1}$ using binary search in $O(\log \frac{n}{k})$ time. Thus, we can find $u_{i,1}$ for every q_i in $O(k \log \frac{n}{k})$ time. Then we can get $x(u_{i,j})$ for $j > 1$ for each q_i in $O(1)$ time as the vertices of Q are stored in an array in order along its boundary. □

By Lemma 8, we can construct the collection of $2k$ sequences implicitly in $O(k \log \frac{n}{k})$ time such that each breakpoint can be accessed in constant time. Our goal is to find the largest breakpoint b^* in the collection such that $\sum_{0\le i<k} q_i(b^*) > 0$.

Kaplan et al. [13] gave a selection algorithm for a row-sorted matrix A with m rows that computes the k smallest items of A in $O(m+k)$ time. Frederickson et al. [9] also gave an $O(m)$-time algorithm for finding the k-th smallest item of A. We describe an algorithm that finds b^* in the collection of $2k$ sequences in $O(k \log^2 \frac{n}{k})$ time. Recall that $n \ge 4k$. We partition each sequence of the collection into blocks of size $\lfloor \frac{n}{4k} \rfloor$. They are partitioned into a number of full blocks, followed possibly by one block of size less than $\lfloor \frac{n}{4k} \rfloor$. Then the number of blocks in the collection is $\Theta(k)$. We set the last element in each block as the representative of the block. We select k smallest representatives among all representatives in $O(k)$ time using the selection algorithm by Kaplan et al. We claim that the k-th smallest representative r is an approximated median of the collection. First, the rank of r in the collection is at least $\frac{n}{8}$, since $\frac{n}{8} \le k\lfloor \frac{n}{4k} \rfloor$ for $n \ge 4k$. Second, the number of blocks containing a breakpoint less than r is at most $3k-1$ in the collection, since r is the k-th smallest representative. Then the rank of r in the collection is less than $n - (3k-1)\lfloor \frac{n}{4k} \rfloor \le n - \frac{n}{2} = \frac{n}{2}$.

We evaluate $F'_-(r) = \sum_{1\le i\le k} q_i(r)$. If $F'_-(r) \le 0$, we shrink the search range of each sequence of the collection to the range of the elements smaller than r. If $F'_-(r) > 0$, we shrink the search range of each sequence of the collection to the range of the elements larger than or equal to r. The number of breakpoints in the collection decreases by a constant factor at each iteration.

To evaluate $\sum_{1\le i\le k} q_i(r)$, we need to locate the position of r in each sequence of the collection, except the sequence where r was selected. For each sequence, we already know the block containing r, and we can find the position of r in the block using binary search in $O(\log \frac{n}{k})$ time. Thus it takes $O(k \log \frac{n}{k})$ time to compute the positions of r in all sequences in total.

After $O(\log \frac{n}{4k})$ iterations, the number of remaining breakpoints in the collection becomes smaller than $4k$. Then we use the algorithm in Section 3.2.1 with all elements in the collection to find b^* taking $O(k)$ time.

Taken together, there are $O(\log \frac{n}{k})$ iterations, each of which takes $O(k \log \frac{n}{k})$ time. Thus it takes $O(k \log^2 \frac{n}{k})$ time using $O(n)$ space to find $x^* = \hat{x} + b^*$.

Theorem 9 *Given a convex polygon P with n vertices stored in an array in order along its boundary and an integer $k = O(n)$, we can find the largest inscribed k-histogon H in P in $O(k \log^2 \frac{n}{k})$ time using $O(n)$ space.*

3.3 Largest inscribed histogon

Now we consider the variation that no restriction is imposed on the width of a largest inscribed histogon in P. We find the largest inscribed histogon H in P.

Recall that $\bar{Q}$ is the union of $\ell \cap Q$ with $|\ell \cap Q| \ge 1$ for all horizontal lines ℓ. Let $\bar{w} = w(\bar{Q})$. Then the largest inscribed histogon has width at most $\lfloor \bar{w} \rfloor + 1$ by Lemma 3.

We now claim that the width of the largest inscribed histogon in P is either $\lfloor \bar{w} \rfloor - 1$, $\lfloor \bar{w} \rfloor$ or $\lfloor \bar{w} \rfloor + 1$. Suppose that the largest inscribed histogon H in P has width $k \le \lfloor \bar{w} \rfloor - 2$. Then there are k vertical lines intersecting $\bar{Q}$ such that all distance between two consecutive lines is 1. Since $\bar{w} - k + 1 \ge 3$, the leftmost vertical line is at distance larger than 1 from the leftmost point of $\bar{Q}$ or the rightmost vertical line is at distance larger than 1 from the rightmost point of $\bar{Q}$. Thus, we can always attach a unit histogon with positive height to the left or right of H and get an inscribed histogon with a larger area in P. Thus, the largest histogon has width $\lfloor \bar{w} \rfloor - 1$, $\lfloor \bar{w} \rfloor$ or $\lfloor \bar{w} \rfloor + 1$. See Figure 2(b–c). Once we compute $\bar{w}$ in $O(\log n)$ time, we can compute the largest histogons

(of width $\lfloor \bar{w} \rfloor - 1$, $\lfloor \bar{w} \rfloor$ and $\lfloor \bar{w} \rfloor + 1$) using the algorithms in Section 3.2 and choose the largest one.

In conclusion, we can compute the largest inscribed histogon of P in $O(n)$ time using $O(n)$ space by Theorem 6. For $\bar{w} = O(n)$, we can compute it in $O(\bar{w} \log^2 \frac{n}{\bar{w}})$ time using $O(n)$ space by Theorem 9.

Theorem 10 *Given a convex polygon P with n vertices stored in an array in order along its boundary, we can find the largest inscribed histogon in P in $O(\min\{n, \bar{w} \log^2 \frac{n}{\bar{w}}\})$ time using $O(n)$ space, where W denotes the width of the largest inscribed histogon in P.*

4 Smallest circumscribed histogon

We consider the problem of covering a convex polygon P with n vertices by a histogon with smallest area and present algorithms for computing the smallest circumscribed histogon of P.

We denote by $\bar{H}(t)$ the smallest unit histogon with the left side at $x = t$ that covers the part of P between $x = t$ and $x = t + 1$. Let l_t denote the intersection between P and the vertical line $x = t$. Observe that $\bar{H}(t)$ is defined if l_t or l_{t+1} has a positive length, and $|\bar{H}(t)| = \max\{|l_t|, |l_{t+1}|\}$.

Let H^* denote the smallest histogon covering P, and let x^* be the x-coordinate of the leftmost vertical side of H^*. Then H^* can be represented by the disjoint union of $w(H^*)$ unit histogons, $\bar{H}(x^*), \bar{H}(x^* + 1), \ldots \bar{H}(x^* + w(H^*) - 1)$.

We denote by $\hat{P}$ the Minkowski sum of P and the horizontal segment with endpoints $(-1, 0)$ and $(0, 0)$. Note that the length of the longest vertical segment contained in $\hat{P}$ at $x = t$ is the same as $|\bar{H}(t)|$.

We define a function $g : \mathbb{R} \mapsto \mathbb{R}$ by $g(t) = |\bar{H}(t)|$ if $\bar{H}(t)$ is defined, otherwise $g(t) = -\infty$. Then g is a concave, piecewise linear function, since $\hat{P}$ is convex and $g(t)$ is the length of the longest vertical segment at $x = t$ contained in $\hat{P}$.

Lemma 11 *Let H^* be a smallest histogon covering P. Then, $w(H^*) = \lceil W \rceil$, where W is the (horizontal) width of P. There is a smallest histogon covering P whose leftmost vertical side contacts the leftmost vertex of P or whose rightmost vertical side contacts the rightmost vertex of P.*

Proof. Let x_l and x_r be the x-coordinates of the leftmost vertex and the rightmost vertex of P, respectively. Any smallest histogon H^* covering P with its leftmost vertical side with $x = x^*$ satisfies $x_l - 1 < x^* \leq x_l$ and $x_r \leq x^* + w(H^*) < x_r + 1$. Thus, $w(H^*) \geq \lceil x_r - x_l \rceil = \lceil W \rceil$ and $w(H^*) \leq \lceil W \rceil + 1$. Suppose that $w(H^*) = \lceil W \rceil + 1$. We define a function $G : I \mapsto \mathbb{R}$ by $G(x) = \sum_{0 \leq i < \lceil W \rceil + 1} g(x + i)$, where I is a maximal interval such that $g(x + i) > 0$ for all integers i with $0 \leq i < \lceil W \rceil + 1$. We minimize $G(x)$ subject to $x \leq x_l$ and $x_r \leq x + \lceil W \rceil + 1$, which P can be circumscribed by the union of $\bar{H}(x), \bar{H}(x+1), \ldots \bar{H}(x + \lceil W \rceil)$. If $G(x)$ has a minimum, $G(x)$ is minimized at $x = x_l$ or $x = x_r - \lceil W \rceil - 1$ since G is also concave and piecewise linear as g. Thus $x^* = x_l$ or $x^* = x_r - \lceil W \rceil - 1$ which means H^* touches either the leftmost vertex or the rightmost vertex of P. Then either $\bar{H}(x^*)$ or $\bar{H}(x^* + \lceil W \rceil)$ does not intersect P, a contradiction. Thus, $w(H^*) = \lceil W \rceil$ and H^* touches either the leftmost vertex or the rightmost vertex of P. □

By Lemma 11, $w(H^*) = \lceil W \rceil$ and there are only two candidate locations for H^*, one with $x^* = x_l$ and one with $x^* = x_r - \lceil W \rceil$. To compute their areas, we can use the method for computing the area of the largest inscribed histogon in Section 3.3. More precisely, we show how to compute the area of the smallest circumscribed histogon with $x^* = x_l$. We construct function g in $O(n)$ time using the vertices of $\hat{P}$ stored in an array in order along its boundary which can be computed in $O(\log n)$ time. For each piece of g, we find in $O(1)$ time the smallest integer s and the largest integer t such that $x_l + s$ and $x_l + t$ are contained in the domain of the piece. Then we can compute $\sum_{s \leq i \leq t} g(x_l + i)$ in $O(1)$ time. By summing the values over all pieces, $\sum_{0 \leq i < \lceil W \rceil} g(x_l + i)$ can be computed in $O(n)$ time. If $W = O(n)$, similar to Lemma 8, we find two edges containing $x = x_l + i$ for all $0 \leq i < \lceil W \rceil$ in $O(W \log \frac{n}{W})$ time and compute $\sum_{0 \leq i < \lceil W \rceil} g(x_l + i)$ in $O(W)$ time. Among two candidates for H^*, the smaller one is the smallest circumscribed histogon of P.

Theorem 12 *Given a convex polygon P with n vertices stored in an array in order along its boundary, we can find the smallest circumscribed histogon of P in $O(\min\{n, O(W \log \frac{n}{W})\})$ time using $O(n)$ space where W denotes the width of the smallest circumscribed histogon in P.*

5 Discussion

We present algorithms for computing the largest inscribed histogon and the smallest circumscribed histogon for a convex polygon. The histogons are required to be axis-aligned. A direction for future work is to consider a generalization of the problem in which the histogons can be of arbitrary orientations.

References

[1] H.-K. Ahn, S. W. Bae, O. Cheong, and J. Gudmundsson. Aperture-angle and hausdorff-approximation of convex figures. *Discrete & Computational Geometry*, 40:414–429, 2008.

[2] H.-K. Ahn, P. Brass, O. Cheong, H.-S. Na, C.-S. Shin, and A. Vigneron. Inscribing an axially symmetric polygon and other approximation algorithms for planar convex sets. *Computational Geometry*, 33:152–164, 2006.

[3] H. Alt, D. Hsu, and J. Snoeyink. Computing the largest inscribed isothetic rectangle. In *Proc. 7th Canad. Conf. Comput. Geom. (CCCG 1995)*, pages 67–72, 1995.

[4] R. P. Boland and J. Urrutia. Finding the largest axis aligned rectangle in a polygon in $O(n \log n)$ time. In *Proc. 13th Canad. Conf. Comput. Geom. (CCCG 2001)*, pages 41–44, 2001.

[5] S. Cabello, O. Cheong, C. Knauer, and L. Schlipf. Finding largest rectangles in convex polygons. *Computational Geometry*, 51:67–74, 2016.

[6] Y. Choi, S. Lee, and H.-K. Ahn. Maximum-area and maximum-perimeter rectangles in polygons. *Computational Geometry*, 94:101710, 2021.

[7] K. Daniels, V. Milenkovic, and D. Roth. Finding the largest area axis-parallel rectangle in a polygon. *Computational Geometry*, 7(1):125–148, 1997.

[8] A. DePano, Y. Ke, and J. O'Rourke. Finding largest inscribed equilateral triangles and squares. In *Proc. 25th Allerton Conf. Commun. Control Comput.*, pages 869–878, 1987.

[9] G. N. Frederickson and D. B. Johnson. Generalized selection and ranking: Sorted matrices. *SIAM Journal on Computing*, 13(1):14–30, 1984.

[10] H. Freeman and R. Shapira. Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Commun. ACM*, 18(7):409?413, jul 1975.

[11] C. A. R. Hoare. Algorithm 65: Find. *Commun. ACM*, 4(7):321?322, jul 1961.

[12] K. Jin and K. Matulef. Finding the maximum area parallelogram in a convex polygon. In *Proc. 23rd Canad. Conf. Comput. Geom. (CCCG 2011)*, 2011.

[13] H. Kaplan, L. Kozma, O. Zamir, and U. Zwick. Selection from heaps, row-sorted matrices and x+y using soft heaps. In *Proc. 2nd Sympos. Simplicity Algo. (SOSA 2019)*, pages 5:1–5:21, 2019.

[14] S. Lee, T. Eom, and H.-K. Ahn. Largest triangles in a polygon. *Computational Geometry*, 98:101792, 2021.

[15] J. O'Rourke, A. Aggarwal, S. Maddila, and M. Baldwin. An optimal algorithm for finding minimal enclosing triangles. *Journal of Algorithms*, 7(2):258–269, 1986.

[16] G. Toussaint. Solving geometric problems with the rotating calipers. *In Proceedings of IEEE MELECON'83*, 83, 02 2000.

Fast Deterministic Approximation of Medoid in $\mathbb{R}^d$

Ovidiu Daescu* Ka Yaw Teo†

Abstract

For a set P of n points in $\mathbb{R}^d$, the *medoid* is the point in P with the minimal sum of distances to P. We present two new deterministic algorithms for approximating the medoid of P within a factor of $(1+\varepsilon)$ in time $O(n\varepsilon^{-d}\log n)$ and $O(n\varepsilon^{-d}+n\log n)$, respectively. Our results rely on a quick approximation of the sum of the distances between P and any given point of P. Our algorithms are simple, versatile, and easily implementable.

1 Introduction

In this paper, we consider the following problem:

Given a set P of n points in $\mathbb{R}^d$, locate a point in P that minimizes the sum of the Euclidean distances between P and the located point.

The optimal point for the problem is commonly referred to as the *medoid*. One would encounter the problem of computing the medoid in various contexts such as clustering in data science [19, 21, 25], optimizing facility location in operations research [10, 11, 15, 22], and quantifying centrality in network analysis [6, 7, 16, 17, 26].

Naively, one can find the medoid of P by simply computing all $\binom{n}{2}$ pairwise distances. However, it has been argued that an exact algorithm does not exist for solving the medoid problem in $o(n^2)$ time [23]. Different approaches have thus been developed to compute the medoid in sub-quadratic time either approximately or exactly under statistical assumptions.

Eppstein and Wang [13] proposed a randomized method that takes $O(n\varepsilon^{-2}\log n)$ distance computations to approximate the medoid within an additive error of $\varepsilon\mathcal{D}$ with high probability, where $\mathcal{D}$ is the diameter of P, which may not be known apriori. This result was later improved by Okamoto et al. [24], whose algorithm requires $O(n^{5/3}\log^{4/3} n)$ distance evaluations to return the exact medoid with high probability under certain statistical assumptions on P. Later on, Newling and Fleuret [23] presented an algorithm for finding the true medoid using $O(n^{3/2}2^{\Theta(d)})$ distance computations under certain assumptions on the distribution of the given points near the medoid. Soon after, a sampling-based algorithm was given by Bagaria et al. [4] for computing the exact medoid with high probability, and their algorithm takes a total of $O(n\log n)$ distance evaluations under a distributional assumption on the input points. By exploiting the underlying structure of the problem, Baharav and Tse [5] derived an improvement to Bagaria et al.'s algorithm, obtaining a gain of two to three orders of magnitude in number of distance computations.

Note that all the algorithms aforementioned have been derived in the context of network analysis, where n is the number of nodes in an undirected graph, and the distance metric is the length of the shortest path between nodes. Nonetheless, the algorithms can be effectively applied to any point set under the Euclidean metric, in which case the time complexity of each said algorithm would be equal to its associated number of distance computations multiplied by a factor of d.

In addition, there are randomized algorithms based on coreset techniques [14, 27] capable of addressing the problem considered herein. Specifically, one can compute an ε-coreset of a point set P in $\mathbb{R}^d$, which is a small weighted subset of P, such that for any point $q\in\mathbb{R}^d$, the distance sum $\sum_{p\in P}\|p-q\|$ can be approximated up to a factor of $(1+\varepsilon)$ by using the distances between q and the weighted points in the coreset. A coreset of size $O(d\varepsilon^{-2})$ and $O(\mathrm{poly}(1/\varepsilon))$ can be constructed in time $O(dn+\log^2 n+d\varepsilon\log n)$ and $O(\mathrm{nnz}(A)+(n+d)\mathrm{poly}(1/\varepsilon)+\exp(\mathrm{poly}(1/\varepsilon)))$, respectively, where $\mathrm{nnz}(A)$ is the number of non-zero entries in the $n\times d$ matrix A of the coordinates of P. As a result, one can find a $(1+\varepsilon)$-approximation to the medoid with high probability in $O(n\cdot\mathrm{poly}(1/\varepsilon))$ time.

For a set P of n points in $\mathbb{R}^2$, Har-Peled et al. [20] obtained an exact algorithm that computes the medoid in $O(n\log^2 n\cdot(\log n\log\log n+c_P))$ expected time, where c_P is the size of the largest subset of P in convex position. When the points of P are located uniformly at random on the unit square, c_P is bounded by $\Theta(n^{1/3})$ in expectation [2], and thus the medoid can be computed in $O(n^{4/3}\log^2 n)$ expected time.

2 Our results

Given an $\varepsilon>0$, a point x is said to be a $(1+\varepsilon)$-approximate solution if the sum of the distances from x

*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

to P is at most $(1+\varepsilon)$ times the sum of the distances from the true medoid. Throughout the paper, we assume that d and ε are fixed constants independent of n; nevertheless, we include them in some of the asymptotic results to indicate their dependencies. In addition, we assume, without loss of generality, that P has been scaled so that it is enclosed within a unit hypercube.

We begin in Section 3 by describing an algorithm to compute a $(1+\varepsilon)$-approximate medoid in time $O(n\varepsilon^{-d}\log n)$. This algorithm uses a new data structure named well separated subset decomposition (WSSD), which extends on the classical idea of the well separated pair decomposition (WSPD) by Callahan [9]. WSSD partitions P into $O(\log n)$ clusters that preserve the distances of P to each of the n candidate points.

In Section 4, we encode the pairwise distances between the points of P by directly using WSPD. We can then estimate the medoid within a factor of $(1+\varepsilon)$ in time $O(n\varepsilon^{-d}+n\log n)$, provided that the pairwise distances associated with the well separated pairs are computed and summed in the right order. If $\varepsilon^{-d}=O(\log n)$, our algorithm would run in $O(n\log n)$ time.

To the best of our knowledge, all the previous approximation methods for solving the medoid problem are randomized, making our algorithms the first *deterministic* fully polynomial-time approximation schemes (FPTASs) with a time complexity near-linear in n.

3 $O(n\varepsilon^{-d}\log n)$-time $(1+\varepsilon)$-approximation

We propose an $O(n\varepsilon^{-d}\log n)$-time approximation algorithm involving the following partitioning scheme.

Well separated subset decomposition (WSSD)

Let C denote a subset of P. Define $s>0$ to be a parameter called separation factor. With respect to a candidate point $p\in P$, for some $r\in\mathbb{R}_{\geq 0}$, if the points of C can be enclosed within a Euclidean ball of radius r such that the closest distance from this ball to p is at least sr, then C is said to be s-well separated from p.

Definition 1 (WSSD) *Given a set P of n points, a point p, and a separation factor $s>0$, an s-well separated subset decomposition (s-WSSD) with respect to p is defined as a collection of subsets of P, denoted by $\{C_1,C_2,\ldots,C_k\}$, such that (I) $C_i\subseteq P$ for $1\leq i\leq k$, (II) $C_i\cap C_j=\emptyset$ for $1\leq i,j\leq k$ and $i\neq j$, (III) $\cup_{i=1}^{k}C_i=P$, and (IV) C_i is s-well separated from p for $1\leq i\leq k$.*

An s-WSSD can be constructed from either a kd-tree [8] or a balanced box decomposition (BBD) tree [3]. Both of these data structures are based on a hierarchical subdivision of space into rectilinear regions called cells. The size of a cell is given by the length of its longest side. For a set P of n points in $\mathbb{R}^d$, it is possible to build, in time $O(n\log n)$, an optimized kd-tree [18] or a BBD-tree with height $O(\log n)$ and space $O(n)$. In either tree, each internal node has two children, and each leaf node contains a single point. Unlike a kd-tree, the cells of a BBD-tree have a bounded aspect ratio, and the sizes of the cells decrease by (at least) a factor of $1/2$ with each descent of $2d$ levels in the tree.

Theorem 1 *For a set P of n points and any $s>0$, with respect to a point p, one can construct an s-WSSD of size $O(s^d\log n)$ in time $O(n\log n+s^d\log n)$.*

Proof. We begin by building a kd-tree or a BBD-tree for P. Each leaf node, which contains a single point, is treated as having an infinitesimally small cell containing its point.

The construction of an s-WSSD, with respect to a point p, is based on a recursive process. Throughout the construction, we maintain a collection of sets that satisfy properties (I), (II), and (III) as stated in Definition 1. When the procedure terminates, all the sets generated will fulfill property (IV). Each set of the s-WSSD will be encoded as a node in the kd-tree or BBD-tree.

Let u denote a node in either tree. Consider the smallest Euclidean ball that encloses the cell of node u. If the ball is s-well separated from point p, then we report node u as an s-well separated subset. Otherwise, we apply the procedure recursively to each child node of u. Let $WSSD(u,p,s)$ denote said procedure.

Note that we can determine whether a node u (i.e., its smallest enclosing Euclidean ball) is s-well separated from point p in $O(1)$ time. This requires computing the smallest Euclidean ball enclosing the cell of node u, either at the time of determining the separation between node u and point p or in advance (when creating the tree data structure).

In the procedure $WSSD()$, we divide a node u only if the call $WSSD(u,p,s)$ is non-terminal – that is, node u is not an s-well separated subset. Each non-terminal call generates at most two recursive calls, through which a terminal call may arise. Note that each terminal call produces at most one well separated subset. Thus, the total number of well separated subsets is at most two times the number of non-terminal calls.

To evaluate the number of s-well separated subsets generated in the recursive process, we use a packing argument (to count the number of non-terminal calls), which slightly differs depending on either an optimized kd-tree or a BBD-tree is used as the basis for the construction of the s-WSSD.

kd-tree-based s-WSSD. Each of the nodes at a given level in an optimized kd-tree is associated with (nearly) the same number of points (which is a result of choosing the median as the cutting value). Consider

the nodes at a given level λ of the kd-tree, where the cell associated with each node contains b_λ points. Let V denote the volume of the cell associated with any node at level λ in the kd-tree. As shown by Friedman et al. [18], the expected volume of each such cell is approximately $E[V] = \frac{b_\lambda}{n+1} \cdot \frac{1}{P_\lambda}$, where P_λ is the probability density averaged over the cell (assuming that the probability distribution of the points within the cell is approximately constant). Let α be the size of the cell. Then, the expected size $E[\alpha]$ of the cell is simply the d-th root of the expected volume of the cell – that is, $E[\alpha] = E[V]^{1/d}$. The expected number of nodes at level λ being divided in the procedure must be bounded from above by the expected number of cells at level λ overlapping the ball of radius sr centered at p – that is, $(1 + \lceil 2sr/E[\alpha] \rceil)^d$. Given that $r = E[\alpha]\sqrt{d}/2$, the upper bound becomes $(1 + s\sqrt{d})^d = O(s^d)$. Since there are $O(\log n)$ levels in the kd-tree, the expected number of non-terminal calls to the procedure *WSSD*() is $O(s^d \log n)$. Hence, the expected number of s-well separated subsets is $2 \cdot O(s^d \log n) = O(s^d \log n)$.

BBD-tree-based s-WSSD. For the case of a BBD-tree, we use a similar packing argument as that for a kd-tree. Recall that point set P has been scaled so that it is enclosed within a unit hypercube. As a result, the cells of the BBD-tree have sizes that are powers of 1/2.

For analysis purposes, we congregate the nodes in the BBD-tree into groups according to the sizes of their associated cells. Define size group i to be the set of nodes whose cell size is $1/2^i$. Note that a node and its child may have the same size, and thus we cannot apply the packing argument directly to each size group. Define base group i to be the subset of nodes in size group i that are leaf nodes or whose children belong to the next smaller size group. The cells corresponding to the nodes in a base group are pairwise interior-disjoint. For each base group i, the number of cells overlapping the ball of radius sr centered at p is bounded from above by $\left(1 + \left\lceil 2sr/\left(1/2^i\right)\right\rceil\right)^d$. Since $r = \left(1/2^i\right)\sqrt{d}/2$, the upper bound becomes $O(s^d)$. Note that at most $2d$ levels of ancestors above the nodes in the base group can be in the same size group. In addition, the BBD-tree is $O(\log n)$ in height, which implies that the total number of base groups is bounded by $O(\log n)$. So, the total number of non-terminal calls to *WSSD*() is $O(2d \cdot s^d \log n) = O(s^d \log n)$. As a result, the total number of s-well separated subsets generated with respect to point p is $2 \cdot O(s^d \log n) = O(s^d \log n)$.

In both cases above, the asymptotic upper bound on the number of s-well separated subsets generated is $O(s^d \log n)$, with the distinction that the upper bound applies to the worst case for the BBD-tree, whereas the upper bound is derived with respect to the average (expected) case for the kd-tree. Together with $O(n \log n)$ time to build either tree, the overall running time is $O(n \log n + s^d \log n)$. □

We now describe a technical lemma associated with an s-WSSD, which will be used later in approximating the medoid.

Lemma 2 (WSSD Utility Lemma) *If subset C is s-well separated from point p, and $c, c' \in C$, then we have $\|c' - p\| \leq \left(1 + \frac{2}{s}\right) \|c - p\|$.*

Proof. Due to the triangle inequality, we have $\|c' - p\| \leq \|c - p\| + \|c - c'\|$. Since C is enclosed within a ball of radius r and is s-well separated from p, we have $\|c' - p\| \leq \|c - p\| + 2r = \|c - p\| + \frac{2r}{sr} sr \leq \|c - p\| + \frac{2}{s}\|c - p\| = \left(1 + \frac{2}{s}\right) \|c - p\|$. □

WSSD-based approximation

This section discusses the usage of a WSSD for approximating the medoid of P. We present the arguments only for the WSSD constructed from a BBD-tree, since the analysis is similar for the case of using a kd-tree, aside from that the resulting time complexity would be of the average case instead of the worst case.

Theorem 3 *Given a set P of n points in $\mathbb{R}^d$, for any $\varepsilon > 0$, a $(1 + \varepsilon)$-approximation to the medoid of P can be computed in time $O(n\varepsilon^{-d} \log n)$.*

Proof. First, we build a BBD-tree for P, using which we construct an s-WSSD with respect to each of the n candidate points in P. According to Theorem 1, the total construction time is bounded by $O(n \log n + ns^d \log n) = O(ns^d \log n)$. We make a small augmentation to the construction of the WSSD as follows. When building a BBD-tree, we associate each node u of the tree with a quantity $|u|$ indicating the number of points lying within its cell. When we output a node u as an s-well separated subset with respect to a point p in the decomposition process, we report $|u|$ and the farthest point within the cell of node u from p (which may not necessarily be a point of P). Since the farthest point within a hypercube from p is one of the 2^d vertices of the hypercube, we can find the farthest point in $O(2^d)$ time, which is just $O(1)$ given that d is treated as a constant. Thus, the overall running time for the construction of the WSSD remains the same as before.

Let $\{C_i \mid 1 \leq i \leq k_p\}$ be the collection of s-well separated subsets with respect to a point p. Let $\phi(C_i)$ denote the farthest point within the cell containing C_i from point p, and let $|C_i|$ be the number of points in C_i. With respect to each candidate point $p \in P$, we compute the distance sum $\sum_i |C_i| \cdot \|\phi(C_i) - p\|$, and output the point p achieving the smallest distance sum.

Suppose that the aforementioned approach yields point x as an approximate medoid for P. Let $\{A_i \mid 1 \leq i \leq k_x\}$ be the set of s-well separated subsets with respect to x. Then, by Lemma 2, for each s-well separated subset A_i, we have $|A_i| \cdot \|\phi(A_i) - x\| \leq \sum_{a \in A_i} \left(1 + \frac{2}{s}\right) \|a - x\|$. Since $\phi(A_i)$ is the farthest point within the cell containing A_i from x, by summing over all i, we obtain

$$\sum_{p \in P} \|p - x\| \leq \sum_i |A_i| \cdot \|\phi(A_i) - x\| \leq \left(1 + \frac{2}{s}\right) \sum_{p \in P} \|p - x\|$$

Let m denote the exact medoid of P. Let $\{B_i : 1 \leq i \leq k_m\}$ be the set of s-well separated subsets with respect to m. We then have

$$\begin{aligned}\sum_{p \in P} \|p - m\| \leq \sum_{p \in P} \|p - x\| &\leq \sum_i |A_i| \cdot \|\phi(A_i) - x\| \\ &\leq \sum_i |B_i| \cdot \|\phi(B_i) - m\| \\ &\leq \left(1 + \frac{2}{s}\right) \sum_{p \in P} \|p - m\|\end{aligned}$$

Given any $\varepsilon > 0$, we set $s = 2/\varepsilon$. Then, we obtain

$$\sum_{p \in P} \|p - m\| \leq \sum_{p \in P} \|p - x\| \leq (1 + \varepsilon) \sum_{p \in P} \|p - m\|$$

This implies that the output point x is a $(1 + \varepsilon)$-approximation to the medoid of P. The overall running time is bounded by $O(n(2/\varepsilon)^d \log n) = O(n\varepsilon^{-d} \log n)$. □

4 $O(n\varepsilon^{-d} + n \log n)$-time $(1 + \varepsilon)$-approximation

In this section, we derive an algorithm for computing a $(1 + \varepsilon)$-approximation to the medoid of P in $O(n\varepsilon^{-d} + n \log n)$ time. First, we use a WSPD to represent the distances between the points of P. After obtaining such a representation, we carefully enumerate the pairwise distances in an order such that the sum of the distances from P to each representative point is approximated correctly.

Well separated pair decomposition (WSPD)

A WSPD [9] is formally defined as follows. Let A and B be subsets of P. Define $s > 0$ to be a separation factor. Denote by r the smallest radius of a Euclidean ball such that each of A and B can be enclosed within such a ball. Set A is said to be s-well separated from B if the closest distance between the two balls enclosing A and B is at least sr.

Definition 2 (WSPD) *For a set P of n points and a separation factor $s > 0$, an s-well separated pair decomposition (s-WSPD) is a collection of pairs of subsets of P, denoted as $\{\{A_1, B_1\}, \{A_2, B_2\}, \ldots, \{A_k, B_k\}\}$, such that (I) $A_i, B_i \subseteq P$ for $1 \leq i \leq k$, (II) $A_i \cap B_i = \emptyset$ for $1 \leq i \leq k$, (III) $\cup_{i=1}^{k} A_i \bigotimes B_i = P \bigotimes P$, and (IV) A_i and B_i are s-well separated for $1 \leq i \leq k$.*

When estimating the medoid of P, we will make use of the following utility property of an s-WSPD.

Lemma 4 (WSPD Utility Lemma) *If pair $\{A, B\}$ is s-well separated, $a, a' \in A$, and $b \in B$, then we have $\|a' - b\| \leq \left(1 + \frac{2}{s}\right) \|a - b\|$.*

Proof. The proof is similar to that of Lemma 2 and thus omitted. □

WSPD-based approximation

Theorem 5 *Given a set P of n points in $\mathbb{R}^d$, for any $\varepsilon > 0$, one can compute a $(1 + \varepsilon)$-approximation to the medoid of P in $O(n\varepsilon^{-d} + n \log n)$ time.*

Proof. We begin by building a compressed octree for P. The octree can be built in $O(n \log n)$ time, and is of size $O(n)$ [1, 12]. For simplicity of arguments, we assume that the octree is not compressed but of size $O(n)$. This allows us to assume that nodes of the same level in the octree have the same cell size. By using the octree, we construct a WSPD for P such that each well separated pair of nodes generated are of the same level in the octree. This requires a slight modification to the original algorithm given in [9] for creating a WSPD. Namely, when we fail to separate a pair of nodes u and v, we proceed to recursively separate the 2^d children of u from those of v, thus keeping the invariant that each pair of nodes considered are of the same size. The algorithm is presented as a pseudocode in Figure 1 (where the code in blue is an augmentation necessary for finding an approximate medoid, which will be discussed later). The initial call is $WSPD(u_0, u_0, s, \emptyset)$, where u_0 is the root of the octree.

To evaluate the total number of well separated pairs in the resulting WSPD, it suffices to count the number of terminal calls to $WSPD()$, each of which can generate $\Theta(2^{2d})$ well separated pairs. Since a terminal call may only arise as a call to $WSPD()$ in a non-terminal call, we instead bound the number of calls to $WSPD()$ made by all the non-terminal calls. We claim that, in any non-terminal call, for every node u_i (iterated in the first outer for loops of the algorithm), the number of calls to $WSPD()$ (as in the final for loop in the algorithm) is bounded by $O(s^d)$. Since there are $O(n)$ nodes in the (compressed) octree, the total number of calls to $WSPD()$ is $O(s^d n)$.

We are now left to establish the claim that for any node u_i, the number of calls to $WSPD()$ is bounded by $O(s^d)$. For a node u_i, a call to $WSPD()$ is made only if u_i is not s-well separated from some node v_j. Let α

```
Algorithm WSPD(u, v, s, par)
1.  let u_1, ..., u_α be the children of u;
2.  let v_1, ..., v_β be the children of v;
3.  for i ← 1 to α
4.      do S ← T ← ∅;
5.         newpar ← par;
6.         for j ← 1 to β
7.             do if (u_i or v_j is empty) then ignore (u_i, v_j);
8.                    else if (u_i and v_j are leaves and u_i = v_j) then ignore (u_i, v_j);
9.                           else if (u_i and v_j are s-well separated)
10.                                then add (u_i, v_j) to S;
11.                                else add (u_i, v_j) to T;
12.                if (S ≠ ∅) then newpar ← u_i;
13.                for each (x, y) ∈ S do output (x, y, par);
14.                for each (x, y) ∈ T do call WSPD(x, y, s, newpar);
```

Figure 1: Augmented algorithm for constructing WSPD.

denote the side length of the cells of nodes u_i and v_j. Let r be the radius of the Euclidean ball enclosing each of the cells of nodes u_i and v_j. Note that $r = \alpha\sqrt{d}/2$. Assume that $s \geq 1$; if $0 < s < 1$, we replace s with $\max(s, 1)$. Let c_{u_i} and c_{v_j} be the centers of the balls enclosing the cells of nodes u_i and v_j, respectively. Since u_i is not s-well separated from v_j, the distance between c_{u_i} and c_{v_j} must be at most $2r + sr \leq 3sr$. Let β denote the ball of radius $3sr$ centered at c_{u_i}. The set of nodes v_j that are not s-well separated from u_i must correspond to the cells of side length α overlapping β. Using a similar packing argument as in the proof of Theorem 1, for a node u_i, the number of such nodes v_j is bounded by $O(s^d)$.

Finally, together with $O(n \log n)$ time to build the octree, the overall time for constructing the WSPD is $O(n \log n + s^d n)$.

For the convenience of the ensuing discussion, each well separated pair (u, v) is represented (and produced by the algorithm *WSPD*()) as an ordered pair, where u is referred to as the *anchor set* of the pair.

Augmenting WSPD construction. In the octree used for constructing the WSPD, we associate each node u with i) a representative point $rep(u)$, which may be chosen arbitrarily among the points lying inside the cell of node u, and ii) a quantity $|u|$ indicating the number of points located within the cell of node u. In addition, we output each well separated pair (u, v) along with a set w, if any, where w is the lowest ancestor of u such that w is s-well separated from some node of the same level as w (in the octree). We call w the *parent set* of (u, v) (and of u), and u a *child set* of w. In the pseudocode *WSPD*(), variables *par* and *newpar* (i.e., code in blue) are used for keeping track of the parent set for each well separated pair.

Finding an approximate medoid. Let Γ denote the set of well separated pairs in an s-WSPD for P. Each ordered pair of points $(p_i, q_i) \in P \times P$ (where $p_i \neq q_i$) occurs in a unique well separated pair (A, B) in Γ. As a result, we could simply compute the approximate distance sum for each point $p_i \in P$ using Γ, and return the point with the minimum distance sum as the approximate solution.

To take a closer look at this idea, let e_i denote any anchor set (of any well separated pair in Γ) that has no child anchor set. Recall that $rep(e_i)$ denotes the representative point associated with e_i. Note that a point $p_i \in P$ must belong to a (unique) childless anchor set e_i, for which p_i may or may not be chosen as the representatives point.

Suppose that $p_i = rep(e_i)$. Let $S_i \subseteq \Gamma$ be the subset of well separated pairs, of which e_i is either the anchor set or a child anchor set. We can obtain an approximate distance sum for point $rep(e_i)$ using S_i. Namely, for each pair $(A, B) \in S_i$, we compute $|B| \cdot \|rep(B) - rep(e_i)\|$. We then take the sum over all the pairs in S_i to be the approximate distance sum for p_i. If $p_i \neq rep(e_i)$, then the approximate distance sum for p_i only differs by at most a factor of $\left(1 + \frac{2}{s}\right)$ from that for $rep(e_i)$, according to Lemma 4.

Hence, we only need to compute the distance sum for each representative point associated with a childless anchor set (which will just be referred to as *representative points* for simplicity hereafter). However, observe that, for any two childless anchor sets e_i and e_j, $S_i \cap S_j$ may not be empty. That is, there could be some well separated pair $(A, B) \in S_i \cap S_j$ such that $e_i \subseteq A$ and $e_j \subseteq A$. In other words, e_i and e_j could have some common ancestor anchor set. This implies, by computing the distance sum for each representative point one at a time, that the running time required to find the representative point with the minimum distance sum could

Algorithm *Approx-Medoid*(Γ)
1. $F \leftarrow \emptyset$;
2. **for** each $\mathsf{C}_i \in \mathsf{C}$ (in increasing order)
3. **do for** each $\mathsf{P}_{ij} \in \mathsf{P}_i$
4. **do** let w be the parent set of the well separated pairs in P_{ij};
5. **for** each $\mathsf{A}_{ijk} \in \mathsf{A}_{ij}$
6. **do** let u be the anchor set of the well separated pairs in A_{ijk};
7. compute $\sigma[u] \mathrel{+}= \sum_{(u,v)\in \mathsf{A}_{ijk}} (|v| \cdot \|rep(v) - rep(u)\|)$;
8. **if** $(w = \emptyset)$ **then** add $(rep(u), \sigma[u])$ to F;
9. **if** $(w \neq \emptyset)$
10. **then** let $A = \{u \mid (u, v) \in \mathsf{P}_{ij}\}$;
11. $u_{\min} \leftarrow \arg\min_{u\in A} \sigma[u]$;
12. $\sigma[w] \leftarrow \sigma[u_{\min}]$;
13. $rep(w) \leftarrow rep(u_{\min})$;
14. output $(x, \sigma) \in F$ with the minimum σ;

Figure 2: Algorithm for approximating medoid using WSPD.

be $\Omega(s^d n)$.

As it turns out, we do not have to compute the distance sums for all the representative points. If we compute and sum the distances for each representative point at each level in a bottom-up fashion, allow the representative point with the minimum "partial" distance sum thus far at each level to overtake the others with the same parent anchor set, then we could find the representative point with the minimum distance sum in $O(s^d n)$ time (since each well separated pair is only used once for distance sum computation).

Here are the details of the procedure. We group the well separated pairs in Γ according to the cell sizes of their corresponding node pairs in the octree. Within each cell-size group, we collect the well separated pairs into groups with common parent sets. Within each such parent-set group, we further congregate the well separated pairs according to their anchor sets. Formally, for a given well separated pair (u, v), we denote by $cel(u, v)$ and $par(u, v)$ the cell size and the parent set of (u, v), respectively. Note that if a well separated pair (u, v) has no parent set, then $par(u, v) = \emptyset$. Define:

i) $\mathsf{C} = \{\mathsf{C}_i \subseteq \Gamma \mid \forall (u, v), (x, y) \in \Gamma,\ cel(u, v) = cel(x, y) \implies (u, v) \in \mathsf{C}_i \wedge (x, y) \in \mathsf{C}_i\}$,

ii) $\mathsf{P}_i = \{\mathsf{P}_{ij} \subseteq \mathsf{C}_i \mid \forall (u, v), (x, y) \in \mathsf{C}_i,\ par(u, v) = par(x, y) \implies (u, v) \in \mathsf{P}_{ij} \wedge (x, y) \in \mathsf{P}_{ij}\}$, and

iii) $\mathsf{A}_{ij} = \{\mathsf{A}_{ijk} \subseteq \mathsf{P}_{ij} \mid \forall (u, v), (x, y) \in \mathsf{P}_{ij},\ u = x \implies (u, v) \in \mathsf{A}_{ijk} \wedge (x, y) \in \mathsf{A}_{ijk}\}$.

That is, $\mathsf{C} = \{\mathsf{C}_i \mid i = 1, 2, \ldots\}$ is the partition of set Γ according to cell size, $\mathsf{P}_i = \{\mathsf{P}_{ij} \mid j = 1, 2, \ldots\}$ is the partition of set $\mathsf{C}_i \in \mathsf{C}$ according to parent set, and $\mathsf{A}_{ij} = \{\mathsf{A}_{ijk} \mid k = 1, 2, \ldots\}$ is the partition of set $\mathsf{P}_{ij} \in \mathsf{P}_i$ according to anchor set. Assume, without loss of generality, that for any $\mathsf{C}_i, \mathsf{C}_j \in \mathsf{C}$, if $i < j$, then $cel(u, v) < cel(x, y)$ for all $(u, v) \in \mathsf{C}_i$ and $(x, y) \in \mathsf{C}_j$. For any anchor set u, let $\sigma[u]$ denote the distance sum computed for u. Initially, we set $\sigma[u] = 0$ for all anchor sets u. We then process Γ as described in the pseudocode given in Figure 2.

Briefly, we iterate the well separated pairs by cell size in ascending order. Within each cell-size group $\mathsf{C}_i \in \mathsf{C}$, we update $\sigma[u]$ for each anchor set u sharing the same parent set w by considering its associated well separated pairs (line 6 of the pseudocode). If $w \neq \emptyset$, then we find, among those having the same parent set w, the anchor set $u_{\min}$ with the minimum distance sum after the update. We record the distance sum for $u_{\min}$ as that for its parent set w, and replace the representative point for the parent set w of $u_{\min}$ with that for $u_{\min}$. When the algorithm terminates, of all anchor sets without a parent set, we report the one with the minimum distance sum.

The time complexity of *Approx-Medoid*() is bounded by $O(s^d n)$, given that each well separated pair is processed by a constant number of operations in the procedure. Along with the construction time for WSPD, the overall time for approximating the medoid of P is $O(n \log n + s^d n)$.

Correctness of algorithm. We now proceed to prove the correctness of the algorithm *Approx-Medoid*() to yield a solution within a multiplicative error of ε.

Let m be the exact medoid of P. Let x be the representative point returned as the approximate solution by the algorithm *Approx-Medoid*(). To establish the correctness of the algorithm, we have to show that

$$\sum_{p\in P} \|p - m\| \leq \sum_{p\in P} \|p - x\| \leq (1 + \varepsilon) \sum_{p\in P} \|p - m\|$$

The first inequality holds because no other point in P can have a smaller distance sum than the exact medoid

m. To prove the second inequality, consider the anchor set U such that i) $m \in U \wedge x \in U$ and ii) $m \notin U' \wedge x \notin U'$ for all child sets U' of U.

First, we examine the set of distance computations for the levels above that of U. Let $\Lambda = \{(U_i'', V_i) \mid i = 1, 2, \dots\}$ denote the set of all well separated pairs such that for each pair $(U_i'', V_i) \in \Lambda$, U_i'' is an ancestor set of U. For each well separated pair $(U_i'', V_i) \in \Lambda$, according to Lemma 4, we have $|V_i| \cdot \|rep(V_i) - x\| \leq \left(1 + \frac{2}{s}\right) |V_i| \cdot \|rep(V_i) - m\|$. If we sum over all the pairs in Λ, we then have

$$\sum_i |V_i| \cdot \|rep(V_i) - x\| \leq \left(1 + \frac{2}{s}\right) \sum_i |V_i| \cdot \|rep(V_i) - m\|$$

Secondly, we examine the distance computations for the levels below that of U. For any anchor set C, let $\sigma[C, c]$ denote the distance sum computed for C in the algorithm, where $c \in C$ is the representative point used in computing the distance sum. Let M be the child set of U such that $m \in M$. Similarly, let X denote the child set of U such that $x \in X$. Recall that $M \cap X = \emptyset$.

Let M' be the lowest descendant set of M such that $m \in M'$. Let m' be the representative point associated with M'. If $m' = m$, then the distance sum computed for M' is $\sigma[M', m'] = \sigma[M', m]$. Otherwise, according to Lemma 4, we have $\sigma[M', m'] \leq \left(1 + \frac{2}{s}\right) \sigma[M', m]$.

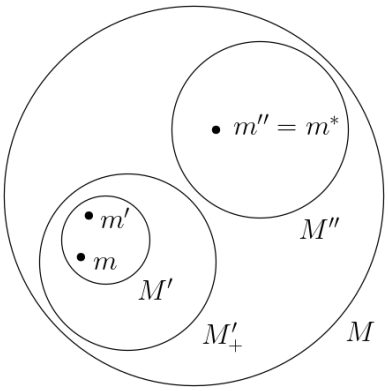


Figure 3: Point $m'' = m^*$ overtakes m' in the distance sum computation during the execution of the algorithm *Approx-Medoid*() such that $\sigma[M, m^*] \leq \sigma[M, m'] \leq \left(1 + \frac{2}{s}\right) \sigma[M, m]$.

Let $m^* \in M$ be the representative point used (in the algorithm) to compute the distance sum for anchor set M. Since x is the representative point produced by the algorithm as the solution, we must have $\sigma[X, x] \leq \sigma[M, m^*]$. If $m^* = m'$, then $\sigma[X, x] \leq \sigma[M, m^*] = \sigma[M, m'] \leq \left(1 + \frac{2}{s}\right) \sigma[M, m]$. Otherwise, at some level between that of M and M', m' must be "overtaken" by some other point m'' such that i) m'' is the representative of some anchor set M'', and ii) $\sigma[M'', m''] \leq \sigma[M'_+, m']$, where M'_+ is an ancestor set of M' and of the same level as M'' (see Figure 3 for an illustration). Clearly, this sort of "overtaking" could happen multiple times as we ascend the levels from that of M' to M in the algorithm. At the end of the ascension, m^* prevails, and we have $\sigma[X, x] \leq \sigma[M, m^*] \leq \sigma[M, m'] \leq \left(1 + \frac{2}{s}\right) \sigma[M, m]$.

As the algorithm terminates, (x, σ) is yielded as the approximate solution, where

$$\sigma = \sum_i |V_i| \cdot \|rep(V_i) - x\| + \sigma[X, x]$$

is the minimum distance sum reported along with point x. By applying Lemma 4, we have

$$\begin{aligned}
&\sum_{p \in P} \|p - x\| \\
&\leq \left(1 + \frac{2}{s}\right) \left(\sum_i |V_i| \cdot \|rep(V_i) - x\| + \sigma[X, x]\right) \\
&\leq \left(1 + \frac{2}{s}\right)^2 \left(\sum_i |V_i| \cdot \|rep(V_i) - m\| + \sigma[M, m]\right) \\
&\leq \left(1 + \frac{2}{s}\right)^3 \sum_{p \in P} \|p - m\| \\
&= \left(1 + \frac{6}{s} + \frac{12}{s^2} + \frac{8}{s^3}\right) \sum_{p \in P} \|p - m\|
\end{aligned}$$

Since $s = \max(s, 1)$, we obtain

$$\sum_{p \in P} \|p - x\| \leq \left(1 + \frac{6}{s} + \frac{20}{s^2}\right) \sum_{p \in P} \|p - m\|$$

Given an $\varepsilon > 0$, if we set $s = \frac{3 + \sqrt{9 + 20\varepsilon}}{\varepsilon}$, then we have

$$\sum_{p \in P} \|p - x\| \leq (1 + \varepsilon) \sum_{p \in P} \|p - m\|$$

□

5 Conclusion

We have presented two deterministic, near-linear time algorithms for approximating the medoid of a point set in fixed dimensions within a factor of $(1 + \varepsilon)$. In the future, we propose to further explore the idea of pair decompositions for solving minsum location-based optimization problems involving more complex geometric objects.

References

[1] S. Aluru. Quadtrees and octrees. In *Handbook of Data Structures and Applications*, pages 309–326. Chapman and Hall/CRC, 2018.

[2] G. Ambrus and I. Bárány. Longest convex chains. *Random Structures & Algorithms*, 35(2):137–162, 2009.

[3] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[4] V. Bagaria, G. Kamath, V. Ntranos, M. Zhang, and D. Tse. Medoids in almost-linear time via multi-armed bandits. In *International Conference on Artificial Intelligence and Statistics*, pages 500–509, 2018.

[5] T. Z. Baharav and D. Tse. Ultra fast medoid identification via correlated sequential halving. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 3655–3664, 2019.

[6] A. Bavelas. Communication patterns in task-oriented groups. *Journal of the Acoustical Society of America*, 22(6):725–730, 1950.

[7] M. A. Beauchamp. An improved index of centrality. *Behavioral Science*, 10(2):161–163, 1965.

[8] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[9] P. B. Callahan. *Dealing with higher dimensions: The well-separated pair decomposition and its applications*. PhD thesis, The Johns Hopkins University, 1995.

[10] Z. Drezner. *Facility location: A survey of applications and methods*. Springer Series in Operations, 1995.

[11] Z. Drezner and H. W. Hamacher. *Facility location: Applications and theory*. Springer Science & Business Media, 2004.

[12] 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.

[13] D. Eppstein and J. Wang. Fast approximation of centrality. *Journal of Graph Algorithms and Applications*, 8(1):39–45, 2004.

[14] D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, pages 569–578, 2011.

[15] R. L. Francis, L. F. McGinnis, and J. A. White. *Facility layout and location: An analytical approach*. Pearson College Division, 1992.

[16] L. C. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1(3):215–239, 1978.

[17] N. E. Friedkin. Theoretical foundations for centrality measures. *American journal of Sociology*, 96(6):1478–1504, 1991.

[18] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, 1977.

[19] J. Han, M. Kamber, and A. K. H. Tung. Spatial clustering methods in data mining: A survey. *Geographic data mining and knowledge discovery*, pages 188–217, 2001.

[20] S. Har-Peled, M. Jones, and S. Rahul. Active-learning a convex body in low dimensions. *Algorithmica*, 83(6):1885–1917, 2021.

[21] L. Kaufman and P. J. Rousseeuw. *Finding groups in data: An introduction to cluster analysis*. John Wiley & Sons, 1990.

[22] P. B. Mirchandani and R. L. Francis. *Discrete location theory*. Wiley-Interscience, 1990.

[23] J. Newling and F. Fleuret. A Sub-Quadratic Exact Medoid Algorithm. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54, pages 185–193, 2017.

[24] K. Okamoto, W. Chen, and X. Y. Li. Ranking of closeness centrality for large-scale social networks. In *International Workshop on Frontiers in Algorithmics*, pages 186–195, 2008.

[25] H. S. Park and C. H. Jun. A simple and fast algorithm for k-medoids clustering. *Expert systems with applications*, 36(2):3336–3341, 2009.

[26] G. Sabidussi. The centrality index of a graph. *Psychometrika*, 31(4):581–603, 1966.

[27] C. Sohler and D. P. Woodruff. Strong coresets for k-median and subspace approximation: Goodbye dimension. In *Proceedings of the 59th Annual Symposium on Foundations of Computer Science*, pages 802–813, 2018.

The Median Line Segment Problem: Computational Complexity and Constrained Variants

Ovidiu Daescu* Ka Yaw Teo†

Abstract

In the *median line segment problem*, we are given a set P of n points in $\mathbb{R}^d$ and a real number $\ell > 0$ with the objective to find a line segment of length ℓ such that the sum of the Euclidean distances from P to the line segment is minimized. We prove that, in general, it is impossible to construct a median line segment for $n \geq 3$ non-collinear points in the plane by using only ruler and compass. We then consider two constrained variants of the median line segment problem in $\mathbb{R}^2$ – i) point-anchored and ii) constant-slope. In the point-anchored variant, an endpoint of the median line segment is given as input, whereas in the constant-slope variant, the orientation of the median line segment is fixed. We present a deterministic $(1+\varepsilon)$-approximation algorithm for solving each constrained variant. For approximating a point-anchored median line segment, we give a space-subdivision method with a time complexity of $O(n\varepsilon^{-2}\alpha_\theta^{-1})$, where α_θ is a parameter dependent on the coordinates of P. For approximating a constant-slope median line segment, a prune-and-search approach is used, and its time complexity is $O(kn \log n)$, where k is inversely proportional to ε.

1 Introduction

The *median line segment problem* is formally defined as follows.

Given a set P of n points in $\mathbb{R}^d$ and a positive real number ℓ, locate a line segment of length ℓ such that the sum of the Euclidean distances from P to the located line segment is minimized.

The problem applies to any real-world scenario that involves finding a best location for any object that could be modeled as a line segment. The problem could arise in many industries and sectors, where we wish to find the optimal placement of various facilities to maximize their efficiency, impact, and profit. These facilities may include highways, railroads, pipelines, telecommunication lines, electronic circuit connectors, and electrodes. In addition to location science, the median line segment problem could have potential applications in other subject areas with less obvious connections such as cluster analysis in data science and pattern recognition in computer vision.

The median line segment problem is closely related to one of the oldest non-trivial problems in facility location theory – the (generalized) *Fermat-Torricelli problem*, which asks to find a point with the minimal sum of distances to a given set of n points. The optimal point is referred to as the Fermat-Torricelli point or simply the (geometric) median. For $n \geq 5$ points in general position, it has been proven that the Fermat-Torricelli point cannot be constructed by strict usage of ruler and compass [1, 7]. In other words, the Fermat-Torricelli problem is unsolvable by radicals over the field of rationals. Consequently, no exact algorithm exists for solving the problem under computational models with basic arithmetic operations and the extraction of k-th roots. This leaves us with only numerical or symbolic approximation methods for $n \geq 5$ points (e.g., see [2, 3, 4]). Furthermore, it remains unclear whether the problem is in $\mathcal{NP}$.

Another problem related to ours is the *median line problem*, which asks to locate a line minimizing the sum of the distances between a given set of n points and the located line. When considering the median line problem in two dimensions, the optimal line has been shown to exhibit the following properties. The median line must divide the given points into two equal halves, and must pass through at least two of the given points [8]. As a consequence, the median line problem could be solved exactly in $O(n^2)$ time (by mainly exploiting the property that the optimal line must contain a pair of given points) [6]. The optimal solution could also be found in $O(h \log n)$ time, where h is the number of halving lines [11, 12]. Currently, the best upper bound for h is $O(n^{4/3})$. However, no exact algorithm is known to solve the median line problem in higher dimensions.

Unlike the Fermat-Torricelli problem and the median line problem, which have been extensively studied over the years, the median line segment problem has not thus far received any proper attention in the literature.

*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

2 Our results

We prove that it is impossible to construct a median line segment for $n \geq 3$ non-collinear points in $\mathbb{R}^2$ by using only ruler and compass (Section 4). We then consider the median line segment problem under different geometric constraints. Particularly, we derive a $(1+\varepsilon)$-approximation algorithm for solving the point-anchored median line segment problem in the plane (Section 5). In this constrained problem, an endpoint of the median line segment is given as part of the input. By essentially dividing the space around the anchor point into $O(n)$ intervals with certain geometric properties, our algorithm finds an approximate solution in $O(n\varepsilon^{-2}\alpha_\theta^{-1})$ time, where α_θ is a parameter dependent on the coordinates of P. Furthermore, we provide an algorithm for computing a $(1+\varepsilon)$-approximate constant-slope median line segment in $\mathbb{R}^2$, where the slope of the median line segment is fixed at input (Section 6). Our algorithm is a tailored extension of the prune-and-search approach given by Bose et al. [2], and its running time is $O(kn \log n)$, where $k = \frac{2\pi}{\cos^{-1}(1+\varepsilon)^{-2}}$.

3 Preliminaries

For any two points a and b in $\mathbb{R}^d$, let ab denote the line segment bounded by a and b, and let $\|ab\| = \|b-a\|$ be the Euclidean distance between a and b.

For any line segment ab in $\mathbb{R}^d$, let H_a (resp. H_b) be the hyperplane containing a (resp. b) and orthogonal to ab. Let S_a (resp. S_b) be the closed half-space bounded by H_a (resp. H_b) and not containing ab. Define $S_{ab} = \mathbb{R}^d \setminus (S_a \cup S_b)$.

For a line segment ab in $\mathbb{R}^2$, let L_{ab} be the line containing ab. Let H^+ denote a closed half-plane bounded by L_{ab}, and let $H^- = \mathbb{R}^2 \setminus H^+$. Define $S_a^+ = S_a \cap H^+$, $S_a^- = S_a \cap H^-$, $S_b^+ = S_b \cap H^+$, $S_b^- = S_b \cap H^-$, $S_{ab}^+ = S_{ab} \cap H^+$, and $S_{ab}^- = S_{ab} \cap H^-$ (Figure 1).

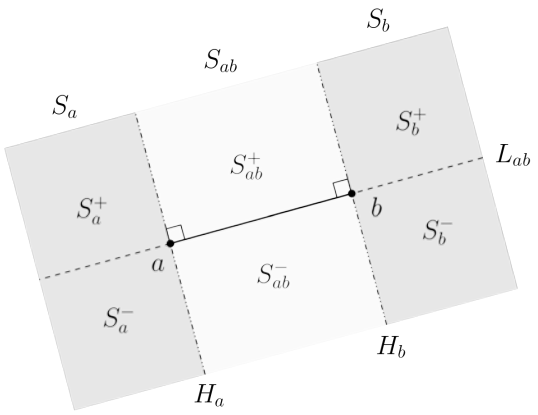


Figure 1: Regions defined with respect to a line segment ab.

We assume, without loss of generality, that the points of P have been uniformly scaled such that the length of the median line segment is $\ell = 1$. Let $\mathcal{D}$ denote the diameter of point set P. Note that if $\ell \geq \mathcal{D}$, then our problem effectively becomes the median line problem. Thus, in this paper, we assume that $\ell < \mathcal{D}$, unless specified otherwise.

A line segment s is said to be a $(1+\varepsilon)$-approximate solution if the sum of the distances from P to s is at most $(1+\varepsilon)$ times that of the optimal line segment.

4 Inconstructibility of the median line segment

Theorem 1 *The construction of a median line segment is, in general, impossible for $n = 3$ and more points in the plane by strict usage of ruler and compass.*

Proof. In order to prove the theorem, we require the following lemma.

Lemma 2 *Let p^* denote the Fermat-Torricelli point for a point set $\{p_1, p_2, p_3\}$. Let $\beta = \arg\max_i \|p^* p_i\|$. For $i \neq \beta$, let η_i be the distance from p_β to the foot of the altitude from p_i in triangle $\triangle p_1 p_2 p_3$.*

A. If $\ell \leq \|p^* p_\beta\|$, then there exists a median line segment $s^* = a^* b^*$ such that its endpoint a^* coincides with p^*, and s^* lies in $p^* p_\beta$ (Figure 2A).

B. If $\ell > \|p^* p_\beta\|$, then there is a median line segment $s^* = a^* b^*$ such that its endpoint a^* coincides with p_β.

(1) $l \leq \min\{\eta_i : i \neq \beta\}$. For $i \neq \beta$, let ϕ_i be the acute angle formed by $b^* p_i$ and the line supporting s^*. Endpoint b^* must be located such that $\phi_i = \phi_j$, where $i, j \neq \beta$ and $i \neq j$ (Figure 2B).

(2) $l > \min\{\eta_i : i \neq \beta\}$. For $i \neq \beta$, let q_i be the closest point on s^* to p_i, and let w_i denote the distance from a^* to q_i. Note that $w_i \in [0,1]$. Let $\bar{d}_i$ denote the vector from q_i to p_i, and let $\bar{h}_i$ be the component of $\bar{d}_i$ normal to s^* multiplied by w_i. For $i, j \neq \beta$ and $i \neq j$, endpoint b^* must be located such that $\|\bar{h}_i\|/\|\bar{d}_i\| = \|\bar{h}_j\|/\|\bar{d}_j\|$.

Proof. We refer to the full paper for the proof. □

Part A of Lemma 2 essentially implies that if $\ell \leq \|p^* p_\beta\|$, then a median line segment s^* can be constructed by using ruler and compass, since the exact Euclidean construction of the Fermat-Torricelli point for $n = 3$ points is possible. However, in part B of Lemma 2 ($\ell > \|p^* p_\beta\|$) – case 1 in particular – in order to construct a median line segment s^*, we have to look

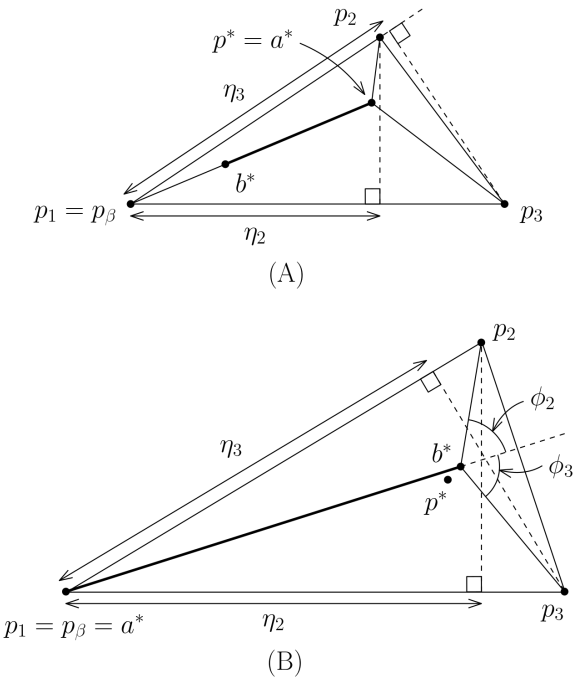


Figure 2: Illustrations for Lemma 2. (A) Part A. (B) Case 1 of part B.

for a point b^* on the circumference of a circle of radius ℓ centered at $a^* = p_\beta$ such that the rays emanating from p_i and p_j, where $i \neq j$ and $i, j \neq \beta$, meeting at b^* make equal angles with the normal at b^*. This is known as (and equivalent to) the Alhazen's billiard problem, to which the general solution has been proven to be inconstructible using only ruler and compass [9]. Briefly, the problem requires solving a quartic equation that is irreducible over $\mathbb{Q}$ (and so does not have constructible solutions). Hence, we conclude that the ruler-and-compass construction of a median line segment is, in general, impossible for $n = 3$ (and more) points. □

5 Approximating the point-anchored median line segment

In this section, we consider the following restricted variant of the median line segment problem.

Given a set P of n points in $\mathbb{R}^2$, a point $q \in \mathbb{R}^2$, and a real number $\ell > 0$, find a line segment of length ℓ with an endpoint at q such that the sum of the Euclidean distances from P to the line segment is minimized.

Remark 1 *It follows from the proof of Theorem 1 that the point-anchored median line segment problem is, in general, not solvable by radicals over $\mathbb{Q}$ for $n \geq 2$ points.*

Theorem 3 *For the point-anchored median line segment problem in $\mathbb{R}^2$, given any $\varepsilon > 0$, one can compute a $(1+\varepsilon)$-approximate solution in time $O(n\varepsilon^{-2}\alpha_\theta^{-1})$, where α_θ is a function dependent on the coordinates of P.*

Proof. Let s denote any line segment of length ℓ with an endpoint fixed at q. Assume, without loss of generality, that the fixed endpoint of line segment s is $a = q = (0, 0)$ (by a translation of P), and the length of line segment s is $\ell = 1$ (through a uniform scaling of P). Let θ be the counterclockwise angle of line segment s with respect to the positive x-axis rooted at a. The sum of the distances from $P = \{p_1, p_2, ..., p_n\}$ to line segment s is given by the following objective function:

$$\begin{aligned} f(\theta) = & \sum_{\substack{1 \leq i \leq n \\ p_i \in S_a}} \sqrt{{x_i}^2 + {y_i}^2} \\ & + \sum_{\substack{1 \leq i \leq n \\ p_i \in S_{ab}^+}} [x_i(-\sin\theta) + y_i \cos\theta] \\ & + \sum_{\substack{1 \leq i \leq n \\ p_i \in S_{ab}^-}} [-x_i(-\sin\theta) - y_i \cos\theta] \\ & + \sum_{\substack{1 \leq i \leq n \\ p_i \in S_b}} \sqrt{(x_i - \cos\theta)^2 + (y_i - \sin\theta)^2} \end{aligned}$$

where x_i and y_i are the x- and y-coordinates of $p_i \in P$, respectively. We consider $\theta \in [0, \pi/2)$ only, and each subsequent quadrant can be handled analogously. The quadrant $[0, \pi/2)$ can be divided into a set T of at most $\Theta(n)$ contiguous intervals, in each of which the subsets of points of P in S_a, S_{ab}^+, S_{ab}^-, S_b^+, and S_b^-, respectively, remain constant. We partition each interval of T into a number of small sub-intervals such that the relative error in computing the sum of the distances from P to a line segment s, whose angle θ is given by a boundary of a sub-interval, does not exceed ε.

To evaluate the number of sub-intervals, we perform the following analysis. Let I denote a sub-interval. Suppose that the optimal line segment s^* lies within I. First, we note that the distance from any given point $p_i \in S_a$ to endpoint a of line segment s remains constant within sub-interval I. For simplicity of notation, the subscript i is dropped, and p is equivalent to p_i hereafter.

For a point $p \in S_{ab}^+$, let $d_p = d(p, s)$ denote its orthogonal distance to a line segment s whose location is defined by a boundary of interval I (Figure 3A). Suppose that $d_p^* = d(p, s^*)$ is the distance from p to the optimal line segment s^*. We rotate the coordinate system such that the positive x-axis contains s, and the first quadrant of the defined xy-plane contains sub-interval I (and thus s^*). Specifically, consider the worst-case scenario where s and s^* are located at the two ends of sub-interval I. Let $\Delta\theta$ be the size of sub-interval I. In addition, let x_p and y_p denote the x- and y-coordinates,

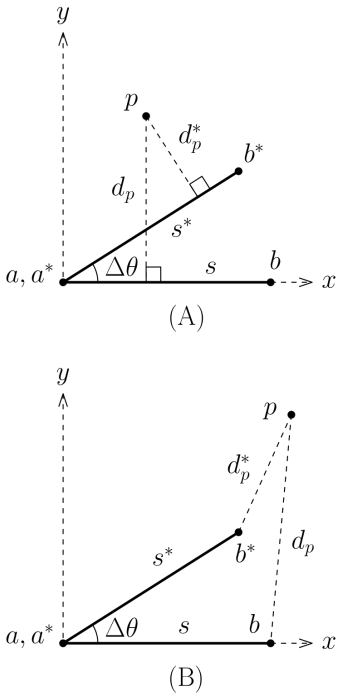


Figure 3: A point $p \in P$ located in (A) S_{ab}^+ or (B) S_b^+.

respectively, of point p. In order to have $d_p \leq (1+\varepsilon)d_p^*$, the following must hold:

$$\begin{aligned} d_p &\leq (1+\varepsilon)\, d_p^* \\ y_p &\leq (1+\varepsilon)\,(-x_p \sin\Delta\theta + y_p \cos\Delta\theta) \\ \frac{1}{1+\varepsilon} &\leq -\frac{x_p}{y_p}\sin\Delta\theta + \cos\Delta\theta \\ &= \sqrt{1+\left(\frac{x_p}{y_p}\right)^2}\cos\left(\Delta\theta + \tan^{-1}\frac{x_p}{y_p}\right) \\ \Delta\theta &\leq \cos^{-1}\left(\frac{1}{(1+\varepsilon)\sqrt{1+\left(\frac{x_p}{y_p}\right)^2}}\right) - \tan^{-1}\frac{x_p}{y_p} \end{aligned}$$

Let $A_{ab,p}^+$ denote the right-hand term of the last inequality above. Given that

$$\begin{aligned} A_{ab,p}^+ &\geq \varepsilon\left(\cos^{-1}\left(\frac{1}{2\sqrt{1+\left(\frac{x_p}{y_p}\right)^2}}\right) - \tan^{-1}\frac{x_p}{y_p}\right) \\ &= \varepsilon\alpha_{ab,p}^+ \end{aligned}$$

for $0 < \varepsilon < 1$, if we have $\Delta\theta = \varepsilon\alpha_{ab,p}^+$, then the desired condition $d_p \leq (1+\varepsilon)d_p^*$ is fulfilled. Note that $\alpha_{ab,p}^+$ is a trigonometric function in terms of the coordinates of point p. We can satisfy $d_p \leq (1+\varepsilon)d_p^*$ for all points $p \in S_{ab}^+$ if we set $\Delta\theta = \varepsilon \cdot \min\{\alpha_{ab,p}^+ : p \in {S_{ab}}^+\}$.

The analysis for S_{ab}^- is similar to that for S_{ab}^+ due to symmetry, and we obtain $\{\alpha_{ab,p}^- : p \in S_{ab}^-\}$ accordingly.

We can also perform a similar analysis for each point $p \in S_b^+$. Let $d_p = d(p,s)$ denote the distance from p to endpoint b of a line segment s located at a boundary of sub-interval I (Figure 3B). Let $d_p^* = d(p,s^*)$ be the shortest distance from p to the optimal line segment s^*. As before, we define a coordinate system on s such that the positive x-axis contains s, and the first quadrant of the xy-plane contains sub-interval I, at whose boundaries s and s^* are positioned. Let $\Delta\theta$ be the size of sub-interval I. If $d_p \leq (1+\varepsilon)d_p^*$, then we have

$$\begin{aligned} & d_p \leq (1+\varepsilon)\, d_p^* \\ & \sqrt{(x_p-1)^2 + {y_p}^2} \\ &\qquad \leq (1+\varepsilon)\sqrt{(x_p - \cos\Delta\theta)^2 + (y_p - \sin\Delta\theta)^2} \\ & \frac{(x_p-1)^2 + {y_p}^2}{(1+\varepsilon)^2} \\ &\qquad \leq (x_p - \cos\Delta\theta)^2 + (y_p - \sin\Delta\theta)^2 \\ &\qquad = {x_p}^2 - 2x_p\cos\Delta\theta + {y_p}^2 - 2y_p\sin\Delta\theta + 1 \\ & -\frac{1}{2}\left(\frac{(x_p-1)^2 + {y_p}^2}{(1+\varepsilon)^2} - {x_p}^2 - {y_p}^2 - 1\right) \\ &\qquad \geq x_p\cos\Delta\theta + y_p\sin\Delta\theta \\ &\qquad = \sqrt{{x_p}^2 + {y_p}^2}\cos\left(\Delta\theta + \tan^{-1}\left(-\frac{x_p}{y_p}\right)\right) \\ & \Delta\theta \leq \tan^{-1}\left(\frac{x_p}{y_p}\right) - \cos^{-1}\left(-\frac{1}{2\sqrt{{x_p}^2 + {y_p}^2}}\right. \\ &\qquad \left.\left(\frac{(x_p-1)^2 + {y_p}^2}{(1+\varepsilon)^2} - {x_p}^2 - {y_p}^2 - 1\right)\right) \end{aligned}$$

Let $A_{b,p}^+$ denote the right-hand side of the last inequality above. Since

$$\begin{aligned} A_{b,p}^+ &\geq \varepsilon^2\left[\tan^{-1}\left(\frac{x_p}{y_p}\right) - \cos^{-1}\left(-\frac{1}{2\sqrt{{x_p}^2 + {y_p}^2}}\right.\right. \\ &\qquad \left.\left.\left(\frac{(x_p-1)^2 + {y_p}^2}{(1+\varepsilon')^2} - {x_p}^2 - {y_p}^2 - 1\right)\right)\right] \\ &= \varepsilon^2\alpha_{b,p}^+ \end{aligned}$$

where $\varepsilon' = \min(1, \varepsilon_p)$,

$$\varepsilon_p = \frac{\sqrt{(x_p-1)^2 + {y_p}^2}}{\sqrt{\left(x_p - \frac{x_p}{\sqrt{{x_p}^2+{y_p}^2}}\right)^2 + \left(y_p - \frac{y_p}{\sqrt{{x_p}^2+{y_p}^2}}\right)^2}} - 1$$

and $0 < \varepsilon \leq \varepsilon' < 1$, if we set $\Delta\theta = \varepsilon^2\alpha_{b,p}^+$, then $d_p \leq (1+\varepsilon)d_p^*$ is satisfied. Note that $\alpha_{b,p}^+$ is a trigonometric function dependent on the coordinates of point p. In

order to uphold $d_p \leq (1+\varepsilon)d_p^*$ for all points $p \in S_b^+$, we can simply set $\Delta\theta = \varepsilon^2 \cdot \min\{\alpha_{b,p}^+ : p \in S_b^+\}$.

Points $p \in S_b^-$ can be handled analogously as those in S_b^+, and we obtain $\{\alpha_{b,p}^- : p \in S_b^-\}$ as the result.

In summary, for each given interval $\tau \in T$, we compute $\alpha_{ab}^+ = \min\{\alpha_{ab,p}^+ : p \in S_{ab}^+\}$, $\alpha_{ab}^- = \min\{\alpha_{ab,p}^- : p \in S_{ab}^-\}$, $\alpha_b^+ = \min\{\alpha_{b,p}^+ : p \in S_b^+\}$, and $\alpha_b^- = \min\{\alpha_{b,p}^- : p \in S_b^-\}$. We then use $\Delta\theta = \min\left\{\varepsilon\alpha_{ab}^+, \varepsilon\alpha_{ab}^-, \varepsilon^2\alpha_b^+, \varepsilon^2\alpha_b^-\right\}$ for partitioning the given interval τ into sub-intervals of size at most $\Delta\theta$.

We now derive an upper bound on the number of sub-intervals as follows. Let $s(\tau)$ denote the set $\{\alpha_{ab}^+, \alpha_{ab}^-, \alpha_b^+, \alpha_b^-\}$ computed for each interval τ of T. Define $\alpha_\theta = \min\{\alpha \in s(\tau) : \tau \in T\}$. Then, we have a total of $2\pi/(\varepsilon^2\alpha_\theta)$ sub-intervals in the worst case. Since it takes $O(n)$ algebraic operations to compute the sum of distances for each candidate line segment (defined by the boundaries of the sub-intervals), we can obtain a solution, whose sum of distances to P is at most $(1+\varepsilon)$ times that of the optimal solution, in $2\pi n/(\varepsilon^2\alpha_\theta) = O(n\varepsilon^{-2}\alpha_\theta^{-1})$ time. □

6 Approximating the constant-slope median line segment

In this section, we address a constrained variant of the median line segment problem stated as follows.

Given a set P of n points in $\mathbb{R}^2$, an angle θ, and a real number $\ell > 0$, find a line segment of length ℓ making angle θ with the abscissa axis such that the sum of the Euclidean distances from P to the line segment is minimized.

Theorem 4 *For the constant-slope median line segment problem in $\mathbb{R}^2$, given any $\varepsilon > 0$, one can find a line segment, whose sum of distances to P is at most $(1+\varepsilon)$ times that of the optimal line segment, in time $O(kn\log n)$, where $k = \frac{2\pi}{\cos^{-1}(1+\varepsilon)^{-2}}$.*

Proof. We denote by $s = ab$ any line segment of length ℓ making angle θ with the positive x-axis. Assume, without loss of generality, that $\theta = 0$ and $\ell = 1$. Let x_a and y_a be the x- and y-coordinates of the endpoint a of line segment s, respectively. Then, the sum of the distances from $P = \{p_1, p_2, ..., p_n\}$ to line segment s can be written as the following objective function:

$$\begin{aligned} f(s) &= f(x_a, y_a) \\ &= \sum_{\substack{1\leq i\leq n \\ p_i\in S_a}} \sqrt{(x_i - x_a)^2 + (y_i - y_a)^2} \\ &\quad + \sum_{\substack{1\leq i\leq n \\ p_i\in S_{ab}^+}} (y_i - y_a) + \sum_{\substack{1\leq i\leq n \\ p_i\in S_{ab}^-}} (y_a - y_i) \\ &\quad + \sum_{\substack{1\leq i\leq n \\ p_i\in S_b}} \sqrt{(x_i - x_a - 1)^2 + (y_i - y_a)^2} \end{aligned}$$

where x_i and y_i are the x- and y-coordinates of $p_i \in P$, respectively.

Remark 2 *f is a piecewise convex function, where each piece consists of a sum of two convex functions and two linear functions, and the transition between any two consecutive pieces corresponds to a point of P moving between S_a, S_{ab}^+, S_{ab}^-, and S_b. Since the number of such transitions is bounded by $\Theta(n^2)$, the minimum of function f can be obtained by solving $\Theta(n^2)$ two-variable convex optimization problems.*

We begin by defining the so-called k-oriented distance function d_k [5, 10] to approximate the Euclidean distance as follows.

***k*-oriented distance.** A cone in $\mathbb{R}^2$ is defined as the intersection of two half-planes, each of whose supporting lines contains the origin O. A simplicial cone c has a diameter bounded by an angle γ if, for any two points p and q in c, we have $\angle pOq \leq \gamma$. Let $C = \{c_1, ..., c_k\}$ be a set of k cones, each of which has a diameter bounded by γ, and C forms a partition of $\mathbb{R}^2$. Note that k is a function of γ. Thus, C could be a set of cones defined by the rays originating at O making angles $\{(i-1)2\pi/k : 1 \leq i \leq k\}$ with respect to the abscissa axis. The two rays that bound a cone c are called the axes of c. For a point $p \in \mathbb{R}^2$, let $t_i(p)$ denote p represented in the coordinate system whose axes are those of c_i. For a point p in a cone c_i, $d_k(O,p) = \|t_i(p)\|$ is called the k-oriented distance from O to p, and is defined as the length of the shortest path from O to p traveling only in the directions parallel to the axes of c_i. For any two points p and q in c_i, we have $d_k(p,q) = d_k(O, q-p)$. Notice that, if $\gamma = \pi/2$, then the corresponding d_k is known as the rectilinear (Manhattan) distance function. For any two points $p, q \in \mathbb{R}^2$, $\|pq\| \leq d_k(p,q) \leq (1+\varepsilon)\|pq\|$, where ε is a positive constant that decreases as k increases.

We now derive an explicit expression for k in terms of ε. Assume, without loss of generality, that point p is located at the origin O (i.e., $p = O$). Let ρ_1 and ρ_2 be the two rays originating at O and defining the cone that contains point q. Recall that the cone has a diameter bounded by angle γ. Consider the case that γ is less than $\pi/2$. Define m to be the line with the same slope as ray ρ_1 and passing through q. Let r be the intersection of m and ρ_2. Note that $d_k(p,q) = \|pr\| + \|rq\|$. Furthermore, according to the law of cosines, we have

$$\|pr\|^2 + \|rq\|^2 - 2\|pr\|\|rq\|\cos(\pi - \gamma) = \|pq\|^2$$
$$\|pr\|^2 + \|rq\|^2 + 2\|pr\|\|rq\|\cos\gamma = \|pq\|^2$$

Given that $0 < \gamma < \pi/2$,

$$\begin{aligned}\left(\|pr\|^2+\|rq\|^2+2\|pr\|\|rq\|\right)\cos\gamma &\leq \|pq\|^2\\ \left(\|pr\|+\|rq\|\right)^2 &\leq \frac{\|pq\|^2}{\cos\gamma}\\ \|pr\|+\|rq\| &\leq \frac{\|pq\|}{\sqrt{\cos\gamma}}\end{aligned}$$

Thus, we have $d_k(p,q) \leq (1+\varepsilon)\|pq\|$, where $\varepsilon = \frac{1}{\sqrt{\cos\gamma}} - 1$. Since $\gamma = 2\pi/k$, we obtain $k = \frac{2\pi}{\cos^{-1}(1+\varepsilon)^{-2}}$ for $0 < \varepsilon < 1$.

Recall that the objective function $f(s)$ denotes the sum of the Euclidean distances from P to s. We can approximate $f(s)$ using

$$\begin{aligned}f_k(s) = &\sum_{\substack{1\leq i\leq n\\ p_i\in S_a}} d_k(p_i,a) + \sum_{\substack{1\leq i\leq n\\ p_i\in S_b}} d_k(p_i,b)\\ &+ \sum_{\substack{1\leq i\leq n\\ p_i\in S^+_{ab}}} y_i - y_a + \sum_{\substack{1\leq i\leq n\\ p_i\in S^-_{ab}}} y_a - y_i\end{aligned}$$

Observe that function $f_k(s)$ is convex and piecewise linear. Hence, we can find the minimum of $f_k(s)$ using the prune-and-search approach described by Bose et al. [2] after some careful modifications.

Prune and search. Consider the set of cones C used in evaluating d_k. Recall that each cone $c \in C$ is defined by two lines. Let L be the set of lines defining C. For each point $p \in P$, we create a point at a distance ℓ to the right of p. Let P' denote the newly created set of points. For each point $p \in P \cup P'$, we construct a copy of L such that each of the lines in L passes through p. The result is an arrangement of lines A. Observe that each cell of A corresponds to a linear piece of the surface f_k. Consequently, f_k reaches a minimum when the endpoint a of line segment s coincides with a vertex of A.

We now describe a prune-and-search algorithm to find the lowest point on the surface f_k. Note that A consists of k sets of parallel lines. Let H_i denote a given set of parallel lines in A, where $1 \leq i \leq k$. We begin by finding a median line $h \in H_i$ that divides H_i into two nearly equal sets. Line h partitions $\mathbb{R}^2$ into two half-planes, h_1 and h_2, one of which contains a minimum of f_k. Suppose, without loss of generality, that h_1 contains the minimum. Then, we can simply ignore all the lines in h_2, and continue to recurse on h_1. This recursive process takes $O(\log n)$ rounds for each set H_i.

In each aforesaid round, we first find a point p_h on h that minimizes f_k. We can then, based on p_h, determine if the minimum lies in h_1 or h_2.

The problem of finding p_h is a one-dimensional instance of our problem (i.e., constrained to line h). Since f_k is piecewise linear, p_h lies on an intersection of h with some other line in $H = \{H_1, ..., H_k\} \setminus h$. Hence, we i) compute all the intersections of h with H, ii) find the median intersection point q_m and the two intersection points q_1 and q_2 that are adjacent to q_m on h, and iii) determine if p_h lies to the left of q_m, right of q_m, or is q_m by evaluating $f_k(q_m)$, $f_k(q_1)$, and $f_k(q_2)$.

Let u be the size of H. The time complexity of finding p_h is given by the recurrence relation $T(u) = T(u/2) + O(u + Q(n))$, where $Q(n)$ denotes the query time to evaluate f_k. This recurrence solves to $O(u+Q(n)\log u)$.

After finding p_h, we determine whether the minimum lies in h_1 or h_2 as follows. Consider two opposite rays r_1 and r_2, which are i) originating at p_h, ii) orthogonal to h, and iii) contained in h_1 and h_2, respectively. We identify the first lines h_{r_1} and h_{r_2} intersected by r_1 and r_2, respectively. Let v_1 (resp. v_2) be the intersection point of h_{r_1} and r_1 (resp. h_{r_2} and r_2). There are three possible cases to be considered: (1) If $f_k(v_1) \leq f_k(p_h) \leq f_k(v_2)$, then a minimum of f_k lies in h_1. (2) If $f_k(v_1) \geq f_k(p_h) \geq f_k(v_2)$, then a minimum of f_k lies in h_2. (3) If $f_k(v_1) > f_k(p_h)$ and $f_k(v_2) > f_k(p_h)$, then p_h is a minimum of f_k. Verifying these cases require the computation of all the intersections of H with r_1 and r_2, and the evaluation of f_k at v_1 and v_2. So, the time complexity of determining whether a minimum lies in h_1 or h_2 is $O(u + Q(n))$.

Observe that $u = O(kn)$. Thus, the time taken by the recursive procedure for each set H_i is given by the recurrence relation $T(n) = T(kn/2) + O(kn + Q(n)\log kn)$, which solves to $O(kn + Q(n)\log kn)$. Given that we have k sets H_i, the overall time taken by the prune-and-search algorithm to compute the point that minimizes f_k is $O(P(n) + k(kn + Q(n)\log kn))$, where $P(n)$ is the preprocessing time to construct the data structure for evaluating f_k, and $Q(n)$ is the query time to evaluate f_k.

We claim that a data structure with a preprocessing time $P(n) = O(kn\log n)$ exists for evaluating f_k in query time $Q(n) = O(k\log n)$ (refer to the full paper for details). As a result, the overall running time of our algorithm is $O(kn\log n)$. □

Remark 3 *Alternatively, the space-subdivision procedure previously used in approximating a point-anchored median line segment could be extended to address the constant-slope variant. The resulting $(1+\varepsilon)$-approximation algorithm would have a time complexity of $O(n^2 + n\varepsilon^{-4}\alpha_{xy})$, where α_{xy} is a function dependent on the coordinates of P.*

7 Conclusion

We have proven that a median line segment is not constructible for $n \geq 3$ non-collinear points in the plane by using only ruler and compass. We have presented a $(1+\varepsilon)$-approximation algorithm for solving the constrained median line segment problem in $\mathbb{R}^2$ where an endpoint or the slope of the median line segment is given at input. These algorithms are space-subdivision and prune-and-search approaches, and their time complexities are near-linear in n. At last, we leave open the question of whether our approximation algorithms for solving the constrained variants can be extended to obtain a $(1+\varepsilon)$-approximate solution to the unconstrained median line segment problem.

References

[1] C. Bajaj. The algebraic degree of geometric optimization problems. *Discrete & Computational Geometry*, 3(2):177–191, 1988.

[2] P. Bose, A. Maheshwari, and P. Morin. Fast approximations for sums of distances, clustering and the Fermat-Weber problem. *Computational Geometry*, 24(3):135–146, 2003.

[3] R. Chandrasekaran and A. Tamir. Algebraic optimization: the Fermat-Weber location problem. *Mathematical Programming*, 46(1):219–224, 1990.

[4] M. B. Cohen, Y. T. Lee, G. Miller, J. Pachocki, and A. Sidford. Geometric median in nearly linear time. In *Proceedings of the 48th Annual ACM Symposium on Theory of Computing*, pages 9–21, 2016.

[5] J. M. Keil. Approximating the complete Euclidean graph. In *Scandinavian Workshop on Algorithm Theory*, pages 208–213, 1988.

[6] N. M. Korneenko and H. Martini. Approximating finite weighted point sets by hyperplanes. In *Scandinavian Workshop on Algorithm Theory*, pages 276–286, 1990.

[7] S. Mehlhos. Simple counter examples for the unsolvability of the Fermat- and Steiner-Weber-problem by compass and ruler. *Beiträge zur Algebra und Geometrie*, 41(1):151–158, 2000.

[8] J. G. Morris and J. P. Norback. A simple approach to linear facility location. *Transportation Science*, 14(1):1–8, 1980.

[9] P. M. Neumann. Reflections on reflection in a spherical mirror. *The American Mathematical Monthly*, 105(6):523–528, 1998.

[10] J. Ruppert and R. Seidel. Approximating the d-dimensional complete Euclidean graph. In *Proceedings of the 3rd Canadian Conference on Computational Geometry*, pages 207–210, 1991.

[11] R. Schieweck and A. Schöbel. Properties and algorithms for line location with extensions. In *Proceedings of the 28th European Workshop on Computational Geometry*, pages 185–188, 2012.

[12] P. Yamamoto, K. Kato, K. Imai, and H. Imai. Algorithms for vertical and orthogonal L_1 linear approximation of points. In *Proceedings of the 4th Annual Symposium on Computational Geometry*, pages 352–361, 1988.

Computational Complexity of Flattening Fixed-Angle Orthogonal Chains

Erik D. Demaine* Hiro Ito† Jayson Lynch‡ Ryuhei Uehara§

Abstract

Planar/flat configurations of fixed-angle chains and trees are well studied in the context of polymer science, molecular biology, and puzzles. In this paper, we focus on a simple type of fixed-angle linkage: every edge has unit length (equilateral), and each joint has a fixed angle of 90° (orthogonal) or 180° (straight). When the linkage forms a path (open chain), it always has a planar configuration, namely the zig-zag which alternating the 90° angles between left and right turns. But when the linkage forms a cycle (closed chain), or is forced to lie in a box of fixed size, we prove that the flattening problem — deciding whether there is a planar noncrossing configuration — is strongly NP-complete.

Back to open chains, we turn to the Hydrophobic–Hydrophilic (HP) model of protein folding, where each vertex is labeled H or P, and the goal is to find a folding that maximizes the number of H–H adjacencies. In the well-studied HP model, the joint angles are not fixed. We introduce and analyze the fixed-angle HP model, which is motivated by real-world proteins. We prove strong NP-completeness of finding a planar noncrossing configuration of a fixed-angle orthogonal equilateral open chain with the most H–H adjacencies, even if the chain has only two H vertices. (Effectively, this lets us force the chain to be closed.)

1 Introduction

In this paper, we introduce and investigate a new model of protein folding. We are given an ***equilateral fixed-angle chain*** ("protein"), where each vertex is marked H or P and has a specified fixed angle, and edges all have unit length. The goal is to embed the chain into a given grid (e.g., 2D square, 3D cube, 2D triangular, or 2D hexagonal) while

1. respecting the fixed angles (but each angle is still free to be a left or right turn in 2D or spin in 3D);
2. without self-crossing in the embedding; and
3. maximizing the number of H–H grid adjacencies.

This is a fixed-angle version of the well-studied HP model of protein folding (where the angles are normally free to take on any value), which is known to be NP-hard in the 2D square grid [4] and 3D cube grid [3]. Fixed angles are motivated by real-world proteins; see [7, Chapters 8–9]. In the 2D square grid or 3D cube grid studied here, we can restrict to ***orthogonal*** fixed-angle chains where all fixed angles are 90° or 180°. For example, the popular "Tangle" toy restricts further to all fixed angles being 90°; see [5].

In the 3D cube grid, NP-hardness of fixed-angle HP protein folding follows from [1] which proves NP-hardness of embedding a fixed-angle orthogonal equilateral chain of n^3 vertices into an $n \times n \times n$ 3D cube grid. If we make all vertices Hs, then a cube embedding is the best way to maximize H–H adjacencies, as the cube uniquely minimizes surface area where potential adjacencies are lost.

In this paper, we prove that the fixed-angle HP protein folding problem is NP-hard in the 2D square grid, even if the chain has only two H vertices and those vertices are its endpoints. In other words, given a fixed-angle orthogonal equilateral HP chain, we prove it is strongly NP-hard to find any planar noncrossing embedding where the endpoints (the two H vertices) are adjacent. This result is tight in the sense that any fixed-angle orthogonal equilateral chain with fewer than two H vertices (and hence can have no H–H adjacencies) has a noncrossing embedding, given by zig-zagging the 90° angles to alternate between left and right turns.

Fixed-angle HP protein folding where only the two endpoints are H vertices is nearly equivalent to finding any planar noncrossing embedding of a ***closed*** fixed-angle chain (where the first and last vertex are identified, and vertices are no longer marked H or P). This is called the ***flattening problem*** for fixed-angle closed chains. The only difference is that, in the flattening problem, the first/last vertex has a fixed-angle constraint, whereas in the HP model, the two necessarily adjacent H vertices could form any angle.

Nonetheless, we show that the flattening problem for fixed-angle orthogonal equilateral closed chains is strongly NP-complete. Past work proved strong NP-hardness when this problem was generalized to fixed-angle orthogonal equilateral caterpillar tree (instead of a chain) or when we allow nonorthogonal fixed angles

*Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, edemaine@mit.edu

†School of Informatics and Engineering, The University of Electro-Communications, itohiro@uec.ac.jp

‡Cheriton School of Computer Science, University of Waterloo, jayson.lynch@uwaterloo.ca

§School of Information Science, JAIST, uehara@jaist.ac.jp

(and working off-grid) [6], but left this case open.

Finally our work also addresses two open problems from [1]. We solve one open problem by proving strong NP-completeness of deciding whether a given fixed-angle orthogonal equilateral chain can be packed into a 2D square (whereas [1] proved an analogous result for a 3D cube). We also prove that this problem remains NP-complete when the chain is only a constant factor longer than the side length of the square (and thus the square is sparsely filled), answering the 2D analog of a 3D question from [1].

2 Preliminaries

A ***linkage*** consists of a ***structure graph*** $G = (V, E)$ and edge-length function $\ell : E \to \mathbb{R}^+$. A ***configuration*** of a linkage in 2D is a mapping $C : V \to \mathbb{R}^2$ satisfying the constraint $\ell(u, v) = \|C(u) - C(v)\|$ for each edge $\{u, v\} \in E$. Let $x(C(u))$ and $y(C(u))$ be the x- and y-coordinate of $C(u)$, respectively. A configuration is ***noncrossing*** if any two edges $e_1, e_2 \in E$ intersect only at a shared vertex $v \in e_1 \cap e_2$.

A linkage is ***equilateral*** if $\ell(e) = 1$ for every $e \in E$. A linkage with n vertices is an ***open chain*** if its structure graph G is a path $(v_0, v_1, \ldots, v_{n-1})$, and it is a ***closed chain*** if G is a cycle $(v_0, v_1, \ldots, v_{n-1}, v_n = v_0)$. A ***fixed-angle chain*** is a chain together with an angle function $\theta : V \to [0°, 180°]$, constraining configurations to have an angle of $\theta(v)$ at every vertex v, except for the two endpoints of an open chain. For notational convenience, we define $\theta(v_0) = \theta(v_{n-1}) = 180°$ for an open chain. A fixed-angle chain is ***orthogonal*** if we have $\theta(v_i) \in \{90°, 180°\}$ for every vertex v_i.

The ***embedding problem*** asks to determine whether a given linkage has a noncrossing configuration in 2D. For general linkages, this problem is $\exists\mathbb{R}$-complete [2]. For fixed-angle orthogonal chains, the problem is in NP: given a binary choice of turning left or right at each vertex, we can construct an embedding (say, placing the first vertex at the origin and the second vertex on the positive x axis), and check for collisions and (for closed chains) closure. In fact, for fixed-angle orthogonal *open* chains, every instance is a "yes" instance:

Observation 1 *Every fixed-angle orthogonal open chain has a noncrossing configuration.*

Proof. Intuitively, we embed the chain in a zig-zag. Precisely, let $P = (v_0, v_1, \ldots, v_{n-1})$ be the path structure graph. First we put v_0 at $(0, 0)$, and v_1 at $(1, 0)$. For each $i = 2, 3, \ldots, n-1$, we define $x(C(v_i))$ and $y(C(v_i))$ as follows. When $\theta(v_i) = 180°$, we have no choice: $x(C(v_i)) = x(C(v_{i-1})) + (x(C(v_{i-1})) - x(C(v_{i-2})))$ and $y(C(v_i)) = y(C(v_{i-1})) + (y(C(v_{i-1})) - y(C(v_{i-2})))$. When $\theta(v_i) = 90°$ and $\overrightarrow{C(v_{i-2})C(v_{i-1})}$ is horizontal, we define $x(C(v_i)) = x(C(v_{i-1}))$ and $y(C(v_i)) = y(C(v_{i-1})) + 1$. If it is vertical, we define $x(C(v_i)) = x(C(v_{i-1})) + 1$ and $y(C(v_i)) = y(C(v_{i-1}))$. The obtained configuration is noncrossing because it proceeds monotonically in x and y, with strict increase in one of the coordinates. □

We note that Observation 1 holds for any fixed-angle orthogonal open chain which is not necessarily equilateral.

In the ***HP model***, the structure graph $G = (V, E)$ has its vertices ***bicolored*** by a color function $\omega : V \to \{H, P\}$. For a configuration C of an equilateral orthogonal linkage, a pair (u, v) of vertices forms an ***H–H contact*** if $\omega(u) = \omega(v) = H$, $\|C(u) - C(v)\| = 1$, and $\{u, v\} \notin E$. The ***HP optimal folding problem*** of a bicolored fixed-angle orthogonal equilateral chain asks to find a noncrossing configuration of the linkage in 2D that maximizes the number of H–H contacts.

A variant of the standard 3SAT problem is ***planar 3SAT***, where the graph $G_\phi = (C \cup V, E)$ of the variable set V and clause set C in a 3SAT formula ϕ, with edges between variables and the clauses that contain them, has a planar embedding. We use a variant of planar 3SAT with additional planarity restrictions: if we add edges to form a Hamiltonian cycle κ of $C \cup V$ that first visits all elements of C and then all elements of V, the resulting graph $G'_\phi = G_\phi \cup \kappa$ must also be planar. The ***linked planar 3SAT problem*** asks, given ϕ, G_ϕ, and κ, whether ϕ is satisfiable. Pilz [8] proved this problem NP-complete.

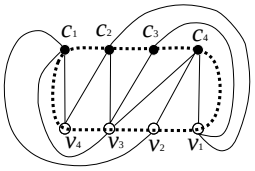


Figure 1: An example instance of linked planar 3SAT, where $c_1 = (\neg v_2 \vee \neg v_3 \vee \neg v_4)$, $c_2 = (v_4 \vee v_3 \vee \neg v_1)$, $c_3 = (\neg v_3 \vee v_1)$, and $c_4 = (v_1 \vee v_2 \vee v_3)$. Hamiltonian cycle κ (drawn dotted) visits $c_1, c_2, c_3, c_4, v_1, v_2, v_3, v_4$ in cyclic order.

3 Embedding Fixed-Angle Orthogonal Equilateral Closed Chains is Strongly NP-complete

In contrast to Observation 1, not all fixed-angle orthogonal equilateral *closed* chains are "yes" instances of the embedding problem. In particular, an orthogonal equilateral closed chain must have an even number of edges to have a configuration in 2D. Even with this property, the length-8 chain $(v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8 = v_0)$ with angles $\theta(v_2) = \theta(v_6) = 180°$ and $\theta(v_i) = 90°$ for $i = 0, 1, 3, 4, 5, 7$ has configurations in 2D but they have crossings at vertices v_2 and v_6. It is not difficult to show

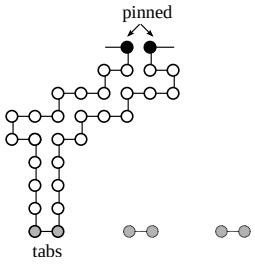
that the embedding problem for fixed-angle orthogonal closed chains is weakly NP-hard by a reduction from the ruler folding problem (see [7, Chap. 2]); this construction requires exponential edge lengths (or equilateral chains with exponentially long straight sections). In this section, we prove that the embedding problem is strongly NP-complete:

Theorem 1 *Embedding a fixed-angle orthogonal equilateral closed chain in 2D is strongly NP-complete.*

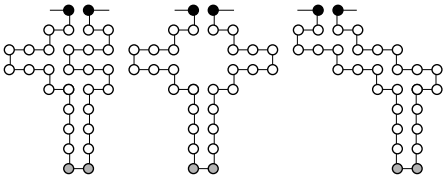
Proof. (Outline.) Section 2 argued membership in NP. To show NP-hardness, we reduce from the linked planar 3SAT problem. We are given a formula ϕ, the associated graph $G_\phi = (C \cup V, E)$, and a Hamiltonian path κ visiting $c_1, c_2, \ldots, c_m, v_1, v_2, \ldots, v_n$ in cyclic order. Because $G_\phi \cup \kappa$ is planar, there is a planar embedding with the clauses $c_1, c_2, \ldots, c_m$ along a single horizontal line from left to right, and the variables $v_1, v_2, \ldots, v_n$ along a lower horizontal line from right to left, as in Figure 1, but with edges routed via orthogonal paths. We can find such an embedding in polynomial time. Note that each edge is either interior or exterior to κ. We can assume that every variable v_i has an incident interior edge and an incident exterior edge, by adding appropriate always-satisfiable clauses $(v_i \vee v_i \vee \neg v_i)$ to κ so that an edge to v_i preserves planarity.

We construct four gadgets that we compose according to the embedding of G_ϕ and κ: the clause gadget, hook gadget, variable gadget, and frame gadget. Some gadgets assume ***pinned*** vertices that cannot move in the plane; we will discuss why they are effectively pinned when we combine the gadgets together.

Figure 2 illustrates the ***clause gadget***. We call the two gray vertices the "tabs" of this gadget. When black vertices are pinned and must turn downward, the tabs have three locations they can be placed. When we flip all vertices in the gadget along the horizontal line through two black vertices, we have three other symmetric options above the horizontal line.



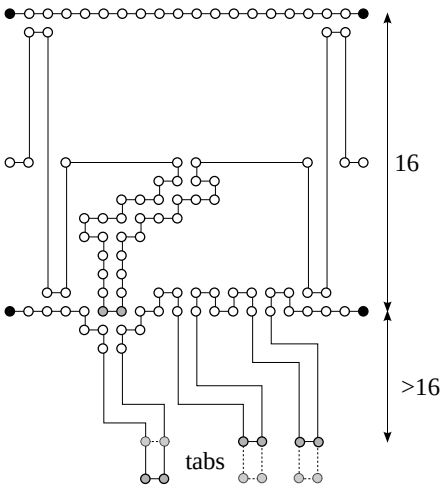
(a) When black vertices are pinned and forced to turn down, the two gray tabs can be placed in one of three places.



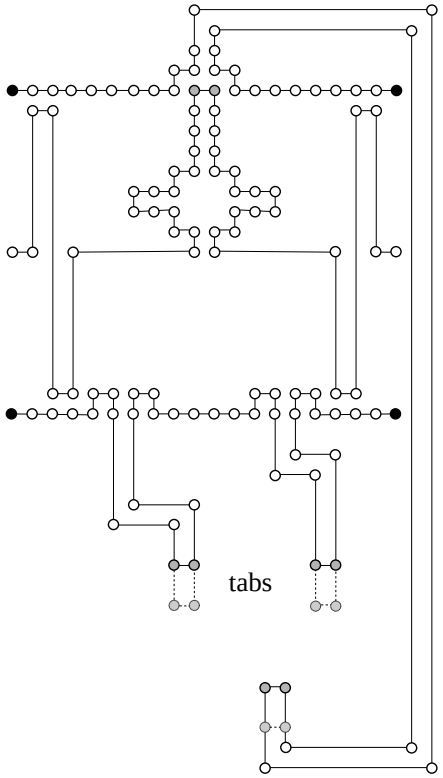
(b) Representative configurations (modulo reflection).

Figure 2: Clause gadget.

We surround each clause gadget with a ***hook gadget***, as shown in Figure 3, which consists of an upper



(a) Hook gadget with three options on the lower half.



(b) Hook gadget with two options on the lower half and one option on the upper half.

Figure 3: Two versions of the hook gadget. (Some vertices are not drawn to simplify the figure.)

half and a lower half to receive tabs of the clause gadget at distance 8 from the pinned vertices of the clause gadget. Again we assume that both endpoints of these upper and lower halves are pinned, which are depicted by black vertices. We add long flaps beside the clause gadget to prevent it from shifting vertically (relative to the hook gadget). The hook gadget limits the clause to three of its six options, which we arrange to be on the upper or lower halves according to which incident edges of the graph are exterior or interior to κ respectively (see Figure 1). We illustrate the two possibilities of this split modulo reflectional symmetry.

In Figure 3a, the three upper options of the clause are prevented by the upper half which is just a horizontal line, which would cross the clause tabs if the tabs were on the upper half. The lower half consists of three subgadgets, each with their own pair of tabs. When the clause gadget chooses one of the downward options for its tabs, it forces the tabs of the corresponding subgadget to be ***extended*** down by 2 (to avoid crossing), while the other tabs can remain ***retracted*** (which will always be better for avoiding crossings). (The figure shows the unused alternate state with dashed lines.) Each pair of tabs in the hook gadget has distance more than 16 from the clause gadget, and the linkage to the tabs is a doubled zig-zag; together, these guarantee that the tabs of a hook gadget cannot be flipped up because this would cross with the upper half. The doubled zig-zag also prevents the tabs from flipping horizontally. Thus each pair of tabs has exactly two placements (retracted and extended) if the black vertices are pinned.

In Figure 3b, one lower option of the clause (the middle) is prevented by the lower half being horizontal there, while the corresponding upper option is allowed by adding a subgadget to the upper half. Using the same arguments, the pair of tabs of the subgadget on the upper half has two exactly placements: retracted and extended. When the clause gadget chooses the available upper option, the pair of tabs of the subgadget is forced to be extended *up* by 2, which is the opposite of each subgadget on the lower half. Moreover, we arrange that no pair of doubled zig-zag corridors to tabs have the same height.[1]

Figure 4 illustrates the ***variable gadget***. The variable gadget for a variable v consists of two zig-zag paths of length $4k + 3$, where k is the number of appearances of v or $\neg v$ as a literal in clauses. The two zig-zag paths are joined by a horizontal baseline, which separates the upper and lower zig-zag paths, forcing only two possible embeddings: the one in the figure and its reflection through the baseline. Both zig-zag paths contain a horizontal segment of length 4 for each appearance of the

[1]Otherwise, unexpected pairs of adjacent corridors may be flipped. For example, consider the pair indicated by an arrow at the top of Figure 6. If these two corridors have the same height, the linkage joining the pair can be flipped up locally.

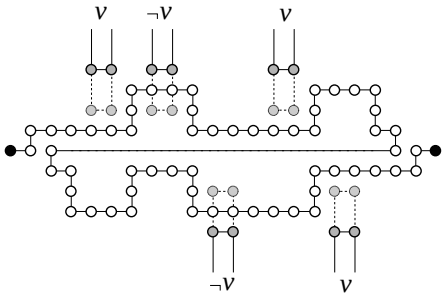


Figure 4: A variable gadget for a variable v that appears five times as v, $\neg v$, $\neg v$, v, and v. The corresponding tabs come from above, above, below, above, and below.

variable. The heights of the segments on the upper and lower zig-zag paths, measured from the baseline, are either 3 and -1 respectively, or 1 and -3 respectively. Which option depends on whether the corresponding literal uses the variable positive or negated, and on whether the corresponding tab of the hook gadget comes from above or below the variable gadget (which corresponds to whether the tab is from the upper or lower half of the clause/hook gadget, i.e., whether the graph edge is exterior or interior to κ). The heights are $(1, -3)$ if and only if either the literal is v and the tab comes from above, or the literal is $\neg v$ and the tab comes from below; in Figure 4, these are the first, third, and fourth pairs of horizontal segments. In the other cases, the heights are $(3, -1)$.

We arrange the variable gadgets with different heights (see Figure 6) so that the minimum vertical distance between two baselines of two variable gadgets is at least $4n$. This minimum distance guarantees that no pair of horizontal segments in variable gadgets for v_i and v_j with $i \neq j$ has the same height, which may cause an unexpected flip between them. In addition, our assumption that every variable has connections to clauses both above and below it means that there is a tab both above and below the variable, forcing an approximately correct height of the baseline.

The last gadget is the ***frame gadget***, shown in Figure 5, which surrounds all other gadgets. For a given closed chain, we consider the minimum rectangle that contains but does not intersect the chain (one step outside the bounding box on all sides). Then we remove an

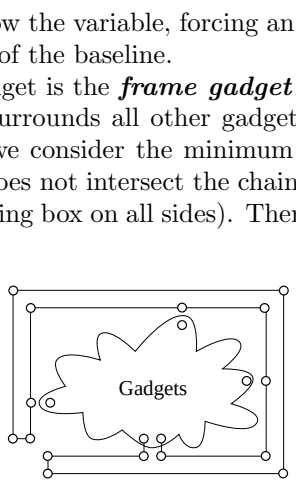


Figure 5: Frame gadget for closed chains.

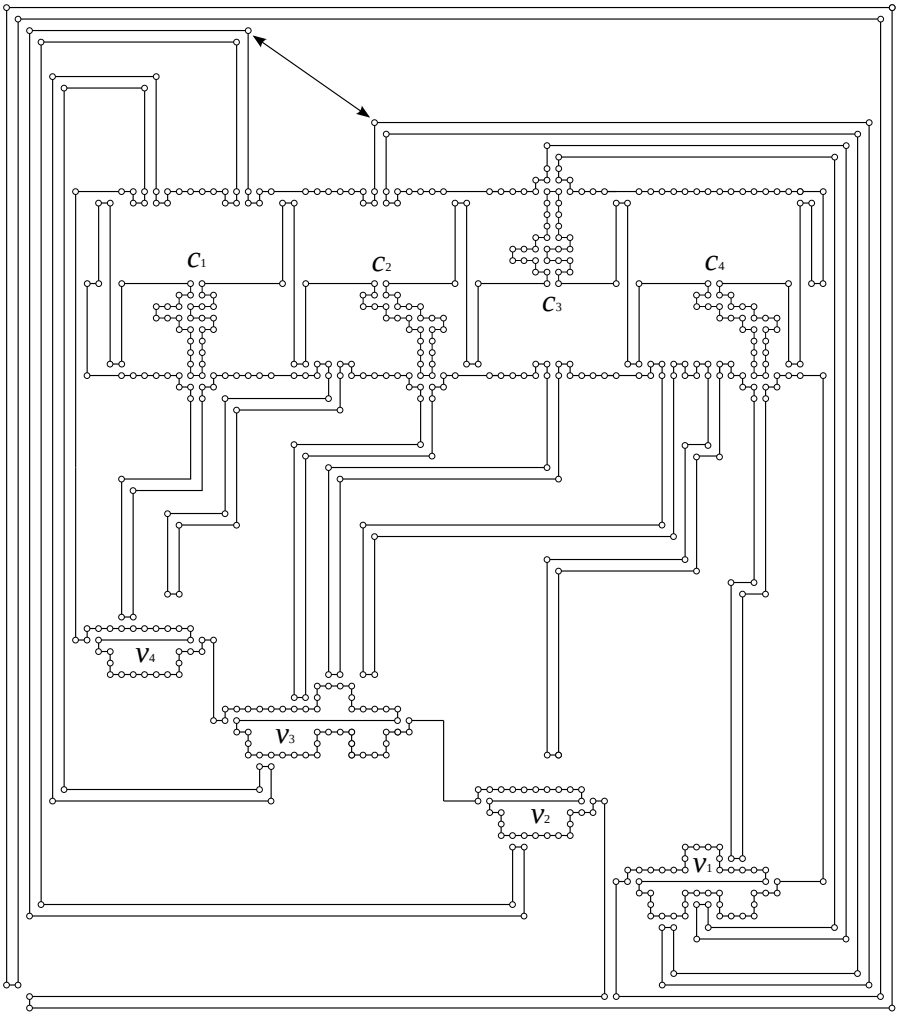


Figure 6: An example of the reduction from the instance in Figure 1, and the solution embedding corresponding to assignment $v_1 = \text{true}$, $v_2 = \text{true}$, $v_3 = \text{true}$, and $v_4 = \text{false}$, where clauses c_1, c_2, c_3, and c_4 choose the variables v_4, v_3, v_1, and v_1 as their true literals, respectively. (Note: vertical distances between two gadgets are not to scale.)

extreme edge $\{u, v\}$ in the gadgets, and attach the frame gadget that essentially doubles the minimum rectangle, as shown in Figure 5. The inside of the frame gadget includes the minimum rectangle, except for three edges, as part of the chain. The doubling prevents any part of the frame from being flipped with respect to the surrounded gadgets. This frame also inhibits the surrounded gadgets from illegal flips to outside the minimum rectangle.[2]

Figure 6 shows how all the gadgets fit together for the example instance from Figure 1. We join together all upper halves of hook gadgets for $c_1, c_2, \ldots, c_m$; all clause gadgets (and their flaps) for $c_m, c_{m-1}, \ldots, c_1$; all lower halves of the hook gadgets for $c_1, c_2, \ldots, c_m$; and all variable gadgets for $v_1, v_2, \ldots, v_n$, in these orders. Finally, we attach the frame gadget by replacing an edge on a path joining the upper halves of the hook gadgets, or an edge on a path joining the variable gadgets.[3]

This reduction can be done in time polynomial in the size of ϕ. It remains to show that an instance (ϕ, G_ϕ, κ) of linked planar 3SAT is satisfiable if and only if the resulting fixed-angle orthogonal equilateral closed chain has a planar embedding. Due to the space limitation, we only outline the proof.

When the linked planar 3SAT instance is satisfiable, at least one literal of each clause is satisfied by the assignment. The clause gadget then chooses the corresponding tabs of the corresponding hook gadget and extends it, while retracting the other tabs. The extended tabs force the corresponding variable gadget to take the true position, to avoid crossing. Because the assignment is satisfiable, all variable gadgets can avoid

[2] In the most common case, including the example in Figure 6, the frame is not necessary, as the hook gadgets will wrap around both sides of the construction.

[3] We omit the case that no edge can be seen from the outside of these gadgets.

crossing with tabs. On the other hand, when the loop has an embedding, all gadgets must be inside the frame gadget. Each clause gadget then has to indicate some tabs to be extended. Because the corresponding variable does not have any crossings, the corresponding variable satisfies the clause. Therefore, the instance of the linked planar 3SAT is satisfiable. □

4 HP Optimal Folding a Fixed-Angle Orthogonal Equilateral Open Chain is Strongly NP-complete

We now turn to orthogonal equilateral open chains in the HP model, where the vertices are bicolored H or P, and we wish to find a noncrossing configuration in 2D that maximizes the number of H–H contacts. In this section, we prove that this problem is NP-complete, despite the chain being open:

Theorem 2 *HP optimal folding of a bicolored fixed-angle orthogonal equilateral open chain is strongly NP-complete, even if the chain has just two H vertices.*

Proof. We use the same reduction in the proof of Theorem 1, except for the frame gadget, which we replace with Figure 7. The inside of the frame gadget covers the minimum rectangle except two edges, but now the bottom doubled edge extends very far to the left, more than 10 times the total length L of all other gadgets. The leftmost two vertices of the bottom doubled edge are H (and the chain is not closed there), and all other vertices in the chain are P.

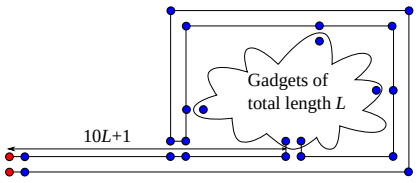


Figure 7: A frame gadget for an HP chain. The two H vertices are drawn red at the far left.

This reduction can be done in polynomial time. Thus it suffices to show that this arrangement of the frame is the only way to obtain the H–H contact at the two H vertices. Because the total length of the gadgets inside of the frame is at most L, we must arrange the two long segments attached to the H vertices in parallel as shown in the figure to make the H–H contact. Thus we must put all other gadgets inside the frame, and hence correctness follows from the proof of Theorem 1. □

5 Packing Fixed-Angle Orthogonal Equilateral Open Chains into Squares is Strongly NP-complete

We now address some of the open questions from [1]. First, the authors ask whether a fixed-angle orthogonal equilateral open chain (or in their terminology, an S–T sequence of squares, where each S square must continue straight and each T square must turn left or right) can be packed into a 2D square. Second, they ask whether the problem remains hard when the chain occupies a small fraction of the volume of the target shape. (They ask this question for the 3D version of the problem, but it naturally extends to the 2D version we consider.) We answer both questions by showing that packing a fixed-angle orthogonal equilateral open chain of length $O(s)$ into an $s \times s$ square is strongly NP-complete. This result is tight up to constant factors: if the chain has length $< s$, then it can be packed into an $s \times s$ square via Observation 1.

Theorem 3 *Embedding a given fixed-angle orthogonal equilateral open chain into an $s \times s$ square is strongly NP-complete, even if the chain has length $O(s)$.*

Proof. We use the same reduction in the proof of Theorem 2, except for the frame gadget, which we replace with Figure 8.

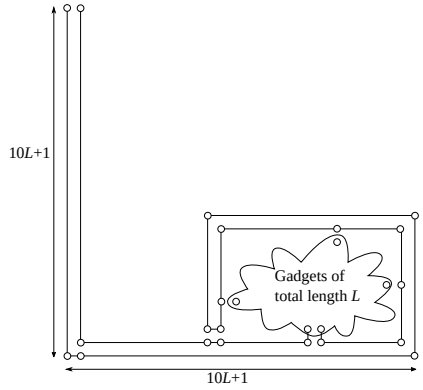


Figure 8: A frame gadget for an open chain which must fit in a $10L+1$ by $10L+1$ square.

This frame gadget starts the chain with two connected segments of length s. Any embedding into the $s \times s$ square must place these segments along two boundary edges of the square, say left and bottom as in the figure. The next two segments on the outside of the frame gadget must turn left to remain within the square. At the other end of the chain, we have a vertical (by parity) segment of length $s-1$ and a horizontal segment of length $> 9L$, which forces these segments against the first two segments. With these segments in place, the prior argument ensures that the rest of the frame and thus the rest of the gadgets are correctly placed.

The chain has length at most $47L+6$ vertices (from the SAT gadgets, the smaller frame, and the three long bars). Thus the length $l = O(s)$. □

It remains open whether the problem of *densely* packing a fixed-angle orthogonal equilateral open chain of

length s^2 into an $s \times s$ square is NP-complete.[4] The analogous problem in 3D is strongly NP-complete [1].

Acknowledgments

This work was initiated at the 3rd Virtual Workshop on Computational Geometry held in March 2022. We thank the other participants of that workshop — in particular Martin Demaine, David Eppstein, Timothy Gomez, and Aaron Williams — for helpful discussions and for providing a fruitful collaborative environment.

References

[1] Z. Abel, E. D. Demaine, M. L. Demaine, S. Eisenstat, J. Lynch, and T. B. Schardl. Finding a Hamiltonian path in a cube with specified turns is hard. *Journal of Information Processing*, 21(3):368–377, 2013.

[2] Z. Abel, E. D. Demaine, M. L. Demaine, S. Eisenstat, J. Lynch, and T. B. Schardl. Who needs crossings? Hardness of plane graph rigidity. In *Proceedings of the 32nd International Symposium on Computational Geometry (SoCG 2016)*, pages 3:1–3:15, Boston, Massachusetts, June 2016.

[3] B. Berger and T. Leighton. Protein folding in the hydrophobic-hydrophilic (*HP*) model is NP-complete. *Journal of Computational Biology*, 5(1):27–40, 1998.

[4] P. Crescenzi, D. Goldman, C. Papadimitriou, A. Piccolboni, and M. Yannakakis. On the complexity of protein folding. *Journal of Computational Biology*, 5(3):423–465, 1998.

[5] E. D. Demaine, M. L. Demaine, A. Hesterberg, Q. Liu, R. Taylor, and R. Uehara. Tangled tangles. In *The Mathematics of Various Entertaining Subjects (MOVES 2015)*, volume 2, pages 141–152. Princeton University Press, 2017.

[6] E. D. Demaine and S. Eisenstat. Flattening fixed-angle chains is strongly NP-hard. In *Proceedings of the 12th Algorithms and Data Structures Symposium (WADS 2011)*, pages 314–325, Brooklyn, New York, August 2011.

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

[8] A. Pilz. Planar 3-SAT with a Clause/Variable Cycle. In *16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*, pages 31:1–31:13. LIPIcs, Dagstuhl, 2018.

[4] In this context, a packing of a square is *densely* if the chain covers all grid points in the square.

A bound for Delaunay flip algorithms on flat tori

Loïc Dubois*

Abstract

We are interested in triangulations of flat tori. A Delaunay flip algorithm performs Delaunay flips on the edges of an input triangulation T until it reaches a Delaunay triangulation. We prove that no sequence of Delaunay flips is longer than $C_\Gamma \cdot n^2 \cdot \Lambda(T)$ where $\Lambda(T)$ is the maximum length of an edge of T, n is the number of vertices of T, and $C_\Gamma > 0$ depends only on the flat torus. The bound improves on the upper bound previously known [4] in three ways: the dependency in the "quality" of the input triangulation is linear instead of quadratic, the bound is tight, and the "quality parameter" is simpler.

Acknowledgments. The author thanks Vincent Despré, Benedikt Kolbe, and Monique Teillaud for their help and discussions.

1 Introduction

Delaunay triangulations are mostly known in the Euclidean plane setting. In this context a triangulation T can be defined as a maximal planar subdivision of a finite set of points P [3, Chapter 9]. If the two bounded faces of T incident to an inner edge e form a strictly convex quadrilateral then the edge e can be replaced, in T, by the other diagonal of the quadrilateral. Such operation is called a flip. The flip graph of P is the graph whose vertices are the triangulations on P and such that two triangulations are linked by an edge if there is a flip transforming one into the other. This graph is connected and its diameter is quadratic in the cardinal of P [5]. A triangulation is Delaunay if the circumdisk of every bounded face contains no point of the triangulation in its interior. A Delaunay flip algorithm takes as input a triangulation and performs Delaunay-flips until it reaches a Delaunay triangulation. Such an algorithm terminates [3, Observation 9.3].

Generalizing Delaunay triangulations [2] [1] and Delaunay flip algorithms [4] to other geometric spaces than the Euclidean plane is a natural question that has been studied (and implemented [7] [6]). In that setting Delaunay flip algorithms present the advantage of handling triangulations containing loops and multi-edges. A flat torus $\mathbb{T}_\Gamma$ is the quotient space of the Euclidean plane under the action of a group Γ generated by two independent translations (Section 2.1). In this paper we are interested in the complexity (number of flips) of Delaunay flip algorithms on flat tori. We prove Theorem 1.

Theorem 1 *Every sequence of Delaunay flips starting from a triangulation T of a flat torus $\mathbb{T}_\Gamma$ has length at most*

$$C_\Gamma \cdot n^2 \cdot \Lambda(T)$$

where $\Lambda(T)$ is the maximum length of an edge of T, n is the number of vertices of T, and $C_\Gamma > 0$ depends only on $\mathbb{T}_\Gamma$. This bound is asymptotically tight.

An upper bound was already proved [4, Theorem 16], together with the connectivity of the flip graph, as a particular (easy) case of a more general result on geometric triangulations of hyperbolic surfaces:

$$C_h \cdot n^2 \cdot \Delta(T)^2$$

where C_h depends only on $\mathbb{T}_\Gamma$ and $\Delta(T)$ is a parameter measuring in some sense how "stretched" T is. The definition of $\Delta(T)$ is not used in this paper but we give it (in the special case of triangulations of flat tori) for the interested reader: the real $\Delta(T)$ is the smallest diameter that can have a domain of $\mathbb{R}^2$ that is the union over every face t of the triangulation T of a lift (Section 2.1) of the face t.

To obtain their bound the authors showed that the edges flipped in a sequence of Delaunay flips cannot be longer than $2\Delta(T)$ [4, Lemma 10]. The upper bound follows from the observation that the number of segments no longer than $L > 0$ between two given points of $\mathbb{T}_\Gamma$ is at most quadratic in L.

Our first (small) improvement is to replace the parameter $\Delta(T)$ by the maximum length $\Lambda(T)$ of an edge in T. The inequality $\Lambda(T) \leq \Delta(T)$ is easily observed to be true. Moreover the definition of $\Delta(T)$ is more intricate than the definition of $\Lambda(T)$ and it is not obvious how to compute $\Delta(T)$.

Our second (main) improvement is to replace the quadratic dependency by a linear dependency in $\Lambda(T)$, obtaining a bound that is asymptotically tight.

*loic.dubois@ens-lyon.fr. LIGM, CNRS, Université Gustave Eiffel, F-77454 Marne-la-Vallée, France. This work was done while the author was working at Université de Lorraine, Inria, LORIA, F-54000 Nancy. It was partially supported by the grant ANR-17-CE40-0033 of the French National Research Agency ANR (project SoS https://sos.loria.fr/). It also was partially supported by ÉNS de Lyon.

2 Background

In this paper $\mathbb{R}^d$, $d \geq 1$, denotes the usual d-dimensional Euclidean space with the L_2 norm. We call *segment* of $\mathbb{R}^d$ the convex hull $[\widetilde{u}, \widetilde{v}]$ of any two distinct points $\widetilde{u}, \widetilde{v} \in \mathbb{R}^d$. We call *interior* of $[\widetilde{u}, \widetilde{v}]$ the set $[\widetilde{u}, \widetilde{v}] \setminus \{\widetilde{u}, \widetilde{v}\}$. The interior of a segment of $\mathbb{R}^d$ is not empty.

2.1 Flat tori

A *flat torus* $\mathbb{T}_\Gamma$ is the quotient of $\mathbb{R}^2$ under the action of a group Γ generated by two independent translations. For the needs of this section we introduce the projection $\rho : \mathbb{R}^2 \to \mathbb{T}_\Gamma$ mapping every point of $\mathbb{R}^2$ to its Γ-orbit.

We call *segment* of $\mathbb{T}_\Gamma$ any projection $s = \rho(\widetilde{s})$ of a segment $\widetilde{s}$ of $\mathbb{R}^2$ such that the restriction of ρ to the interior of $\widetilde{s}$ is injective. If $\widetilde{u}$ and $\widetilde{v}$ are the endpoints of $\widetilde{s}$ then $\rho(\widetilde{u})$ and $\rho(\widetilde{v})$ are the (possibly equal) *endpoints* of s. We call *interior* of s the image by ρ of the interior of $\widetilde{s}$.

A *lift* of a point $p \in \mathbb{T}_\Gamma$ is any point $\widetilde{p}$ in the Γ-orbit $\rho^{-1}(p)$. A lift of a segment s of $\mathbb{T}_\Gamma$ is any segment $\widetilde{s}$ of $\mathbb{R}^2$ whose interior is, through ρ, in one-to-one correspondence with the interior of s.

The *length* $l(s)$ of a segment s of $\mathbb{T}_\Gamma$ is the length of a lift of s in $\mathbb{R}^2$. It is independent of the lift.

2.2 Delaunay triangulations and flip algorithms

A topological triangulation of a flat torus $\mathbb{T}_\Gamma$ is any embedding of a finite undirected graph onto $\mathbb{T}_\Gamma$ such that each resulting face is homeomorphic to an open disk and is bounded by exactly three distinct edge-embeddings. Observe that this graph may have loops or multiple edges. A geometric triangulation of $\mathbb{T}_\Gamma$ is a topological triangulation in which each edge is embedded as a segment of $\mathbb{T}_\Gamma$. In this paper every triangulation is geometric so we just use the term *triangulation*.

The lift of a triangulation T of $\mathbb{T}_\Gamma$ is the infinite triangulation of $\mathbb{R}^2$ whose vertices and edges are the lifts of the vertices and edges of T.

A *Delaunay triangulation* of $\mathbb{T}_\Gamma$ is a triangulation T of $\mathbb{T}_\Gamma$ whose lift $\widetilde{T}$ is a Delaunay triangulation of $\mathbb{R}^2$ (Figure 1). In other words for each face $\widetilde{t}$ of $\widetilde{T}$ the disk circumscribing $\widetilde{t}$ contains no vertex of $\widetilde{T}$ in its interior. We refer to the literature for an introduction to Delaunay triangulations of $\mathbb{R}^2$ [3, Chapter 9].

Consider a triangulation T of $\mathbb{T}_\Gamma$, an edge e of T and a lift $\widetilde{e}$ of e. The segment $\widetilde{e}$ of $\mathbb{R}^2$ is an edge of the lift $\widetilde{T}$ of T and $\widetilde{e}$ is incident with two faces $\widetilde{t_1}$ and $\widetilde{t_2}$ of $\widetilde{T}$. Let $\widetilde{D_1}$ and $\widetilde{D_2}$ be the open disks of $\mathbb{R}^2$ circumscribing $\widetilde{t_1}$ and $\widetilde{t_2}$ respectively. Let also $\widetilde{v}_1$ be the vertex of $\widetilde{t_1}$ that is not a vertex of $\widetilde{t_2}$, and $\widetilde{v}_2$ be the vertex of $\widetilde{t_2}$ that is not a vertex of $\widetilde{t_1}$. The condition $\widetilde{v}_1 \in \widetilde{D_2}$ is equivalent to $\widetilde{v}_2 \in \widetilde{D_1}$. If it is satisfied we say that the edge e is *Delaunay-flippable* in the triangulation T and this definition is independent of the choice of the lift $\widetilde{e}$. In such a case the union of the closures of $\widetilde{t_1}$ and $\widetilde{t_2}$ is a convex quadrilateral and replacing, in the triangulation T, the edge e by the segment $\rho([\widetilde{v}_1, \widetilde{v}_2])$ of $\mathbb{T}_\Gamma$ yields another triangulation T' of $\mathbb{T}_\Gamma$. We say that the triangulation T' results from the *Delaunay flip* of the edge e in the triangulation T.

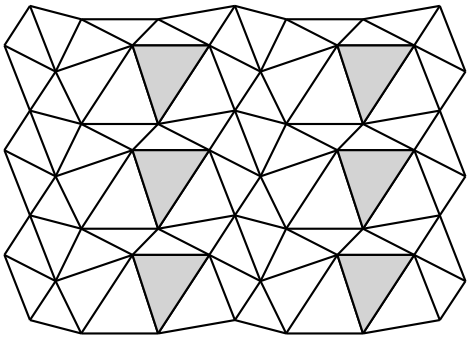


Figure 1: A portion of the lift of a Delaunay triangulation of a flat torus. (Gray) Six lifts of a single face.

We call *sequence of Delaunay flips* any sequence $T_0, \ldots, T_m$ of triangulations of $\mathbb{T}_\Gamma$, for some $m \geq 0$, such that for every $k \in \{1, \ldots, m\}$ the triangulation T_k results from the Delaunay flip of an edge in the triangulation T_{k-1}. We say that m is the *length* of the sequence.

Every *Delaunay flip algorithm* takes as input a triangulation of $\mathbb{T}_\Gamma$ and flips Delaunay-flippable edges until there is none left to flip. Such an algorithm terminates and outputs a Delaunay triangulation [4].

2.3 Stereographic projection and Delaunay flips

In $\mathbb{R}^3$ let $\mathbb{S}_2$ denote the 2-dimensional sphere of radius 1 centered at $(0, 0, 0)$. The point $P = (0, 0, -1)$ belongs to $\mathbb{S}_2$. We identify $\mathbb{R}^2$ with the plane of $\mathbb{R}^3$ containing the points whose third coordinate is 1. Given $\widetilde{p} \in \mathbb{R}^2$ we denote by $I_{\widetilde{p}}$ the unique line of $\mathbb{R}^3$ containing the points $\widetilde{p}$ and P (Figure 2).

The *stereographic projection* π is a bijection from $\mathbb{R}^2$ to $\mathbb{S}_2 \setminus P$. It maps every point $\widetilde{p} \in \mathbb{R}^2$ to the unique intersection of the line $I_{\widetilde{p}}$ with $\mathbb{S}_2 \setminus P$.

A triangle in $\mathbb{R}^3$ is the convex hull of three points that do not belong to a common line. We call *triangular surface* any connected union of triangles satisfying the following properties. Firstly if the intersection of any two distinct triangles of the union is not empty then it is either a vertex or an edge of both of the two triangles. Secondly every edge belongs to at most two triangles. Finally the triangles incident to a common vertex v can be either circularly or linearly ordered so that two such triangles share an edge e that is incident to v if and only if the two triangles are adjacent in the (circular or linear) ordering.

Every infinite triangulation $\mathcal{T}$ of $\mathbb{R}^2$ is mapped uniquely to a triangular surface S as follows. The vertices of S are the images of the vertices of $\mathcal{T}$ under π and the triangles of S are in one-to-one correspondence with the faces of $\mathcal{T}$: the three vertices $\widetilde{v_1}, \widetilde{v_2}$ and $\widetilde{v_3}$ of a face of $\mathcal{T}$ are mapped to the three vertices $\pi(\widetilde{v_1}), \pi(\widetilde{v_2})$, and $\pi(\widetilde{v_3})$ of a triangle of S. We say that such a triangular surface (issued of an infinite triangulation of $\mathbb{R}^2$) is *standard*.

We emphasize that every standard triangular surface shares no other point with the sphere $\mathbb{S}_2$ than its vertices. In fact if a point belongs to, but is not a vertex of, a standard triangular surface then it is at distance less than one from the point $(0, 0, 0)$.

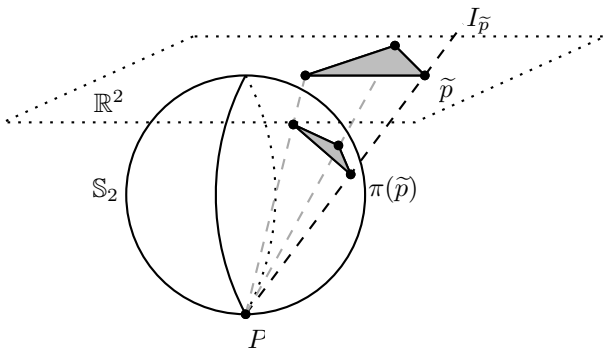


Figure 2: Mapping a lift of a triangulation of flat torus to a standard triangular surface.

Every standard triangular surface S induces bijection $\pi_S : \mathbb{R}^2 \to S$ sending every $\widetilde{p} \in \mathbb{R}^2$ to the unique intersection with S of the line $I_{\widetilde{p}}$. Given two standard triangular surfaces S and S' (possibly with $S = S'$) we say that S *is above* S' if for every $\widetilde{p} \in \mathbb{R}^2$ the point $\pi_{S'}(\widetilde{p})$ lies on the closed segment $[P, \pi_S(\widetilde{p})]$ of $\mathbb{R}^3$, on the line $I_{\widetilde{p}}$. The aboveness relation is a partial order on the set of standard triangular surfaces. Lemma 2 is folklore and follows from the fact that every circle on $\mathbb{R}^2$ is mapped under the stereographic projection to a circle on $\mathbb{S}_2 \setminus P$, the latter being the intersection with $\mathbb{S}_2 \setminus P$ of a plane of $\mathbb{R}^3$.

Lemma 2 *Assume that a triangulation T of a flat torus $\mathbb{T}_\Gamma$ results from the Delaunay flip of an edge e' in a triangulation T' of $\mathbb{T}_\Gamma$ and let e be the edge of T resulting from the flip. Let S and S' be the standard triangular surfaces associated to the lifts of T and T', respectively. Then S is above S'. Let also $p \in \mathbb{T}_\Gamma$ be the intersection point of the interiors of e and e' and $\widetilde{p} \in \mathbb{R}^2$ be any lift of p. Then $\pi_S(\widetilde{p}) \neq \pi_{S'}(\widetilde{p})$.*

3 Lower bound

On a flat torus $\mathbb{T}_\Gamma$ the length of a sequence of Delaunay flips ending at a Delaunay triangulation cannot be bounded from above by a function depending only on the number of vertices of the starting triangulation. This fact follows from two observations. The first observation is that it is easy to construct an infinite set of triangulations of $\mathbb{T}_\Gamma$ all having a single common vertex, say v, as their vertex set (Figure 3). The second observation is that there can only be a finite number of Delaunay triangulations of $\mathbb{T}_\Gamma$ having v as their unique vertex[1].

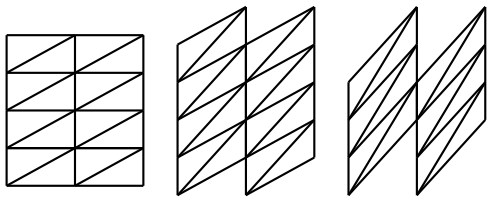


Figure 3: On a flat torus, three portions of the lifts of three triangulations with a single common vertex.

To understand this phenomenon more precisely, we consider a second parameter of the starting triangulation T: the maximum length $\Lambda(T)$ of an edge in T. We exhibit in Proposition 3 a family of starting triangulations T for which we prove a lower bound on the length of every sequence of Delaunay flips starting from T and ending at a Delaunay triangulation.

We are interested in a particular flat torus. Consider the two independent translations by the vectors $(1, 0)$ and $(0, 1)$ respectively. We are interested in the flat torus $\mathbb{T}_\square$ that is the quotient of $\mathbb{R}^2$ under the action of the group generated by those two translations. We denote by $\rho_\square$ the canonical projection from $\mathbb{R}^2$ to $\mathbb{T}_\square$. We say that $\mathbb{T}_\square$ is the *unit flat torus*.

Proposition 3 *For every $n \geq 1$ and every $\Lambda_0 > 0$ there is a triangulation T of the unit flat torus $\mathbb{T}_\square$ such that every sequence of Delaunay flips starting from T and ending at a Delaunay triangulation is longer than*

$$c \cdot n^2 \cdot \Lambda(T)$$

where $\Lambda(T) > \Lambda_0$ is the maximum length of an edge in T, n is the number of vertices of T, and $c > 0$ is a constant.

The quadratic dependence in the number of vertices is also a consequence of a more general fact about flips (not necessarily Delaunay flips) of triangulated polygons in the plane [5, Theorem 3.8]. Our construction is inspired from one previously known in that setting [5].

[1] Pick's theorem [8] infers the existence of $\Lambda_1 > 0$ depending only on $\mathbb{T}_\Gamma$ such that in $\mathbb{R}^2$ every disk of diameter Λ_1 intersects a lift of v. It follows that the edges of any Delaunay triangulation of $\mathbb{T}_\Gamma$ with vertex set $\{v\}$ are not longer than Λ_1. There can only be a finite number of such edges.

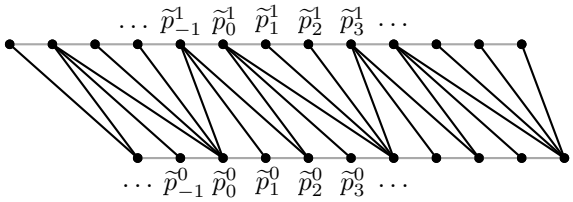


Figure 4: A portion of the lift of a triangulation belonging to $\mathcal{F}$ in the proof of Proposition 3. The fixed edges are in gray.

Proof. We fix $n \geq 1$ and $\Lambda_0 > 0$. See Figure 4.

For every $z \in \mathbb{Z}$ and every $\epsilon \in \{0,1\}$ we define the point $\widetilde{p}^\epsilon_z = (\frac{z}{n}, \epsilon)$ in $\mathbb{R}^2$ and the point p_z of $\mathbb{T}_\square$ by $p_z = \rho_\square(\widetilde{p}^0_z)$. Observe that if $z, z' \in \mathbb{Z}$ are such that $z \equiv z' \mod n$ then $p_z = p_{z'}$ and the points $\widetilde{p}^0_z, \widetilde{p}^1_z, \widetilde{p}^0_{z'}$, and $\widetilde{p}^1_{z'}$ are all lifts of p_z. For every $z, z' \in \mathbb{Z}$ we define the segment $s_{z,z'}$ of $\mathbb{T}_\square$ as $\rho_\square([\widetilde{p}^0_z, \widetilde{p}^1_{z'}])$.

We are interested in the set $\mathcal{F}$ of the triangulations of $\mathbb{T}_\square$ satisfying the following. The vertices of every triangulation $T \in \mathcal{F}$ are $p_1, \ldots, p_n$ and the edges of T are partitioned as follows: T contains n edges that we call *fixed* and $2n$ edges that we call *free*. For $k \in \{1, \ldots, n\}$ the k^{th} fixed edge of T is $\rho_\square([\widetilde{p}^0_{k-1}, \widetilde{p}^0_k])$. The only restriction on the free edges of T is that they must belong to $\{s_{z,z'} : z, z' \in \mathbb{Z}\}$.

Claim 1. For every $T \in \mathcal{F}$ the following holds:

(a) The fixed edges of T are not Delaunay-flippable.

(b) The Delaunay flip of a free edge e in T results in a triangulation $T' \in \mathcal{F}$.

(c) Such a Delaunay flip replaces the edge e in T by an edge e' in T' such that $l(e') \geq l(e) - 2/n$.

(d) The lengths of two free edges of T cannot differ by more than 2.

Claim 2. There is a triangulation in $\mathcal{F}$ having a free edge longer than Λ_0.

Claim 3. There is a constant $\Lambda_1 > 0$ such that the edges of every Delaunay triangulation in $\mathcal{F}$ are not longer than Λ_1.

Claims 2 and 3 are straightforward. We will prove Claim 1 in the end. We first show that those claims imply the result. By Claim 2 there is a triangulation $T_0 \in \mathcal{F}$ having a free edge longer than Λ_0. Let $\Lambda(T)$ denote the maximum length of an edge in T_0; $\Lambda(T)$ is the length of a free edge of T_0. Indeed the free edges of T_0 have length at least 1 while the fixed edges of T_0 have length $1/n$.

We assign to every triangulation $T \in \mathcal{F}$ a weight $\omega(T)$ that is the sum of the lengths of its edges. By Claim 1.d $\omega(T_0) \geq 1 + 2n(\Lambda(T) - 2)$. Indeed T_0 has n fixed edges of length $1/n$ and $2n$ free edges of length at least $\Lambda(T) - 2$.

Consider a sequence $T_0, \ldots, T_m$ of Delaunay flips for some $m \geq 0$ that starts from T_0 and ends at a Delaunay triangulation T_m. By Claims 1.a and 1.b all the triangulations $T_0, \ldots, T_m$ belong to $\mathcal{F}$. By Claim 1.c holds $\omega(T_m) \geq \omega(T_0) - 2m/n$. By Claim 3 there is a constant $\Lambda_1 > 0$ such that $\omega(T_m) \leq 3n\Lambda_1$. Thus

$$2m \geq n(\omega(T_0) - \omega(T_m)) \geq n + (2\Lambda(T) - 3\Lambda_1 - 4)n^2.$$

That proves the result. Now we prove Claim 1.

Proof of Claim 1. To prove (a) consider a fixed edge e of the triangulation T. There is $k \in \{1, \ldots, n\}$ such that the segment $\widetilde{e}$ of $\mathbb{R}^2$ between $\widetilde{p}^0_{k-1} = (\frac{k-1}{n}, 0)$ and $\widetilde{p}^0_k = (\frac{k}{n}, 0)$ is a lift of e. Consider the two faces $\widetilde{t}_1$ and $\widetilde{t}_2$ of the lift $\widetilde{T}$ of T that are incident to $\widetilde{e}$. Let $\widetilde{v}_1$ be the vertex of $\widetilde{t}_1$ that is not a vertex of $\widetilde{t}_2$ and let $\widetilde{v}_2$ be the vertex of $\widetilde{t}_2$ that is not a vertex of $\widetilde{t}_1$. Up to renaming $\widetilde{v}_1$ and $\widetilde{v}_2$ there are $z, z' \in \mathbb{Z}$ such that $\widetilde{v}_1 = \widetilde{p}^1_z = (\frac{z}{n}, 1)$ and $\widetilde{v}_2 = (\frac{z'}{n}, -1)$. It is straightforward to check that the open disk whose boundary contains $\widetilde{p}^0_{k-1}$,$\widetilde{p}^0_k$, and $\widetilde{v}_1$ does not contain $\widetilde{v}_2$.

To prove (b) and (c) consider a free edge e of the triangulation T and assume that e is Delaunay-flippable. There are $z, z' \in \mathbb{Z}$ such that $e = s_{z,z'}$. The segment $\widetilde{e} = [\widetilde{p}^0_z, \widetilde{p}^1_{z'}]$ is a lift of e so it is incident to two faces $\widetilde{t}_1$ and $\widetilde{t}_2$ of the lift $\widetilde{T}$ of T. Let $\widetilde{v}_1$ be the vertex of $\widetilde{t}_1$ that is not a vertex of $\widetilde{t}_2$ and let $\widetilde{v}_2$ be the vertex of $\widetilde{t}_2$ that is not a vertex of $\widetilde{t}_1$. Up to renaming $\widetilde{v}_1$ and $\widetilde{v}_2$ there is $\epsilon \in \{1, -1\}$ such that $\widetilde{v}_1 = \widetilde{p}^0_{z-\epsilon}$ and $\widetilde{v}_2 = \widetilde{p}^1_{z'+\epsilon}$: every other case would contradict the fact that both T and the triangulation resulting from the flip of e in T are indeed triangulations. The edge e' resulting from the lift of e in T admits the segment $[\widetilde{v}_1, \widetilde{v}_2]$ as a lift and $l(e') \geq l(e) - 2/n$.

To prove (d) consider a lift $\widetilde{e}$ of a free edge e of T and the two vertices $\widetilde{v}_1$ and $\widetilde{v}_2$ of $\widetilde{e}$. Let τ_1 be the translation by the vector $(1, 0)$ (one of the two translations defining $\mathbb{T}_\square$). The four points of $\mathbb{R}^2$ that are $\widetilde{v}_1$, $\widetilde{v}_2$, $\tau_1(\widetilde{v}_2)$ and $\tau_1(\widetilde{v}_1)$ are the vertices of a closed parallelogram $P_\diamond$. The closed parallelogram $P_\diamond$ contains a lift of every free edge of T. Indeed every free edge f of T distinct from e admits a lift $\widetilde{f}$ whose interior intersects the interior of $P_\diamond$ [2], and the interior of $\widetilde{f}$ cannot intersect a side of $P_\diamond$ because that would imply that the interior of the edge f intersects another edge of the triangulation T. To conclude observe that by construction the sides of $P_\diamond$ are of length 1 (for the sides $\widetilde{v}_1\tau_1(\widetilde{v}_1)$ and $\widetilde{v}_2\tau_1(\widetilde{v}_2)$) and of length $l(e)$ (for the sides $\widetilde{v}_1\widetilde{v}_2$ and $\tau_1(\widetilde{v}_1)\tau_1(\widetilde{v}_2)$)). Thus every free edge of T has its length between $l(e)-2$ and $l(e)+2$. □

[2] The closed parallelogram $P_\diamond$ is a fundamental domain for the flat torus $\mathbb{T}_\square$.

4 Upper bound

In Section 3 we exhibited a family of triangulations T for which the length of a sequence of Delaunay flips starting from T and ending at a Delaunay triangulation is bounded from below (Proposition 3). In this section we show that our construction was actually "the worst possible" and that the lower bound of Proposition 3 is asymptotically matched by a general upper bound over all possible starting triangulations on a flat torus. This upper bound comes from an observation formalized by Proposition 4. Informally, given two "long" edges e_1 and e_2 among the edges flipped in a sequence of Delaunay flips, if e_1 and e_2 have "comparable" lengths then they must be "roughly parallel".

4.1 Statement of Proposition 4

Consider a flat torus $\mathbb{T}_\Gamma$. We say that a segment s of $\mathbb{T}_\Gamma$ *follows* a segment s' of $\mathbb{T}_\Gamma$ (possibly with $s = s'$) if there are triangulations T and T' of $\mathbb{T}_\Gamma$ (possibly with $T = T'$) such that s is an edge of T, s' is an edge of T', and there is a sequence of Delaunay flips (possibly of length 0) starting from T' and ending at T.

We map every segment s of $\mathbb{T}_\Gamma$ to a pair $\{\widetilde{p}, -\widetilde{p}\}$ of opposite nonzero vectors of $\mathbb{R}^2$ as follows. We consider the endpoints $\widetilde{u}$ and $\widetilde{v}$ of a lift of s and define the point $\widetilde{p}$ as the image of $0_{\mathbb{R}^2}$ under the translation that maps $\widetilde{u}$ to $\widetilde{v}$. The resulting pair $\{\widetilde{p}, -\widetilde{p}\}$ does not depend on the choice of $\widetilde{u}$ and $\widetilde{v}$. We call these two points the *signature points* of the segment s.

Consider two segments s and s' of $\mathbb{T}_\Gamma$ and assume that s and s' have the same endpoints u and v (u and v may be equal) and the same signature points $\widetilde{p}$ and $-\widetilde{p}$. Consider also a lift $\widetilde{u}$ of u. For $\epsilon \in \{1, -1\}$ let $\widetilde{v}_\epsilon$ denote the image of $\widetilde{u}$ under the translation that maps $0_{\mathbb{R}^2}$ to $\epsilon\widetilde{p}$. There are $\epsilon, \epsilon' \in \{1, -1\}$ such that the segment $[\widetilde{u}, \widetilde{v}_\epsilon]$ of $\mathbb{R}^2$ is a lift of s and such that the segment $[\widetilde{u}, \widetilde{v}_{\epsilon'}]$ of $\mathbb{R}^2$ is a lift of s'. If $\epsilon = \epsilon'$ then $s = s'$. Thus there cannot be more than two distinct segments of $\mathbb{T}_\Gamma$ having the same endpoints and the same signature points.

Proposition 4 *Given a flat torus $\mathbb{T}_\Gamma$ there are $\kappa > 0$ and $l_0 > 0$ depending only on $\mathbb{T}_\Gamma$ such that the following holds. If a segment s of $\mathbb{T}_\Gamma$ follows a segment s' of $\mathbb{T}_\Gamma$ and if $l(s) > l_0$ and $l(s') \in [l(s)/2, 2l(s)]$ then the signature points of s' are at distance at most κ from the line containing the signature points of s.*

See Figure 5 for an illustration of Proposition 4.

4.2 Proof of Proposition 4

Lemma 5 *Assume that a segment s of a flat torus $\mathbb{T}_\Gamma$ follows a segment s' of $\mathbb{T}_\Gamma$ and consider a lift $\widetilde{s}$ of s and a lift $\widetilde{s'}$ of s'. If $\widetilde{s}$ and $\widetilde{s'}$ intersect in their respective interiors and if there is an open disk $\widetilde{D}$ whose boundary $\partial\widetilde{D}$ contains the two endpoints of $\widetilde{s}$ and one of the endpoints of $\widetilde{s'}$ then the other endpoint of $\widetilde{s'}$ lies outside $\widetilde{D}$.*

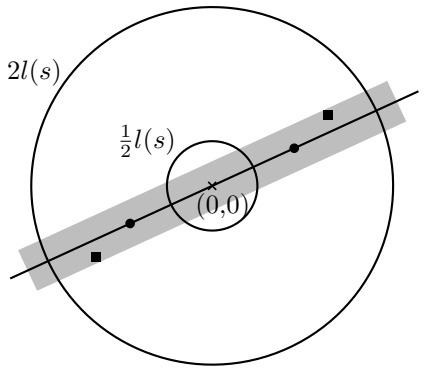


Figure 5: Illustration of Proposition 4. (Black disks) Signature points of s. (Black squares) Signature points of s'. (Gray) Points at distance at most κ from the line containing the signature points of s.

Observe that in Lemma 5 if a point lies outside the open disk $\widetilde{D}$ it may still lie within the boundary circle $\partial\widetilde{D}$. In particular the conclusion of the lemma holds when $s = s'$ and $\widetilde{s} = \widetilde{s'}$.

Proof. Let $\widetilde{u}, \widetilde{v}$ denote the two endpoints of $\widetilde{s}$, and $\widetilde{u'}, \widetilde{v'}$ denote the two endpoints of $\widetilde{s'}$. Assume that the points $\widetilde{u}, \widetilde{v}$, and $\widetilde{u'}$ belong to the circle $\partial\widetilde{D}$. The projection $\pi(\partial\widetilde{D})$ is the intersection with $\mathbb{S}_2 \setminus P$ of a plane $\mathcal{P} \subset \mathbb{R}^3$. The plane $\mathcal{P}$ bounds two closed half-spaces whose union is $\mathbb{R}^3$ and whose intersection is $\mathcal{P}$. We will show that $\pi(\widetilde{v'})$ belongs to the half-space $\mathcal{R}$ containing the point P.

There are triangulations T and T' of $\mathbb{T}_\Gamma$ such that s is an edge of T, s' is an edge of T', and there is a sequence of Delaunay flips starting from T' and ending at T. The lift $\widetilde{T}$ of T and the lift $\widetilde{T'}$ of T' are infinite triangulations of $\mathbb{R}^2$; $\widetilde{s}$ is an edge of $\widetilde{T}$ and $\widetilde{s'}$ is an edge of $\widetilde{T'}$. Let S and S' be the standard triangular surfaces associated to $\widetilde{T}$ and $\widetilde{T'}$ respectively. Lemma 2 and the transitivity of the aboveness relation imply that S is above S' (possibly with $S = S'$). Thus any point $\widetilde{p} \in \mathbb{R}^2$ of the intersection of $\widetilde{s}$ and $\widetilde{s'}$ is such that $\pi_{S'}(\widetilde{p})$ lies on the segment of $\mathbb{R}^3$ $[P, \pi_S(\widetilde{p})]$ on the line $I_{\widetilde{p}}$. (Section 2.3). The point $\pi_S(\widetilde{p})$ is the intersection with the line $I_{\widetilde{p}}$ of an edge of S: this edge is the segment of $\mathbb{R}^3$ $[\pi(\widetilde{u}), \pi(\widetilde{v})]$. This segment is fully contained in the plane $\mathcal{P}$ since its endpoints $\pi(\widetilde{u})$ and $\pi(\widetilde{v})$ both belong to $\mathcal{P}$. In particular $\pi_S(\widetilde{p})$ belongs to $\mathcal{P}$ and $\pi_{S'}(\widetilde{p})$ belongs to the half-space $\mathcal{R}$. Since $\pi_{S'}(\widetilde{p})$ is distinct from $\pi(\widetilde{u'})$ and belongs to the segment of $\mathbb{R}^3$ $\left[\pi(\widetilde{u'}), \pi(\widetilde{v'})\right]$ and since both $\pi_{S'}(\widetilde{p})$ and $\pi(\widetilde{u'})$ belong to $\mathcal{R}$ then so does $\pi(\widetilde{v'})$. □

Lemma 6 *Let $\varepsilon > 0$ and $d > 20\varepsilon$. Let $\widetilde{u} \in \mathbb{R}\times]-\infty, 0[$ and $\widetilde{v} \in \mathbb{R}\times]0, +\infty[$ such that $\|\widetilde{u}\| < \varepsilon$ and $\|\widetilde{v} - \widetilde{u}\| < 4d$. There is a unique open disk $\widetilde{D}$ whose boundary contains $\widetilde{u}$ and the points $(d, 0)$ and $(-d, 0)$. If $\widetilde{v}$ lies outside $\widetilde{D}$ then $y_{\widetilde{v}} < 100\varepsilon$ where $y_{\widetilde{v}}$ denotes the second coordinate of $\widetilde{v}$.*

Observe that in Lemma 6 if the point $\widetilde{v}$ lies outside the open disk $\widetilde{D}$ it may, still, belong to its boundary. See Figure 6 for an illustration of Lemma 6.

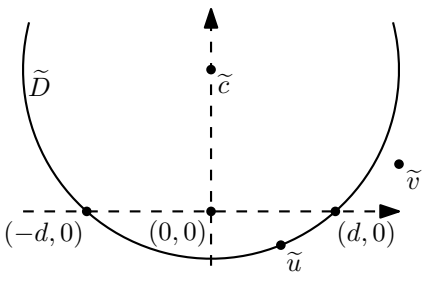


Figure 6: Illustration of Lemma 6.

We put the proof of Lemma 6 in Appendix 5. Now we prove Proposition 4 using Lemmas 5 and 6.

Proof. *(Proof of Proposition 4)*
Assume that a segment s of a flat torus $\mathbb{T}_\Gamma$ follows a segment s' of $\mathbb{T}_\Gamma$. Consider a lift $\widetilde{s}$ of s. Up to a rotation and a translation we assume that $\widetilde{s}$ is a horizontal segment whose center is the point $(0, 0)$. We claim that there exist $\varepsilon > 0$ depending only on $\mathbb{T}_\Gamma$ and a lift $\widetilde{s'}$ of s' whose endpoints $\widetilde{u} = (x_{\widetilde{u}}, y_{\widetilde{u}})$ and $\widetilde{v} = (x_{\widetilde{v}}, y_{\widetilde{v}})$ satisfy the three following conditions: $\|\widetilde{u}\| < \varepsilon$, $y_{\widetilde{u}} < 0$, and $y_{\widetilde{u}} \leq y_{\widetilde{v}}$. To prove this claim start with any lift of s' and let $\widetilde{p} = (x_{\widetilde{p}}, y_{\widetilde{p}})$ and $\widetilde{q} = (x_{\widetilde{q}}, y_{\widetilde{q}})$ denote the endpoints of this lift. Up to renaming $\widetilde{p}$ and $\widetilde{q}$ we assume $y_{\widetilde{p}} \leq y_{\widetilde{q}}$. There is $\varepsilon > 0$ such that any open disk of diameter ε intersects the Γ-orbit of $\widetilde{p}$. Hence there is a point $\widetilde{u} \in \mathbb{R}^2$ at distance less than $\varepsilon/2$ from the point $(0, -\varepsilon/2)$ and a translation $\tau \in \Gamma$ such that $\tau(\widetilde{p}) = \widetilde{u}$. Setting $\widetilde{v} = \tau(\widetilde{q})$ proves the claim.

The signature points of s belong to the line $\mathbb{R} \times \{0\}$. We set $\kappa = 101\varepsilon$ and consider one of the two signature points of s', namely $\widetilde{v} - \widetilde{u}$. Since $-\varepsilon < y_{\widetilde{u}} < 0$ and $y_{\widetilde{u}} \leq y_{\widetilde{v}}$ proving $y_{\widetilde{v}} < 100\varepsilon$ will infer the proposition. Having $y_{\widetilde{v}} \leq 0$ would conclude so we assume $y_{\widetilde{v}} > 0$. There are two cases: either $\widetilde{s}$ and $\widetilde{s'}$ intersect in their interiors or they do not.

First assume that $\widetilde{s}$ and $\widetilde{s'}$ intersect in their interiors. We set $d = l(s)/2$ and we can enforce that $d > 20\varepsilon$. Indeed we assumed $l(s) > l_0$ and we can choose l_0 large enough with respect to ε (recall that ε depends only on $\mathbb{T}_\Gamma$). Lemma 5 implies that $\widetilde{v}$ lies outside the open disk $\widetilde{D}$ whose boundary contains $\widetilde{u}$ and the endpoints $(d, 0)$ and $(-d, 0)$ of $\widetilde{s}$. Thus the conditions of Lemma 6 are satisfied and $y_{\widetilde{v}} < 100\varepsilon$.

If $\widetilde{s}$ and $\widetilde{s'}$ do not intersect in their interiors then $\widetilde{v}$ lies outside $\widetilde{D}$ and the conditions of Lemma 6 are satisfied again. □

4.3 Proof of the upper bound

Lemma 7 is folklore. We prove it in Appendix 6 for completeness.

Lemma 7 *Consider a flat torus $\mathbb{T}_\Gamma$, an integer $m \geq 0$, and a sequence of Delaunay flips $T_0, \ldots, T_m$. For every $k \in \{1, \ldots, m\}$ we let e_k denote the edge of T_{k-1} that is flipped to obtain T_k. The segments $e_1, \ldots, e_m$ of $\mathbb{T}_\Gamma$ are pairwise distinct.*

The edges flipped in a sequence of Delaunay flips are not longer than $2\Delta(T)$ where $\Delta(T)$ is a parameter measuring in some sense how "stretched" the starting triangulation T is [4, Lemma 10]. The arguments yielding a bound in terms of $\Delta(T)$ easily infer a bound in terms of the maximum length of an edge in T. This new bound is stated by Lemma 8. As the proof of Lemma 8 is only a slight adaptation of the anterior proof [4, Lemma 10] we omit it here and put it in Appendix 7 for completeness.

Lemma 8 *Consider triangulations T and T' of a flat torus $\mathbb{T}_\Gamma$ and assume that there is a sequence of Delaunay flips starting from T' and ending at T. Then the edges of T cannot be more than twice as long as a longest edge of T'.*

Now we prove Theorem 1.

Proof. *(Proof of Theorem 1)* Consider $m \geq 0$ and a sequence of Delaunay flips $T_0, \ldots, T_m$ such that $T_0 = T$. For every $k \in \{0, \ldots, m\}$ the edges of T_k constitute a set E_k of segments of $\mathbb{T}_\Gamma$. We are interested in the union E of the sets $E_0, \ldots, E_m$. By Lemma 7 the cardinal of E is not smaller than m. We partition the elements of E into $n(n+1)/2$ subsets according to their endpoints, as follows. For every unordered pair $\{u, v\}$ of vertices of the triangulation T we consider the set of segments in E that end at u and v. For every single vertex v of T we consider the set of segments in E that admit v as their unique endpoint. Proving that each of those subsets contains at most $C_\Gamma \cdot \Lambda(T)$ segments will infer the result.

So consider such a subset $F \subseteq E$ in the partition that we just described and let u and v be the (possibly equal) endpoints of the segments in F. Let $\kappa > 0$ and $l_0 > 0$ be given by Proposition 4.

As explained in Section 4.1 there cannot be more than two distinct segments of $\mathbb{T}_\Gamma$ having the same endpoints and the same signature points. Fix a lift $\widetilde{u}$ of u and a lift $\widetilde{v}$ of v. For any signature point $\widetilde{p}$ of a segment in F there is $\tau \in \Gamma$ such that either $\widetilde{p}$ or $-\widetilde{p}$ is equal to $\tau(\widetilde{v}) - \widetilde{u}$. Thus there is a finite number of such signature points

that are at distance at most l_0 from the point $(0,0)$, and this finite number depends only on $\mathbb{T}_\Gamma$ (recall that l_0 depends only on $\mathbb{T}_\Gamma$). That implies that there is only a finite number of segments in F that are not longer than l_0.

Consequently we let $F' \subseteq F$ be the set of segments in F that are longer than l_0: we will now bound the cardinality of F'. By Lemma 8 no segment in F' is longer than $2\Lambda(T)$. We partition the segments in F' by their lengths as follows. We consider $j_0 = l_0 < j_1 < \cdots < j_N = 2\Lambda(T)$ for some $N \geq 1$ such that for every $k \in \{1, \ldots, N\}$ the reals j_{k-1} and j_k differ by a factor of at most 2. For every $k \in \{1, \ldots, N\}$ we let F'_k denote the set of segments in F' whose length belongs to $]j_{k-1}, j_k]$. We now fix k and claim that F'_k contains at most $C'_\Gamma \cdot (j_k - j_{k-1})$ segments, where $C'_\Gamma > 0$ is a constant that depends only on $\mathbb{T}_\Gamma$.

To prove this claim observe that if F'_k is not empty then it contains a segment s that follows every other segment $s' \in F'_k \setminus \{s\}$. For such another segment s' Proposition 4 states that the signature points of s' are at distance at most κ from the line containing the signature points of s. Also the distance to $(0,0)$ of the two signature points of s' is the length of s' and thus lies between j_{k-1} and j_k. Consequently the number of signature points of elements of F'_k is at most linear in $j_k - j_{k-1}$ and the constant coefficient depends only on $\mathbb{T}_\Gamma$.

To clarify this statement observe that the signature points of elements of F'_k all belong, by definition, to the Γ-orbit $\mathcal{O}$ of some point of $\mathbb{R}^2$. We just proved that such signature points also belong to the set $\mathcal{D}$ of points of $\mathbb{R}^2$ (1) that are at distance κ from the line containing the signature points of s and (2) whose distance to $(0,0)$ lies between j_{k-1} and j_k. The cardinality of the intersection of $\mathcal{O}$ and $\mathcal{D}$ is linear in $j_k - j_{k-1}$, and the constant coefficient depends only on $\mathcal{O}$ and κ, that both depend only on $\mathbb{T}_\Gamma$.

That, together with Proposition 3 for the lower bound, concludes the proof of Theorem 1. □

References

[1] M. Bogdanov, M. Teillaud, and G. Vegter. Delaunay Triangulations on Orientable Surfaces of Low Genus. In S. Fekete and A. Lubiw, editors, *32nd International Symposium on Computational Geometry (SoCG 2016)*, volume 51 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:17, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[2] M. Caroli and M. Teillaud. Delaunay Triangulations of Closed Euclidean d-Orbifolds. *Discrete Computational Geometry*, 55:827–853, 2016.

[3] M. de Berg, M. V. Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry, Algorithms and Applications*. Springer, 2000.

[4] V. Despré, J.-M. Schlenker, and M. Teillaud. Flipping geometric triangulations on hyperbolic surfaces. In *Proceedings 36th International Symposium on Computational Geometry (SoCG'20)*, pages 35:1–35:16, 2020.

[5] F. Hurtado, M. Noy, and J. Urrutia. Flipping edges in triangulations. *Discrete and Computational Geometry*, 22:333–346, 1999.

[6] I. Iordanov and M. Teillaud. 2D Periodic Hyperbolic Triangulations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.4 edition, 2022.

[7] N. Kruithof. 2D Periodic Triangulations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.4 edition, 2022.

[8] J. Trainin. An elementary proof of Pick's theorem. *Mathematical Gazette*, 91(522):536–540, 2007.

Appendix

5 Proof of Lemma 6

Proof. We write $\widetilde{u} = (x_{\widetilde{u}}, y_{\widetilde{u}})$ and $\widetilde{v} = (x_{\widetilde{v}}, y_{\widetilde{v}})$ and recall that $y_{\widetilde{v}} > 0$ and $y_{\widetilde{u}} < 0$ both hold by assumption. The latter enforces the existence of the open disk $\widetilde{D}$. Now let $\widetilde{c}$ denote the center of $\widetilde{D}$. The segment $[-d, d] \times \{0\}$ is a chord of $\widetilde{D}$ and its midpoint is the point $(0, 0)$. Thus the first coordinate of $\widetilde{c}$ is 0 and the radius of $\widetilde{D}$ is $\sqrt{y_{\widetilde{c}}^2 + d^2}$ where $y_{\widetilde{c}}$ denotes the second coordinate of $\widetilde{c}$. One easily gets $y_{\widetilde{c}} > 0$ from the assumptions $y_{\widetilde{u}} < 0$, $\|\widetilde{u}\| < \varepsilon$, and $d > \varepsilon$. See Figure 6.

We first prove a few inequalities that may seem arbitrary at first but will be used in the end of the proof. Pythagorean Theorem gives $(y_{\widetilde{c}} - y_{\widetilde{u}})^2 + x_{\widetilde{u}}^2 = y_{\widetilde{c}}^2 + d^2$ which simplifies to $-2y_{\widetilde{u}}y_{\widetilde{c}} = d^2 - x_{\widetilde{u}}^2 - y_{\widetilde{u}}^2$. We assumed $\|\widetilde{u}\| < \varepsilon$ and $d > \sqrt{2}\varepsilon$, that implies $x_{\widetilde{u}}^2 + y_{\widetilde{u}}^2 < d^2/2$ and $-y_{\widetilde{u}} < \varepsilon$ and thus

$$4\varepsilon y_{\widetilde{c}} > d^2. \tag{1}$$

Equation (1) combined with the assumption that $d > 20\varepsilon$ implies

$$y_{\widetilde{c}} > 100\varepsilon. \tag{2}$$

The triangular inequality gives $\|\widetilde{v}\| \leq \|\widetilde{v} - \widetilde{u}\| + \|\widetilde{u}\|$. The later is smaller than $4d + \varepsilon < 5d$ by assumptions. So $\|\widetilde{v}\|^2 < 25d^2$ and by Equation 1 we obtain

$$\|\widetilde{v}\|^2 < 100\varepsilon y_{\widetilde{c}}. \tag{3}$$

Equation (3) and Equation (2) imply

$$\|\widetilde{v}\| < y_{\widetilde{c}}. \tag{4}$$

Now we prove $y_{\widetilde{v}} < 100\varepsilon$. Since $\widetilde{v}$ lies outside $\widetilde{D}$ then $(y_{\widetilde{c}} - y_{\widetilde{v}})^2 + x_{\widetilde{v}}^2 \geq y_{\widetilde{c}}^2 + d^2$, which simplifies to $y_{\widetilde{v}}^2 - 2y_{\widetilde{c}}y_{\widetilde{v}} + x_{\widetilde{v}}^2 - d^2 \geq 0$. We study this inequality to derive a bound on $y_{\widetilde{v}}$. Equation (4) implies $4(y_{\widetilde{c}}^2 + d^2 - x_{\widetilde{v}}^2) > 0$ hence the polynomial $X^2 - 2y_{\widetilde{c}}X + x_{\widetilde{v}}^2 - d^2$ univariate in X admits two real roots $y_{\widetilde{c}} \pm \sqrt{y_{\widetilde{c}}^2 + d^2 - x_{\widetilde{v}}^2}$. Equation (4) enforces $y_{\widetilde{v}} \leq y_{\widetilde{c}} - \sqrt{y_{\widetilde{c}}^2 + d^2 - x_{\widetilde{v}}^2}$, which implies

$$y_{\widetilde{v}} < y_{\widetilde{c}} \left(1 - \sqrt{1 - x_{\widetilde{v}}^2 / y_{\widetilde{c}}^2}\right).$$

Equation (3) and Equation (2) successively infer

$$y_{\widetilde{v}} < y_{\widetilde{c}} \left(1 - \sqrt{1 - 100\varepsilon / y_{\widetilde{c}}}\right) \leq 100\varepsilon.$$

That proves the lemma. □

6 Proof of Lemma 7

Proof. Assume there are $k, k' \in \{1, \ldots, m\}$ such that $k < k'$ and $e_k = e_{k'}$. Let S_{k-1}, S_k and $S_{k'-1}$ be the standard triangular surfaces associated to the lifts of T_{k-1}, T_k and $T_{k'-1}$, respectively. Consider the edge f of T_k resulting from the Delaunay flip of the edge e_k in T_{k-1}. Let $p \in \mathbb{T}_\Gamma$ be the intersection point of the interiors of f and e_k. Let also $\widetilde{p} \in \mathbb{R}^2$ be a lift of p.

Since $e_k = e_{k'}$ then $\pi_{S_{k-1}}(\widetilde{p}) = \pi_{S_{k'-1}}(\widetilde{p})$. By Lemma 2 $S_{k'-1}$ is above S_k and S_k is above S_{k-1}. We deduce $\pi_{S_k}(p) = \pi_{S_{k-1}}(\widetilde{p}) = \pi_{S_{k'-1}}(\widetilde{p})$. But Lemma 2 also gives $\pi_{S_k}(\widetilde{p}) \neq \pi_{S_{k-1}}(\widetilde{p})$ hence a contradiction. □

7 Proof of Lemma 8

Proof. Let $\Lambda(T')$ be the maximum length of an edge of T' and assume that there is an edge e of T such that $l(e) > 2\Lambda(T')$. Consider a lift $\widetilde{e}$ of e and let $\widetilde{p} \in \mathbb{R}^2$ be the middlepoint of $\widetilde{e}$. There is a face $\widetilde{t'}$ of the lift $\widetilde{T'}$ of T' such that $\widetilde{p}$ belongs either to $\widetilde{t'}$ or to the boundary of $\widetilde{t'}$. The three edges of the triangle $\widetilde{t'}$ are not longer than $\Lambda(T)$ so, by the triangular inequality, the distance from $\widetilde{p}$ to any vertex of $\widetilde{t'}$ is not greater than $\Lambda(T)$ and the *closed* disk $\widetilde{D} \subset \mathbb{R}^2$ of diameter $\Lambda(T)$ and centered at $\widetilde{p}$ contains $\widetilde{t'}$. Also the two endpoints $\widetilde{u}$ and $\widetilde{v}$ of $\widetilde{e}$ lie outside $\widetilde{D}$.

Consider the standard triangular surfaces S and S' associated to the lifts of T and T', respectively. The projection $\pi(\partial\widetilde{D})$ of the boundary $\partial\widetilde{D}$ of $\widetilde{D}$ is the intersection with $\mathbb{S}_2 \setminus P$ of a plane $\mathcal{P} \subset \mathbb{R}^3$. The plane $\mathcal{P}$ bounds two *open* half-spaces. The points $\pi(\widetilde{u})$ and $\pi(\widetilde{v})$ both belong to the half-space R that contains P. Thus $\pi_S(\widetilde{p}) \in R$. The vertices $\widetilde{w_1}, \widetilde{w_2}$ and $\widetilde{w_3}$ of $\widetilde{t'}$ all belong to $\partial\widetilde{D}$ thus $\pi(\widetilde{w_1}), \pi(\widetilde{w_2})$ and $\pi(\widetilde{w_3})$ all belong to $\mathcal{P}$ and $\pi_{S'}(\widetilde{p}) \in \mathcal{P}$. Consequently $\pi_{S'}(\widetilde{p})$ does not lie on the segment $[P, \pi_S(\widetilde{p})]$ of $\mathbb{R}^3$, contradicting Lemma 2. □

Computing Batched Depth Queries and the Depth of a Set of Points

Stephane Durocher* Alexandre Leblanc† Sachini Rajapakse*

Abstract

Simplicial depth and Tukey depth are two common measures for expressing the depth of a point q relative to a set P of points in $\mathbb{R}^d$. We introduce definitions that generalize these notions to express the depth of a set Q of points relative to a set P of points in $\mathbb{R}^d$, and we examine algorithms for computing these in $\mathbb{R}^2$, capitalizing on the relative cardinalities of P and Q.

1 Introduction

Depth measures quantify the centrality of an object relative to a set of objects. For univariate quantitative data, a natural definition for the depth of a point q relative to a set P of points in $\mathbb{R}$ is to measure how deeply nested q is in P by the lesser of the number of points of P less than q, and the number of points of P greater than q. By this measure, outliers relative to P have low depth, whereas a median of P has maximum depth. Various generalizations to higher dimensions exist, including simplicial depth and Tukey depth.

The *simplicial depth* of a query point q relative to a set P of points is the number of simplices determined by points in P that contain q:

Definition 1.1 (Simplicial depth [14]) *Given a set P of n points in $\mathbb{R}^d$ and a point q in $\mathbb{R}^d$, the* simplicial depth *of q relative to P is*

$$SD_P(q) = \sum_{S \in \mathcal{S}} I(q \in S), \tag{1}$$

where $\mathcal{S}$ denotes the set of $\binom{n}{d+1}$ closed simplices, each of which is the convex hull of $d+1$ points from P, and I is an indicator function such that $I(A) = 1$ if A is true and $I(A) = 0$ otherwise.

The *Tukey depth* of a query point q relative to a set P of points is the minimum number of points of P in any closed half-space containing q:

Definition 1.2 (Tukey depth [19]) *Given a set P of n points in $\mathbb{R}^d$ and a point q in $\mathbb{R}^d$, the* Tukey depth *(or* half-space depth*) of q relative to P is*

$$TD_P(q) = \min_{\substack{H \in \mathcal{H} \\ H \cap q \neq \varnothing}} |H \cap P|, \tag{2}$$

where $\mathcal{H}$ is the set of all closed half-spaces in $\mathbb{R}^d$.

Given q and P in $\mathbb{R}^2$, the simplicial depth and the Tukey depth of q relative to P can be computed in $O(n \log n)$ time, respectively, where $n = |P|$ [9, 17, 10], both of which have matching lower bounds of $\Omega(n \log n)$ worst-case time [1].

A *depth median* of a set P is a point of maximum depth relative to P for a given depth measure. We refer to a *simplicial median* and *Tukey median*, which can be computed in $O(n^4)$ time [2] and $O(n \log^3 n)$ time [12] in $\mathbb{R}^2$, respectively. An *in-sample median* of P is a point of P with maximum depth, which can be identified in $O(n^2)$ time in $\mathbb{R}^2$ for simplicial depth [9].

Depth measures are typically defined to describe the location of a single query point (an individual) relative to a set of points (a population). In this paper, we examine (1) computing a batch of depth queries relative to a common set of points, and (2) deriving a single estimator for the depth of a set of query points relative to another set of points.

Computing a batch of depth queries by iteratively running an algorithm designed to calculate the depth of a single query point can be inefficient. To address this, we present algorithms to compute a batch of depth queries; the choice of which algorithm to apply to minimize running time depends on the relative cardinalities of the query point set Q to the input point set P. Defining and evaluating the depths of a set of query points has various applications in data analysis, e.g., finding a center-outward ordering of a set Q relative to a set P. Next, we derive a single estimator to express the depth of Q relative to P. Applications include (1) measuring the centrality of Q relative to P (e.g., the position of one soccer team relative to the opposing team), (2) classifying a set Q selected from the same distribution as the sets $P_1, \ldots, P_m$ to determine within which set P_i the set Q is most deeply contained.

Our results In Section 3.1 we present three algorithms for computing a batch of k simplicial depth queries in $\mathbb{R}^2$ in $O(kn \log n)$ time, $O(n^2 + nk)$ time, and $O(n^4 + k \log n)$, respectively. The first algorithm is fastest when

*Department of Computer Science, University of Manitoba, stephane.durocher@umanitoba.ca, rajapak1@myumanitoba.ca

†Department of Statistics, University of Manitoba, alex.leblanc@umanitoba.ca

$k \in O(\frac{n}{\log n})$, the second when $k \in \Omega(\frac{n}{\log n})$ and $k \in O(n^3)$, and the third when $k \in \Omega(n^3)$. In Section 3.2 we present two algorithms for computing a batch of k Tukey depth queries in $\mathbb{R}^2$ in $O(kn \log n)$ time and $O(n^2 + k \log n)$ time, respectively. The first algorithm is fastest when $k \in O(\frac{n}{\log n})$, and the second when $k \in \Omega(\frac{n}{\log n})$.

In Section 4 we introduce definitions for the simplicial depth and Tukey depth of a set Q relative to a set P, which can be computed in $\mathbb{R}^2$ by applying the algorithms above. Finally, we examine properties and probabilistic interpretations for the simplicial depth of a set of points.

2 Related Work

2.1 Simplicial Depth and Simplicial Median

Multiple algorithms compute the simplicial depth of a point q relative to a set P of n points in $\mathbb{R}^2$ in $O(n \log n)$ time [9, 17, 10]. Given the radial ordering of P around q, the simplicial depth of q can be computed in $O(n)$ time [9]. Given a set $P = \{p_1, \ldots, p_n\}$ of points in $\mathbb{R}^2$, Lee and Ching [13] showed that the radial order of $P \setminus \{p_i\}$ with respect to p_i for all $i \in \{1, \ldots, n\}$ can be determined in $O(n^2)$ time. Consequently, the simplicial depths of all points in P can be obtained in $O(n^2)$ time [9], and an in-sample simplicial median can be identified in $O(n^2)$ time [9]. Khuller and Mitchell studied a similar problem independently [10].

When defined in terms of closed simplices, a simplicial median lies at an intersection of simplex boundaries [2]. Rousseeuw and Ruts described how to find a simplicial median by searching the set of intersection points in $O(n^5 \log n)$ time [17]. Aloupis et al. [2] derived a faster algorithm to compute a simplicial median in $O(n^4 \log n)$ time, which they further reduced to $O(n^4)$ time. We apply a technique similar to that of Aloupis et al. [2] in Algorithm S.III in Section 3.1.

2.2 Tukey Depth and Tukey Median

The Tukey depth of a point q relative to a set P of n points in $\mathbb{R}^2$ can be computed in $O(n \log n)$ time [17]. Tukey depth contours are a collection of nested polygons that partition the plane into regions of equal Tukey depth, which can be computed in $O(n^2)$ time [15]. A Tukey median can be found in $O(n \log^3 n)$ time [12].

2.3 Depth of a Set of Points

Recently, Pilz and Schnider introduced a definition for the Tukey depth of a set of points [16]:

Definition 2.1 (Generalized Tukey depth [16]) *The* generalized Tukey depth *of a set $Q \subseteq \mathbb{R}^d$ with respect to a set $P \subseteq \mathbb{R}^d$ is*

$$GTD_P(Q) = \min_{\substack{H \in \mathcal{H} \\ Q \cap H \neq \varnothing}} \frac{|H \cap P|}{|H \cap Q|}, \tag{3}$$

where $\mathcal{H}$ is the set of all closed half-spaces in $\mathbb{R}^d$.

Definition 2.1 differs from our Definition 4.2 introduced in Section 4.2. Definition 2.1 selects a single non-empty half-space that minimizes the ratio (3), i.e., the number of points of P in the half-space H relative to the number of points of Q in H. On the other hand, Definition 4.2 incorporates the respective Tukey depths for each point in Q, i.e., different half-spaces may be selected for each point.

Depth histograms provide a characterization of the combinatorial structure of a point set [6, 4]. Bertschinger et al. studied Tukey depth histograms of k-flats [4] and defined variations of Tukey depth for a set Q relative to P, including affine Tukey depth and convex Tukey depth.

Recently, Barba et al. [3] introduced a definition for the cardinal simplicial depth[1] of a set of points:

Definition 2.2 (Cardinal simplicial depth [3]) *The* cardinal simplicial depth *of a set $Q \subseteq \mathbb{R}^d$ with respect to a set $P \subseteq \mathbb{R}^d$ is*

$$CSD_P(Q) = \sum_{S \in \mathcal{S}} I(Q \cap S \neq \varnothing), \tag{4}$$

where $\mathcal{S}$ denotes the set of $\binom{n}{d+1}$ closed simplices, each of which is the convex hull of $d+1$ points from P, and I is an indicator function such that $I(A) = 1$ if A is true and $I(A) = 0$ otherwise.

Definition 2.2 differs from our Definition 4.1 introduced in Section 4.1. Definition 2.2 counts the number of non-empty simplicies (the cardinality of the set of non-empty simplicies), whereas Definition 4.1 is a normalized sum of the number of points of Q contained in each simplex. See further discussion in Section 4.1. Barba et al. gave an algorithm to compute $CSD_P(Q)$ for given sets P and Q in $O(N^{7/3} \log^{O(1)} N)$ time, where $N = |P| + |Q| = n + k$.

3 Computing a Batch of Depth Queries

In this section, we describe algorithms that compute a batch of simplicial depth queries or Tukey depth queries for k points in a set Q relative to a set P of n points, where $P \cup Q$ is in general position in $\mathbb{R}^2$. For simplicial depth we propose three algorithms: Algorithm S.I is not

[1] To disambiguate between Definitions 2.2 and 4.1, we refer to Definition 2.2 as the *cardinal simplicial depth* because it corresponds to the cardinality of the set of non-empty simplicies.

new [9, 17, 10]; Algorithms S.II and S.III are new. For Tukey depth we propose two algorithms: Algorithms T.I and T.II apply techniques used in existing algorithms for Tukey depth and Tukey depth contours.

3.1 Computing a Batch of Simplicial Depth Queries

3.1.1 Algorithm S.I

In Section 2.1 we mentioned algorithms for computing the simplicial depth of a single query point q relative to a set P of n points in $\mathbb{R}^2$ in $O(n \log n)$ time [9, 17, 10]. When the number of query points k is small relative to n, a straightforward approach for computing the depths of k points is to iteratively compute the simplicial depth of each query point using one of these existing algorithms. Using this approach, we can compute the simplicial depth of all k points in $O(kn \log n)$ time and $O(n)$ space to store the angular order of P around each query point (this space is reused for each query point). Due to the lower bound of $\Omega(n \log n)$ on the worst-case time required for computing the simplicial depth of a single point [1], this approach is optimal when $k \in O(1)$.

Lemma 1 *Given a set P of n points and a set Q of k query points in general position in $\mathbb{R}^2$, Algorithm S.I computes $SD_P(q)$ for every $q \in Q$ in $O(kn \log n)$ total time and $O(n + k)$ space.*

3.1.2 Algorithm S.II

Algorithm S.I is efficient when k is small relative to n, but more efficient approaches are possible for larger values of k. We describe an algorithm that computes the simplicial depths of points in Q relative to P in $O(n^2 + nk)$ time and $O(n^2)$ space. Using an approach similar to the in-sample simplicial median algorithm of Gil et al. [9] (Step 1), we compute the radial order of the n points of P around each point in Q, and (Step 2) we use this ordering to compute the simplicial depth of each point in Q.

To perform Step 1, we modify the method described by Gil et al. [9] and Khuller and Mitchell [10]. First, the sets P and Q are transformed into sets of lines L_P and L_Q in the dual plane, respectively. The sorted order of P around a point q can be obtained by considering the intersection order of L_P with the dual-line L_q using a method described in [13]. This step requires $O(n)$ time for each point in Q. The planar graph construction method in [5] can be implemented to find the line intersection order of L_P set with each line in L_q. We construct a graph G of the arrangement of lines induced by L_P incrementally by introducing one line at a time, and construct the doubly connected edge list of L_P, which requires $O(n^2)$ time and $O(n^2)$ space. Then we continue this process by temporarily inserting each line in L_q to G, and finding the order of intersections of lines in L_P with L_q by traversing the sequence of edges in G along L_q, which takes $O(n)$ time. Then, applying a method analogous to that described in [13], the angular sorted order of P around each point q can be obtained in $O(n)$ time. Step 1 requires $O(n^2)$ time and $O(n^2)$ space for preprocessing. In Step 2, the simplicial depth of each point $q \in Q$ relative to P can be found in $O(n)$ time using the angular order of points of P around q [9]. This takes $O(nk)$ time, giving a total time of $O(n^2 + nk)$.

Step 1 requires finding the order of intersections between L_P and each line in L_q. Finding the order of intersections between one line and a set of m lines can be achieved using one of various methods: (a) incremental planar graph construction [5] in $O(m^2)$ time and $O(m^2)$ space, (b) line sweeping [18] in $O(m^2 \log m)$ time [8], or (c) topological sweeping [7] in $O(m^2)$ time and $O(m)$ space. Despite its lower costs as a function of m, when applied to our problem, topological sweeping takes $O(n^2 + k^2)$ time and $O(n + k)$ space because it processes additional intersections in L_P and L_Q that are not needed for Step 1. The most efficient method for finding the ordered intersections between L_P and each L_q line is incremental planar graph construction, which takes $O(n^2 + nk)$ time and $O(n^2)$ space.

Lemma 2 *Given a set P of n points and a set Q of k query points in general position in $\mathbb{R}^2$, Algorithm S.II computes $SD_P(q)$ for every $q \in Q$ in $O(n^2 + nk)$ total time and $O(n^2 + k)$ space.*

3.1.3 Algorithm S.III

When k is large relative to n, construct the arrangement L formed by lines between every pair of points in P. This arrangement partitions the plane into $\Theta(n^4)$ convex cells in which every point within a cell has equal simplicial depth. By modifying the $O(n^4)$-time simplicial median algorithm of Aloupis et al. [2], we can compute the depths of all cells in $O(n^4)$ time. Aloupis et al. consider the arrangement of line segments connecting every pair of points in P, which also has $O(n^4)$ intersections and $O(n^4)$ cells. This method computes the number of points on each side of each line segment of P in $O(n^3)$ time. Further, Aloupis et al. showed that starting from a known depth value on a line segment, by processing each intersection point in $O(1)$ time, the simplicial depth along the line segment can be computed in $O(n)$ time [2]. We adapt this depth-finding method along a line segment to find the simplicial depth of cells in our arrangement L as described below.

Each line l in L is partitioned into three by the two points p_1 and p_2 in P that determine l: the line segment between p_1 and p_2 (colour this segment blue) and two rays (colour the rays red) on l rooted at p_1 and p_2, respectively. In the arrangement determined by L, only the blue segments are boundaries of simplices. There-

fore, when crossing from one cell to an adjacent cell, the depth changes if the two cells share a blue line segment on their common boundary. Similarly, if the two cells share a red segment on their common boundary, then both cells have the same simplicial depth.

We compute the number of points on each side of each line in L in $O(n^3)$ time. Starting from a cell C_i with known simplicial depth, the algorithm traverses the arrangement, calculating the simplicial depth of each cell relative to the depth of an adjacent cell whose depth was already computed. To find the simplicial depth of a cell C_j that shares a blue edge with C_i, subtract the number of points in P on the side of C_i to the blue edge and add the number of points in P on the side C_j to the blue edge. The simplicial depth inside C_i includes simplicies (triangles) bounded by the blue line segment and points in P on the side of C_i. When crossing the blue edge to C_j, we exit (subtract) one set of triangles and enter (add) a new set of triangles based on the blue line segment and points of P on the C_j side of the blue edge. If depth on simplex boundaries is required, then the depth on the blue edge is calculated by adding depth in C_i to the number of points in the side of C_j; no query point lies on a simplex boundary when $P \cup Q$ is in general position.

All cells outside the convex hull of P have depth zero; we can initiate our algorithm at any of these cells. The algorithm proceeds to compute the depths of all cells by traversing the planar graph determined by L starting from an extreme cell (with depth zero) using the technique described above. The depth of each individual cell is computed in $O(1)$ time. Therefore, the traversal takes time and space proportional to the number of cells: $\Theta(n^4)$.

Finally, for each point q in Q we apply a point location algorithm to identify the cell in the arrangement determined by L that contains q. Kirkpatrick's point location algorithm can be implemented in a t-edge planar subdivision with $O(t)$ preprocessing time, $O(t)$ space, and $O(\log t)$ query time [11]. In our case, $t \in \Theta(n^4)$, corresponding to $\Theta(n^4)$ cells in the planar subdivision determined by L (the number of edges is also $\Theta(n^4)$). Therefore, Kirkpatrick's point location algorithm can be used to find the locations of each point in Q in $O(n^4)$ preprocessing time, $O(n^4)$ space and $O(k \log n)$ query time. The simplicial depths of all points in Q can be computed in $O(n^4 + k \log n)$ time and $O(n^4)$ space.

Lemma 3 *Given a set P of n points and a set Q of k query points in general position in $\mathbb{R}^2$, Algorithm S.III computes $SD_P(q)$ for every $q \in Q$ in $O(n^4 + k \log n)$ total time and $O(n^4 + k)$ space.*

Lemmas 1–3 give:

Theorem 4 *Given a set P of n points and a set Q of k query points in general position in $\mathbb{R}^2$, the simplicial depths of points in Q with respect to P can be computed in $O(\min\{kn \log n, n^2 + nk, n^4 + k \log n\})$ time.*

3.2 Computing a Batch of Tukey Depth Queries

In this section, we describe two methods for computing a batch of Tukey depth queries based on previous work related to computing Tukey depth [17] and Tukey depth contours [15].

3.2.1 Algorithm T.I

In $\mathbb{R}^2$, the Tukey depth of a point q relative to a set P of n points can be computed in $O(n \log n)$ time [17]. Similar to Algorithm S.I in Section 3.1, a straightforward method for computing the Tukey depths of k query point is to apply a Tukey depth algorithm iteratively for each point of Q. This process take $O(kn \log n)$ time and $O(n)$ space to store the sorted order of P around each point of Q.

Lemma 5 *Given a set P of n points and a set Q of k query points in general position in $\mathbb{R}^2$, Algorithm T.I computes $TD_P(q)$ for every $q \in Q$ in $O(kn \log n)$ total time and $O(n + k)$ space.*

3.2.2 Algorithm T.II

Algorithm T.I is efficient when k is small relative to n, but more efficient approaches are possible for larger values of k. Algorithm T.II begins by computing the Tukey depth contours of P using the algorithm of Miller et al. in $O(n^2)$ time and space [15]. Miller et al. showed how to build a point location data structure on the contours in $O(n^2)$ time to support $O(\log n)$-time Tukey depth queries. Therefore, the Tukey depths of k points can be calculated in $O(n^2 + k \log n)$ time and $O(n^2)$ space.

Lemma 6 *Given a set P of n points and a set Q of k query points in general position in $\mathbb{R}^2$, Algorithm T.II computes $TD_P(q)$ for every $q \in Q$ in $O(n^2 + k \log n)$ total time and $O(n^2 + k)$ space.*

Lemmas 5 and 6 give:

Theorem 7 *Given a set P of n points and a set Q of k query points in general position in $\mathbb{R}^2$, the Tukey depths of points in Q with respect to P can be computed in $O(\min\{kn \log n, n^2 + k \log n\})$ time.*

4 Depth of a Set of Query Points

We introduce definitions for the simplicial depth and Tukey depth of a *set* Q of points relative to a set P of points. As we discuss below, our new definitions differ from previous definitions introduced by Barba et al. [3] and Pilz and Schnider [16].

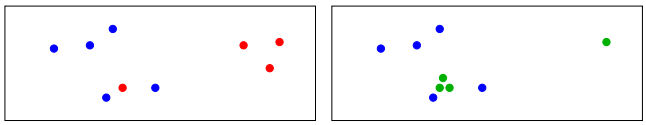


Figure 1: Relative to the blue set, the green and red sets have the same cardinal simplicial depth (Definition 2.2). However, by Definition 4.1, the simplicial depth of the green set is triple that of the red set. An analogous property holds for Tukey depth: the green and red sets have the same generalized Tukey depth (by Definition 2.1) relative to the blue set, but their Tukey depths differ (by Definition 4.2).

4.1 Simplicial Depth of a Set of Query Points

We define the simplicial depth of a set Q relative to a set P as the normalized sum of the number of points of Q contained in each simplex determined by points in P:

Definition 4.1 (Simplicial depth of a set of points) *Given a set P of n points and a set Q of k points in $\mathbb{R}^d$, the* simplicial depth *of Q relative to P is*

$$SD^*_P(Q) = \frac{1}{|Q|} \sum_{S \in \mathcal{S}} |Q \cap S|, \tag{5}$$

where $\mathcal{S}$ denotes the set of $\binom{n}{d+1}$ closed simplicies, each of which is the convex hull of $d+1$ points from P.

$SD^*_P(Q)$ can be expressed as the average simplicial depth of points in Q:

$$SD^*_P(Q) = \frac{1}{|Q|} \sum_{q \in Q} SD_P(q). \tag{6}$$

A derivation of (6) is given in Appendix A. (6) implies that $SD^*_P(Q)$ also has a natural probabilistic interpretation. If q is selected uniformly at random from Q, the expected value of the simplicial depth of q relative to P is $SD^*_P(Q)$.

Definition 4.1 differs from $CSD_P(Q)$ (Definition 2.2) introduced by Barba et al. [3]. $CSD_P(Q)$ counts the number of non-empty simplicies, which can result in similar depth values for significantly different point sets. Specifically, a small subset of points in the set Q can determine the depth of Q relative to P. See Figure 1. On the other hand, Definition 4.1 is a normalized sum of the number of points contained in each simplex. Equation (6) also suggests a family of measures that can be used to define the simplicial depth of a set Q with respect to a set P by substituting the average with another summary statistic of the distribution of the depths of points in Q. We discuss this briefly in Section 5.

We can compute $SD^*_P(Q)$ by computing the simplicial depth $SD_P(q)$ for each point $q \in Q$, and taking the average of these depth values. This can be achieved efficiently using the algorithms introduced in Section 3.1, which gives the following corollary.

Corollary 8 *Given a set P of n points and a set Q of k points in general position in the plane, $SD^*_P(Q)$ can be computed in $O(\min\{kn \log n, n^2 + nk, n^4 + k \log n\})$ time.*

As mentioned earlier, $CSD_P(Q)$ can be computed in $O(N^{7/3} \log^{O(1)} N)$ time, where $N = n + k$. By Corollary 8, the simplicial depth, $SD^*_P(Q)$, introduced in this paper can be computed asymptotically faster for any values of n and k.

Next, we consider another generalization of simplicial depth to sets, which we show is equivalent to Definion 4.1. For this, we introduce the *normalized simplicial depth* (NSD) of a query point q relative to P as

$$NSD_P(q) = \frac{1}{|\mathcal{S}|} \sum_{S \in \mathcal{S}} I(q \in S) = \frac{SD_P(q)}{|\mathcal{S}|}, \tag{7}$$

that is, it is the proportion of simplices from $\mathcal{S}$ that contain q. Interestingly, this normalized simplicial depth can also be interpreted as the probability that the query point q lies in a simplex whose vertices are selected at random from P or, equivalently,

$$NSD_P(q) = \mathbb{P}(q \in S), \tag{8}$$

where S is selected uniformly at random from $\mathcal{S}$. Liu [14] argued that this is an estimator of the probability that the query point q lies in a simplex formed from $d+1$ independent random points selected from a common distribution F in $\mathbb{R}^d$.

Now, consider generalizing the idea described above by selecting a simplex at random from $\mathcal{S}$, but by instead focusing on the expected number of points of Q that lie in that simplex. This depth measure, which we denote $ERS_P(Q)$ (Expected number of points of Q in a Random Simplex from P) is then

$$ERS_P(Q) = \mathbb{E}[Y_Q(S)], \tag{9}$$

where S is again randomly selected from $\mathcal{S}$, and where the random variable $Y_Q(S)$ denotes the number of points of Q that lie inside S. This is a reasonable measure of the depth of Q with respect to P, has an elegant and intuitive interpretation, and reduces to (8) when Q contains a single point. Indeed, when Q contains a single point, $\mathbb{E}[Y_Q(S)] = \mathbb{E}[I(q \in S)] = \mathbb{P}(q \in S)$, the normalized simplicial depth of q. We now justify that ERS_P and SD^*_P are equivalent measures of depth.

The number of points of Q that lie inside a simplex S constructed from points of P can be expressed as

$$Y_Q(S) = \sum_{q \in Q} I(q \in S), \tag{10}$$

and takes values in $\{0, 1, \ldots |Q|\}$. Also, the proportion of simplices in $\mathcal{S}$ that contain exactly y points of Q is

$$P_{\mathcal{S}}(y) = \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} I\big[Y_Q(s) = y\big], \tag{11}$$

for $y = 0, 1, \ldots, |Q|$. This also corresponds to the probability that the simplex constructed from three points selected at random from P contains exactly y points of Q.

In this context, the expectation of $Y_Q(S)$, which corresponds to the mean of the probability distribution in (11), can be shown to satisfy (see Appendix A)

$$SD^*_P(Q) = \frac{|\mathcal{S}|}{|Q|} ERS_P(Q). \tag{12}$$

From this, the simplicial depth of Q, as defined in Definition 4.1, is equivalent to $ERS_P(Q)$, the expected number of points in Q that lie a randomly selected simplex constructed from points of P, as the two depth measures are always proportional to each other.

We conclude this section by highlighting how CSD_P, defined in (4) as the number of simplices constructed from points of P that contain at least one point of Q, relates to the discussion above. Specifically, it is possible to write (see Appendix A)

$$CSD_P(Q) = |\mathcal{S}|\, \mathbb{P}(Y_Q(S) > 0). \tag{13}$$

This implies that, as a measure of depth, $CSD_P(Q)$ is equivalent to $\mathbb{P}(Y_Q(S) > 0)$, the probability that a random simplex contains at least one point of Q. In the case where Q contains a single point, this further reduces to $\mathbb{P}(Y_Q(S) > 0) = \mathbb{P}(q \in S)$ and justifies that $CSD_P(Q)$ is also a direct generalization of simplicial depth that applies to sets, but differs from $ERS_P(Q)$.

4.2 Tukey Depth of a Set of Query Points

We define the Tukey depth of a set Q relative to a set P as follows:

Definition 4.2 (Tukey depth of a set of points) *Given a set P of n points and a set Q of k points in $\mathbb{R}^d$, the* Tukey depth *of Q relative to P is*

$$TD^*_P(Q) = \frac{1}{|Q|} \sum_{q \in Q} TD_P(q). \tag{14}$$

As with (6), (14) corresponds to the average Tukey depth of points in Q relative to P, and carries the same probabilistic interpretation as for simplicial depth: (14) corresponds to the expected depth of a point selected uniformly at random from Q.

To compute $TD^*_P(Q)$, we can compute the Tukey depth of each point in Q relative to P using the algorithms introduced in Section 3.2, and take the average of those depth values. Therefore, we have the following corollary.

Corollary 9 *Given a set P of n points and a set Q of k points in general position in the plane, $TD^*_P(Q)$ can be computed in $O(\min\{kn \log n, n^2 + k \log n\})$ time.*

As mentioned in Section 2.3, Pilz and Schnider [16] introduced the generalized Tukey depth of a set Q relative to a set P, $GTD_P(Q)$. This definition can result in similar depth values for significantly different point sets. Specifically, a small subset of points in the set Q can determine the depth of Q relative to P. See Figure 1. Pilz and Schnider did not provide an algorithm to compute $GTD_P(Q)$, but based on Definition 2.1, a straightforward iterative approach for computing $GTD_P(Q)$ would require $O(n^3 + k^3)$ time. This time can likely be reduced to $O(n^2 + k^2)$ time by constructing the arrangement of lines determined by pairs of points in $Q \cup P$, and traversing the arrangement to examine all possible subsets of $Q \cup P$ contained in a half-plane; traversing from one cell in the arrangement to a neighbouring cell corresponds to adding or removing $O(1)$ points from $Q \cup P$.

5 Discussion and Directions for Future Research

In this paper, we introduced new definitions for the simplicial depth and Tukey depth of a set Q of points relative to a set P of points in $\mathbb{R}^d$, and we presented algorithms for computing these in $\mathbb{R}^2$.

This work suggests various possible generalizations of simplicial depth and Tukey depth to measure the depth of a query set Q. As the computation of these depth measures involves computing the depth of each point in Q, we could instead define a depth measure as a function of a different summary of the distributions of the simplicial depths and Tukey depths of individual points of Q relative to P. For instance, we could summarize the distribution of depths using a median, a minimum, a maximum, or a measure of spread, such as variance, range, skewness, or quantiles of this distribution. These different summaries of the constructed depth distributions over the points of Q can all be computed in the same time and space complexities as in Corollaries 8 and 9. One could also define the depth of a set using another depth for individual points altogether.

Future work is warranted to investigate the characterization of these depth measures of sets of points. SD^*_P and TD^*_P are invariant under affine transformations and vanish at infinity. TD^*_P is consistent across dimensions. Other properties such as convexity, stability, and robustness remain to be analyzed, requiring appropriate generalizations for the depth of a set of points. Finally, some questions remain unanswered with respect to the possibility of improving the running times of the algorithms presented in Theorems 4 and 7. In particular, can we show corresponding lower bounds on the worst-case running time expressed in terms of n and k?

References

[1] G. Aloupis, C. Cortés, F. Gómez, M. Soss, and G. Toussaint. Lower bounds for computing statistical depth. *Computational Statistics & Data Analysis*, 40(2):223–229, 2002.

[2] G. Aloupis, S. Langerman, M. Soss, and G. Toussaint. Algorithms for bivariate medians and a Fermat-Torricelli problem for lines. *Computational Geometry*, 26(1):69–79, 2003.

[3] L. Barba, S. Lochau, A. Pilz, and P. Schnider. Simplicial depth for multiple query points. In *Proc. European Workshop on Computational Geometry*, pages 29:1–29:7, 2019.

[4] D. Bertschinger, J. Passweg, and P. Schnider. Tukey depth histograms. *arXiv preprint arXiv:2103.08665*, 2021.

[5] B. Chazelle, L. J. Guibas, and D.-T. Lee. The power of geometric duality. *BIT Numerical Mathematics*, 25(1):76–90, 1985.

[6] S. Durocher, R. Fraser, A. Leblanc, J. Morrison, and M. Skala. On combinatorial depth measures. *International Journal of Computational Geometry & Applications*, 28(04):381–398, 2018.

[7] H. Edelsbrunner and L. J. Guibas. Topologically sweeping an arrangement. *Journal of Computer and System Sciences*, 38(1):165–194, 1989.

[8] H. Edelsbrunner and E. Welzl. Constructing belts in two-dimensional arrangements with applications. *SIAM Journal on Computing*, 15(1):271–284, 1986.

[9] J. Gil, W. Steiger, and A. Wigderson. Geometric medians. *Discrete Mathematics*, 108(1-3):37–51, 1992.

[10] S. Khuller and J. S. Mitchell. On a triangle counting problem. *Information Processing Letters*, 33(6):319–321, 1990.

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

[12] S. Langerman and W. Steiger. Optimization in arrangements. In *Proc. Symposium on Theoretical Aspects of Computer Science*, pages 50–61. Springer, 2003.

[13] D. T. Lee and Y.-T. Ching. The power of geometric duality revisited. *Information Processing Letters*, 21(3):117–122, 1985.

[14] R. Y. Liu. On a notion of data depth based on random simplices. *The Annals of Statistics*, pages 405–414, 1990.

[15] K. Miller, S. Ramaswami, P. Rousseeuw, J. A. Sellares, D. Souvaine, I. Streinu, and A. Struyf. Efficient computation of location depth contours by methods of computational geometry. *Statistics and Computing*, 13(2):153–162, 2003.

[16] A. Pilz and P. Schnider. Extending the centerpoint theorem to multiple points. *Leibniz International Proceedings in Informatics*, 123:53–1, 2018.

[17] P. J. Rousseeuw and I. Ruts. Bivariate location depth. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 45(4):516–526, 1996.

[18] M. I. Shamos and D. Hoey. Geometric intersection problems. In *Proc. Symposium on Foundations of Computer Science*, pages 208–215. IEEE, 1976.

[19] J. W. Tukey. Mathematics and the picturing of data. In *Proc. International Congress of Mathematicians*, volume 2, pages 523–531, 1975.

A Proofs

In this Appendix we include complete details of proofs and arguments omitted from the main text due to space constraints.

Derivation of Equation (6)**:** Starting with Definition 4.1, and noting that

$$|Q \cap S| = \sum_{q \in Q} I(q \in S),$$

we see that

$$\begin{aligned} SD^*_P(Q) &= \frac{1}{|Q|} \sum_{S \in \mathcal{S}} |Q \cap S| \\ &= \frac{1}{|Q|} \sum_{S \in \mathcal{S}} \sum_{q \in Q} I(q \in S) \\ &= \frac{1}{|Q|} \sum_{q \in Q} \sum_{S \in \mathcal{S}} I(q \in S) \\ &= \frac{1}{|Q|} \sum_{q \in Q} SD_P(q), \end{aligned}$$

as claimed.

Derivation of Equation (12)**:** To avoid confusion in what follows, we reserve S to denote a randomly selected simplex and use s otherwise. First, we note that

$$\begin{aligned} \mathbb{E}[Y_Q(S)] &= \sum_{y=0}^{|Q|} y\, P_{\mathcal{S}}(y) \\ &= \sum_{y=0}^{|Q|} \frac{y}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} I\big[Y_Q(s) = y\big] \\ &= \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \sum_{y=0}^{|Q|} y\, I\big[Y_Q(s) = y\big] \\ &= \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} Y_Q(s). \end{aligned} \tag{15}$$

Now, using (10), we can further simplify (15) to get

$$\begin{aligned} \mathbb{E}[Y_Q(S)] &= \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \sum_{q \in Q} I(q \in s) \\ &= \frac{1}{|\mathcal{S}|} \sum_{q \in Q} \sum_{s \in \mathcal{S}} I(q \in s) \\ &= \frac{1}{|\mathcal{S}|} \sum_{q \in Q} SD_P(q) \\ &= \frac{|Q|}{|\mathcal{S}|}\, SD^*_P(Q). \end{aligned} \tag{16}$$

Finally, (9) and (16) together imply that

$$ERS_P(Q) = \frac{|Q|}{|\mathcal{S}|} SD^*_P(Q),$$

which is equivalent to (12).

Derivation of Equation (13)**:** First, we write

$$CSD_P(Q) = \sum_{s \in \mathcal{S}} I\big[Y_Q(s) > 0\big].$$

Then, making use of (11), derivations similar to those provided above allow one to see that

$$\begin{aligned} CSD_P(Q) &= \sum_{s \in \mathcal{S}} \sum_{y=1}^{|Q|} I\big[Y_Q(s) = y\big] \\ &= \sum_{s \in \mathcal{S}} \sum_{y=0}^{|Q|} I\big[y > 0\big] I\big[Y_Q(s) = y\big] \\ &= \sum_{y=0}^{|Q|} I\big[y > 0\big] \sum_{s \in \mathcal{S}} I\big[Y_Q(s) = y\big] \\ &= |\mathcal{S}| \sum_{y=0}^{|Q|} I\big[y > 0\big] P_{\mathcal{S}}(y) \\ &= |\mathcal{S}|\, \mathbb{E}\big[I(Y_Q(S) > 0)\big] \\ &= |\mathcal{S}|\, \mathbb{P}(Y_Q(S) > 0), \end{aligned}$$

as claimed.

Curve Stabbing Depth: Data Depth for Plane Curves*

Stephane Durocher† Spencer Szabados†

Abstract

Measures of data depth have been studied extensively for point data. Motivated by recent work on analysis, clustering, and identifying representative elements in sets of trajectories, we introduce *curve stabbing depth* to quantify how deeply a given curve Q is located relative to a given set $\mathcal{C}$ of curves in $\mathbb{R}^2$. Curve stabbing depth evaluates the average number of elements of $\mathcal{C}$ stabbed by rays rooted on Q. We describe an $O(nm^2 + n^2 m \log^2 m)$-time algorithm for computing curve stabbing depth when Q is an m-vertex polyline and $\mathcal{C}$ is a set of n polylines, each with $O(m)$ vertices.

1 Introduction

Processes that generate functional or curve data are becoming increasingly common within various domains, including medicine (e.g., ECG signals [16] and analysis of nerve fibres in brain scans [11]), GIS techniques for generating and processing positional trajectory data (e.g., tracking migratory animal paths [4], air traffic control [8], and clustering of motion capture data [12]), and in the food industry (e.g., classification of nutritional information via spectrometric data [13]). In this paper, we consider depth measures for curve data.

Traditional depth measures are defined on multidimensional point data and seek to quantify the centrality or the outlyingness of a given object relative to a set of objects or to a sample population. Common depth measures include simplicial depth [18], Tukey (half-space) depth [23], Oja depth [20], convex hull peeling depth [3], and regression depth [21]. See [19] and [22] for further discussion on depth measures for multivariate point data. Previous work exists defining depth measures for sets of functions and functional data [13, 10, 16, 9], often with a focus on classification. Despite the fact that curves can be expressed as functions, depth measures for functions typically do not generalize to curves, as they are often sensitive to the specific parameterization and most are restricted to functions whose range is $\mathbb{R}$, which can only represent x-monotone curves.

New methods are required for efficient analysis of trajectory and curve data. Recent work has examined identifying representative elements [4] and clustering in a set of trajectories [5, 12]. In this work, we introduce curve stabbing depth, a new depth measure defined in terms of stabbing rays to quantify the degree to which a given curve is nested within a given set of curves.

Our main contributions are:

- In Section 2, we define *curve stabbing depth*, a new depth measure for curves in $\mathbb{R}^2$, and we describe a general approach for evaluating the curve stabbing depth of a given curve Q relative to a set $\mathcal{C}$ of curves in $\mathbb{R}^2$.
- In Section 3, we present an $O(nm^2 + n^2 m \log^2 m)$-time algorithm for computing the curve stabbing depth of a given m-vertex polyline Q relative to a set $\mathcal{P}$ of n polylines in $\mathbb{R}^2$, each with $O(m)$ vertices.
- In Section 4, we discuss properties of a deepest curve (depth median) for curve stabbing depth, discuss the consistency of generalizations to higher dimensions, and outline possible directions for future research.

2 Definitions and Preliminary Analysis

Definition 1 (Plane Curve) *A* plane curve *is a continuous function* $Q : [0, 1] \to \mathbb{R}^2$.

Definition 2 (Polyline) *A* polyline *(polygonal chain) is a piecewise-linear curve consisting of the line segments* $\overline{p_1p_2}, \overline{p_2p_3}, \ldots, \overline{p_{m-1}p_m}$ *determined by the sequence of points* $(p_1, p_2 \ldots, p_m)$ *in* $\mathbb{R}^2$.

Definition 3 (Stabbing Number) *Given a ray* $\overrightarrow{q_\theta}$ *rooted at a point* q *in* $\mathbb{R}^2$ *that forms an angle* θ *with the* x*-axis, the* stabbing number *of* $\overrightarrow{q_\theta}$ *relative to a set* $\mathcal{C}$ *of plane curves, denoted* $\text{stab}_{\mathcal{C}}(\overrightarrow{q_\theta})$*, is the number of elements in* $\mathcal{C}$ *intersected by* $\overrightarrow{q_\theta}$.

Definition 4 (Curve Stabbing Depth) *Given a plane curve* Q *and a set* $\mathcal{C}$ *of plane curves, the* curve stabbing depth *of* Q *relative to* $\mathcal{C}$*, denoted* $D(Q, \mathcal{C})$*, is*

$$\frac{1}{\pi L(Q)} \int_{q \in Q} \int_0^{\pi} \min\{\text{stab}_{\mathcal{C}}(\overrightarrow{q_\theta}), \text{stab}_{\mathcal{C}}(\overrightarrow{q_{\theta+\pi}})\}\, d\theta\, ds, \quad \text{(D.c)}$$

where $L(Q) = \int_{q \in Q} ds$ *denotes the arc length of* Q.

*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, szabados@myumanitoba.ca

As $L(Q)$ approaches zero (the curve Q becomes a point q), the value of (D.c) approaches

$$\frac{1}{\pi}\int_0^{\pi} \min\{\text{stab}_{\mathcal{C}}(\overrightarrow{q\theta}), \text{stab}_{\mathcal{C}}(\overrightarrow{q_{\theta+\pi}})\}\, d\theta. \qquad \text{(D.p)}$$

Curve stabbing depth corresponds to the average depth of points $q \in Q$. The depth of a point q relative to $\mathcal{C}$, given by (D.p), is the average stabbing number in all directions θ around q, where for each θ, either the stabbing number of the ray $\overrightarrow{q\theta}$ or its reflection $\overrightarrow{q_{\theta+\pi}}$ is applied, generalizing the one-dimensional notion of depth that counts the lesser of the number of elements less than vs. greater than the query point (outward rank).

As a ray $\overrightarrow{q\theta}$ rotates about a point q, $\text{stab}_{\mathcal{C}}(\overrightarrow{q\theta})$ partitions the range $\theta \in [0, \pi)$ into intervals, such that for all values θ in a given interval, $\overrightarrow{q\theta}$ intersects the same subset of $\mathcal{C}$. These intervals partition the plane around q into *wedges*. We generalize this notion and define the wedges determined by a point q relative to a set $\mathcal{C}$ of curves.

Definition 5 (Wedge) *The* wedge *of the curve C relative to the point q is the region determined by all rays rooted at q that intersect C:*

$$w(q, C) = \bigcup_{\substack{\overrightarrow{q\theta} \cap C \neq \varnothing \\ \theta \in [0, 2\pi)}} \overrightarrow{q\theta}.$$

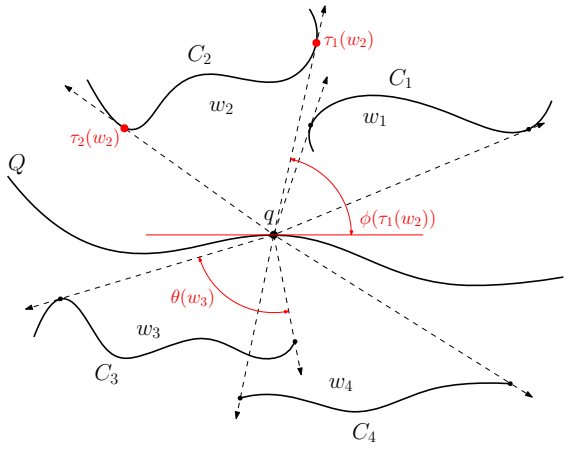


Figure 1: The set of wedges w_1, w_2, w_3, and w_4 induced by curves C_1, C_2, C_3, and C_4 rooted at the point q on the curve Q. Moving counterclockwise around q, the positive angle between $\tau_1(w_2)$ with the horizontal is indicated by $\phi(\tau_1(w_2))$, the tangent points of w_2 are labelled $\tau_1(w_2)$ and $\tau_2(w_2)$, and the internal angle of w_3 is highlighted by $\theta(w_3)$.

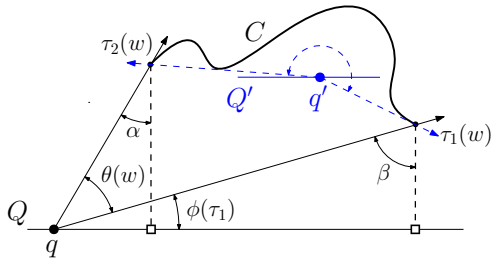
Definition 6 (Tangent Points) *When $C \cup \{q\}$ is in general position in $\mathbb{R}^2$, the* tangent points *of the wedge $w = w(q, C)$, denoted $\tau(w) = \{\tau_1, \tau_2\}$, are those points of C incident with the boundary of w; i.e., $\tau(w) = \partial w \cap C$, where ∂w denotes the boundary of w. (If C is a curve for which all rays from q intersect, the tangents points of $w(q, C)$ are taken to be coincident on C, with an internal wedge angle of 2π radians.) $\tau_1(w)$ denotes the tangent point that is the most clockwise of the two around q. The angles between the horizontal and each tangent point of w are denoted by $\phi(\tau_1(w))$ and $\phi(\tau_2(w))$, with $\theta(w)$ denoting the interior angle of w.*

See Figures 1 and 2. The sequence of wedges determines an ordering of the curves stabbed about a given point q. A ray $\overrightarrow{q\theta}$ always stabs the associated curve C as $\overrightarrow{q\theta}$ sweeps through the wedge determined by the extreme points of C. For a given set $\mathcal{C}$ of curves and associated wedges $\mathcal{W}_{\mathcal{C}}$ rooted at a common point q,

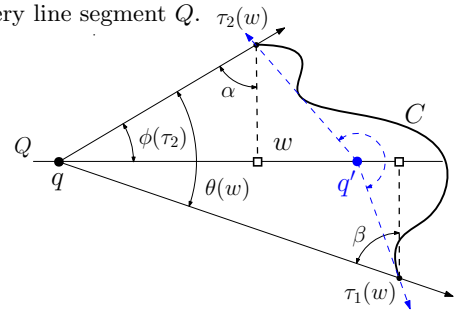
$$\text{stab}_{\mathcal{C}}(\overrightarrow{q\theta}) = |\{w \in \mathcal{W}_{\mathcal{C}} \mid \theta \in [\phi(\tau_1(w)), \phi(\tau_2(w))]\}|. \qquad (1)$$

That is, $\text{stab}_{\mathcal{C}}(\overrightarrow{q\theta})$ is the number of wedges that contain the ray $\overrightarrow{q\theta}$, where each wedge is associated with a curve in $\mathcal{C}$. See Figure 1.

Our algorithm for computing curve stabbing depth requires calculating the interior angle $\theta(w)$ of each wedge w, which we now describe. We consider two cases for the relative positions of a given query line segment Q, a curve C, and the wedge $w(q, C)$ rooted at a point q: (1) when $q \notin \text{CH}(C)$, where $\text{CH}(C)$ denotes the convex hull of C, i.e., Q does not pass through the interior of C, and (2) when $q \in \text{CH}(C)$. When points and curves are in general position, C cannot coincide with a bounding edge of w. See Figure 2.



(a) A curve C existing entirely above (below) the query line segment Q.



(b) A curve C crossing through the query line segment Q.

Figure 2: Two ways a query line segment C and a wedge rooted at a point on C can be arranged under general position. Case 1 is drawn in black while Case 2 is outlined in blue.

In Case 1, when C lies entirely above or below Q the angles formed between the tangent points, root, and horizontal can be evaluated as

$$\theta(w) + \phi(\tau_1) = \frac{\pi}{2} - \alpha \quad = \frac{\pi}{2} - \tan^{-1}\left(\left|\frac{q_x - \tau_2(w)_x}{\tau_2(w)_y - q_y}\right|\right),$$
$$\phi(\tau_1) = \frac{\pi}{2} - \beta \quad = \frac{\pi}{2} - \tan^{-1}\left(\left|\frac{q_x - \tau_1(w)_x}{\tau_1(w)_y - q_y}\right|\right).$$

Where the interior angle of w is found to be

$$\theta(w) = \tan^{-1}\left(\frac{q_x - \tau_2(w)_x}{\tau_2(w)_y - q_y}\right) - \tan^{-1}\left(\frac{q_x - \tau_1(w)_x}{\tau_1(w)_y - q_y}\right). \tag{A.1}$$

This can be done rather than evaluating distinct cases due to the order in which the signs of each inverse tangent change while q transitions past each dropped perpendicular. When C crosses in front of Q, as illustrated in Figure 2b, we calculate

$$\theta(w) = \pi - \left|\tan^{-1}\left(\frac{q_x - \tau_2(w)_x}{\tau_2(w)_y - q_y}\right) + \tan^{-1}\left(\frac{q_x - \tau_1(w)_x}{\tau_1(w)_y - q_y}\right)\right|. \tag{A.2}$$

Once q enters $\mathrm{CH}(C)$, we transition to Case 2, in which the calculations are similar to those of Case 1, except for modifications needed to account for taking an angle greater than π radians, as shown in Figure 2b in blue. Every case considered by our algorithm reduces to Case 1 or Case 2. We sometimes limit discussion to instances of Case 1 depicted in Figure 2a to simplify the presentation; our results apply to all cases.

Definition 7 (Circular Partition) *The* circular partition *induced by the set of wedges* $\mathcal{W}_\mathcal{C} = \{w_1, w_2, \ldots, w_n\}$ *rooted at a common point* q *is the sequence* $0 = \theta_0 < \theta_1 < \cdots < \theta_{4n} < 2\pi$ *of angles, corresponding to the ordered sequence of bounding edges of wedges in* $\mathcal{W}_\mathcal{C}$*; i.e., it is the ordered sequence of values in* $\{\theta_i \mid \theta_i \in \{\phi(\tau_j), \phi(\tau_j) + \pi \bmod 2\pi\}, \tau_j \in \tau(w), w \in \mathcal{W}_\mathcal{C}\}$. *Denote this sequence by* $\sigma(\mathcal{W}_\mathcal{C}) = (\theta_0, \theta_1, \ldots, \theta_{4n})$.

See Figure 3. Applying Equation (1) to Definition 7, we arrive at the following observation:

Observation 1 *Given a set* $\mathcal{W}_\mathcal{C}$ *of wedges and induced partition* $\sigma(\mathcal{W}_\mathcal{C}) = (\theta_0, \theta_1, \ldots, \theta_{4n})$ *for a given point* q *and set* $\mathcal{C}$ *of curves, for every* $i \in \{1, \ldots, 4n-1\}$ *and every* $\phi_1, \phi_2 \in (\theta_i, \theta_{i+1})$*, the set of curves in* $\mathcal{C}$ *intersected by* $\overrightarrow{q_{\phi_1}}$ *is the same as that intersected by* $\overrightarrow{q_{\phi_2}}$.

Observation 1 remains true when the point q at the root of the wedges moves within a bounded neighbourhood: given a curve Q and a set $\mathcal{C}$ of curves, for each point q on Q, the relative ordering of wedge boundaries in the circular partition of q remains unchanged when q moves along some interval of Q. By partitioning Q into such cyclically invariant segments, this property allows us to calculate the curve stabbing depth of Q relative to $\mathcal{C}$ discretely. Formally:

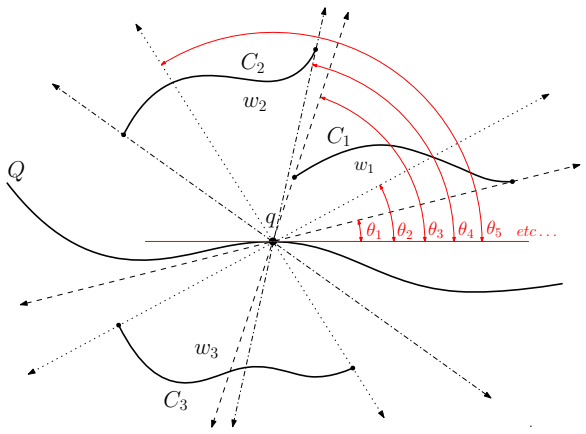


Figure 3: A configuration similar to that shown in Figure 1 for three curves C_1, C_2, and C_3 is depicted, with their respective wedge boundaries extended through the origin. The circular partition induced is shown by the sequence of angles towards the right-hand side of the figure.

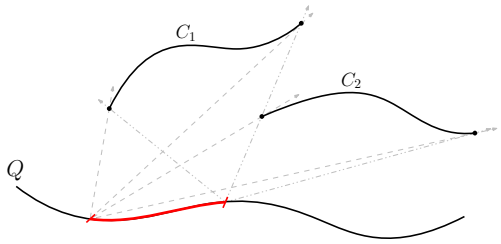


Figure 4: A configuration similar to that shown in Figure 1 for two curves C_1 and C_2 is depicted, the highlighted segment being cyclically invariant with respect to the given population, as can be seen by inspecting the wedge boundaries

Definition 8 (Cyclically Invariant Segments) *A segment along a curve that maintains the same cyclic ordering of boundaries within the circular partitions of each point along its length, is called* cyclically invariant. *Specifically, for a given curve* Q*, a segment* $I \subseteq Q$ *is* cyclically invariant *provided* $\sigma(\mathcal{W}_\mathcal{C})$ *has the same ordering of wedge boundaries as* $\sigma(\mathcal{W}'_\mathcal{C})$*, for all* $\mathcal{W}_\mathcal{C}$ *and* $\mathcal{W}'_\mathcal{C}$ *defined relative to any* $q, q' \in I$ *respectively.*

See Figure 4. Clearly such segments exist when $\{Q\} \cup \mathcal{P}$ is a set of polylines in $\mathbb{R}^2$. This property does not hold more generally for all plane curves[1]. For the remainder of this article, we assume $\{Q\} \cup \mathcal{P}$ is a set of polylines.

[1]We use $\mathcal{C}$ to denote a general set of plane curves, and $\mathcal{P}$ to denote a set of polylines in $\mathbb{R}^2$.

Lemma 1 (Invariant Segments along Polylines) *Given a polyline Q and a set $\mathcal{P}$ of polylines, Q can be partitioned into line segments, each of which is cyclically invariant with respect to $\mathcal{P}$.*

Proof. Consider any line segment L in Q, and assume without loss of generality that every element of $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$ lies above the line determined by L. An analogous argument can be applied to polylines that lie below L (any polyline that crosses L can be partitioned into separate polylines above and below L). The tangent points in a circular partition can only undergo a change in relative positions when the reference point (root) q becomes collinear with one of the common tangents between a pair of polylines defining the associated wedges, common to the convex hulls of each polyline. Consequently, as at most four such tangents exist for each pair of polylines, the set of points along L that trigger change in wedge orderings must be finite. Therefore, L can be partitioned into cyclically invariant segments, each of which is a maximal line segment on Q between two consecutive points that trigger changes. □

By Observation 1 and Lemma 1, the double integral in (D.c) can be reformulated as a sum of integrals measuring the total angular area swept out by the wedges of $\mathcal{P}$ with stabbing number weights along all cyclically invariant segments. This reformulation, which is made explicit in Section 3.2, is possible due to the fact that stabbing numbers remain constant within circular partitions, and that the cyclic ordering of each circular partition remains unchanged along each invariant segment.

3 Computing Curve Stabbing Depth for Polylines

In the following section we develop an algorithm for computing the curve stabbing depth of a given polyline Q relative to a given set $\mathcal{P}$ of polylines, based on the identification of critical curve features, such as tangent update points for curve wedges and the partitioning Q into invariant segments.

3.1 One Invariant Segment and One Polyline

We first describe an algorithm for computing the curve stabbing depth of one cyclically invariant segment $I = \overline{q_1 q_2}$ on a query polyline Q relative to another polyline $P = (p_1, p_2, \ldots, p_m)$, before generalizing the algorithm to the complete polyline Q and a set $\mathcal{P}$ of polylines.

The wedge $w(q, P)$ associated with polyline P and a given point q is determined by the tangent points of P (see Figure 5) which can be found by computing the convex hull of P and examining its vertices relative to q using binary search in $O(\log m)$ time. Thus, start by computing the convex hull $\mathrm{CH}(P)$, which can be completed in $O(m \log m)$ total time [14, 6].

In Case 1, begin by deriving the initial tangent points τ_1 and τ_2 of $w(q, P)$ for $q = q_1 \in I$ by using $\mathrm{CH}(P)$ as described in the previous paragraph. Additionally, determine all points of intersection between I and the set of lines corresponding to the extension of all line segments that form, $\partial\,\mathrm{CH}(P)$, the boundary of $\mathrm{CH}(P)$. Denote this set of intersection points along I by T. The points of T signal when and how tangent points of $w(q, P)$ need to be updated as q traverses along I; see Figure 7. The cyclical invariance of I allows the angular area swept out by w along each subsegment $I_i = \overline{ab}$ of I formed by points of T to be evaluated as

$$A_i = \int_{q \in I_i} \theta(w(q, P))\, ds. \qquad \text{(WA)}$$

We can apply a coordinate transform to render I collinear with the x-axis, which for Case 1(a) using (A.1) in the integral results in (WA) becoming

$$A_i = \int_{a'}^{b'} \left[\tan^{-1}\left(\frac{x - \tau_2(w')_x}{\tau_2(w')_y}\right) - \tan^{-1}\left(\frac{x - \tau_1(w')_x}{\tau_1(w')_y}\right) \right] dx,$$

for the transformed points a', b' and resulting wedge w' defined by the tangent points associated with the points of T delineating I_i. This being an integral with known antiderivative

$$\begin{aligned} A_i = \Bigg[& (\tau_1(w')_x - x) \tan^{-1}\left(\frac{\tau_2(w')_x - x}{\tau_2(w')_y}\right) \\ & + (x - \tau_1(w')_x) \tan^{-1}(\tau_1(w')_y(\tau_1(w')_x - x)) \\ & + \frac{1}{2\tau_1(w')_y} \ln(\tau_1(w')_y(\tau_1(w')_x^2 - 2\tau_1(w')_x x + x^2) + 1) \\ & - \frac{1}{2}\tau_2(w')_y \ln(\tau_2(w')_x^2 - 2\tau_2(w')_x x + \tau_2(w')_y^2 + x^2) \Bigg]_{a'}^{b'}. \end{aligned}$$

As a consequence of the circular partition induced by w being straightforward and w having stabbing number one, we find $D(I, P) = \sum_{I_i \in I} A_i / \pi L(I)$. Analogous analysis can be applied using (A.2) for problems in Case 1(b) reassembling that depicted in Figure 2b.

In Case 2, where $q \in I$ is in the interior of $\mathrm{CH}(P)$, begin by processing P to identify points of self intersection, some of which form closed loops (closed regions). Let L denote the set of points of self intersections of P. The planar subdivision formed by P and $\partial\,\mathrm{CH}(P)$ consists of polygonal faces, each of which can include at most one *window* edge on its boundary, i.e., an edge of $\partial\,\mathrm{CH}(P)$ that is not on P, as well as subpaths of P that do not cross into other faces.[2] This planar subdivision

[2] Observe that the faces of the planar decomposition are effectively simple polygons. Any polyline that protrudes into the interior of a face could be twinned to form a proper simple polygonal face.

can be computed in $O(m^2)$ time using a line segment intersection algorithm (e.g., [2]) and updating a doubly-connected edge list each time a point of intersection is identified.

Next, we construct the shortest geodesic path query data structure given in [15] augmented using the result from [7] in linear time for each face of the planar subdivision, taking $O(m^2)$ total time.

For any endpoint of I within $\mathrm{CH}(P)$ and for every intersection point a between I and the boundary of a face of the planar subdivision (or when I crosses an edge of P while remaining in the same face), we query the two shortest geodesic paths between a and the endpoints of the window edge on $\partial\,\mathrm{CH}(P)$ belonging to the current face. When q is in a face with no window edge, no visibility computation is required as all rays rooted at q stab P. The intersections between I and the extended segments along the shortest paths identify when and which tangent points of the visibility wedges that look out of $\mathrm{CH}(P)$ need to be updated. If the two shortest paths intersect at a vertex of P, then q loses external visibility after one of the two update points corresponding to these extended intersecting segments. Shortest geodesic path queries can be performed in $O(\log m^2 + t)$ time, where t is the number of turns on the reported shortest path. Intersection testing between extended segments and I takes at most $O(t)$ time per path. Thus, this step takes $O(m)$ worst-case time for each such query along I.

The depth for the portion of I within $\mathrm{CH}(P)$ can be calculated as a discrete sum of the depth accumulated by each subsegment I_i of I that result from partitioning I by shortest path update points, by calculating the total wedge area of the difference between 2π and the window visibility wedge at each point along I_i. A calculation that is otherwise analogous to those discussed for Case 1 above.

3.2 A Polyline Q and a Set $\mathcal{P}$ of Polylines

We generalize the algorithm described in Section 3.1 to a query polyline $Q = (q_1, q_2, \ldots, q_m)$ and a set of polylines $\mathcal{P} = \{P_1, P_2, \ldots, P_n\}$, with $P_i = (p_{i,1}, p_{i,2}, \ldots, p_{i,m})$ for $i = 1, \ldots, n$.

The algorithm is organized into three stages: an initial preprocessing stage applied to $\mathcal{P}$, a separate preprocessing method applied to Q based on results of the first stage, and the final computation of $D(Q, \mathcal{P})$.

Preprocessing $\mathcal{P}$. Begin by computing the convex hull $\mathrm{CH}(P_i)$ of each polyline $P_i \in \mathcal{P}$ to determine wedge tangent points, as done in Section 3.1; see Figure 5. Let $\mathcal{H}$ denote the resulting set of convex hulls. This stage can be completed in $O(nm \log m)$ total time.

Having determined $\mathcal{H}$, compute the collection $\tau(\mathcal{H})$ of all common tangent lines that separate each pair of convex hulls. See Figure 6. That is, compute

$$\tau(\mathcal{H}) = \{\text{lines } l \mid \text{for some } \{P_i, P_j\} \subseteq \mathcal{P} \\ (l \cap \mathrm{CH}(P_i)) \cup (l \cap \mathrm{CH}(P_j)) = \{p_{i,i'}, p_{j,j'}\}\},$$

where $p_{i,i'}$ and $p_{j,j'}$ are vertices of $\mathrm{CH}(P_i)$ and $\mathrm{CH}(P_j)$, respectively.

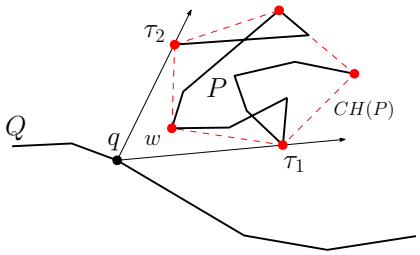


Figure 5: A query polyline Q and tangent points of P highlighted along the boundary of $\mathrm{CH}(P)$. The tangent points and boundary rays for the wedge $w(q, P)$ are also shown.

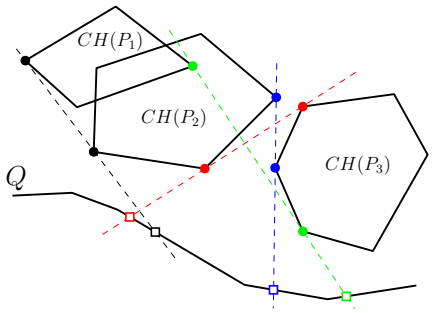


Figure 6: Illustration of the common tangents between convex hulls $\mathrm{CH}(P_1), \mathrm{CH}(P_2)$, and $\mathrm{CH}(P_3)$. To simplify the figure, only those tangents that intersect Q are shown, with their points of intersection marked along Q by boxes.

There are three distinct cases to consider when computing these common tangents: (1) the two convex hulls are disjoint, (2) their boundaries intersect, and (3) one convex hull entirely contains the other. Case 1 is the simplest, in which the common tangents between two convex hulls $\mathrm{CH}(P_1)$ and $\mathrm{CH}(P_2)$ can be computed in $O(\log |\,\mathrm{CH}(P_1)| + \log |\,\mathrm{CH}(P_2)|)$ time [17]. Case 2 requires $O(m)$ time to compute in the worst case. However, if the two convex hull boundaries intersect at most twice, the common tangents can be found in $O(\log(|\,\mathrm{CH}(P_1)| + |\,\mathrm{CH}(P_2)|) \log k)$ time, where $k = \min\{|\,\mathrm{CH}(P_1) \cap \mathrm{CH}(P_2)|, |\,\mathrm{CH}(P_1) \cup \mathrm{CH}(P_2)|\}$ [17]. In Case 3, no computation is performed after identifying that the hulls are nested. It takes up to $O(m)$ time to identify which of the three cases must be applied. Thus, this stage can be computed in $O(n^2 m \log^2 m)$ time.

Preprocessing Q. After preprocessing $\mathcal{P}$, mark the points of intersection between elements of $\tau(\mathcal{H})$ and Q, which, per the proof of Lemma 1, partition Q into cyclically invariant segments. Additionally, as outlined in

Section 3.1, determine all points of intersection between Q and the set of lines corresponding to the extension of all line segments on $\partial\,\mathrm{CH}(P_i)$ for each $i = 1, \ldots, n$. See Figure 7 for the latter. If the intersection between one of these extended segments and Q occurs on the boundary of the convex hull, Q must pass into the interior of the convex hull. Here we enter Case 2 of the algorithm described in Section 3.1, and perform the same computations. In the worst case, Q passes through the convex hulls of all n polylines in $\mathcal{P}$, leading to $O(nm^2)$ worst-case processing time.

This yields two point sets on Q, say S and T, that respectively identify when wedge stabbing numbers and tangent points need to be updated relative to the position of q along Q. Let $\mathcal{I}$ denote the resulting partition of Q into cyclically invariant segments by points of S, after further refinement from the vertices of Q itself. Likewise, for all $I \in \mathcal{I}$, let $I_i \in I$ denote a subdivision of I delineated by tangent update points of T. There are at most $O(n^2)$ many points in S as there are at most four common tangents for each pair of convex hulls. Additionally, there are at most $O(m)$ segments composing each of the n convex hulls, and $O(m^2)$ internal update points for each crossed convex hull, so T contains at most $O(nm^2)$ points. Consequently, this step takes $O(n^2m + nm^2)$ worst-case time to compute all possible intersections.

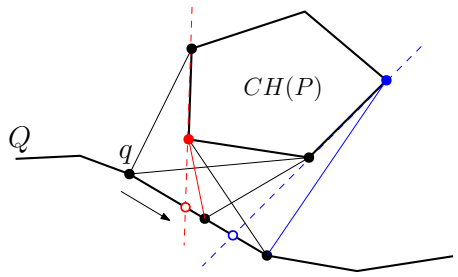


Figure 7: Depiction of a query polyline Q with tangent update points of a polyline P shown along its length as open circles. These points are derived from the intersection between Q and lines passing though the parameter segments of $\mathrm{CH}(P)$. Only those lines that intersect Q are shown. Then q traverses the length of Q (in the indicated direction) the tangent points of the wedge $w(q, P)$ change whenever one such point is crossed. The update points are color matched with the resulting tangent point (line) change.

Computing the curve stabbing depth of Q. Let $\vec{u}$ denote the unit direction vector associated with a line segment $I = \overline{q_1q_2} \in \mathcal{I}$ along Q. Construct the matrix $P_{\vec{u}} + B$, composed of the transition matrix $P_{\vec{u}}$ from the standard basis of $\mathbb{R}^2$ to the orthonormal basis $\{\vec{u}, -1/\vec{u}\}$, and a vertical translation matrix B that displaces I to have height zero after the transformation. Applying this transformation to $\mathcal{P}$ pointwise for each $I \in \mathcal{I}$ allows us to calculate the area swept out by wedges along a path as described in Section 3.1. Let $\mathcal{P}' = \{P'_1, P'_2, \ldots, P'_n\}$ be the set of transformed polylines.

Starting at q_1, construct the set of wedges $\mathcal{W}_{\mathcal{P}'}$. This is accomplished by calculating the tangent points of each convex hull within $\mathcal{H}$ relative to q_1 using binary search in $O(n \log m)$ time. The set $\mathcal{W}_{\mathcal{P}'}$ is updated incrementally by monitoring the points of T crossed by q while traversing I. Each update takes $O(1)$ time by walking one vertex clockwise or counterclockwise around the perimeter of the convex hull depending on the relative motion between q translating along I and the convex hull.

Afterwards, construct $\sigma(\mathcal{W}_{\mathcal{P}'}) = (\theta_0, \theta_1, \ldots, \theta_{4n})$ by sorting the lines associated to each tangent point by slope, treating the portion of the line extended through the origin separately. During this process, take note of which regions overlap to calculate the stabbing numbers of each angular region in the partition (subdivided wedges) as in (1) and Observation 1. These stabbing numbers are iteratively updated by monitoring the points of S crossed in $O(1)$ time per event as is done for tangent points above. From the circular partition $\sigma(\mathcal{W}_{\mathcal{P}'})$, select a minimizing subset iteratively by defining the indicator variable (bit sequence)

$$\delta_i = \begin{cases} 1 & \text{if } \mathrm{stab}_{\mathcal{P}}(\overrightarrow{q_{\theta^*}}) \leq \mathrm{stab}_{\mathcal{P}}(\overrightarrow{q_{\theta^*+\pi}}) \\ & \quad \text{for } \theta_{i-1} \leq \theta^* \leq \theta_i \\ 0 & \text{otherwise,} \end{cases}$$

for $i = 1, \ldots, 4n$. This selection procedure performs the same task as the minimization operation within (D.c).

At last, we can compute the depth of Q accumulated along I, by reformulating (D.c) in terms of summations over all $I_i \in I$, specifically,

$$D_I = \frac{1}{\pi L(Q)} \sum_{I_i \in \mathcal{I}} \sum_{j=1}^{4n} \delta_j \,\mathrm{stab}_{\mathcal{P}'}(\overrightarrow{q_{\theta_j^*}}) A_j, \qquad \text{(D.i)}$$

for any q along I_i and sample angle $\theta_j^* \in [\theta_{j-1}, \theta_j)$, and the angular area A_j swept out by the wedge bounded between the angles $[\theta_{j-1}, \theta_j]$ while q is translated across I_i, as calculated above using (WA).

The total depth of Q is found by evaluating the sum $D(Q, \mathcal{P}) = \sum_{I \in \mathcal{I}} D_I$.

Forming the partition $\sigma(\mathcal{W}_{\mathcal{P}'})$ and selecting the chosen subset takes at most $O(n \log m)$ time. The query polyline Q contains at most $O(n^2 + nm^2)$ update points from S and T which are used during the computation of D_I, and at most $O(m)$ directional transitions from its m constitutional line segments where each transformation to the set $\mathcal{P}$ takes $O(nm^2)$ time. Thus, this final stage takes $O(n^2 + nm^2)$ time.

These results are summarized in the following theorem:

Theorem 2 *Given an m-vertex polyline Q and a set $\mathcal{P}$ of n polylines, each with m vertices, we can compute the curve stabbing depth of Q relative to $\mathcal{P}$ in $O(nm^2 + n^2 m \log^2 m)$ time.*

Expressed differently, the running time is $O(k^2)$, where k denotes the total number of vertices in the input polylines $\{Q\} \cup \mathcal{P}$.

4 Discussion and Directions for Future Research

In this section, we discuss depth medians, possible generalizations of curve stabbing depth to higher dimensions, and other possible measures of curve depth. Due to space constraints, discussion of properties has been omitted (e.g., stability, robustness, etc.).

4.1 Median Curves and Depth Median Points

The depth for any particular curve in a set can be computed by treating it as a query curve Q. The comparison of all the resulting depth scores allows for a median outwards ranking of all curves.

Moreover, observe not all points along the length of a curve Q contribute equally to the curve stabbing depth of Q relative to the set $\mathcal{C}$ of curves. The depth of a point (a degenerate curve) is given by (D.p). It follows that for some point q on Q, $D(q,\mathcal{C}) \geq D(Q,\mathcal{C})$. Consequently, this gives:

Observation 2 *For any given set $\mathcal{C}$ of plane curves, there exists a point $m \in \mathbb{R}^2$ that is a depth median of $\mathcal{C}$. That is,*

$$D(m,\mathcal{C}) = \max_{Q \in \mathcal{Q}} D(Q,\mathcal{C}),$$

where $\mathcal{Q}$ denotes the set of all plane curves.

4.2 Generalizations to Higher Dimensions

When a curve Q and a set $\mathcal{C}$ of curves lie in a k-dimensional flat of $\mathbb{R}^d$ for some $k < d$, the d-dimensional curve stabbing depth of Q as calculated using a ray relative to $\mathcal{C}$ is zero; whereas, the k-dimensional curve stabbing depth of Q relative to $\mathcal{C}$ is non-zero in general, meaning that the straightforward generalization of Definition 4 is not consistent across dimensions.

Alternatively, another natural generalization of Definition 4 to higher dimensions is to replace the rotating stabbing ray by a k-dimensional half-hyperplane, and to measure the number of curves it intersects as it rotates. This second generalization is consistent across dimensions.

4.3 Alternative Definitions

Alternative possible definitions for the stabbing depth of curves considered by the authors include:

$$\int_{q \in Q} \min_{0 \leq \theta < \pi} \min\{\mathrm{stab}_{\mathcal{C}}(\overrightarrow{q_\theta}), \mathrm{stab}_{\mathcal{C}}(\overrightarrow{q_{\theta+\pi}})\}\, ds, \qquad (2)$$

which differs from (D.c) by a minimum in place of the second integral (maximum was also considered). Equation (2) often gives a zero depth value regardless of the position of Q relative to $\mathcal{C}$. For example, consider a set $\mathcal{C}'$ of n parallel line segments of equal length. Each of these line segments has depth zero relative to $\mathcal{C}'$ by (2) because every point on every segment is the root of some ray that does not intersect any other segment in $\mathcal{C}'$. Conversely, using Definition 4 instead, the line segment at the centre (median) of $\mathcal{C}'$ has greatest depth, with depth values decreasing monotonically toward the two line segments on the outside of $\mathcal{C}'$, which are the only two curves in $\mathcal{C}'$ with depth zero.

4.4 Approximation Algorithms using Randomization

Definition 4 suggests that efficient approximate computation by Monte Carlo methods is likely possible using a random sample of rays rooted along the query curve Q. One possible direction for future research is to bound the expected quality of approximation and the expected running time as functions of the number of random rays selected.

4.5 Upper Envelopes of Sets of Pseudolines

Our algorithm for computing curve stabbing depth involves identifying the extreme points of each curve $P \in \mathcal{P}$ relative to a point q that follows the query curve Q. When P is a polyline, the extreme points can be identified by computing the upper and lower envelopes of the angle formed by each vertex of P relative to q as a function of the position of q on Q. These functions are a set of pseudolines when Q is a line segment; it may be possible to compute upper and lower envelopes of this set efficiently by constructing the convex hull of a set of points dual to the set of pseudolines (e.g., [1]), which may lead to a simpler and more efficient algorithm for computing curve stabbing depth.

References

[1] P. K. Agarwal and M. Sharir. Pseudo-line arrangements: Duality, algorithms, and applications. *SIAM Journal on Computing*, 34(3):526–552, 2005.

[2] I. J. Balaban. An optimal algorithm for finding segments intersections. In *Proceedings of the Eleventh Annual Symposium on Computational Geometry (SoCG)*, pages 211—219, 1995.

[3] V. Barnett. The ordering of multivariate data. *Journal of the Royal Statistical Society. Series A (General)*, 139(3):318–355, 1976.

[4] K. Buchin, M. Buchin, M. van Kreveld, M. Löffler, R. I. Silveira, C. Wenk, and L. Wiratma. Median trajectories. *Algorithmica*, 66:595–614, 2013.

[5] K. Buchin, M. Buchin, M. J. van Kreveld, B. Speckmann, and F. Staals. Trajectory grouping structure. *Journal of Computational Geometry*, 6(1):75–98, 2015.

[6] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16:361–368, 1996.

[7] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(3):485–524, 1991.

[8] J. Chu, X. Ji, Y. Li, and C. Ruan. Center trajectory extraction algorithm based on multidimensional hierarchical clustering. *Journal of Mechatronics and Artificial Intelligence in Engineering*, 2(2):63–72, 2021.

[9] G. Claeskens, M. Hubert, L. Slaets, and L. Vakili. Multivariate functional halfspace depth. *Journal of the American Statistical Association*, 109(505):411–423, 2014.

[10] F. A. Cuevas and R. Fraiman. Robust estimation and classification for functional data via projection-based depth notions. *Computational Statistics*, 22:481–496, 2007.

[11] P. L. de Micheaux, P. Mozharovskyi, and M. Vimond. Depth for curve data and applications. *Journal of the American Statistical Association*, 116(536):1881–1897, 2021.

[12] S. Durocher and M. Y. Hassan. Clustering moving entities in euclidean space. In *17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT)*, volume 162 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 22:1–22:14, 2020.

[13] F. Ferraty and P. Vieu. Curves discrimination: a nonparametric functional approach. *Computational Statistics & Data Analysis*, 44(1-2):161–173, 2003.

[14] R. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, 1972.

[15] L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989.

[16] F. Ieva and A. M. Paganoni. Depth measures for multivariate functional data. *Communications in Statistics - Theory and Methods*, 42(7):1265–1276, 2013.

[17] D. Kirkpatrick and J. Snoeyink. Computing common tangents without a separating line. In *Proceedings of the 4th International Workshop on Algorithms and Data Structures (WADS)*, pages 183–193, Berlin, Heidelberg, 1995. Springer-Verlag.

[18] R. Liu. On a notion of data depth based on random simplices. *The Annals of Statistics*, pages 405–414, 1990.

[19] R. Y. Liu, J. M. Parelius, and K. Singh. Multivariate analysis by data depth: Descriptive statistics, graphics and inference. *The Annals of Statistics*, 27(3):783–840, 1999.

[20] H. Oja. Descriptive statistics for multivariate distributions. *Statistics & Probability Letters*, 1(6):327–332, 1983.

[21] P. J. Rousseeuw and M. Hubert. Regression depth. *Journal of the Amerian Statistical Association*, 94(446):388–402, 1999.

[22] R. Serfling. Depth functions in nonparametric multivariate inference. In R. Y. Liu, R. Serfling, and D. L. Souvaine, editors, *Data Depth: Robust Multivariate Analysis, Computational Geometry and Applications, Proceedings of a DIMACS Workshop*, volume 72 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 1–16, 2003.

[23] J. W. Tukey. Mathematics and the picturing of data. In R. D. James, editor, *Proceedings of the International Congress of Mathematicians*, volume 2, pages 523–531, 1975.

Reflections in an Octagonal Mirror Maze

David Eppstein*

Abstract

Suppose we are given an environment consisting of axis-parallel and diagonal line segments with integer endpoints, each of which may be reflective or non-reflective, with integer endpoints, and an initial position for a light ray passing through points of the integer grid. Then in time polynomial in the number of segments and in the number of bits needed to specify the coordinates of the input, we can determine the eventual fate of the reflected ray.

1 Introduction

There are many problems in graphics and visibility testing where it is of interest to determine the path that would be taken by a ray of light, through an environment that may contain mirrors. Figure 1 gives a simple example of a problem of this type. Even for very restricted environments such as the one depicted, where the starting point of the ray and all endpoints of mirrored segments have integer coordinates and where the mirrors are all either axis-aligned or at 45° angles to the axes, the path of such a ray may be very complicated, taking a number of reflections that may depend on the geometry of the input and not merely on its combinatorial complexity. For instance, a ray that bounces diagonally between two parallel mirrors on opposite sides of a long thin rectangle will only exit the rectangle after a number of bounces proportional to the aspect ratio of the rectangle, even though such an environment consists of only two segments. Nevertheless, in that simple two-mirror example, the eventual path of the ray is easy to predict, without resorting to separately simulating each bounce. What about for environments of more than two mirrors? Is it still easy to ray-trace reflected rays in such environments?

We formalize this problem as follows. The input environment is described as a collection of line segments, with integer endpoints and either parallel to a coordinate axis or at a 45° angle to the axes. Each side of each line segment may be marked as reflective or non-reflective. We are also given an integer position for the start of a light ray, and an integer vector describing the initial direction of the light ray. The restricted orientation of the mirrors ensures that each reflection of the ray in one of the reflective segments continues to have integer slope, on a line through infinitely many points of the integer grid. If, after repeated reflections, the ray eventually hits a non-reflective segment, the endpoint of a segment, or its initial position and orientation, it stops; otherwise, it must eventually escape the environment along an unobstructed infinite ray. The output of the problem is the eventual fate of the ray: the point where it stops, or the ray along which it escapes. Our main result is that we can determine this outcome in polynomial time.

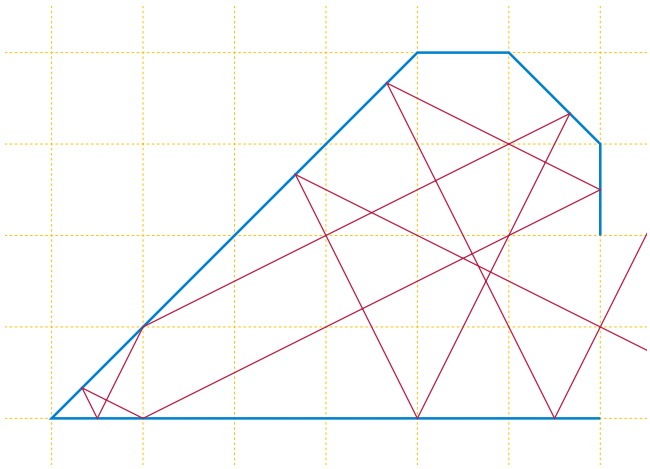


Figure 1: The reflections of a light ray (red) among mirrors that are axis-aligned or at 45° angles to the axes (blue)

Let n denote the number of segments of the input, and suppose that all of the integers in the input specification (including the ones specifying the initial vector of the traced ray) have magnitude at most N. For these problem size parameters, it would be trivial to solve the problem in time polynomial in N – simply trace the ray one reflection at a time – but this time bound is not polynomial, as it is exponentially larger than the input size. Our time bound is weakly polynomial, but not strongly polynomial: it is a polynomial of the number of bits required to specify the input, which is $O(n \log N)$. For the purposes of polynomial-time complexity, it would be equivalent to allow input coordinates that are rational numbers, rather than integers, with numerators and denominators of magnitude at most N. Clearing denominators would produce an integer input whose

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

coordinates have magnitude $N^{O(n)}$ (the product of the input numerators and denominators), still requiring only a polynomial number of bits to specify, $O(n^2 \log N)$.

The main idea of our algorithm is to transform the problem into one of determining the iterated behavior of a certain type of discrete one-dimensional dynamic system, which in a related recent paper [4] we called an *iterated integer interval exchange transformation*. In turn, following that paper, we can transform the iterated integer interval exchange transformation problem into a previously-studied problem in computational topology, of following paths along *normal curves* on triangulated topological surfaces. To solve this path-following problem on normal curves in triangulated surfaces, we apply algorithms of Erickson and Nayyeri [5].

The general topic of visibility and ray-shooting with reflection is one with much prior work, both heuristic as part of the computer graphics rendering pipeline and with more rigorous bounds in computational geometry, for which see, e.g., [1–3, 6–10]. However, this past work has a combinatorial complexity that blows up with the number of reflections. In contrast, our results give a polynomial time algorithm whose complexity is independent of the number of reflections.

2 Iterated interval exchange transformations

In a recent paper of the author [4] we investigated a broad class of problems, involving computing the nth iterate of a polynomial-time bijective function. One motivation for this investigation was in ray-tracing problems like the one studied here: if an environment consists only of mirrors, with no absorbing barriers for light, then (modulo representational issues involving whether reflections preserve the integer nature of a light ray) the mapping from each reflected position and direction of a light ray to the next is just such a polynomial-time bijection. Most of the problems considered in our recent paper have high computational complexity. However, our paper also identified a special class of bijections, the *integer interval exchange transformations*, for which iterates can be computed in polynomial time. We will use the resulting *iterated integer interval exchange transformation problem* as a subroutine in our algorithm for finding the result of a sequence of reflections. In this section we summarize the definitions needed to apply this problem, following our previous paper.

We define an integer interval exchange transformation to be a certain type of piecewise-linear bijective mapping on a range of consecutive integers. It may be defined by a partition of the range into subintervals, and by a permutation of those subintervals. The transformation then translates each subinterval (meaning that it acts on this interval by addition of the same value to each integer in the interval), so that the translated subintervals again form a partition of the same range, reordered into the given permutation. An example, used in Fig. 2, is the transformation on $[0, 15)$ that permutes the intervals $a = [0, 3], b = [4, 5], c = [6], d = [7, 14]$ into the permuted order b, d, c, a. This permutation describes the function

$$x \mapsto \begin{cases} x + 11, & \text{for } x \in [0, 3] \\ x - 4, & \text{for } x \in [4, 5] \\ x + 4, & \text{for } x \in [6] \\ x - 5, & \text{for } x \in [7, 14]. \end{cases}$$

A transformation of this type, with m intervals on the range $[0, M - 1]$, can be specified by $O(m \log M)$ bits of information, specifying the endpoints and permuted position of each subinterval. The resulting integer function may be evaluated on any integer x in its range in time $O(m)$, by a sequential search of the listed subintervals to find the one containing x, and a second scan of the subintervals to determine which ones have permuted positions before the one containing x and contribute to the translation offset for x. Even faster evaluations are possible if the intervals are stored in sorted order with their translation offsets. The *iterated interval exchange transformation problem* takes as input an integer interval exchange transformation μ, represented in either of these ways, an integer x in its range, and another non-negative integer k. The goal is to compute $\mu^{(k)}(x)$, the result of repeatedly replacing x by its transformed value, k times.

Following a suggestion of Mark Bell, our paper [4] shows that, for every integer interval exchange transformation, it is possible to find a corresponding triangulated two-dimensional manifold, and a *normal curve* on the surface, such that tracing the normal curve for a specified number of steps corresponds to evaluating the integer interval exchange transformation (Figure 2). Here, a normal curve is a curve through the triangles of the surface, avoiding triangle vertices and passing straight across each triangle from edge to edge, without crossing itself. It can be specified by a system of numbers on each edge counting the number of segments of the curve that cross that edge; this specification must obey certain consistency constraints (the numbers of crossings on the three edges of each triangle must obey the triangle inequality and sum to an even number). This specification is sufficient to reconstruct the curve itself, up to topological equivalence.

In the transformation from our paper [4], the integers in the range $[0, M - 1]$ of an integer interval exchange transformation are represented as the sequence of M crossings of a normal curve, along a central horizontal edge of a triangulated surface. Each of these crossing points x is connected by the normal curve, along a path of exactly s segments for a number s determined as part of the construction, to its image $\mu(x)$ according to the integer interval exchange transformation. Following this

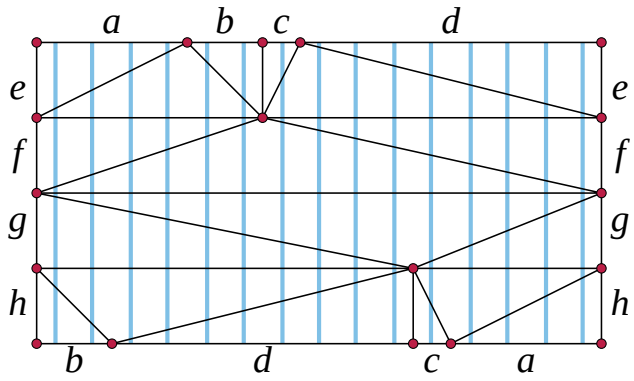


Figure 2: A normal curve (light blue) on a triangulated double torus (black triangles and red vertices, glued from top to bottom and from left side to right side with the pairing indicated by the letters). Traversing the normal curve upwards from its central horizontal line, through the glued edges from top to bottom, and continuing upwards back to the same central line, permutes the branches of the curve according to the integer interval exchange transformation that maps $[0,3] \mapsto [11,14]$, $[4,5] \mapsto [0,1]$, $6 \mapsto 10$, and $[7,14] \mapsto [2,9]$. From [4].

construction, the iterated interval exchange transformation problem can then be reduced to finding the crossing point that is sk steps ahead of x along the normal curve. This problem, of tracing paths for a given number of steps on a normal curve, has been given a polynomial-time solution by Erickson and Nayyeri [5]. It follows that the iterated interval exchange transformation problem can also be solved in polynomial time. More precisely, the time is $O(m^2 \log M)$, after an initial step in which the input parameter k is reduced modulo the total number of steps in (a component of) the normal curve [4]. As an integer division of a $\log k$-bit number by a $\log M$-bit number, this reduction step can be performed in time $O(\log k \log M)$ using naive division algorithms.

3 Partial integer interval exchange

Reflection in a mirror is a reversible transformation on systems of rays, but absorption by a non-reflective surface is not: many different rays could be absorbed at the same point. To mimic this non-reversibility in an integer exchange problem, while still allowing the polynomial-time procedure from our previous paper to apply, it is convenient to generalize the integer interval exchange problem to allow transformations that are only partially defined, as follows.

We define a *partial integer interval exchange transformation*, for the range $[0, M-1]$, to be a system of disjoint subintervals of this range, together with a transformation that offsets each of these subintervals to another system of disjoint subintervals (necessarily of equal lengths). For instance, by omitting the subinterval [6] from the previous example, we obtain a partial integer exchange transformation that maps that maps $[0,3] \mapsto [11,14]$, $[4,5] \mapsto [0,1]$, and $[7,14] \mapsto [2,9]$. The *domain* of the transformation is the union of the given subintervals, and the *codomain* is the union of their target subintervals. This example has domain $[0,5] \cup [7,14]$ and codomain $[0,9] \cup [11,14]$.

Lemma 1. *If a partial integer interval exchange transformation is repeatedly applied to an input x that does not belong to the codomain, it eventually reaches a transformed value that does not belong to the domain.*

Proof. Consider the directed graph that connects each value to its transformed image. This graph has in-degree and out-degree at most one at each vertex, and has finitely many vertices, so it consists of a disjoint union of directed paths and directed cycles. An input x that does not belong to the codomain has no incoming edge, so it is the starting vertex of a path, and is eventually transformed into the ending vertex of the same path, a value that does not belong to the domain. □

We define the *iterated partial integer interval exchange transformation problem* to be a computational task that takes as input the description of a partial integer interval exchange transformation (as a system of subintervals and their targets) and a value x that does not belong to the codomain, and that produces the corresponding value that does not belong to the domain, according to Lemma 1.

Lemma 2. *The iterated partial integer interval exchange transformation problem can be solved in time $O(m^2 \log M + \log^2 M)$, polynomial in the input representation size.*

Proof. We transform the problem in polynomial time into an equivalent instance of the (non-partial) iterated integer interval exchange transformation problem. Let $I_1, I_2, \ldots$ be the intervals of the given partial transformation f, and let $f(I_1)$ etc. denote their images after the transformation. Suppose also that the given partial transformation operates on the range $[0, M-1]$ of length M. Let m denote the total number of elements in this range that are missed by f: they are not in its domain. We define a new transformation $\bar{f}$ that operates on the range $[0, Mm+M+m-1]$ of length $Mm+M+m$, as follows:

- For each subinterval I_i in the given transformation, with image $f(I_i)$, we include in $\bar{f}$ the mapping $I_i \mapsto f(I_i)$.
- For each maximal subinterval J_i of $[0, M-] \setminus \cup I_i$ (a subinterval not in the domain of f) we include in $\bar{f}$ a mapping from J_i to a subinterval of $[M, M+m-1]$,

so that the images of these subintervals are disjoint and completely cover $[M, M+m-1]$.

- We include in $\bar{f}$ the mapping $[M, Mm+M-1] \mapsto [M+m, Mm+M+m-1]$. Iterating this mapping eventually transforms any value in $[M, M+m-1]$ to a value in $[Mm+M, Mm+M+m-1]$, but it takes M iterations to do so.
- For each maximal subinterval K_i of $[0, M-] \setminus \cup f(I_i)$ (a subinterval not in the codomain of f) we include in $\bar{f}$ a mapping from a subinterval of $[Mm+M, Mm+M+m-1]$ to K_i, so that the preimages of these mappings are disjoint and completely cover $[Mm+M, Mm+M+m-1]$.

For instance, for the example partial integer interval exchange transformation f given above, $M=15$ and $m=1$ (there is only one missing value from the transformation), and we have $\bar{f}$ mapping $[0,3] \mapsto [11,14]$, $[4,5] \mapsto [0,1]$, $[7,14] \mapsto [2,9]$; $[6] \mapsto [15]$; $[15,29] \mapsto [16,30]$; and $[30] \mapsto [10]$.

Suppose we apply the algorithm to the iterated integer interval exchange transformation problem, with transformation $\bar{f}$, on an input value x that does not belong to the codomain, and that the output of this algorithm is a value z. If we iterate $\bar{f}$ for a total of M iterations, starting with a value x that does not belong to the codomain, it will reach a value y that does not belong to the domain in fewer than M iterations, by Lemma 1. The next iteration will map y into a value y' in the subinterval $[M, M+m-1]$, and the subsequent (again fewer than M) iterations will each add m to this value y'. We may therefore obtain y' by z as the unique value in the subinterval $[M, M+m-1]$ that is congruent to z modulo m. From y', we may obtain the desired value y as $\bar{f}^{-1}(y')$.

The time bound is obtained by plugging in the number of pieces of the resulting transformation, $O(m)$, the range of its values, $O(Mm)$, and the number of iterations, $O(M)$, into the previous time bound for iterated integer interval exchange transformations. □

4 Converting reflection to partial integer interval exchange

The reason that we have restricted our attention to reflections in line segments that are horizontal, vertical, and diagonal is that these kinds of reflections preserve the integrality of the reflected rays. We formalize these observations as follows.

Lemma 3. *If a ray whose direction is specified by a vector (x, y) is reflected by a sequence of horizontal, vertical, or diagonal mirrors, then the resulting ray's direction can be specified by one of the eight vectors $(\pm x, \pm y)$ or $(\pm y, \pm x)$.*

Figure 3: Cases for reflection of a ray from a horizontal, vertical, or diagonal mirror

Proof. Vertical mirrors negate the first coordinate, horizontal mirrors negate the second coordinate, and diagonal mirrors exchange the two coordinates (possibly also negating both of them); see Fig. 3. The set of eight vectors of the lemma are preserved by these operations, so no other direction is possible. □

Along with these eight directions, it will also be important to keep track of the left-right orientation of each ray; that is, whether an image that follows a thickened copy of the ray has its orientation preserved or flipped. This changes on each reflection, and cannot be determined only from the ray's direction: a ray that reflects perpendicularly from a mirror will have its orientation changed, whereas a ray that returns from a corner-reflector (two perpendicular mirrors) will not.

Lemma 4. *If a ray that passes through infinitely many points of the integer grid is reflected by a sequence of horizontal, vertical, or diagonal mirrors, each with integer endpoints, then the reflected ray again passes through infinitely many points of the integer grid.*

Proof. When a ray is reflected by a mirror, it passes after the reflection through the same sequence of grid points that the unreflected ray would pass through in the reflection of the grid. But with mirrors of the type described by the lemma, the reflection of the grid is an integer grid with the same points. □

Observation 5. *Let $\mathcal{L}$ be the system of all lines in the plane that pass through integer points in the direction of a vector (x, y), where x and y are integers in lowest terms (their greatest common divisor is one). Then the lines of $\mathcal{L}$ subdivide each vertical unit segment of the integer grid into x equal-length pieces, and each horizontal unit segment of the grid into y equal-length pieces. They subdivide each diagonal segment of the grid, of length $\sqrt{n}$ with slope of the same sign as the slope of the lines in $\mathcal{L}$, into $|x-y|$ equal-length pieces, and each diagonal segment of the opposite slope into $|x+y|$ equal -length pieces. (See Fig. 4.)*

Putting these observations together, we can transform any octagonal mirror maze into an equivalent partial integer interval exchange transformation.

Lemma 6. *Suppose we are given an environment, described as a collection of line segments, each side of*

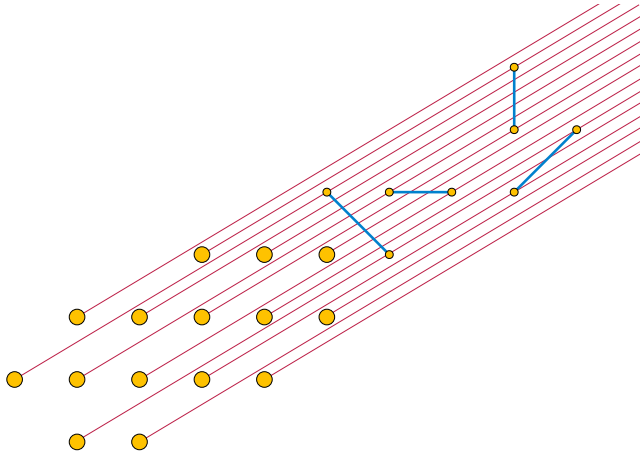


Figure 4: Subdivision of grid segments and diagonals (blue) by the system of all lines through integer points in the direction of the vector $(5, 3)$ (red).

which may be marked as reflective or non-reflective, with integer endpoints, an integer position for the start of a light ray, and an integer vector describing the initial direction of the light ray. Then in polynomial time we can construct a partial integer interval exchange transformation f whose values correspond to points of the environment (either the starting point or points along the segments of the environment) and directions of a reflected light ray emanating from that point, such that:

- *If a value v belongs to the domain of f, then the ray described by v is eventually reflected by the environment, with its first reflection at a position and direction described by $f(v)$.*
- *If a value v does not belong to the domain of f, then the ray described by v hits an absorbing barrier before being reflected, or escapes to infinity.*

Proof. We surround the given environment with a non-reflective bounding box that will catch all escaping rays. By Lemma 3 and Lemma 4, we need only consider rays through integer points, in eight directions and two left-right orientations. By Observation 5, we need only consider a discrete evenly-spaced set of possible reflection points along each segment of the environment, of a size that can be described by an integer with polynomially many bits. (Recall that we are not restricting the lengths of our grid segments or our initial direction to have polynomial magnitude, only to be integers with a polynomial number of bits.) We will create a partial integer interval exchange transformation f whose range is partitioned into subintervals, one for each combination of an environment segment, a direction of the emanating ray, and a left-right orientation, with the length of these subintervals obtained by multiplying the length of the segment by the number of reflection points per unit length given by Observation 5. For each of these subintervals, we further partition it into sub-subintervals, in polynomial time, according to the next object in the environment that a ray in that direction would hit, as depicted in Fig. 5. (This is simply a lower envelope of n disjoint line segments, for a projection direction determined by the ray direction.) When the next object to be hit is reflecting, we map a sub-subintervals to the interval describing the corresponding reflection points along that object. When, instead, it is absorbing, we omit that sub-subinterval from the partial integer interval exchange transformation. □

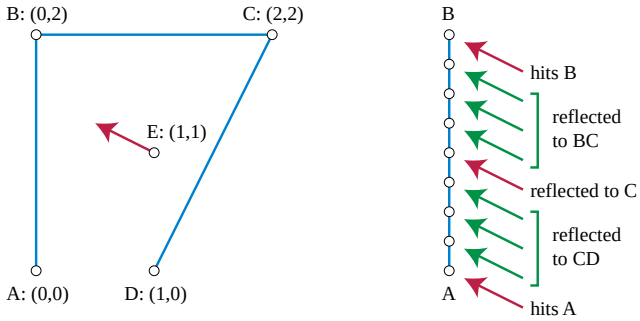


Figure 5: Reflection points along segment AB of a given environment, partitioned into subintervals according to the next object in the reflected path for incoming rays of slope $-\frac{1}{2}$.

5 The main result

Putting these components together, we have the following result:

Theorem 7. *Suppose we are given an environment, described as a collection of line segments, each side of which may be marked as reflective or non-reflective, with integer endpoints, an integer position for the start of a light ray, and an integer vector describing the initial direction of the light ray. Then in time polynomial in the number of segments and in the number of bits needed to specify the integers of the input, we can determine whether the reflected ray is eventually absorbed or escapes to infinity. If it is absorbed, we can determine where it is absorbed, what direction it comes from when it is absorbed, and how many bounces it makes before this happens. If it escapes to infinity, we can determine its eventual escape path, and how many bounces it takes before reaching this path. The time bound for these algorithms is $O(n^2 \log N + \log^2 N)$.*

Proof. We convert the input to an equivalent partial integer interval exchange transformation according to Lemma 6, and then apply the polynomial-time algo-

rithm for the iterated partial integer interval exchange transformation problem of Lemma 2.

An input of size n and coordinate magnitude N can be converted into a partial integer interval exchange transformation whose number of subintervals is $O(n)$ (each comes from a trapezoid in four trapezoidal decompositions with sides parallel to one of the directions of the reflected rays) and whose range is $O(N^2)$ (combining the magnitude of the environment coordinates with the number of reflection points along each grid segment). For inputs of this size, the time to apply an algorithm for the iterated partial integer interval exchange transformation is $O(n^2 \log N + \log^2 N)$. □

It would be of interest to determine to what extent this algorithm can be generalized. Can we determine the geometric length of the reflected path of a light ray, and not just its number of bounces? Are there other systems of slopes, beyond the axis-aligned and diagonal slopes, for which similar algorithms can work? How does the complexity of the algorithm depend on the system of slopes? For slopes that do not preserve the rationality of reflected rays, what can be said about the computational complexity of the problem?

References

[1] Mridul Aanjaneya, Arijit Bishnu, and Sudebkumar Prasant Pal. Directly visible pairs and illumination by reflections in orthogonal polygons. In Sylvain Petitjean, editor, *Collection of abstracts of the 24th European Workshop on Computational Geometry*, pages 241–244. INRIA-LORIA, March 18–20 2008. URL: `https://physbam.stanford.edu/~aanjneya/mridul_files/papers/vis.pdf`.

[2] Boris Aronov, Alan R. Davis, Tamal K. Dey, Sudebkumar P. Pal, and D. Chithra Prasad. Visibility with multiple reflections. *Discrete & Computational Geometry*, 20(1):61–78, 1998. `doi:10.1007/PL00009378`.

[3] Boris Aronov, Alan R. Davis, Tamal K. Dey, Sudebkumar P. Pal, and D. Chithra Prasad. Visibility with one reflection. *Discrete & Computational Geometry*, 19(4):553–574, 1998. `doi:10.1007/PL00009368`.

[4] David Eppstein. The complexity of iterated reversible computation. Electronic preprint arxiv:2112.11607, 2021.

[5] Jeff Erickson and Amir Nayyeri. Tracing compressed curves in triangulated surfaces. *Discrete & Computational Geometry*, 49(4):823–863, 2013. `doi:10.1007/s00454-013-9515-z`.

[6] Subir Kumar Ghosh, Partha P. Goswami, Anil Maheshwari, Subhas C. Nandy, Sudebkumar Prasant Pal, and Swami Sarvattomananda. Algorithms for computing diffuse reflection paths in polygons. *The Visual Computer*, 28(12):1229–1237, 2012. `doi:10.1007/s00371-011-0670-z`.

[7] Salma Sadat Mahdavi, Ali Mohades, and Bahram Kouhestani. Computing k-link visibility polygons in environments with a reflective edge. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry, Toronto, Ontario, Canada, August 10-12, 2011*, 2011. URL: `https://www.cccg.ca/proceedings/2011/papers/paper63.pdf`.

[8] Joseph O'Rourke and Octavia Petrovici. Narrowing light rays with mirrors. In *Proceedings of the 13th Canadian Conference on Computational Geometry, University of Waterloo, Ontario, Canada, August 13-15, 2001*, pages 137–140, 2001. URL: `https://www.cccg.ca/proceedings/2001/orourke-13443.ps.gz`.

[9] D. Chithra Prasad, Sudebkumar P. Pal, and Tamal K. Dey. Visibility with multiple diffuse reflections. *Computational Geometry: Theory and Applications*, 10(3):187–196, 1998. `doi:10.1016/S0925-7721(97)00021-7`.

[10] Arash Vaezi and Mohammad Ghodsi. Visibility extension via mirror-edges to cover invisible segments. *Theoretical Computer Science*, 789:22–33, 2019. `doi:10.1016/j.tcs.2019.02.011`.

Locked and Unlocked Smooth Embeddings of Surfaces

David Eppstein*

Abstract

We study the continuous motion of smooth isometric embeddings of a planar surface in three-dimensional Euclidean space, and two related discrete analogues of these embeddings, polygonal embeddings and flat foldings without interior vertices, under continuous changes of the embedding or folding. We show that every star-shaped or spiral-shaped domain is unlocked: a continuous motion unfolds it to a flat embedding. However, disks with two holes can have locked embeddings that are topologically equivalent to a flat embedding but cannot reach a flat embedding by continuous motion.

1 Introduction

Much research in computational geometry has concerned smooth deformations of a shape that preserve its geometric structure. This includes bloomings, continuous and collision-free unfoldings from polyhedra to flat nets that preserve the shape of each face [6, 9, 15, 18, 21, 26], the carpenter's rule problem on continuous collision-free motions that straighten a polygonal chain while preserving segment lengths [8, 20, 22], and the closely related study of continuous rigid motions of single-vertex origami patterns [1, 19, 23]. When the carpenter's rule problem is generalized to to more complex linkages or three dimensions it can have *locked* configurations, unable to reach a straightened configuration even though there is no topological obstacle to their reconfiguration [3–5, 7, 11, 16]. Demaine, Devadoss, Mitchell, and O'Rourke studied "folded states" of simple planar polygons in 3d, consisting of a surface-distance-preserving mapping to 3D together with a consistent "local stacking order" at parts of the polygon that are mapped onto each other. As they show, any folded state can be continuously transformed to any other folded state: the configuration space of these states is connected [10, 12].

In this work we examine reconfigurability for three natural restricted forms of folded states:

- Smooth embeddings into $\mathbb{R}^3$, where the embedded surface is doubly differentiable (having a tangent plane everywhere) without self-contact.
- Polygonal (piecewise linear) embeddings into $\mathbb{R}^3$ without interior vertices, so all "fold lines" where the mapping is not locally linear extend entirely across the surface. There should be finitely many connected linear pieces and no self-contact.
- Planar folded states (flat foldings) without interior vertices. We again require that the mapping be piecewise linear with finitely many pieces and that the fold lines extend entirely across the surface.

At each interior point of a smoothly embedded flat surface in $\mathbb{R}^3$ that is not locally flat, the surface bends along a straight "bend line" that continues to the surface's boundary [14]. As an everyday example of this phenomenon, when holding a slice of pizza at its crust, keeping the crust flat allows the slice to droop, but bending it extends rigid bend lines lengthwise through the slice, preventing drooping [25]. Our restriction against interior vertices of polygonal embeddings and flat foldings provides a combinatorial model of the same phenomenon. We have studied flat foldings with this restriction (but not their reconfiguration) in previous work [13].

In all three models of bending and folding we allow continuous motions that stay in the same model; in particular, in the polygonal embedding model, folds may "roll" along the surface rather than remaining fixed in place. Our folded states are special cases of the ones previously considered by Demaine et al. [10, 12], and we retain their notion of a continuous motion as a mapping from the closed unit interval $[0, 1]$ to folded states that is continuous under the sup-norm of the distances of mapped points and (for flat foldings) consistent with respect to the local stacking order. The initial configuration of a motion is the mapping for the parameter value 0, and the final configuration is the mapping for the parameter value 1. For all three of our restricted models of folded states, we prove the following new results:

- A compact subset of the plane with a continuous shrinking motion has a connected space of restricted folded states: every folded state can be continuously unfolded to a flat state. These sets are topological disks and include the star-shaped domains.
- There exist subsets of the plane, topologically equivalent to a disk with two holes, that can be locked: they have embeddings that are topologically equivalent (ambient isotopic) to their flat embedding, but cannot be continuously deformed to become flat.

*Department of Computer Science, University of California, Irvine. This work was inspired by discussions at the 3rd Virtual Workshop on Computational Geometry, held in March 2022, for which we thank the organizers and participants. Research supported in part by NSF grant CCF-2212129.

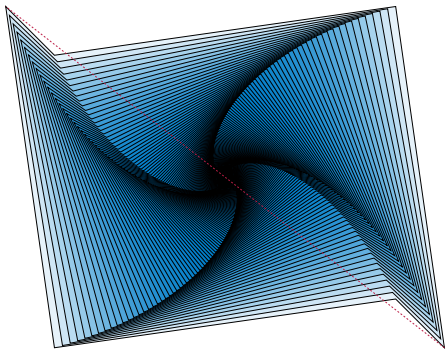


Figure 1: A polygon that has a continuous shrinking motion into itself, but that is not star-shaped.

2 Shapes that can shrink into themselves

A *star-shaped* polygon, or more generally a star-shaped domain, is a subset S of the plane such that, with an appropriate choice of one point of the plane to be the origin, every scaled copy pS for $p \in [0, 1]$ is a subset of S itself. These are widely studied in computational geometry, and can be recognized in linear time [17]. However, these are not the only shapes that have a continuous shrinking motion of scaled copies of the shape into themselves. Fig. 1 depicts a different type of continuous shrinking motion, in which the shape spirals inwards while shrinking. Such a motion can be described by coordinatizing the plane by complex numbers, again for an appropriately chosen origin (the center of the spiral motion), choosing a complex number q of absolute value less than one, and considering the family of scaled copies $e^p qS$ for $p \in [0, \infty)$. The linear shrinking motion of star-shaped domains is a special case of this type of motion in which q is a positive real number. If any shape S has any continuous shrinking motion of its scaled copies into itself, the start of the motion can be approximated to first order by an inward-spiraling shrinking motion of this type, which can then be continued to a complete inward-spiraling shrinking motion of the same shape. In this sense, this family of continuous shrinking motions is completely general, in the sense that all shapes with a continuous shrinking motion have a continuous shrinking motion of this type, although we will not use this fact. Following Aharonov et al. [2], we call a compact set S that has a continuous shrinking motion of this type a *spiral-shaped domain*.

In an inward-spiraling shrinking motion, each point of the set S traces out a logarithmic spiral, which meets every ray from the center of the motion in a fixed angle θ. The existence of a spiraling motion for a given simple polygon and a fixed choice of θ can be tested by intersecting polygonal regions, one for each edge, that describe the set of locations for the center where a spiral of this angle would leave the edge towards the interior of the polygon, rather than towards the exterior. This characterization leads to a polynomial-time algorithm for testing the existence of an inward-spiraling motion, by continuously varying θ over the range $(-\pi, \pi)$ and determining the combinatorial changes in the corresponding intersection of polygonal regions. It is plausible that finding a feasible angle θ and a center point for that choice of θ is an LP-type problem of bounded dimension, solvable in linear time, but we have not proved this.

The main results of this section are that every spiral-shaped domain S has connected spaces of smooth embeddings, polygonal embeddings without interior vertices, and flat-foldings without interior vertices. Equivalently, every embedding can be continuously unfolded. The simplest case concerns smooth embeddings.

Theorem 1. *Every smooth embedding of a spiral-shaped domain has a continuous motion, through smooth embeddings, to a flat embedding.*

Proof. Let S be the given spiral-shaped domain, and $f : S \to \mathbb{R}^3$ be its smooth embedding. Parameterize an inward-spiraling shrinking motion of S as $s_i : S \to S$ where $i \in (0, 1]$, s_1 is the identity, and each s_i scales S by a factor of i, converging as $i \to 0$ to a single central point of the motion (which may or may not be on the boundary of S).

Our proof converts this parameterized family of scalings to a parameterized family of smooth embeddings of S at a single scale, by composing s_i, f, and a function that expands $\mathbb{R}^3$ by a factor of $1/i$ to restore the original size of S. The obvious expansion function $\mathbb{R}^3$ by $(x, y, z) \mapsto (x/i, y/i, z/i)$ does not work, because of the following issues:

- Composing s_i, f, and an expansion by $1/i$ does provide a continuous motion of smooth embeddings on the half-open interval $(0, 1]$, whose curvature tends towards zero as the parameter goes to zero. However, we need a continuous motion on the closed interval $[0, 1]$ for which the limiting embedding at parameter value zero exists and is completely flat.

- When the origin of $\mathbb{R}^3$ does not belong to all of the rescaled and smoothly embedded copies of S, the composition with $\mathbb{R}^3$ by $(x, y, z) \mapsto (x/i, y/i, z/i)$, as $i \to 0$, will produce smooth embeddings of S whose distance from the origin is inversely proportional to i, preventing them from having a limit. We can prevent this by choosing coordinates for $\mathbb{R}^3$ that have as their origin $f(p)$, where p is the limit point of the inward-spiraling shrinking motion on S. In this way, the composition of s_i, f, and an expansion by $1/i$ will act as the identity on this point, and more strongly will preserve the tangent plane of the surface at that point.

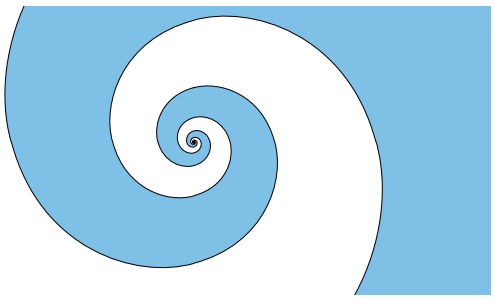


Figure 2: A spiral-shaped domain between two logarithmic spirals, whose shrinking motion cannot be shortcut to a linear shrinking motion.

- This still does not complete the proof, because a spiral inward-shrinking motion of the domain, composed with f and $(x, y, z) \mapsto (x/i, y/i, z/i)$, will cause the embeddings to rotate at increasing speed as $i \to 0$, preventing the continuous motion from having a flat limiting surface at $i = 0$. For the polygonal domain of Fig. 1, this problem can be circumvented by switching from the spiral inward-shrinking motion to a linear scaling transformation in S, once the scaled copies of S become small enough that this linear scaling stays entirely within S. However, for some other shapes, such as the domain between two logarithmic spirals depicted in Fig. 2, switching to linear scaling is never possible.

  Instead, we address this third issue by choosing a reference vector tangent to S at $f(p)$. We compose s_i, f, an expansion by $1/i$, and a rotation of $\mathbb{R}^3$ (with axis perpendicular to the tangent plane at $f(p)$) that restores this vector to its original direction.

Let f_i denote the resulting composition of s_i, f, an expansion of $\mathbb{R}^3$ with a careful choice of origin, and a rotation that restores the original directions of vectors in the tangent plane to the embedded surface. Then f_i, for values of i in the half-open interval $(0, 1]$, describes a continuous motion with $f_1 = f$ as one endpoint of the motion. The maximum curvature of the surface $f_i(S)$ equals i times the maximum curvature within the i-scaled copy of S within $f(S)$, which in the limit becomes arbitrarily close to i times the curvature at p in $f(S)$ and therefore has limiting value zero. The transformations f_i preserve the tangent plane to the surface and directions within the tangent plane.

For each point $q \in S$, $f_i(q)$ can be obtained by the exponential map: follow a curve on $f_i(S)$ of length $|p-q|$, starting from p, in the direction given by the image of the tangent vector $q - p$. This length and direction are invariant through the motion, and as $i \to 0$ the curvature of this path approaches zero. Therefore, f_i converges pointwise to a flat embedding $f_0(S)$, obtained by the exponential map on the tangent plane of $f(S)$ at p. Appending f_0 to our continuous sequence of smooth embeddings f_i for $i \in (0, 1]$ gives us a continuous sequence on $i \in [0, 1]$, flat at $i = 0$ and equal to our starting embedding at $i = 1$, which therefore shows that these two embeddings are reconfigurable to each other. □

This proof uses compactness to ensure that the limit point of the spiral shrinking motion is asymptotically flat and that sufficiently small copies of the entire domain fit into any neighborhood of that point. Non-compact surfaces can have self-similar embeddings, smooth everywhere except the limit point, that are invariant under shrinking and re-expansion. For polygonal embeddings we handle the same issue of avoiding self-similar embeddings differently, using the requirement that these embeddings have finitely many connected linear pieces.

Theorem 2. *Every polygonal embedding without interior vertices of a bounded spiral-shaped domain has a continuous motion, through polygonal embeddings without interior vertices, to a flat embedding.*

Proof. The same idea as above comes close to working: compose the inward-spiraling shrinking motion of the domain, the initial embedding f, an expansion of $\mathbb{R}^3$ centered at the limit point of the motion, and a rotation of $\mathbb{R}^3$ that cancels any spinning motion the inward-spiraling motion might have. However, the limit point p of the inward-spiraling shrinking motion might be a point on a fold line of the polygonal embedding, or worse, it might be a boundary point of S where multiple fold lines meet. In this case, there is not a unique tangent plane of $f(S)$ at p, and when the same composition can be made to have a limit, this limit will be folded at p in the same way as it was in $f(p)$ rather than being flat.

To address these issues, when p is a folding point of the embedding $f(S)$, we choose one of the polygonal faces incident to p to determine the tangent plane for the previous construction. Then as above we compose the inward-spiraling shrinking motion of the domain, the initial embedding f, an expansion of $\mathbb{R}^3$ centered at the limit point of the motion, and a rotation of $\mathbb{R}^3$ that cancels any spinning motion of this tangent plane. The resulting composition defines a continuous motion over the half-open interval of parameter values $(0, 1]$, which can be extended with a well-defined limit at 0, a polygonal folded state that has the same fold lines and fold angles as $f(S)$ at p and is flat everywhere else.

We distinguish three cases:

- If p is not a folding point of the embedding $f(S)$, then the previous proof applies directly.

- If p belongs to a single fold line of the embedding $f(S)$, then we can concatenate two continuous motions. First we perform the motion from the previous proof that transforms $f(S)$ into a folded state

that contains this fold line. Because this folded state has only one fold line, it cannot self-intersect, and forms a valid polygonal embedding. Next, we perform an additional continuous motion that linearly changes the angle of this fold from its initial value to π (unfolded into a flat angle). Again, none of the intermediate states of this second continuous motion can self-intersect.

- In the remaining case, p is a boundary point of S that belongs to multiple fold lines within S. For a line from p to intersect S in a line segment, it must necessarily be the case that the inward-spiraling shrinking motion is actually the linear shrinking motion of a star-shaped domain. The *link* of this folded state at p, the intersection of $f_0(S)$ with a small sphere centered at p, is a polygonal chain consisting of arcs of great circles. Because it remains invariant throughout the motion, it does not self-intersect. Any two distinct points of S at distance d from p lie on distinct points of a scaled copy of this link, on a sphere of radius d, and for this reason cannot coincide. Therefore, the folded state at f_0 is again a valid polygonal embedding.

  By known results on the spherical carpenter's rule problem, there exists a continuous motion of the link, as a polygonal chain of fixed-length great-circle arcs on the sphere, from its folded state to a completely flat state. This motion induces a continuous motion of f_0 to a flat-folded state [23]. Performing the continuous motion from $f(S)$ to $f_0(S)$, and then using this carpenter's rule solution on the resulting single-vertex surface, produces a combined continous motion from $f(S)$ to a flat state. □

To apply the same method to flat foldings without interior vertices, we cannot use the carpenter's rule problem, as 1d flat-folded polygonal chains with fixed edge lengths cannot change continuously. Instead, we use a one-dimensional version of Theorem 2:

Lemma 3. *Let P be line segment, folded flat by a piecewise-isometry $f : P \to R$ with a finite number of fold points and a consistent above-below relation for points with the same image. Then there exists a continuous motion of flat foldings of P that transforms folding f into an unfolded state.*

Proof. Chose any point p of P that is not a fold point, and form a continuous family of one-dimensional folded states of P, $f_i(P)$ for $i \in (0, 1]$, by scaling P by a factor of i centered at p, applying f, and scaling the result by a factor of $1/i$ centered at $f(p)$. When i becomes less than the distance from p to the nearest fold, the result will be an unfolding of P, so this provides a continuous transformation from f to an unfolding. □

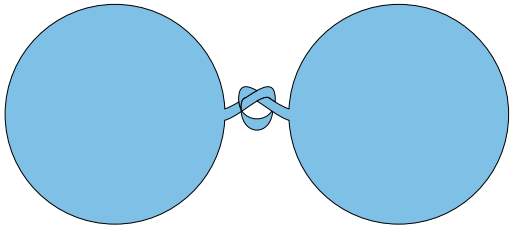


Figure 3: Two disks connected by a knotted band. This surface can be flattened by a continuous motion of smooth embeddings.

Theorem 4. *Every flat folding without interior vertices of a bounded spiral-shaped domain has a continuous unfolding motion through foldings of the same type.*

Proof. The proof follows the same outline as Theorem 2, combining shrinking and unshrinking to reach a folded state in which all folds pass through the center p of the spiral shrinking transformations. If the result has a single fold through p we roll this fold to the boundary of S rather than changing its fold angle. If p lies on the boundary of the domain and belongs to multiple fold lines, we apply Lemma 3 in place of the spherical carpenter's rule. □

3 An instructive example

Fig. 3 shows two large disks connected by a short thin band, embedded with the disks spread flat and the band tied into an open overhead knot. If the disks could be crumpled, it would be easy to untie and flatten, by crumpling the disks into small enough balls that they could be passed through the knot, and then uncrumpling. However, our model of smooth surface embeddings does not allow crumpling. In every smooth embedding, the center point of each disk lies in a flat subset of the disk with large diameter: either a diameter of the disk, or a triangular subset of the disk with the vertices of this triangle on the boundary of the disk.

Rolling up either disk around a diameter makes that diameter act like a rigid line segment, but gives the rest of the disk a smaller overall shape. Rolling both disks in this way can produce an embedding like the locked polygonal chain with long "knitting needles" at its ends from Figure 1 of Biedl et al. [4]. Any other rigid configuration for the two disks would be similarly locked. However, this surface is unlocked! It can be unfolded through the following sequence of transformations:

- Let D be a diameter of one of the two disks, touching the boundary of the disk at its attachment point p with the knotted band. Roll up the disk around D, starting at one of the points of the disk that is farthest from D, and leaving the semicircle opposite that point exposed on the outside of the roll.

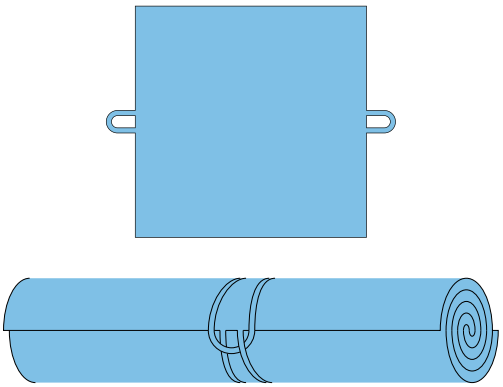


Figure 4: A flat surface with two holes and a configuration that cannot be flattened. Intuitively, the two interlocked loops prevent the rolled center region from unrolling, the bend lines of the roll make it act like a rigid rod, and the length of this rod prevents the loops from being pulled around its ends. For the way the two loops on the hidden side of the roll interlock, see Fig. 5.

- Poke p into the hole made by the knotted band, so that if the rolled-up disk around D were not rigid, it could pass through the hole and untie the knot. However, because D is made rigid by the bending of the embedding as it rolls around D, only the very end of diameter D near p can pass into the hole.
- Continuously spin the parallel family of bend lines on the rolled-up disk, so that it rolls up around a different diameter than D. Choose the direction of spin that causes p to travel along the exposed semicircle along the rolled-up disk. As it does so, this will allow more of the diameter of the rolled-up disk to poke through the hole in the knot.
- When the bend lines have spun on the disk through an angle of π, causing p to reach the other end of the exposed semicircle, the rolled-up diameter will have traveled all of the way through the hole in the knot, which will become unknotted.
- Unroll the disk so that it lies flat with the rest of the surface.

Thus, although bend lines of a smoothly embedded surface are rigid, the underlying pattern of these lines on the surface can change continuously, complicating the search for a proof that a surface is locked.

4 A locked surface

Fig. 4 depicts a smoothly-embedded unit square with two small loops on midpoints of opposite sides, wrapped into a spiral roll with the loops interlocked. Fig. 5 provides a cutaway view showing how the loops interlock.

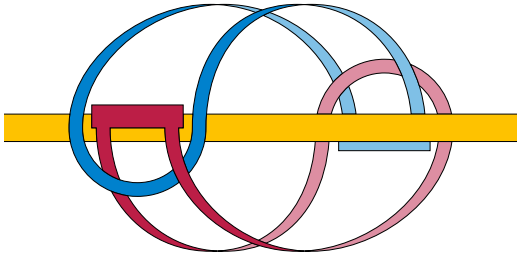


Figure 5: Cutaway view of the two loops and centerline of the rolled-up part of the surface of Fig. 4, showing their topological equivalence to the Borromean rings

This rolled-up and interlocked surface is topologically equivalent (ambient isotopic) to a flattened surface. As can be seen in Fig. 5, the interlocking pattern is that of the Borromean rings, three unknotted loops in space that cannot be separated from each other, in which any two of the loops become unlinked if the third loop is removed. In the figure, the role of one of these three loops is taken by the center of the spiral roll, which does not actually form a loop, so this embedding is not topologically linked. From the arrangement of the loops in Fig. 5, it is possible to unlink it by pulling the right half of the red band to the right and up, around the right end of the yellow spiral center, passing this part of the red band around the right half of the blue band, and then passing the same part of the red band back to the right and down around the right end of the yellow spiral center. This sequence of motions reverses the front-back order of the two red-blue crossings on the right half of the red band, after which it is straightforward to flatten the whole surface. We have also verified that this configuration is topologically unlocked using a physical model of two rubber bands attached to a pen.

As a smoothly-embedded surface, Fig. 4 is locked: it cannot be flattened while preserving its geometry. To prove this, we use three interlocked properties of the embedding that, like the Borromean rings, are interlinked: as long as any two of the properties remain valid, the third one must remain valid as well, so none of the three properties can be the first to break in any continuous motion of the embedding. The first two properties are parameterized by a parameter ε, which we will be able to make arbitrarily small by making the length of the loops sufficiently small:

A. The two loops are bounded within distance ε of each other.

B. The nearest bend line on the square to its center point has distance $\leq \varepsilon$ to the center point, and crosses the top and bottom sides of the square at distance $\geq 1/4 - \varepsilon$ from its left and right sides.

C. The nearest bend line on the square to its center

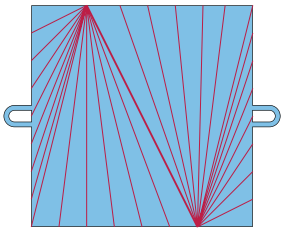


Figure 6: Bend lines for a deformed version of Fig. 4 in which the (heavier) bend line through the center of the square crosses the top and bottom sides of the square at distance 1/4 from the sides. For these bend lines, the two loops still meet near the center of the rolled surface.

point can be completed into a loop by a curve, on a sphere with it as diameter, forming Borromean rings with the other two loops.

The reason for the 1/4 in property B is that it is possible to deform Fig. 4, keeping the two loops interlocked in the center of the roll, so that the center bend line crosses the top and bottom sides of the square at distance 1/4, as shown in Fig. 6. However, as we will prove, it is not possible to move this center bend line significantly farther from vertical.

Lemma 5 ($A \wedge B \Rightarrow C$)**.** *For all sufficiently small loop lengths δ and all sufficiently small ε, continuous motion through states where $A(\varepsilon)$ and $B(\varepsilon)$ hold, starting from a state where C holds, cannot reach a state where C does not hold.*

Proof. Let b be the bend line of property B. Then (Fig. 7) the attachment point p of the right loop must lie within the intersection of two spheres centered at the endpoints of b, with radii equal to the distances from p to those endpoints, because the embeddings we consider are not allowed to increase distances. Similarly, the attachment point of the left loop must lie within the intersection of another two spheres. By property A, these attachment points must be near each other, and until C stops holding, they must also lie near line b. This limits their nearby locations to points of line b that are far from its endpoints on the square, so they cannot reach the sphere through the endpoints of b on which the connecting curve of property C lies.

As the smooth embedding deforms continuously, the bend line nearest the center point can change, but (as long as B continues to hold) only by small amounts, and the connecting curve can be changed by similar small amounts to maintain property C. As the other two loops cannot reach the sphere containing this curve, they cannot cross this curve (even though it does not form a physical obstacle to them) and cannot change the knotted topology that it forms with them. □

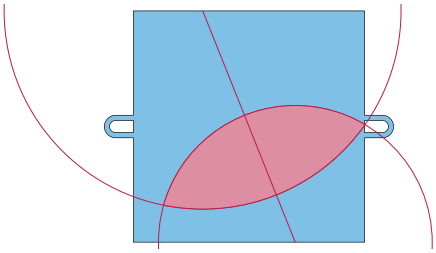


Figure 7: For a bend line through the square's center, the attachment point of the right loop must be in the shaded intersection of circles centered at the bend endpoints, to avoid having greater distance to those endpoints than in the flattened surface.

Lemma 6 ($A \wedge C \Rightarrow B$)**.** *For all sufficiently small ε there exists a δ such that, for loop lengths less than δ, continuous motion through states where $A(2\varepsilon)$ and C hold, starting from a state where $B(\varepsilon)$ holds, cannot reach a state where $B(\varepsilon)$ does not hold.*

Proof. Let b be the bend line nearest the center of the square. While A and C hold, b must pass through the Borromean link formed by it and the two loops, and so (if the center point itself does not lie on a bend line) it must lie on a flattened part of the surface whose width is at most proportional to the loop length. Therefore, b is close to the center point, as part of property B demands. For a bend line that is close to the center point, the same reasoning used in Fig. 7 and Lemma 5 shows that it must be at distance at least $1/4 - \varepsilon$ from the left and right sides, for otherwise the two intervals on this bend line where the left and right loop attachment points must be near would not intersect. □

Lemma 7 ($B \wedge C \Rightarrow A$)**.** *For all sufficiently small ε there exists a δ such that, for loop lengths less than δ, continuous motion through states where $B(2\varepsilon)$ and C hold, starting from a state where $A(\varepsilon)$ holds, cannot reach a state where $A(\varepsilon)$ does not hold.*

Proof. By assumption C, the two loops of the surface and a third loop formed by bend line b form Borromean rings, and by assumption B, line b is near vertical and near the square's center, forcing the attachment points of the two loops to be far from the endpoints of b.

Each loop of the surface has small diameter, so if the two loops could be far from each other it would be possible to make two small spheres (of radius ε), one containing each loop. The loop containing b lies on a straight line within each of these two spheres. Although Borromean rings can have two loops in two disjoint spheres [24] (Fig. 8), the third Borromean ring cannot pass through either sphere as a straight line. If it could, we could deform a loop within its sphere to a circle (as it is not pairwise linked with the line passing through the

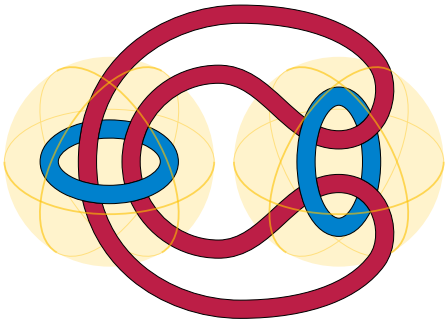


Figure 8: Borromean rings with links in separate spheres

sphere) and span it by a disk not crossed by the other two loops, impossible for the Borromean rings. This contradiction shows that the two loops cannot be separated by a distance larger than ε, as stated in property A. □

These properties together prove that the surface of Fig. 4 is locked: for versions of this surface with short enough loops, it is impossible to deform it as a smoothly embedded surface to its flattened state. The same is true for the same reasons for approximations to this surface by polyhedral embeddings or flat foldings without interior vertices. As a result, we have the following theorem:

Theorem 8. *For smooth embeddings, polyhedral embeddings without interior vertices, and flat foldings without interior vertices, there exist flat surfaces with the topology of a disk with two holes that are ambient isotopic to their flattened form but cannot reach that form by a continuous sequence of folded states staying within the same class of folded states.*

Proof. Choose a sufficiently small $\delta > 0$ and $\varepsilon > 0$ for the lemmas above. The surface of Fig. 4, in the configuration of the figure, has properties $A(\varepsilon)$, $B(\varepsilon)$, and C. As it continuously moves, all three properties remain true; none can be the first to fail, because at the instant it failed, the weaker properties $A(2\varepsilon)$ and $B(2\varepsilon)$ would still be true, which by the lemmas would imply all three of C, $A(\varepsilon)$, and $B(\varepsilon)$. However, the flattened configuration does not obey the three properties, so it cannot be reached. □

5 Conclusions and open problems

We have shown that, for smooth embeddings, polyhedral embeddings without interior vertices, and flat foldings without interior vertices, every spiral-shaped domain can be flattened. However, there exist more complex planar shapes whose configuration spaces are disconnected: they have locked embeddings that, although topologically equivalent, cannot be flattened.

Is every topological disk flattenable? Can the method that we used to flatten Fig. 3 be generalized to other disks? What about surfaces with a single hole? Additionally, we have only investigated the existence of flattenings, but not their algorithmic complexity. For polyhedral embeddings and flat foldings, how hard is it to determine whether a continuous flattening exists? What about for smooth embeddings represented as a piecewise cylindrical and conical surface?

References

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

[2] Dov Aharonov, Mark Elin, and David Shoikhet. Spiral-like functions with respect to a boundary point. *Journal of Mathematical Analysis and Applications*, 280(1):17–29, 2003. doi:10.1016/S0022-247X(02)00615-7.

[3] Brad Ballinger, David Charlton, Erik D. Demaine, Martin L. Demaine, John Iacono, Ching-Hao Liu, and Sheung-Hung Poon. Minimal locked trees. In Frank K. H. A. Dehne, Marina L. Gavrilova, Jörg-Rüdiger Sack, and Csaba D. Tóth, editors, *Algorithms and Data Structures, 11th International Symposium, WADS 2009, Banff, Canada, August 21-23, 2009. Proceedings*, volume 5664 of *Lecture Notes in Computer Science*, pages 61–73. Springer, 2009. doi:10.1007/978-3-642-03367-4_6.

[4] Therese Biedl, Erik D. Demaine, Martin L. Demaine, Sylvain Lazard, Anna Lubiw, Joseph O'Rourke, Mark Overmars, Steve Robbins, Ileana Streinu, Godfried Toussaint, and Sue Whitesides. Locked and unlocked polygonal chains in three dimensions. *Discrete & Computational Geometry*, 26(3):269–281, 2001. doi:10.1007/s00454-001-0038-7.

[5] Therese Biedl, Erik D. Demaine, Martin L. Demaine, Sylvain Lazard, Anna Lubiw, Joseph O'Rourke, Steve Robbins, Ileana Streinu, Godfried Toussaint, and Sue Whitesides. A note on reconfiguring tree linkages: trees can lock. *Discrete Applied Mathematics*, 117(1-3):293–297, 2002. doi:10.1016/S0166-218X(01)00229-3.

[6] Therese Biedl, Anna Lubiw, and Julie Sun. When can a net fold to a polyhedron? *Computational Geometry*, 31(3):207–218, 2005. doi:10.1016/j.comgeo.2004.12.004.

[7] Robert Connelly, Erik D. Demaine, Martin L. Demaine, Sándor P. Fekete, Stefan Langerman, Joseph S. B. Mitchell, Ares Ribó, and Günter Rote. Locked

and unlocked chains of planar shapes. *Discrete & Computational Geometry*, 44(2):439–462, 2010. doi:10.1007/s00454-010-9262-3.

[8] Robert Connelly, Erik D. Demaine, and Günter Rote. Straightening polygonal arcs and convexifying polygonal cycles. *Discrete & Computational Geometry*, 30(2):205–239, 2003. doi:10.1007/s00454-003-0006-7.

[9] Erik D. Demaine, Martin L. Demaine, Vi Hart, John Iacono, Stefan Langerman, and Joseph O'Rourke. Continuous blooming of convex polyhedra. *Graphs and Combinatorics*, 27(3):363–376, 2011. doi:10.1007/s00373-011-1024-3.

[10] 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.

[11] Erik D. Demaine, Stefan Langerman, Joseph O'Rourke, and Jack Snoeyink. Interlocked open and closed linkages with few joints. *Computational Geometry*, 26(1):37–45, 2003. doi:10.1016/S0925-7721(02)00171-2.

[12] 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/.

[13] David Eppstein. Realization and connectivity of the graphs of origami flat foldings. *Journal of Computational Geometry*, 10(1):257–280, 2019. doi:10.20382/jocg.v10i1a10.

[14] Dmitry Fuchs and Serge Tabachnikov. Lecture 14: Paper Möbius band. In *Mathematical Omnibus: Thirty Lectures on Classic Mathematics*, pages 199–206. American Mathematical Society, Providence, Rhode Island, 2007. doi:10.1090/mbk/046.

[15] Yue Hao, Yun hyeong Kim, and Jyh-Ming Lien. Synthesis of fast and collision-free folding of polyhedral nets. In *Proceedings of the 2nd ACM Symposium on Computational Fabrication*, June 2018. doi:10.1145/3213512.3213517.

[16] John Hopcroft, Deborah Joseph, and Sue Whitesides. Movement problems for 2-dimensional linkages. *SIAM Journal on Computing*, 13(3):610–629, 1984. doi:10.1137/0213038.

[17] D. T. Lee and F. P. Preparata. An optimal algorithm for finding the kernel of a polygon. *Journal of the ACM*, 26(3):415–421, 1979. doi:10.1145/322139.322142.

[18] Ezra Miller and Igor Pak. Metric combinatorics of convex polyhedra: Cut loci and nonoverlapping unfoldings. *Discrete & Computational Geometry*, 39(1-3):339–388, 2008. doi:10.1007/s00454-008-9052-3.

[19] 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.

[20] John Pardon. On the unfolding of simple closed curves. *Transactions of the American Mathematical Society*, 361(4):1749–1764, 2009. doi:10.1090/S0002-9947-08-04781-8.

[21] Guang Song and Nancy M. Amato. A motion-planning approach to folding: From paper craft to protein folding. *IEEE Transactions on Robotics and Automation*, 20(1):60–71, February 2004. doi:10.1109/tra.2003.820926.

[22] Ileana Streinu. A combinatorial approach to planar non-colliding robot arm motion planning. In *Proceedings of the 41st IEEE Symposium on Foundations of Computer Science*, pages 443–453, 2000. doi:10.1109/SFCS.2000.892132.

[23] 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 Computer Science*, pages 161–173. Springer-Verlag, 2005. doi:10.1007/11589440_17.

[24] Morwen B. Thistlethwaite. On the algebraic part of an alternating link. *Pacific Journal of Mathematics*, 151(2):317–333, 1991. doi:10.2140/pjm.1991.151.317.

[25] Nicholas Turner, Bill Goodwine, and Mihir Sen. A review of origami applications in mechanical engineering. *Proceedings of the Institution of Mechanical Engineers, Part C*, 230(14):2345–2362, August 2015. doi:10.1177/0954406215597713.

[26] Zhonghua Xi and Jyh-Ming Lien. Continuous unfolding of polyhedra – a motion planning approach. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, September 2015. doi:10.1109/iros.2015.7353828.

Orthogonal Dissection into Few Rectangles

David Eppstein*

Abstract

We describe a polynomial time algorithm that takes as input a polygon with axis-parallel sides but irrational vertex coordinates, and outputs a set of as few rectangles as possible into which it can be dissected by axis-parallel cuts and translations. The number of rectangles is the rank of the Dehn invariant of the polygon.

1 Introduction

Slicing an orthogonal polygon horizontally through each vertex partitions it into rectangles, but may use more rectangles than necessary. Instead, a partition into a minimum number of rectangles can be found in polynomial time, even for polygons with holes. The algorithm finds axis-parallel segments through pairs of non-convex vertices, and uses bipartite matching to find a maximum independent set in the intersection graph of segments. The resulting independent slices, with one additional slice through each remaining non-convex vertex, minimize the number of rectangles [6, 7, 14, 16].

What if we allow sliced pieces to be rejoined? Slicing a polygon into pieces and rejoining them into another polygon is called *dissection*. For example, the Greek cross of Fig. 1 requires three rectangles when partitioned, but has a three-piece dissection into one rectangle, as shown. In fact, every polygon (orthogonal or not) can be dissected into every other polygon of the same area; this is the *Wallace–Bolyai–Gerwien theorem* [2, 9, 11, 20]. Therefore, a dissection into one rectangle always exists. However, this dissection may rotate pieces and use non-axis-aligned cuts, unnatural for orthogonal polygons. Instead, we ask: if we consider dissections that use only axis-parallel slices, translations, and rejoining of the sliced pieces, without rotations, how few rectangles can we dissect a given shape into? For instance, the figure demonstrates that the answer for the Greek cross is one: it can be dissected into a single rectangle. We call this restricted class of dissections *orthogonal dissections*.[1]

Polyominoes such as the Greek cross always have an orthogonal dissection into one rectangle: just subdivide into some number n of squares and rearrange into a $1 \times n$ rectangle. However, we consider polygons with irrational coordinates, for which an orthogonal dissection into a single rectangle may not exist. To address the computational issues that this entails, we assume that all coordinates are presented as rational linear combinations of a *rational basis*, a set of real numbers for which no nontrivial rational linear combination sums to zero. Our main result is an algorithm that in polynomial time computes the minimum number of rectangles into which a given orthogonal polygon can be dissected, and constructs a family of rectangles of that minimum size into which it may be dissected. As we show, this has strong implications for the possibility of dissecting a polygon into a prototile that can tile the plane: such a dissection exists if and only if the minimum number of rectangles is one or two.

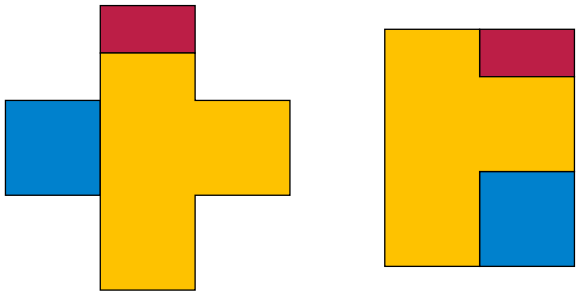


Figure 1: Dissection of a Greek cross into a rectangle, using only axis-parallel cuts and translation of pieces.

The main technical tool that we use for this task is a form of the *Dehn invariant*. The Dehn invariant is a value living in an infinite-dimensional tensor space, usually used for three-dimensional polyhedral dissection problems. One polyhedron can be dissected into another if and only if they have the same volumes and Dehn invariants, and a polyhedron can be dissected to tile space if and only if its Dehn invariant is zero [3, 5, 12, 19]. Another version of the Dehn invariant has also been used for orthogonal dissection of rectangles to rectangles, in order to prove that such a dissection exists if and only if the two rectangles have equal areas and rationally related sides [1, 4, 17, 18]. For instance, because the Greek cross of Fig. 1 (scaled to form a pentomino, with side length one) has an orthogonal dissection into a rectangle with dimensions $2 \times 2\,\frac{1}{2}$, it cannot also be orthogonally dissected into a $\sqrt{5} \times \sqrt{5}$ square. (Instead, it can be

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

[1] It would be natural to allow also 90°-rotations, but our results do not directly extend to dissections that allow such rotations.

dissected into a square using only two straight but not axis-parallel cuts [8].)

Our key insight is that, as order-two tensors, Dehn invariants have significant structure beyond merely being equal or unequal to each other or zero. In particular, like matrices, they have a rank, and this rank is geometrically meaningful. We prove that, for the orthogonal Dehn invariant, the rank equals the minimum number of rectangles that can be obtained from an orthogonal dissection. For the Dehn invariant of polyhedra, we do not have as precise a relation, but the rank of the Dehn invariant (if nonzero) provides a lower bound on the minimum number of edges in a polyhedron to which the given polyhedron can be dissected.

2 Model of computation

We consider inputs that are *orthogonal polygons*, bounded regions of the plane whose boundary consists of finitely many axis-parallel line segments, allowing polygons with holes. We assume that the input specifies the coordinates of each vertex (endpoint of a boundary segment). Because the problems we consider are nontrivial only for polygons with irrational coordinates, it is necessary to say something about how those coordinates are represented and how we compute with them. We assume that the input is described in terms of a *rational basis*, finitely many real numbers with the property that if a linear combination of basis elements with rational-number coefficients adds to zero, all coefficients must be zero.[2] We allow different bases for the x-coordinates and the y-coordinates (an x-basis and a y-basis) or a single combined basis. Each vertex coordinate is a rational linear combination of basis elements, represented as a vector of rational-number coefficients, one for each basis element. The size of the input is the number of rational coefficients needed to describe all of the polygon vertices: the product of the number of vertices with the sum of the sizes of the x-basis and y-basis.

To compute the minimum number of rectangles in an orthogonal dissection, no additional information about the basis elements is necessary. Our algorithm for this version of the problem uses only rational-number arithmetic, and performs a polynomial number of arithmetic operations: essentially, only Gaussian elimination applied to a matrix whose coefficients are quadratic combinations of input coefficients. However, we need additional assumptions that allow computation with basis elements in order to verify that the input describes a polygon without edge crossings, or to construct the rectangles into which it can be dissected. To do these things, we need the following additional primitive operations:

- Find the sign of a rational combination of basis elements, or of a rational combination of products of x-basis elements and y-basis elements.
- Given any two rational combinations of basis elements, or of products of x-basis elements and y-basis elements, find a rational number between them.

We are not aware of past use of this specific computational model. However, exact computation using algebraic numbers is common in computational geometry implementation libraries [13, 15], and it is standard to represent such numbers as rational combinations of roots of a Galois polynomial, a special case of a rational basis. We have stated our results in a more general model that does not specify the algebraic nature of the numbers, so that they can be applied as well to coordinates involving transcendental numbers such as π and e.

3 The orthogonal Dehn invariant

In terms of the given rational basis, the Dehn invariant $\mathcal{D}(P)$ of an orthogonal polygon P can be described as a matrix of rational numbers, with rows indexed by y-basis elements and columns indexed by x-basis elements, constructed as follows:

- Express the given polygon as a linear combination of rectangles R_i. For instance, if coordinate comparisons are available, we may slice the polygon horizontally through each non-convex vertex. If comparisons are unavailable, we may instead choose the line through one horizontal side as a base and consider the family of signed rectangles between each other horizontal side and this base line.
- Express the width w_i and height h_i of each rectangle R_i as a linear combination of basis elements with rational coefficients. The width is the difference of x-coordinates of right and left sides of the rectangle, the height is the difference of y-coordinates of top and bottom sides, and the difference of two linear combinations of basis elements produces another linear combination.
- Construct a matrix M_i, the outer product of the expressions for w_i and h_i. The coefficient of this matrix, for the column corresponding to an x-basis element x_j and the row corresponding to a y-basis element y_k, is a rational number, the product of the coefficient of x_j in w_i and the coefficient of y_k in h_i.
- The Dehn invariant of the polygon is the sum of matrices $\sum_i M_i$.

For example, for the blue polygon in Fig. 2 and the rational basis $\{1, 2^{1/3}, 2^{2/3}\}$, this computation would

[2] A note on terminology: A rational basis does not generally consist of rational numbers. On the contrary, at most one member of the basis can be rational.

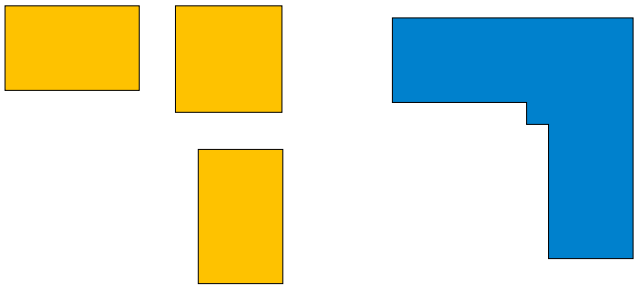


Figure 2: Three rectangles with dimensions $2^{2/3} \times 1$, $2^{1/3} \times 2^{1/3}$, and $1 \times 2^{2/3}$ (yellow), and a polygon formed by gluing them together (blue)

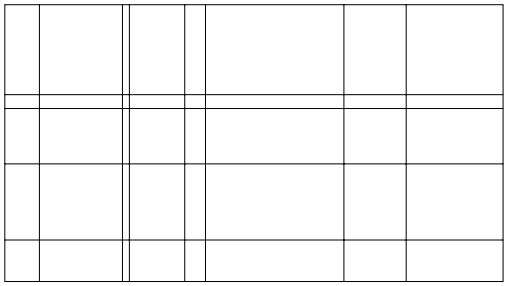


Figure 3: Illustration for Lemma 1: subdividing a rectangle into a grid of smaller rectangles does not change its Dehn invariant.

yield as the Dehn invariant the matrix

$$\begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix},$$

as can be seen from its dissection into yellow rectangles in the figure. These have basis elements as widths and heights, and they each contribute a single 1 coefficient to the total.

Instead of using a specific basis, one can describe the same thing in a basis-free way by writing that the Dehn invariant is an element of the tensor product of $\mathbb{Q}$-vector spaces $\mathbb{R} \otimes_{\mathbb{Q}} \mathbb{R}$, and can be determined as a sum of elements of this tensor product:[3]

$$\mathcal{D}(P) = \sum_i h_i \otimes w_i.$$

It is an invariant of P, in the sense that its value (either thought of as a matrix for a specific basis or as a tensor) does not depend on the decomposition into rectangles used to compute it, and remains unchanged under orthogonal dissections; see Section 4.

In contrast to the polyhedral Dehn invariant, the area of an orthogonal polygon P can be recovered from its Dehn invariant under any basis, as the sum

$$\sum_j \sum_k \mathcal{D}(P)_{kj} x_j y_k$$

of products of matrix coefficients, x-basis elements, and y-basis elements. In this sense, it is meaningful to speak of the area of a Dehn invariant, rather than the area of a polygon.

[3]The Dehn invariant is often written as an element of a tensor product of abelian groups, rather than of vector spaces, using the notation $\mathbb{R} \otimes_{\mathbb{Z}} \mathbb{R}$ or, for the polyhedral invariant, $\mathbb{R} \otimes_{\mathbb{Z}} \mathbb{R}/\mathbb{Z}$. The group notation makes more sense for some contexts; for instance, it works for the polyhedral invariant in hyperbolic or spherical geometry, where linear scaling of polyhedra is not possible. But for our use of tensor rank, vector space notation is more convenient. For the equivalence of matrices and tensors see [10].

4 Invariance of the Dehn invariant

Previous works on the orthogonal Dehn invariant only appear to have considered it with regard to rectangles, rather than for orthogonal polygons more generally. Therefore, for completeness, we prove that it is an invariant of orthogonal polygons under dissection, although we believe this to be implicit in previous work [1,4,17,18]. Throughout this section, we use the abstract tensor space formulation of the orthogonal Dehn invariant; everything carries directly over to the formulation in any particular basis, according to standard principles on the invariance of linear algebra under different choices of basis.

Lemma 1. *Let R be a rectangle with height h and width w. Suppose R is subdivided arbitrarily by vertical and horizontal lines into a rectangular grid of smaller rectangles of heights h_j and widths w_k, as depicted in Fig. 3. For all such subdivisions, $h \otimes w = \sum h_j \otimes w_k$.*

Proof. This follows immediately from the facts that $\sum h_j = h$ and that $\sum w_k = w$, and from the bilinearity of tensors. □

Lemma 2. *Let P be any orthogonal polygon. Then regardless of how P is subdivided into rectangles R_i of height h_i and width w_i, the value $\sum h_i \otimes w_i$ will be unchanged. That is, $\mathcal{D}(P) = \sum h_i \otimes w_i$ is well-defined as an invariant of P.*

Proof. Consider any two different subdivisions into rectangles R_i and R'_i, and refine both subdivisions into a common subdivision by extending vertical and horizontal lines through all vertices of both R_i and R'_i. By Lemma 1, this refinement does not change the sum over the rectangles in either subdivision. Because both of the sums coming from the initially given subdivisions are equal to the sum coming from their common refinement, they must be equal to each other. □

Lemma 3. *If two orthogonal polygons P and P' are related by an orthogonal dissection, then $\mathcal{D}(P) = \mathcal{D}(P')$.*

That is, the Dehn invariant remains invariant under orthogonal dissections.

Proof. We can refine any orthogonal dissection into a dissection for which all pieces are rectangles, and use those rectangles to calculate $\mathcal{D}(P)$ and $\mathcal{D}(P')$. Translating a rectangle obviously does not change its height or width, so the result follows from Lemma 2. □

5 The rank of the Dehn invariant

Any tensor has a rank, the minimum number of terms needed to express it as a sum of tensor products. The Dehn invariants we are considering are order-two tensors over the field of rational numbers, and for any order-two tensor over any field, the rank of the tensor equals the rank of any matrix representing it for any basis over that field. As the rank of a matrix, it equals the minimum number of terms in an expression of the matrix as a sum of outer products of vectors [10]. Therefore, the rank of the Dehn invariant is just the rank of the matrix computed in Section 3. It does not depend on the basis chosen to construct this matrix, and it can be computed using any standard algorithm for matrix rank, such as Gaussian elimination.

If an orthogonal polygon P has an orthogonal dissection into r rectangles with height h_i and width w_i, we have seen that its Dehn invariant can be expressed as

$$\mathcal{D}(P) = \sum_{i=1}^{r} h_i \otimes w_i.$$

This is an expression as a sum of r products, so the Dehn invariant has rank at most r. Conversely, if an orthogonal polygon P has a Dehn invariant with rank r, then it has an expression of exactly this form. However, not all terms of such an expression may be interpreted as describing rectangles. To come from a rectangle, a term $h_i \otimes w_i$ must have $h_i \cdot w_i > 0$, in which case it can come from any rectangle of height $q \cdot |h_i|$ and width $|w_i|/q$ for any positive rational number q. All of these different rectangles produce the same value $h_i \otimes w_i$. But if the product $h_i \cdot w_i$ is a negative number, then $h_i \otimes w_i$ cannot be the Dehn invariant of a rectangle or of any polygon, because it would have negative area. For this reason, the rank of the Dehn invariant lower bounds the number of rectangles that can be obtained in an orthogonal dissection, but it requires an additional argument to prove that these two numbers are equal.

6 Geometric realizability

In the case of the polyhedral Dehn invariant, not every tensor in the space describing these invariants comes from the Dehn invariant of a polyhedron. There exists a surjective homomorphism of groups from the tensor space $\mathbb{R} \otimes_{\mathbb{Z}} \mathbb{R}/\mathbb{Z}$ onto the group $\Omega^1_{\mathbb{R}/\mathbb{Q}}$ of Kähler differentials, such that the tensors coming from Dehn invariants are exactly those mapped to the group identity. The preimages of nonzero Kähler differentials are tensors that do not come from Dehn invariants [5]. In contrast, for the orthogonal Dehn invariant, the only obstacle to geometric realizability is area:

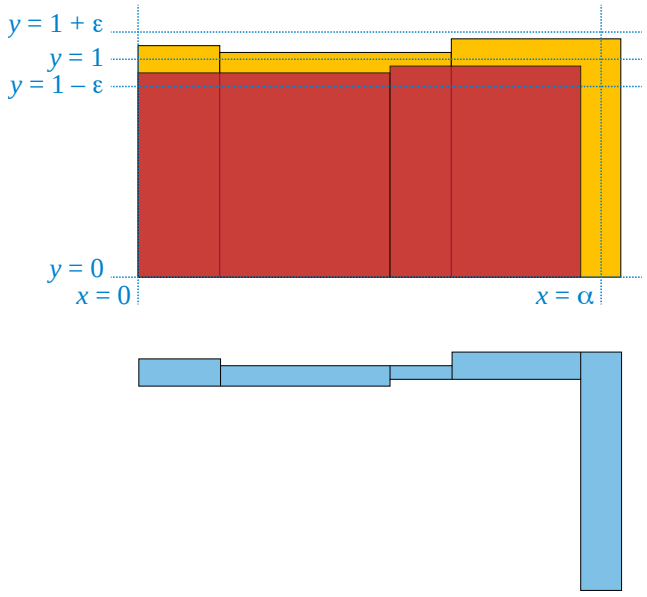


Figure 4: Illustration for Lemma 4: realizing each term in a tensor by a rectangle of height near one, forming the difference of the positive and negative rectangles, and repartitioning the result into rectangles, produces a set of r rectangles having a given Dehn invariant of rank r.

Lemma 4. *Let $D = \sum_{i=1}^{r} h_i \otimes w_i$ be a tensor of rank r in $\mathbb{R} \otimes_{\mathbb{Q}} \mathbb{R}$, and suppose that the putative area $a(D) = \sum_{i=1}^{r} h_i \cdot w_i$ is positive. Then D is the Dehn invariant of a disjoint union of r rectangles.*

Proof. Partition the range of indices $[1, r]$ into two subsets I^+ and I^-, where $i \in I^+$ if $h_i \cdot w_i > 0$ and in i^- otherwise. (Because each term contributes to the rank, it is not possible for $h_i \cdot w_i$ to equal zero.) Let $a^+ = \sum_{i \in I^+} h_i \cdot w_i$ and $a^- = -\sum_{i \in I^+} h_i \cdot w_i$, so that $a(D) = a^+ - a^-$. By assumption this is positive. We may assume without loss of generality that both sets of indices are non-empty: I^+ non-empty is needed to make $a(D)$ positive, and if I^- is empty then we can represent D using the disjoint union of rectangles of height $|h_i|$ and width $|w_i|$ without any additional construction. We can find two rational numbers α and $\varepsilon > 0$ such that $a^- < \alpha < a^+$, with $a^- < \alpha(1-\varepsilon)$ and $a^+ > \alpha(1+\varepsilon)$.[4] These numbers are illustrated with the dashed blue axis-parallel lines in Fig. 4.

[4] Computationally, this uses the assumption from our model of computation that we can find a rational number between two products of combinations of basis elements.

Let A be a rectangle with unit height and with width α. For each index $i \in I^+$, find a positive rational number q_i such that $1 < q_i \cdot |h_i| < 1+\varepsilon$, and construct a rectangle of height $q_i \cdot |h_i|$ and width $|w_i|/q_i$, whose Dehn invariant is $h_i \otimes w_i$. Arranging these rectangles side by side on a common baseline produces an orthogonal polygon P^+ whose height varies between 1 and $1+\varepsilon$, whose area is a^+, and whose Dehn invariant is $\sum_{i\in I^+} h_i \otimes w_i$. In order to achieve area a^+ with height everywhere less than $1+\varepsilon$, P^+ must have width greater than $a^+/(1+\varepsilon) < \alpha$, so it completely covers A. These side-by-side rectangles are shown in yellow in the top part of Fig. 4.

In the same way, for each index $i \in I^-$, find a positive rational number q_i such that $1-\varepsilon < q_i \cdot |h_i| < 1$, and construct a rectangle of height $q_i \cdot |h_i|$ and width $|w_i|/q_i$, whose Dehn invariant is $h_i \otimes w_i$. Arranging these rectangles side by side on a common baseline produces an orthogonal polygon P^- whose area is a^- and whose Dehn invariant is $-\sum_{i\in I^+} h_i \otimes w_i$, entirely within A, the red rectangles in the top part of Fig. 4.

Arranging P^+ and P^- so they share the same bottom left vertex, and computing the set-theoretic difference $P^+ \setminus P^-$, produces a polygon P whose Dehn invariant is D. It can be sliced vertically at each vertex whose x-coordinate is intermediate between its smallest and largest x-coordinate, as shown in the bottom part of Fig. 4. There are $r-1$ slices (one for each side where two rectangles from I^+ meet, and one for each left side of a rectangle from I^-), so the result is a set of r rectangles with total Dehn invariant D, as required. □

7 Dissectability

Long after the work of Dehn, Sydler proved that the polyhedral Dehn invariant is a complete invariant: any two polyhedra with the same volumes and Dehn invariants can be dissected to each other [19]. We need an analogous result for the orthogonal Dehn invariant. We do not bound the number of pieces in a dissection; a polynomial bound is not possible, because even the trivial dissection of a $1 \times n$ rectangle into an $n \times 1$ rectangle requires n pieces, a non-polynomial number.

Theorem 5. *Any two orthogonal polygons with the same Dehn invariant have an orthogonal dissection.*

Proof. We may assume without loss of generality that the two polygons P_1 and P_2 have already been dissected into (different) disjoint sets of rectangles R_1 and R_2. We use induction on the size of rational bases for the heights and widths of these rectangles (which may be a superset of a basis for the Dehn invariant). As a base case, if these bases have size one, all rectangles have heights and widths that are rational multiples of each other. In this case we can scale the x and y coordinates separately to clear denominators in these coordinates and make all rectangle side lengths integers, allowing a dissection using unit squares.

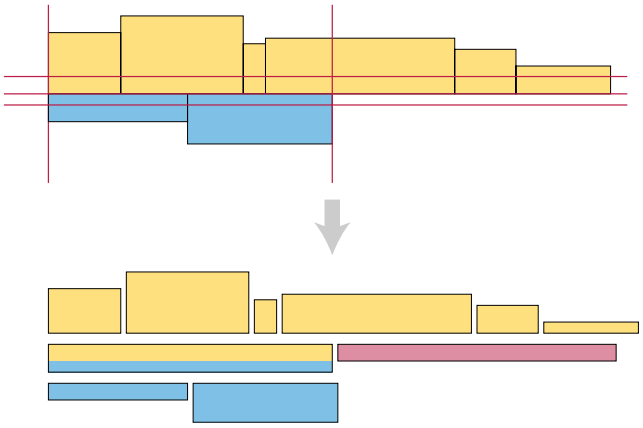


Figure 5: Illustration for Theorem 5. The horizontal red lines are (from top to bottom) $y = \varepsilon^+$, $y = 0$, and $y = -\varepsilon^-$; the vertical lines are (left to right) $x = 0$ and $x = \min\{c_i^+, c_i^-\}$. Slicing the rectangles in R_i^+ (yellow) and R_i^- (blue) by these lines dissects them into a family of rectangles whose heights do not depend on $\hat{y}$ (the bottom blue and yellow rectangles) together with a single rectangle whose coefficient of $\hat{y}$ is ± 1 (red).

Otherwise, by the symmetry of heights and widths, we can assume without loss of generality that the y-basis has at least two elements; let $\hat{y}$ be one of them. For each rectangle in R_1 and R_2, of width w_i and height h_i let q_i be the coefficient of $\hat{y}$ in the expansion of h_i as a rational combination of basis elements. Whenever $q_i \neq 0$, apply the base case of the theorem (for the one-element bases $\{w_i\}$ and $\{h_i\}$) to dissect that rectangle into another rectangle of width $q_i \cdot w_i$ and height h_i/q_i. After this step, for all rectangles in R_1 and R_2, the coefficient of $\hat{y}$ in the rectangle height belongs to $\{-1, 0, 1\}$. Let R_i^+ be the rectangles in R_i for which this coefficient is 1, R_i^- be the rectangles for which it is -1, and R_i^0 be the rectangles for which it is 0. By composition of dissections, a dissection of these modified sets of rectangles into each other will lead to a dissection of P_1 and P_2 into each other.

For each of P_1 and P_2, translate the rectangles of R_i^+ so they are placed side by side, with their bottom sides all placed on the x-axis, with the left side of the leftmost rectangle placed on the y-axis. Similarly translate the rectangles of R_i^- so they are side by side, with their top sides all placed along the x-axis, and again with the left side of the leftmost rectangle placed on the y-axis. Let ε be the smallest height of any rectangle in either R_1 or R_2. Choose two numbers $0 < \varepsilon^+ < \varepsilon$ and $0 < \varepsilon^- < \varepsilon$, so that both of these numbers are expressible as a rational combination of elements of the y-basis, with the coefficient of $\hat{y}$ in ε^+ equal to 1 and the coefficient of $\hat{y}$ in ε^- equal to -1. (The ability to

make this choice hinges on the fact that the rational multiples of any remaining basis element are dense in the real number line.) Let c_i^+ be the x-coordinate of the right end of the rightmost rectangle in the placement of R_i^+, and define c_i^- symmetrically.

Now that the rectangles have been placed in this way, slice them by the horizontal lines $y = \varepsilon^+$ and $y = -\varepsilon^-$. This leaves a hexagonal region between these two lines, which we dissect into two rectangles by slicing it with the vertical line $x = \min\{c_i^+, c_i^-\}$ (two different lines, one for R_1 and the other for R_2). The dissection is shown in Fig. 5.

The rectangles in R_i^0, the remaining parts of rectangles in R_i^+ above the line $x = \varepsilon^+$, and the remaining parts of rectangles in R_i^- below the line $x = \varepsilon^-$, all have heights whose rational expansion in terms of the y-basis does not use $\hat{y}$. The rectangle to the left of the vertical slice line, and between the two horizontal slice lines, has height $\varepsilon^+ + \varepsilon^-$; here, the coefficients of $\hat{y}$ cancel leaving a rectangle height whose expansion in terms of the basis does not use $\hat{y}$. This leaves all dependence on $\hat{y}$ concentrated in one remaining rectangle, to the left of the vertical slice line, with height ε^+ or ε^- and width $|c_i^+ - c_i^-|$. Let $\hat{w}_i$ denote this width.

Because all remaining pieces except this rectangle have heights that do not depend on $\hat{y}$, it follows that the coefficients of $\mathcal{D}(P_i)$, in the row of the coefficient matrix corresponding to basis element $\hat{y}$, are exactly the coefficients in the rational expansion of $\hat{w}_i$ for a rectangle of height ε^+, or the negation of those coefficients for a rectangle of height ε^-. By the assumption that $\mathcal{D}(P_1) = \mathcal{D}(P_2)$, these matrix coefficients must be equal. The widths $\hat{w}_1$ and $\hat{w}_2$ of the rectangles can be recovered, up to their signs, as the number represented by these coefficients using the y-basis. It is not possible for the two signs to be different, because both $\hat{w}_1$ and $\hat{w}_2$ are non-negative. Therefore, the two remaining rectangles must both have the same height, ε^+ or ε^-, and the same width, $\hat{w}_1 = \hat{w}_2$, and need no more dissection to be transformed into each other.

We have shown that P_1 and P_2 can be dissected into two congruent rectangles whose height expansion uses $\hat{y}$, and into a larger number of additional rectangles whose height expansion does not use $\hat{y}$. These remaining rectangles have a smaller basis for their heights and (because we have removed a congruent rectangle from each polygon) have equal Dehn invariants. The result follows from the induction hypothesis. □

8 Putting the pieces together

We are now ready to prove our main results:

Theorem 6. *The minimum number of rectangles into which an orthogonal polygon can be dissected by axis-parallel cuts and translation equals the rank of its orthogonal Dehn invariant.*

Proof. This number of rectangles is lower-bounded by the rank, by the discussion in Section 5. If the rank is r, then there exists a set of r rectangles with the same invariant as the polygon, by Lemma 4. The given polygon can be dissected into these rectangles, by Theorem 5. □

Theorem 7. *We can compute the minimum number of rectangles into which an orthogonal polygon can be dissected, given a representation for its coordinates over a rational basis, in a polynomial number of rational-arithmetic operations. We can construct a minimal set of rectangles into which it can be dissected, in a polynomial number of operations using arithmetic over the given rational basis.*

Proof. To compute the rank, we compute the Dehn invariant as described in Section 3, and apply any polynomial-time algorithm for computing the rank of a rational-number matrix, such as Gaussian elimination. To construct the rectangles, we follow the steps in the proof of Lemma 4, which uses only a polynomial number of operations involving comparing linear combinations of basis elements and finding rational approximations to them. □

We remark that, as well as counting rectangles, the rank of the orthogonal Dehn invariant can also count vertices. Every set of r disjoint rectangles can be glued together to form an orthogonal polygon with $2r + 2$ vertices, and every orthogonal polygon with this many vertices can be sliced at its non-convex vertices into r rectangles. Therefore, the minimum number of vertices that a polygon of Dehn rank r can be orthogonally dissected into is $2r + 2$.

9 Dissection into prototiles

Another use of the polyhedral Dehn invariant, besides dissection of one shape into another, involves tiling. Any polyhedron that tiles space must have Dehn invariant zero, and any polyhedron with Dehn invariant zero can be dissected into a different polyhedron that tiles space. For the axis-parallel polygonal Dehn invariant we study, things don't work out quite so neatly. The Greek cross can tile, but has nonzero Dehn invariant. More, any axis-parallel polygon can be cut into multiple rectangles, and these can tile space aperiodically by grouping them into rows of the same type of rectangle (Fig. 6). So the Dehn invariant cannot be used to prove that such a thing is impossible, because it is always possible. If we could rotate pieces, we could also rearrange certain sets of more than two rectangles, such as the three rectangles of Fig. 2, into a single-piece axis-parallel hexagon that could tile the plane periodically (Fig. 7).

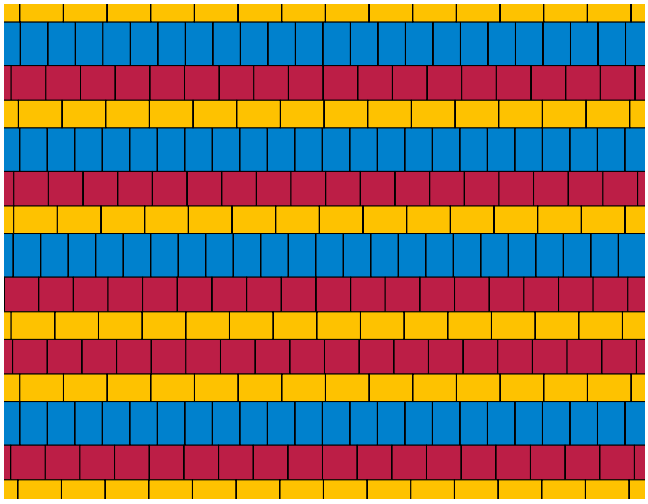


Figure 6: Construction of a non-periodic tiling from an arbitrary dissection into rectangles

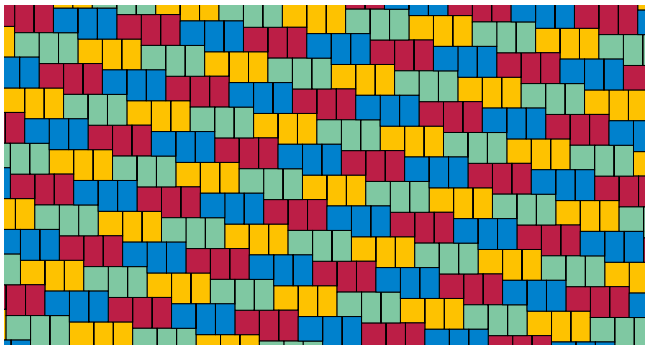


Figure 7: With orthogonal rotations of pieces, the polygon of Fig. 2 can be dissected to form the prototile of a periodic tiling of the plane.

However, for the orthogonal dissections considered here, without rotation, the rank of the Dehn invariant does produce a limitation on the ability to tile periodically without rotation:

Theorem 8. *An orthogonal polygon P, or any finite number of copies of P, has an orthogonal dissection to a prototile that can tile the plane periodically if and only if the rank of its Dehn invariant is at most two.*

Proof. If the rank of the Dehn invariant is one, P can be dissected to a rectangle, which tiles periodically. If the rank is two, P can be dissected into two rectangles, and reassembled into a hexagon, which (like the prototiles of Fig. 7) tile periodically.

Combining n copies of P multiplies the Dehn invariant by the scalar n, which does not change the rank. Every periodic tiling of the plane has a fundamental region in the shape of an axis-parallel hexagon, like the prototiles of Fig. 7. (Because it tiles by translation, this fundamental region may combine a finite number of prototiles of the tiling.) If copies of P could be dissected to the prototile of a tiling, they could also be dissected to this fundamental region, which has Dehn invariant at most two. □

In particular, as a shape whose Dehn invariant has rank three, the orthogonal polygon of Fig. 2 has no orthogonal dissection to a prototile for a periodic tiling of the plane.

10 Conclusions

We have shown that the rank of the orthogonal Dehn invariant of an orthogonal polygon controls the number of rectangles into which it can be dissected by axis-parallel slices and translation, leading to a polynomial time algorithm to compute this number of rectangles or to construct an optimal set of rectangles into which it can be dissected. The dissection itself may require a non-polynomial number of pieces. The number of rectangles, in turn, controls the ability to dissect a polygon into a shape that tiles the plane. One natural question for future research is whether these results can be extended to dissections that allow 90° rotations.

The rank of the polyhedral Dehn invariant, similarly, provides a lower bound on the number of edges of a polyhedron into which a given polyhedron may be dissected, because every polyhedron's Dehn invariant is defined as a sum of tensors over its edges, with rank at most the number of edges in the sum. It is tempting to guess that this lower bound provides a constant-factor approximation to the minimum number of edges in a dissection, but we have been unable to prove this. What would be needed is a construction of a polyhedron with a given Dehn invariant and with a number of edges proportional to the rank of the invariant, analogous to Lemma 4, but this is made more difficult by the fact that not all tensors are realizable as polyhedral Dehn invariants.

Order-two tensors have additional invariants beyond their rank, obtained as the coefficients of the characteristic polynomial or as any function of those coefficients. (The rank can be obtained in this way from the difference in degrees of the highest-degree and lowest-degree nonzero coefficients.) Our work naturally raises the question: which of these invariants are meaningful for dissection problems, and what do they mean?

References

[1] David Benko. A new approach to Hilbert's third problem. *American Mathematical Monthly*, 114(8):665–676, 2007. doi:10.1080/00029890.2007.11920458.

[2] Wolfgang Bolyai. Transmutatio figurarum quoad areas; et hinc reductio earum ad formam rectae. In

Tentamen iuventutem studiosam in elementa matheseos purae elementaris ac sublimioris methodo intuitiva evidentiaque huic propria introducendi, cum appendici triplici, volume 2, pages 108–111. Sumptibus Academiae Scientiarum Hungaricae, 2nd edition, 1904. URL: https://archive.org/details/wolfgangibolyai01akadgoog/page/108.

[3] Max Dehn. Ueber den Rauminhalt. *Mathematische Annalen*, 55(3):465–478, 1901. doi:10.1007/BF01448001.

[4] Max Dehn. Über Zerlegung von Rechtecken in Rechtecke. *Mathematische Annalen*, 57:314–332, 1903. doi:10.1007/BF01444289.

[5] Johan L. Dupont. *Scissors congruences, group homology and characteristic classes*, volume 1 of *Nankai Tracts in Mathematics*. World Scientific, River Edge, New Jersey, 2001. doi:10.1142/9789812810335.

[6] David Eppstein. Graph-theoretic solutions to computational geometry problems. In Christophe Paul and Michel Habib, editors, *Graph-Theoretic Concepts in Computer Science, 35th International Workshop, WG 2009, Montpellier, France, June 24–26, 2009, Revised Papers*, volume 5911 of *Lecture Notes in Computer Science*, pages 1–16. Springer, 2009. doi:10.1007/978-3-642-11409-0_1.

[7] L. Ferrari, P. V. Sankar, and J. Sklansky. Minimal rectangular partitions of digitized blobs. *Computer Vision, Graphics, and Image Processing*, 28(1):58–71, 1984. doi:10.1016/0734-189X(84)90139-7.

[8] Greg N. Frederickson. *Dissections: Plane & Fancy*. Cambridge University Press, 1997. For the Greek cross to square dissection, see pp. 105–106. doi:10.1017/CBO9780511574917.

[9] Paul Gerwien. Zerschneidung jeder beliebigen Anzahl von gleichen geradlinigen Figuren in dieselben Stücke. *Journal für die reine und angewandte Mathematik*, 10:228–234, January 1833. doi:10.1515/crll.1833.10.228.

[10] Wolfgang Hackbusch. *Tensor Spaces and Numerical Tensor Calculus*, volume 42 of *Springer Series in Computational Mathematics*. Springer, 2012. See in particular the identification of matrices with tensors on pp. 5, and 55–56, and the equivalence of multiple definitions of matrix rank on p. 23. doi:10.1007/978-3-642-28027-6.

[11] W. H. Jackson. Wallace's theorem concerning plane polygons of the same area. *American Journal of Mathematics*, 34(4):383–390, October 1912. doi:10.2307/2370498.

[12] Børge Jessen. The algebra of polyhedra and the Dehn–Sydler theorem. *Mathematica Scandinavica*, 22(2):241–256, 1968. URL: https://www.jstor.org/stable/24489773, doi:10.7146/math.scand.a-10888.

[13] Menelaos I. Karavelas. Exact geometric and algebraic computations in CGAL. In Komei Fukuda, Joris van der Hoeven, Michael Joswig, and Nobuki Takayama, editors, *Mathematical Software – ICMS 2010, Third International Congress on Mathematical Software, Kobe, Japan, September 13–17, 2010, Proceedings*, volume 6327 of *Lecture Notes in Computer Science*, pages 96–99. Springer, 2010. doi:10.1007/978-3-642-15582-6_20.

[14] Witold Lipski, Jr., Elena Lodi, Fabrizio Luccio, Cristina Mugnai, and Linda Pagli. On two-dimensional data organization II. *Fundamenta Informaticae*, 2:245–260, 1979. doi:10.3233/FI-1978-2116.

[15] Kurt Mehlhorn and Stefan Schirra. Exact computation with leda_real – theory and geometric applications. In Götz Alefeld, Jiří Rohn, Siegfried Rump, and Tetsuro Yamamoto, editors, *Symbolic Algebraic Methods and Verification Methods*, pages 163–172. Springer, 2001. doi:10.1007/978-3-7091-6280-4_16.

[16] T. Ohtsuki. Minimum dissection of rectilinear regions. In *Proc. IEEE Int. Symp. Circuits and Systems*, pages 1210–1213, 1982.

[17] Jeroen Spandaw. Dissecting cuboids into cuboids. *American Mathematical Monthly*, 111(5):425–429, 2004. doi:10.2307/4145269.

[18] John Stillwell. 5.6: The Dehn invariant. In *Numbers and geometry*, Undergraduate Texts in Mathematics, pages 161–165. Springer-Verlag, New York, 1998. doi:10.1007/978-1-4612-0687-3.

[19] Jean-Pierre Sydler. Conditions nécessaires et suffisantes pour l'équivalence des polyèdres de l'espace euclidien à trois dimensions. *Commentarii Mathematici Helvetici*, 40:43–80, 1965. doi:10.1007/bf02564364.

[20] William Wallace and John Lowry. Question 269. In Thomas Leybourn, editor, *New Series of the Mathematical Repository*, volume 3, pages 44–46. W. Glendinning, London, 1814. URL: https://archive.org/details/mathematicalrep00leybgoog/page/n54.

Minimum Enclosing Spherical/Cylindrical Shells in High-Dimensional Streams

Mohammad Javad Eslami-Bidgoli* Hamid Zarrabi-Zadeh†

Abstract

Given a point set P in $\mathbb{R}^d$, the minimum enclosing spherical (resp., cylindrical) shell problem is to find a sphere (resp., a cylinder) best fitting P. We show that in the single-pass streaming model, any algorithm for the minimum enclosing spherical/cylindrical shell problem with an approximation factor better than $d^{1/3}/4$ requires a memory size exponential in d.

1 Introduction

Shape fitting is a fundamental problem in computational geometry, with various applications to machine learning, data mining, statistics, and computer vision. The objective in the shape fitting problem is to find a shape from a certain family of shapes, best fitting a given point set. Examples of geometric shape fitting problems include minimum enclosing ball, width, minimum-radius enclosing cylinder, and minimum-width enclosing spherical/cylindrical shell.

In this paper, we consider shape fitting problems in the streaming model, where input points arrive one at a time, and the algorithm has a limited storage, so it cannot keep all the points received so far in memory. Moreover, we are considering the problems in high dimensions, where the dimension, d, can be arbitrarily large.

In fixed dimensions, a $(1+\varepsilon)$-approximation for many geometric shape fitting problems can be computed efficiently in linear time using the idea of core-sets [1]. The idea can be extended to the streaming model as well, leading to efficient $(1+\varepsilon)$-approximation streaming algorithms that require only $1/\varepsilon^{O(d)}$ space [1, 4, 7].

In high dimensions, however, the above $(1+\varepsilon)$-factor approximation algorithms are not applicable, due to their exponential dependency in d. Nevertheless, there are a few results on shape fitting in high-dimensional streams. For the minimum enclosing ball problem, Zarrabi-Zadeh and Chan [8] presented a simple 3/2-approximation algorithm that works in any dimension, using only $O(d)$ space. Agarwal and Sharathkumar [2] presented another $O(d)$-space streaming algorithm with an approximation factor of $\frac{1+\sqrt{3}}{2}+\varepsilon \approx 1.37$. The approximation factor of their algorithm was later improved to 1.22 by Chan and Pathak [5], getting very close to the lower bound of $\frac{1+\sqrt{2}}{2} \approx 1.207$ known for the problem [2]. For the minimum enclosing cylinder problem, Chan [4] gave an elegant $(5+\varepsilon)$-approximation streaming algorithm using only $O(d)$ space. For the width problem, Agarwal and Sharathkumar [2] proved a lower bound of $d^{1/3}/8$ on the approximation factor of any streaming algorithm for the problem whose memory size is sub-exponential in d.

In this paper, we study two other important shape fitting problems in high dimensions, namely the minimum enclosing spherical shell and the minimum enclosing cylindrical shell in the data stream model. We show that any (randomized) streaming algorithms that approximates the minimum enclosing spherical/cylindrical shell to within a factor better than $d^{1/3}/4$ (with probability at least 2/3) requires a memory size exponential in d.

To prove our lower bounds, we use the well-known lower bound on the one-round communication complexity of the index problem [6], in the same way used by Agarwal and Sharathkumar [2]. However, technical details of our proofs are non-trivial, and require carefully exploiting geometric properties of spherical/cylindrical shells. To the best of our knowledge, the two problems studied in this paper—despite being central in geometric shape fitting—have not been studied before in high-dimensional streams. As such, this work is an important step towards better understanding the limitations of shape fitting in high-dimensional data streams.

2 Preliminaries

Given a point $c \in \mathbb{R}^d$ and two positive real numbers r_1 and r_2 such that $r_1 < r_2$, the closed region between two concentric spheres of radii r_1 and r_2 centered at c is called a *spherical shell*. The *width* of the spherical shell is defined as $r_2 - r_1$. Given a line ℓ in $\mathbb{R}^d$ and two positive real numbers r_1 and r_2 such that $r_1 < r_2$, the closed region between two coaxial cylinders with axis ℓ and radii r_1 and r_2 is called a *cylindrical shell*. Likewise, the value $r_2 - r_1$ is called the *width* of the cylindrical shell. The *minimum enclosing spherical (resp., cylindri-*

*Department of Computer Engineering, Sharif University of Technology. Email: bidgoli@ce.sharif.edu.

†Department of Computer Engineering, Sharif University of Technology. Email: zarrabi@sharif.edu.

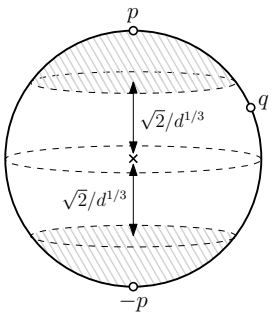


Figure 1: Two points p and q on $\mathbb{S}^{d-1}$.

cal) shell problem is to find a minimum-width spherical (resp., cylindrical) shell that encloses the whole input point set.

Let $\mathbb{S}^{d-1}$ be the a sphere in $\mathbb{R}^d$ centered at origin. Namely, $\mathbb{S}^{d-1} = \{p \in \mathbb{R}^d \mid \|op\| = 1\}$, where o denotes the origin. We call a point set P *symmetric* if for every point p in P, $-p$ is also in P. Given two points/vectors u and v in $\mathbb{R}^d$, we denote by $u \cdot v$ the inner product of u and v. We also denote by $\exp(x)$ the function e^x, where e is the natural number. The following lemma, which is stated in a slightly different form in [2], will be used to derive our lower bounds.

Lemma 1 *There is a symmetric point set $K \subseteq \mathbb{S}^{d-1}$ of size $\Omega(\exp(d^{1/3}))$ such that for every pair of distinct points $p, q \in K$, if $q \neq -p$, then $|p \cdot q| \leq \sqrt{2}/d^{1/3}$.*

Proof. Let $p \in \mathbb{S}^{d-1}$ and $0 < \delta \leq 1$. The hyperplane at distance δ from the origin and normal to p partitions $\mathbb{S}^{d-1}$ into two parts. We denote the smaller part by $\text{cap}(p, \delta)$. Now, consider the point set K returned by the following algorithm.

Algorithm 1 WELL-SEPARATED POINT SET

```
F ← S^{d−1}, K ← ∅
while F ≠ ∅ do
    pick an arbitrary point p in F
    K ← K ∪ {p, −p}
    F ← F \ (cap(p, √2/d^{1/3}) ∪ cap(−p, √2/d^{1/3}))
return K
```

Let p and q be two points in K, such that $q \neq -p$. Suppose that p is added to K before q by the algorithm. Then, it is clear by our construction that q is at distance at most $\sqrt{2}/d^{1/3}$ from the hyperplane passing through the origin and normal to p, which means $|p \cdot q| \leq \sqrt{2}/d^{1/3}$. (See Figure 1.)

To analyze the size of K, let $S(X)$ denote the area of a surface X. It is well-known [3] that for any $p \in \mathbb{S}^{d-1}$ and any $0 \leq \delta \leq 1$,

$$\frac{S(\text{cap}(p, \delta))}{S(\mathbb{S}^{d-1})} \leq \exp(-d\delta^2/2).$$

Therefore, at any iteration of the algorithm, no more than $2/\exp(d^{1/3})$ of the surface of $\mathbb{S}^{d-1}$ is removed from F. Hence, $|K| = \Omega(\exp(d^{1/3}))$. □

3 Spherical Shell

In this section, we provide a lower bound on the approximation factor of any streaming algorithm for the minimum enclosing spherical shell problem whose storage size is sub-exponential in d. The following lemma provides the main ingredient of our proof.

Lemma 2 *Let $\{\vec{u}_1, \ldots, \vec{u}_d\}$ be a set of orthogonal unit vectors in $\mathbb{R}^d$. Then any spherical shell that encloses the point set $P = \{o, \pm\vec{u}_1\} \cup \bigcup_{i=2}^{d} \{\pm\sqrt{d}\vec{u}_i, \pm(\sqrt{d}+2)\vec{u}_i\}$ has width at least $\frac{\sqrt{2}}{2}$.*

Proof. Suppose that the minimum spherical shell enclosing P is centered at a point $a = (a_1, \ldots, a_d)$, where $a_i = a \cdot \vec{u}_i$. Due to symmetry of P around the origin, we can assume without loss of generality that $a_i \geq 0$ for all $i \in \{1, \ldots, d\}$. Moreover, by symmetry of definition of P on dimensions 2 to d, we can assume without loss of generality that $a_2 \geq a_i$ for all $i \in \{3, \ldots, d\}$, i.e., a_2 is the biggest number among $a_2, \ldots, a_d$. We consider the following four cases:

(i) $a_1 \leq a_2\sqrt{d}$ and $a_2 \geq 1$

(ii) $a_1 \leq a_2\sqrt{d}$ and $a_2 \leq 1$

(iii) $a_1 \geq a_2\sqrt{d}$ and $a_1 \geq 1$

(iv) $a_1 \geq a_2\sqrt{d}$ and $a_1 \leq 1$

Let $w_a(P)$ denote the width of the minimum spherical shell centered at a enclosing P, i.e., $w_a(P) = \max_{p \in P} \|pa\| - \min_{p \in P} \|pa\|$. To prove the lemma, it is enough to show that in all the above four cases, $w_a(P) \geq \frac{\sqrt{2}}{2}$. We will prove the first case here, and leave the rest to the appendix. By the definition of $w_a(P)$ we have

$$\begin{aligned} w_a(P) &\geq w_a\Big(\{\pm\sqrt{d}\vec{u}_2\}\Big) \\ &= \sqrt{a_1^2 + (a_2 + \sqrt{d})^2 + a_3^2 + \cdots + a_d^2} \\ &\quad -\sqrt{a_1^2 + (a_2 - \sqrt{d})^2 + a_3^2 + \cdots + a_d^2} \\ &= \sqrt{\Sigma_{i=1}^d a_i^2 + d + 2a_2\sqrt{d}} \\ &\quad -\sqrt{\Sigma_{i=1}^d a_i^2 - d + 2a_2\sqrt{d}}. \end{aligned}$$

Note that the function $f(x) = \sqrt{x + c} - \sqrt{x}$ is decreasing in x, for all positive c. Therefore, in the above expression, if we increase both values under the roots by the

same amount, the expression becomes smaller. Therefore,

$$\begin{aligned}
w_a(P) &\geq \sqrt{2da_2^2 + d + 4a_2\sqrt{d}} - \sqrt{2da_2^2 + d} \\
&\quad \text{(as } a_2\sqrt{d} \geq a_1 \text{ and } a_2 \geq a_i \text{ for } i = 3, \ldots, d\text{)} \\
&\geq \sqrt{3da_2^2 + 4a_2\sqrt{d}} - \sqrt{3da_2^2} \qquad \text{(as } a_2 \geq 1\text{)} \\
&\geq \sqrt{3}\left(\sqrt{da_2^2 + a_2\sqrt{d} + \frac{a_2\sqrt{d}}{3}} - \sqrt{da_2^2}\right) \\
&\geq \sqrt{3}\left(\sqrt{da_2^2 + a_2\sqrt{d} + \frac{1}{4}} - \sqrt{da_2^2}\right) \\
&\quad \text{(as } a_2, d \geq 1 \text{ and } a_2\sqrt{d}/3 > 1/4\text{)} \\
&= \sqrt{3}\left(\sqrt{\left(a_2\sqrt{d} + \frac{1}{2}\right)^2} - \sqrt{da_2^2}\right) \\
&= \frac{\sqrt{3}}{2},
\end{aligned}$$

which implies $w_a(P) \geq \frac{\sqrt{2}}{2}$ in Case (i). The proof of the other three cases are provided in the appendix. □

Now, we are ready to provide our lower bound. Suppose $\mathbb{A}$ is a streaming algorithm that approximates the minimum enclosing spherical shell within a factor better than $d^{1/3}/4$ with probability at least 2/3. We reduce from the following problem in communication complexity:

Index Problem. *Alice has a binary string $a_1 \ldots a_k$ and Bob has an index $i \in \{1, \ldots, k\}$. Alice can send Bob a message to inform him about her string and then Bob should determine whether a_i is 0 or 1.*

It is known [6] that in any algorithm for the index problem that succeeds with probability at least 2/3, the size of the message sent by Alice to Bob is $\Omega(k)$. By reducing from the index problem to the minimum enclosing spherical shell problem, we will show that the space required by $\mathbb{A}$ is $\Omega(\exp(d^{1/3}))$.

Let d be the smallest integer such that $k < \exp(d^{1/3})$. By Lemma 1, it is possible to choose a set K of k pair of well-separated points on $\mathbb{S}^{d-1}$. Let K^+ be the subset of points of K lying on the hemisphere $x_1 \geq 0$, and let $f : \{1, \ldots, k\} \to K^+$ be a one-to-one function. We assume that Alice and Bob are both aware of f and K, as these are independent of Alice's string or Bob's index. Alice gives the points $\{\pm f(j) \mid a_j = 1\}$ to $\mathbb{A}$ and then sends the working space of $\mathbb{A}$ to Bob. Bob then gives the set of points $\{o\} \cup \bigcup_{j=2}^{d} \{\pm\sqrt{d}\vec{u}_j, \pm(\sqrt{d}+2)\vec{u}_j\}$ to $\mathbb{A}$, where o is the origin and $\{\vec{u}_2, \ldots, \vec{u}_d\}$ is a set of orthogonal unit vectors which are all orthogonal to the point $f(i)$. If the output of $\mathbb{A}$ is a spherical shell of width less than $\frac{\sqrt{2}}{2}$, then Bob claims that $a_i = 0$, otherwise, he claims $a_i = 1$. We show that with probability at least 2/3 he is true in his claim.

Suppose $a_i = 0$. In this case, all of the points that Alice and Bob have given to $\mathbb{A}$ are at most at distance $\sqrt{2}/d^{1/3}$ from the hyperplane passing through the origin and normal to the point $f(i)$. Therefore, a spherical shell centered at a point along $f(i)$ lying at infinity encloses all of the points with a width of at most $2\sqrt{2}/d^{1/3}$. Hence, due to the approximation factor of $\mathbb{A}$, the output will be a spherical shell of width less than $\frac{\sqrt{2}}{2}$, with probability at least 2/3.

On the other hand, if $a_i = 1$, due to Lemma 2, there is no spherical shell of width less than $\frac{\sqrt{2}}{2}$ that encloses Bob's points plus the pair of points $\pm f(i)$. Thus, the output that Bob receives has width at least $\frac{\sqrt{2}}{2}$.

Therefore, the protocol presented for the index problem works with probability at least 2/3. Hence, the size of the working space that Alice sends to Bob is $\Omega(k) = \Omega(\exp(d^{1/3}))$. We can conclude the following theorem.

Theorem 3 *Given a set P of n points in $\mathbb{R}^d$, any streaming algorithm that approximates the minimum enclosing spherical shell of P to within a factor better than $d^{1/3}/4$ with probability at least 2/3 requires $\Omega(\min\{n, \exp(d^{1/3})\})$ space.*

4 Cylindrical Shell

In this section, we prove a lower bound on the approximation factor of any streaming algorithm for the minimum enclosing cylindrical shell problem whose memory size is sub-exponential in d. Again, we start with the following crucial lemma.

Lemma 4 *Let $\{\vec{u}_1, \ldots, \vec{u}_d\}$ be a set of orthogonal unit vectors in $\mathbb{R}^d$. Then any cylindrical shell that encloses the point set $P = \{o, \pm\vec{u}_1\} \cup \bigcup_{i=2}^{d} \{\pm\sqrt{d}\vec{u}_i\}$ has width at least $\frac{\sqrt{2}}{2}$.*

Proof. Let ℓ be a line in $\mathbb{R}^d$, α be a point on ℓ, and $\vec{n}$ be a unit vector in the direction of ℓ. Then, any point on ℓ can be represented by $\alpha + t\vec{n}$, where t is a real number. Note that such a presentation is not unique. In particular, we can choose the point α such that α is orthogonal to $\vec{n}$. This condition does not lose generality, as every point $\alpha + t\vec{n}$ can be rewritten as $\alpha' + t'\vec{n}$, where $\alpha' = \alpha - \frac{\alpha \cdot \vec{n}}{\vec{n} \cdot \vec{n}}\vec{n}$, $t' = t + \frac{\alpha \cdot \vec{n}}{\vec{n} \cdot \vec{n}}$, and α' is orthogonal to $\vec{n}$. Henceforth, we assume that the standard form of presenting a line has the above condition.

Now, let $\ell = \{\alpha + t\vec{n}\}$, where α is orthogonal to $\vec{n}$. We write α and $\vec{n}$ as $(a_1, \ldots, a_d)$ and $(n_1, \ldots, n_d)$, respectively, where $a_i = \alpha \cdot \vec{u}_i$ and $n_i = \vec{n} \cdot \vec{u}_i$. The perpendicularity condition ensures that $\sum_{i=1}^{d} a_i n_i = 0$. Let $\mathrm{dist}(p, \ell)$ denote the distance of a point p to the line ℓ.

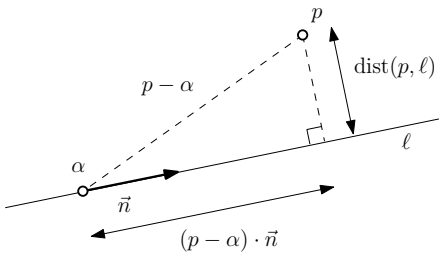


Figure 2: Computing $\mathrm{dist}(p, \ell)$.

Likewise, we write p as $(p_1, \ldots, p_d)$, where $p_i = p \cdot \vec{u}_i$. By the Pythagorean theorem,

$$\begin{aligned}
\mathrm{dist}^2(p,\ell) &= |p-\alpha|^2 - ((p-\alpha)\cdot\vec{n})^2 \quad \text{(see Fig. 2)} \\
&= \sum_{i=1}^{d}(p_i-a_i)^2 - \left(\sum_{i=1}^{d} p_i n_i - \sum_{i=1}^{d} a_i n_i\right)^2 \\
&= \sum_{i=1}^{d}(p_i-a_i)^2 - \left(\sum_{i=1}^{d} p_i n_i\right)^2
\end{aligned}$$

In particular, when $p = p_k\vec{u}_k$, we have:

$$\mathrm{dist}^2(p,\ell) = \sum_{i=1}^{d} a_i^2 + p_k^2 - 2a_k p_k - p_k^2 n_k^2 \qquad (1)$$

Note that the minimum width of a cylindrical shell with axis ℓ enclosing point set P is equal to $\max_{p\in P}\mathrm{dist}(p,\ell) - \min_{p\in P}\mathrm{dist}(p,\ell)$. We denote this value by $w_\ell(P)$. Suppose that ℓ is the axis of a minimum-width cylindrical shell enclosing P. We can assume without loss of generality that $a_i \geq 0$ for all $i = 1, \ldots, d$, and $a_2 \geq a_i$ for all $i = 3, \ldots, d$, due to symmetry of P. We consider two cases: (i) $a_1 \geq a_2\sqrt{d}$, and (ii) $a_1 \leq a_2\sqrt{d}$.

To prove the first case, we first show that $n_1^2 \leq \frac{1}{2}$. Since $\sum_{i=1}^{d} a_i n_i = 0$, we have

$$a_1|n_1| \;=\; \left|\sum_{i=2}^{d} a_i n_i\right| \;\leq\; \sum_{i=2}^{d} a_i|n_i|.$$

Therefore,

$$|n_1| \;\leq\; \sum_{i=2}^{d}\frac{a_i}{a_1}|n_i| \;\leq\; \frac{1}{\sqrt{d}}\sum_{i=2}^{d}|n_i|.$$

By the Cauchy-Schwarz inequality, for any two vectors u and v, $|u\cdot v|^2 \leq (u\cdot u)(v\cdot v)$. As such,

$$n_1^2 \;\leq\; \frac{1}{d}\left(\sum_{i=2}^{d}|n_i|\right)^2 \;\leq\; \frac{d-1}{d}\sum_{i=2}^{d} n_i^2.$$

Thus,

$$n_1^2 + \frac{d-1}{d}n_1^2 \;\leq\; \frac{d-1}{d}\sum_{i=1}^{d} n_i^2 \;=\; \frac{d-1}{d},$$

which implies $n_1^2 \leq \frac{d-1}{2d-1} \leq \frac{1}{2}$. Now,

$$\begin{aligned}
w_\ell(P) &\geq \mathrm{dist}(-\vec{u}_1,\ell) - \mathrm{dist}(o,\ell) \\
&= \sqrt{\Sigma_{i=1}^{d} a_i^2 + 2a_1 + (1-n_1^2)} \qquad \text{(by (1))} \\
&\quad -\sqrt{\Sigma_{i=1}^{d} a_i^2} \\
&\geq \sqrt{\Sigma_{i=1}^{d} a_i^2 + 2a_1 + \frac{1}{2}} - \sqrt{\Sigma_{i=1}^{d} a_i^2} \\
&\geq \sqrt{2a_1^2 + 2a_1 + \frac{1}{2}} - \sqrt{2a_1^2} \\
&\quad (\text{as } a_1 \geq a_i\sqrt{d}, \text{ for } i = 2, \ldots, d) \\
&= \sqrt{2}\left(\sqrt{a_1^2 + a_1 + \frac{1}{4}} - a_1\right) \\
&= \sqrt{2}\left(\sqrt{\left(a_1 + \frac{1}{2}\right)^2} - a_1\right) \\
&= \frac{\sqrt{2}}{2},
\end{aligned}$$

which completes the proof of Case (i). The second case is proved in the appendix. □

Now, suppose $\mathbb{A}$ is a streaming algorithm that approximates the minimum enclosing cylindrical shell to within a factor better than $d^{1/3}/4$ with probability at least $2/3$. Like the previous section, we use a reduction from the index problem. Alice gives the points $\{\pm f(j) \,|\, a_j = 1\}$ to $\mathbb{A}$, and sends the working space to Bob. Bob then gives the point set $\{o\} \cup \bigcup_{j=2}^{d}\{\pm\sqrt{d}\vec{u}_j\}$ to $\mathbb{A}$, where $\{\vec{u}_2, \ldots, \vec{u}_d\}$ is a set of orthogonal unit vectors which are all orthogonal to $f(i)$.

Let H be the hyperplane passing through the origin and normal to $f(i)$. If $a_i = 0$, all of the points given to $\mathbb{A}$ are at most at distance $\sqrt{2}/d^{1/3}$ from H. In this case, a cylindrical shell with an axis parallel to H infinitely far from H encloses all the input points with width $2\sqrt{2}/d^{1/3}$. Thus, according to the approximation factor of $\mathbb{A}$, the output of $\mathbb{A}$ is a cylindrical shell of width less than $\frac{\sqrt{2}}{2}$. On the other hand, if $a_i = 1$, then by Lemma 4, there is no cylindrical shell of width less than $\frac{\sqrt{2}}{2}$ that encloses Bob's points plus the pair of points $\pm f(i)$.

Therefore, Bob can check the output shell, and report $a_i = 0$, if the width is less than $\frac{\sqrt{2}}{2}$, and report $a_i = 1$, otherwise. This solves the index problem with probability at least $2/3$. Therefore, the working space of $\mathbb{A}$ must be $\Omega(k) = \Omega(\exp(d^{1/3}))$.

Theorem 5 *Given a set P of n points in $\mathbb{R}^d$, any streaming algorithm that approximates the minimum enclosing cylindrical shell of P to within a factor better than $d^{1/3}/4$ with probability at least $2/3$ requires $\Omega(\min\{n, \exp(d^{1/3})\})$ space.*

References

[1] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *Journal of the ACM*, 51(4):606–635, 2004.

[2] P. K. Agarwal and R. Sharathkumar. Streaming algorithms for extent problems in high dimensions. *Algorithmica*, 72(1):83–98, 2015.

[3] K. Ball. An elementary introduction to modern convex geometry. *Flavors of geometry*, 31:1–58, 1997.

[4] T. M. Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Computational Geometry*, 35(1-2):20–35, 2006.

[5] T. M. Chan and V. Pathak. Streaming and dynamic algorithms for minimum enclosing balls in high dimensions. *Computational Geometry*, 47(2):240–247, 2014.

[6] E. Kushilevitz. Communication complexity. *Advances in Computers*, 44:331–360, 1997.

[7] H. Zarrabi-Zadeh. An almost space-optimal streaming algorithm for coresets in fixed dimensions. *Algorithmica*, 60(1):46–59, 2011.

[8] H. Zarrabi-Zadeh and T. M. Chan. A simple streaming algorithm for minimum enclosing balls. In *Proceedings of the 18th Canadian Conference on Computational Geometry*, pages 139–142, 2006.

Appendix

Proof of Lemma 2

Case (ii).

$$\begin{aligned} w_a(P) &\geq w_a\Big(\{o, (\sqrt{d}+2)\vec{u}_2\}\Big) \\ &= \sqrt{a_1^2 + ((\sqrt{d}+2)-a_2)^2 + a_3^2 + \cdots + a_d^2} \\ &\quad -\sqrt{\Sigma_{i=1}^d a_i^2} \\ &\geq \sqrt{a_1^2 + (\sqrt{d}+a_2)^2 + a_3^2 + \cdots + a_d^2} \\ &\quad -\sqrt{\Sigma_{i=1}^d a_i^2} \\ &= \sqrt{\Sigma_{i=1}^d a_i^2 + 2\sqrt{d}a_2 + d} - \sqrt{\Sigma_{i=1}^d a_i^2} \\ &\geq \sqrt{2da_2^2 + 2\sqrt{d}a_2 + \frac{1}{2}} - \sqrt{2da_2^2} \\ &= \frac{\sqrt{2}}{2}. \end{aligned}$$

Case (iii).

$$\begin{aligned} w_a(P) &\geq w_a(\{\pm\vec{u}_1\}) \\ &= \sqrt{(a_1-(-1))^2 + a_2^2 + \cdots + a_d^2} \\ &\quad -\sqrt{(a_1-1)^2 + a_2^2 + \cdots + a_d^2} \\ &= \sqrt{\Sigma_{i=1}^d a_i^2 + 2a_1 + 1} - \sqrt{\Sigma_{i=1}^d a_i^2 - 2a_1 + 1} \\ &\geq \sqrt{2a_1^2 + 4a_1} - \sqrt{2a_1^2} \\ &\geq \sqrt{2}(\sqrt{a_1^2 + \sqrt{2}a_1 + \frac{1}{2}} - a_1) \\ &= 1 \\ &> \frac{\sqrt{2}}{2}. \end{aligned}$$

Case (iv).

$$\begin{aligned} w_a(P) &\geq w_a\Big(\{o, -\sqrt{d}\vec{u}_2\}\Big) \\ &= \sqrt{a_1^2 + (a_2-(-\sqrt{d}))^2 + a_3^2 + \cdots + a_d^2} \\ &\quad -\sqrt{\Sigma_{i=1}^d a_i^2} \\ &= \sqrt{\Sigma_{i=1}^d a_i^2 + 2\sqrt{d}a_2 + d} - \sqrt{\Sigma_{i=1}^d a_i^2} \\ &\geq \sqrt{2a_1^2 + d} - \sqrt{2a_1^2} \\ &\geq \sqrt{2+d} - \sqrt{2}, \end{aligned}$$

which is at least $\frac{\sqrt{2}}{2}$, for all $d > 2$. □

Proof of Lemma 4

Case (ii). We first prove that $d(1-n_2^2) \geq \frac{1}{2}$. Since $\sum_{i=1}^d a_i n_i = 0$, we have

$$a_2|n_2| \;=\; \left|a_1 n_1 + \sum_{i=3}^d a_i n_i\right| \;\leq\; a_1|n_1| + \sum_{i=3}^d a_i|n_i|.$$

Therefore,

$$|n_2| \;\leq\; \frac{a_1}{a_2}|n_1| + \sum_{i=3}^d \frac{a_i}{a_2}|n_i| \;\leq\; \sqrt{d}|n_1| + \sum_{i=3}^d |n_i|,$$

and hence,

$$\begin{aligned} n_2^2 &\leq dn_1^2 + \sum_{i=3}^d 2\sqrt{d}|n_1||n_i| + \left(\sum_{i=3}^d |n_i|\right)^2 \\ &\leq dn_1^2 + \sum_{i=3}^d \left(n_1^2 + dn_i^2\right) + (d-2)\sum_{i=3}^d n_i^2 \\ &= (2d-2)\left(n_1^2 + \sum_{i=3}^d n_i^2\right). \end{aligned}$$

Thus,

$$n_2^2 + (2d-2)n_2^2 \;\leq\; (2d-2)\sum_{i=1}^d n_i^2 \;=\; 2d-2.$$

Therefore, $n_2^2 \leq \frac{2d-2}{2d-1}$, and hence, $d\left(1-n_2^2\right) \geq \frac{d}{2d-1} \geq \frac{1}{2}$. We can thus conclude that

$$\begin{aligned} w_\ell(P) &\geq \operatorname{dist}(-\sqrt{d}\vec{u}_2, \ell) - \operatorname{dist}(o, \ell) \\ &= \sqrt{\Sigma_{i=1}^d a_i^2 + 2a_2\sqrt{d} + d(1-n_1^2)} - \sqrt{\Sigma_{i=1}^d a_i^2} \\ &\geq \sqrt{2da_2^2 + 2a_2\sqrt{d} + \frac{1}{2}} - \sqrt{2da_2^2} \\ &= \frac{\sqrt{2}}{2}. \end{aligned}$$

□

On the Geometry of Stable Steiner Tree Instances

James Freitag* Neshat Mohammadi† Aditya Potukuchi‡ Lev Reyzin§

Abstract

In this work we consider the Steiner tree problem under Bilu-Linial stability. We give strong geometric structural properties that need to be satisfied by stable instances. We then make use of, and strengthen, these geometric properties to show that 1.59-stable instances of Euclidean Steiner trees are polynomial-time solvable by showing it reduces to the minimum spanning tree problem. We also provide a connection between certain approximation algorithms and Bilu-Linial stability for Steiner trees.

1 Introduction and previous work

In this work, we initiate the study of Steiner tree instances that are stable to multiplicative perturbations to the distances in the underlying metric. Our analysis lies in the Bilu-Linial stability [9] setting, which provides a way to study tractable instances of NP-hard problems.

Instances that are γ-stable in the Bilu-Linial model have the property that the structure of the optimal solution is not only unique, but also does not change even when the underlying distances among the input points are perturbed by a multiplicative factor $\gamma > 1$. In their original paper, Bilu and Linial analyzed MAX-CUT clustering, and since their seminal work, other problems have been analyzed including center-based clustering [4, 6, 7], multi-way cut problems [15], and metric TSP [17].[1]

Here, we look at the metric Steiner tree problemand also the more restricted Euclidean version. For general metrics, the Steiner tree problem is known to be APX-hard in the worst case [10]. For the Euclidean metric, a PTAS is known [3].

In this paper we begin by providing strong geometric structural properties that need to be satisfied by stable instances. These point to the existence of algorithms for non-trivial families. We then make use of, and strengthen, these geometric properties to show that 1.59-stable instances of Euclidean Steiner trees are polynomial-time solvable. Finally, we discuss the connections between certain approximation algorithms and Bilu-Linial stability for Steiner trees.

2 Model and definitions

In this section, we recall the relevant definitions. Fist we define the Steiner tree problem, which is among Karp's 21 original NP-hard problems [13]. It has various applications including in network design, circuit layouts, and phylogenetic tree reconstruction.

Definition 1 (the Steiner tree problem). *Consider an undirected graph $G = (V, E)$ with edge weights $w_e \in \mathbb{R}_0^+$ for every edge $e \in E$, and a set $T \subseteq V$ of terminals. A Steiner tree S is a tree in the graph G that spans all terminal vertices T and may contain some of the non-terminals (also called Steiner points). The goal is to find such a tree of lowest weight, which we call* OPT,

$$\text{OPT} = \arg\min_S \sum_{e \in S} w_e.$$

We can assume without loss of generality[2] that the vertices are points in a metric space and the weights of the edges are given by the distance function – when the input is in the form of a metric, we call this the **metric Steiner tree problem**. Our results use properties of metric spaces, but move freely between the metric space and graph representations of the problem. When the metric is Euclidean, this is called the **Euclidean Steiner tree problem**.

Now we move on to defining Bilu-Linial stability for the Steiner tree problem on metrics.

*Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, jfreitag@uic.edu. Supported in part by NSF CAREER award 1945251.

†Department of Computer Science, University of Illinois at Chicago, nmoham24@uic.edu. Supported in part by NSF grant CCF-1934915.

‡Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, adityap@uic.edu. Supported in part by NSF grant CCF-1934915.

§Department of Mathematics, Statistics, and Computer Science, University of Illinois at Chicago, lreyzin@uic.edu. Supported in part by NSF grant CCF-1934915.

[1] Bilu-Linial stability is one among other notions of data stability studied in the literature [1, 5]. This is in contrast to notions of algorithmic stability, which focus on properties algorithms as opposed to data, see e.g. [2, 8, 11, 14, 16].

[2] For any graph with distances specified on edges, a metric can be formed by taking the vertices to be points and considering the shortest path distances in the graph between pairs of vertices. Solving (or approximating) the Steiner tree problem on a metric formed in this matter solves (or approximates) the problem on the original graph. See Vazirani [18] for further discussion of this issue.

Definition 2 (Bilu-Linial γ-stabile instances). *Let $I = (G, w)$ be an instance of a metric Steiner tree problem and $\gamma > 1$. I is γ-stable if for any function $w' : V \times V \to \mathbb{R}_0^+$ such that $\forall u, v \in V$,*

$$w_{uv} \leq w'_{uv} \leq \gamma w_{uv},$$

the optimal Steiner tree OPT′ *under w' is equal to the optimal Steiner tree* OPT *under w.*

We note that the perturbations can be such that instances originally satisfying the metric or Euclidean properties no longer have to satisfy these properties after perturbation. We also note that due to the triangle inequality, no instances have stability 2 or greater in the metric setting.

Notation: For a graph G, w_{ab}^G is the weight of edge ab in G. We abbreviate $w_{ab} = w_{ab}^G$ and $w'_{ab} = w_{ab}^{G'}$. Let $\mathrm{OPT} \subseteq E(G)$ denote the minimum weight Steiner tree of G, let $w(\mathrm{OPT}) = w^G(\mathrm{OPT})$ denote the weight of the Steiner tree.

3 Structural properties in general metrics

In this section, we work in the context of a general metric space, and we develop interesting restrictions on the types of problems with γ-stable solutions, for various values of γ.

The techniques of this section *do not* give, in complete generality, an efficient algorithm for finding the optimal Steiner tree for any value of γ less than 2, a problem we leave open. However, when more information about the metric space is available, one can use the structural results here to give restrictions on the arrangements of Steiner points which does yield a definitive solution. In particular,

1. In Section 4, we use Lemma 3 to give an algorithm for the Euclidean metric when $\gamma > 2^{2/3}$.

2. More generally, in the case that no two Steiner points are adjacent in the optimal solution, Lemma 10 together with the other results of the section can be used to give an efficient and very simple algorithm to find the minimal weight Steiner tree. Other more general situations can be efficiently handled via only slightly more elaborate arguments - e.g. if one has a bound on the length of the longest path of Steiner points in the optimal solution.

Lemma 3. *The degree of any Steiner point in the optimal solution is greater than $\frac{2}{2-\gamma}$.*

Proof. Consider a Steiner node s in the optimal solution, that is connected to ($m \neq n$) other points, $a_1, ..., a_m$. Let $\overline{w} = \sum_{i=1}^m \frac{w_{sa_i}}{m}$, and let w_{sa_1} and w_{sa_m} be such that $w_{sa_1} + w_{sa_m} \geq 2\overline{w}$. Let G' be obtained by perturbing each edge sa_i by a factor of γ. Let

$$\mathrm{T}' := (\mathrm{OPT} \setminus \{sa_1, \ldots, sa_m\}) \cup \{a_1a_2, \ldots, a_{m-1}a_m\}.$$

Clearly, T′ is also a Steiner tree. Triangle inequality gives us $w_{a_ia_{i+1}} \leq w_{sa_i} + w_{s_{a_{i+1}}}$. So, we have

$$\begin{aligned} w'(\mathrm{T}') &\leq w'(\mathrm{OPT}) - \sum_{i=1}^{m} \gamma w_{sa_i} \\ &\quad + \sum_{i=1}^{m-1} \left(w_{sa_i} + w_{sa_{i+1}}\right) \\ &= w'(\mathrm{OPT}) - \sum_{i=1}^{m} \gamma w_{sa_i} + \sum_{i=2}^{m-1} 2w_{sa_i} \\ &\quad + w_{sa_1} + w_{sa_m}. \end{aligned}$$

Using the fact that $w'(\mathrm{T}') > w'(\mathrm{OPT})$, we have

$$\sum_{i=1}^{m} \gamma w_{sa_i} < \sum_{i=2}^{m-1} 2w_{sa_i} + w_{sa_1} + w_{sa_m}$$

or

$$\gamma \cdot \overline{w}m < (2m-2)\overline{w}.$$

Rearranging, we have

$$\frac{2}{2-\gamma} < m.$$

□

Now we state some additional structural properties of optimal Steiner trees in γ-stable instances. These are not used in Section 4. Nevertheless, we hope that they are of independent interest.

Proposition 4. *If $a, b \in V(\mathrm{OPT})$ are nearest neighbors in the graph, then the edge ab is in the optimal solution.*

Lemma 5. *Suppose $ab, bc \in$ OPT, then*

1. $w_{ac} > \gamma \cdot \max\{w_{ab}, w_{bc}\}$.
2. $\frac{2}{\gamma} \cdot w_{ac} > w_{ab} + w_{bc}$.
3. $(\gamma - 1) \cdot w_{ab} < w_{bc}$, $(\gamma - 1) \cdot w_{bc} < w_{ab}$.

Proof. We handle the three parts in turn:

1. Assume w.l.o.g. $w_{ab} \geq w_{bc}$. Suppose that $w_{ac} \leq \gamma \cdot \max\{w_{ab}, w_{bc}\}$, let G' be obtained by perturbing ab by a factor of γ. Then $(\mathrm{OPT} \setminus \{ab\}) \cup \{ac\}$ is also a Steiner tree in G' of weight $w'(\mathrm{OPT})$ contradicting stability. This completes the proof of 1.

2. The proof of 2. follows from 1. and the fact that $\max\{w_{ab}, w_{bc}\} \geq \frac{w_{ab}+w_{bc}}{2}$.

3. Let G' be obtained by perturbing bc by a factor of γ. Then $\mathrm{T}' := \mathrm{OPT} \setminus \{bc\} \cup \{ac\}$ is also a Steiner tree of weight

$$w'(\mathrm{T}') = w(\mathrm{OPT}) - w_{bc} + w_{ac} \leq w(\mathrm{OPT}) + w_{ab}. \quad (1)$$

On the other hand, stability gives us that

$$w'(\mathrm{T}') > w'(\mathrm{OPT}) = w(\mathrm{OPT}) + (\gamma - 1)w_{bc}. \quad (2)$$

Putting (1) and (2) together gives us that $(\gamma - 1) \cdot w_{bc} \leq w_{ab}$.

Repeating the same argument but swapping bc for ab gives us $(\gamma - 1) \cdot w_{ab} \leq w_{bc}$.

□

Lemma 6. *Let H be a subgraph of* OPT *with at least one edge. Let $ab \in H$. Fix any vertex $c \in V(\mathrm{OPT}) \setminus V(H)$ satisfying $w_{ca} \leq \gamma(\gamma - 1) \cdot w_{ab}$; then we have $ca \in$* OPT*.*

Proof. If $ca \notin$ OPT, then adding the edge ac to OPT produces a cycle which includes edge ac. Suppose that the cycle also includes ab. Let G' be obtained by perturbing ab by a factor of γ. Then $(\mathrm{OPT} \setminus \{ab\}) \cup \{ac\}$ is a Steiner tree of weight at most $w'(\mathrm{OPT})$, contradicting stability.

If the cycle does not include ab, it includes some edge other than ac which has an endpoint at a. This edge, call it ad, is in OPT. By Lemma 5, $w_{ad} > (\gamma - 1)w_{ba}$. Let G' be obtained by perturbing ad by a factor of γ. We have $w'_{ad} > \gamma(\gamma-1)w_{ba} \geq w_{ac}$. Then $(\mathrm{OPT} \setminus \{ad\}) \cup \{ac\}$ is a Steiner tree of weight less than $w'(\mathrm{OPT})$, again contradicting stability. □

Lemma 7. *Let $\gamma > \frac{1+\sqrt{5}}{2}$. Let $ab \in H$, a subgraph of* OPT*. Suppose that c is a vertex with $w_{ca} \geq \gamma \cdot w_{ab}$, then $ca \notin$* OPT*.*

Proof. Let $\gamma' = \frac{w_{ca}}{w_{ab}}$. Note that $\gamma' \geq \gamma$ is some real number larger than $\frac{1+\sqrt{5}}{2}$. If $ac \in$ OPT, then by part 1. of Lemma 5, we must have

$$\frac{w_{bc}}{w_{ac}} > \gamma.$$

On the other hand,

$$\begin{aligned} \frac{w_{bc}}{w_{ac}} &\leq \frac{w_{ab} + w_{ac}}{w_{ac}} \\ &\leq \frac{w_{ab} + \gamma' w_{ab}}{\gamma' w_{ab}} \\ &\leq \frac{1 + \gamma'}{\gamma'}. \end{aligned}$$

We now have a contradiction as long as $\frac{1+\gamma'}{\gamma'} < \gamma$. The function $f(x) = \frac{1+x}{x}$ is decreasing for $x > 0$ and $f(x) < x$ for any $x \geq \frac{1+\sqrt{5}}{2}$. So, we have that

$$\frac{1 + \gamma'}{\gamma'} < \frac{1 + \gamma}{\gamma} < \gamma$$

as desired. □

Proposition 8. *Let H be a subgraph of* OPT *with at least one edge. Suppose that $ab \in H$ and suppose that $c \in V(\mathrm{OPT}) \setminus V(H)$ with $w_{bc} < \gamma(\gamma - 1)w_{ab}$. Then we must have $w_{bc} < \frac{w_{ab}}{\gamma - 1}$ and $w_{ab} < \frac{w_{bc}}{\gamma - 1}$.*

Proof. By Lemma 6, we must have that $bc \in$ OPT. Therefore, property 3. of Lemma 5 gives us the desired inequalities. □

When $\gamma(\gamma - 1)^2 > 1$ Proposition 8 strengthens the bounds of Lemma 6. This holds, for instance, when $\gamma > 1.755$. In this case, we obtain:

Proposition 9. *Assume that $\gamma(\gamma - 1)^2 > 1$. Assume that H is a subgraph of* OPT *with at least one edge. Let $ab \in H$. Fix any vertex $c \in V(\mathrm{OPT}) \setminus V(H)$. Then we have $w_{ca} < \frac{1}{\gamma - 1} \cdot w_{ab}$ if and only if $ca \in$* OPT*.*

Proof. By Lemma 6 and the assumption that $\gamma(\gamma - 1) > \frac{1}{\gamma - 1}$, we must have that $ac \in$ OPT. If $w_{ca} \geq \frac{1}{\gamma - 1} \cdot w_{ab}$, we can not have edge ac in OPT by Lemma 5 part 3.

□

Let $B = \{b_1, \ldots, b_m\}$ be vertices (either terminal or Steiner points). For a vertex a, we denote by $T(a, B)$ the tree on vertex set a, B in which a is connected to each element of B. Let the *average weight of* $T(a, B)$ be

$$\frac{\sum_{i=1}^{m} w_{ab_i}}{m}.$$

Suppose that H is a subgraph of OPT. We call $T(a, B)$ a *terminal component fan* relative to H if a is a Steiner point and B are all terminals or vertices in distinct connected components of H each with at least two vertices. We call the collection of components of H together with the terminals not in H the *terminal components of H*.

Lemma 10. *Let $\gamma > 1.755$ and suppose that H is a subgraph of* OPT *and in the optimal solution, no two Steiner points are adjacent. Suppose that $T(a, B)$ with $B = \{b_1, \ldots, b_m\}$ is a terminal component fan such that:*

- *The average weight of $T(a, B)$ is less than all edges not in H which connect two terminal components of H,*
- *the average weight of $T(a, B)$ is minimal among all terminal component fans, and*
- *the weights of the edges of $T(a, B)$ are all within a factor of $\frac{1}{\gamma - 1}$ of each other.*

Then $T(a, B)$ is a subgraph of OPT.

Proof. Suppose that the fan $T(a, B)$ is not in OPT. Specifically, if there are $k < m$ edges of $T(a, B)$ which are not in OPT, then there are at least k edges of OPT$\setminus H$ such that in OPT$\cup T(a, B)$ we may remove these k edges and still have a Steiner tree.[3] Moreover, since no two Steiner points are adjacent, these edges are either

- terminal to terminal edges, or
- part of a terminal component fan.

In the first case, the terminal to terminal edges have weight at least $\frac{\sum_{i=1}^{m} w_{ab_i}}{m}$. In this case perturb this edge by a factor of γ, and swap it with one edge of the terminal component fan $T(a, B)$. Since the weights of edges of T are within a factor of $\frac{1}{\gamma-1}$ of each other and their average weight is $\frac{\sum_{i=1}^{m} w_{ab_i}}{m}$, this swap decreases the weight of the resulting Steiner tree after the perturbation.

Similarly in the case that one of the k edges is in another terminal component fan, T_1, the average weight of edges in that fan is at least $\frac{\sum_{i=1}^{m} w_{ab_i}}{m}$, and applying part 3. of Lemma 5, the minimal weight edge in T_1 is at least $(\gamma - 1) \cdot \frac{\sum_{i=1}^{m} w_{ab_i}}{m}$. Now, perturb such an edge by a factor of γ to make the weight at least $\gamma \cdot (\gamma - 1) \cdot \frac{\sum_{i=1}^{m} w_{ab_i}}{m}$, which is larger than the weight of the largest weight edge of $T(a, B)$, which is a most $\frac{1}{\gamma-1} \cdot \frac{\sum_{i=1}^{m} w_{ab_i}}{m}$ because $\gamma > 1.755$.

Performing any of these k swaps yields a lower weight Steiner tree than OPT under the above perturbations, contradicting γ-stability. □

4 Euclidean Steiner trees

In this section, we consider the restriction of the Steiner tree problem to the Euclidean metric.

Definition 11 (angle)**.** *Let a_1, a_2, b be points on a Euclidean metric. Then we call $\angle a_1 b a_2$ the* angle *between a_1, a_2 at b.*

Under the assumption of γ-stability the minimum angle between two terminal points at their common Steiner neighbor can be bounded from below as a function of γ.

Lemma 12. *For a γ-stable instance of a Euclidean Steiner tree, the angle between two terminal points at their common Steiner neighbor in the tree should be greater than $2\sin^{-1}(\gamma/2)$.*

[3] In the case that $k = m$, there may be only $m - 1$ such edges, as a may not be in OPT, but the argument works identically in that case.

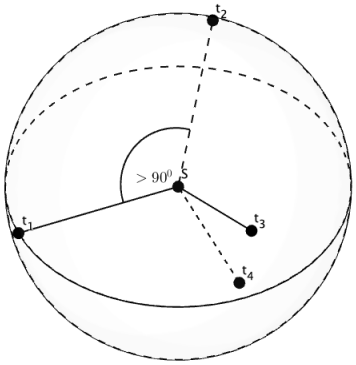


Figure 1: An example of points t_1, t_2, t_3, and t_4 surrounding Steiner point s at angles over $\theta > 90$ degrees. No more than $\frac{-1}{\cos\theta}$ can fit, independent of the dimension.

Proof. Let's assume, for a γ-stable instance of Steiner tree, the angle between two terminal points a_1, and a_2 at a Steiner point b is θ. Without loss of generality, let $w_{a_1 b} =: w \geq w_{a_2 b}$. Clearly $w_{a_1 a_2} > \gamma w$, since otherwise, perturbing edge $a_1 b$ by a factor of γ allows one to replace $a_1 b$ by $a_1 a_2$ in a minimal Steiner tree, contradicting stability. Let us use α to denote the angle $\angle a_1 a_2 b$. Clearly, $\alpha \geq \pi/2 - \theta/2$. Thus by the sine rule, we have

$$\frac{\gamma w}{\sin\theta} < \frac{w_{a_1 a_2}}{\sin\theta} = \frac{w}{\sin\alpha} \leq \frac{w}{\sin(\pi/2 - \theta/2)}.$$

Rearranging, we have

$$\begin{aligned} \gamma &< \frac{\sin\theta}{\sin(\pi/2 - \theta/2)} \\ &= \frac{2\sin(\theta/2)\cos(\theta/2)}{\cos(\theta/2)} \\ &= 2\sin(\theta/2) \end{aligned}$$

as desired. □

Thus we immediately get the following Corollary.

Corollary 13. *For a γ-stable instances of Steiner tree where $\gamma > \sqrt{2}$, the angle between two terminal points at their common neighbor in the optimal Steiner tree is greater than $\pi/2$.*

We say that a matrix $M \in \mathbb{R}^{d\times d}$ is *positive semidefinite* if for every $v \in \mathbb{R}^d$, it holds that $v^T R v \geq 0$.

Lemma 14. *If there are N points in $\mathbb{R}^d$ such that the angle between every pair with respect to a point u is at least $\theta > (\pi/2)$, then $N \leq 1 - \frac{1}{\cos\theta}$.*

Proof. Let $\theta > \pi/2$ and let $v_1, \ldots, v_N \in \mathbb{R}^d$ be unit vectors in $\mathbb{R}^d$ such that $\langle v_i, v_j\rangle \leq \cos\theta$. Consider the matrix V whose columns are the v_i's. By construction

V^TV is positive semi-definite. Indeed, for any $u \in \mathbb{R}^d$, we have $u^T(V^TV)u = \langle Vu, Vu\rangle \geq 0$.

If $N - 1 > \frac{-1}{\cos\theta}$, then the sum of every row is negative. This is because each diagonal entry of V^TV is 1, and every non-diagonal entry is at most $\cos\theta$. So we have that $\mathbf{1}^T(V^TV)\mathbf{1} < 0$ where $\mathbf{1} = (1, 1, \ldots, 1)$. This contradicts the positive semidefiniteness of V^TV. So it must be the case that $N \leq 1 - \frac{1}{\cos\theta}$. □

Corollary 15. *For $\gamma > \sqrt{2}$ the degree of a Steiner node in the optimal solution is at most $\frac{\gamma^2}{\gamma^2-2}$.*

Proof. Consider any two neighbors u, w of a given vertex v, and assume that $\angle uvw = \theta$. From Lemma 12 we have

$$\gamma < 2\sin(\theta/2).$$

So

$$\gamma^2 < 4\sin^2(\theta/2)$$

and so $\gamma^2/2 < 2\sin^2(\theta/2)$ or $1 - \gamma^2/2 > 1 - 2\sin^2(\theta/2)$. Since $\cos(\theta) = 1 - 2\sin^2(\theta/2)$, we have

$$\cos(\theta) < 1 - \gamma^2/2$$

or

$$\theta > \cos^{-1}(1 - \gamma^2/2).$$

Since the angle between any two neighbors of v is at least $\cos^{-1}(1-\gamma^2/2)$, Lemma 14 gives us that there are at most $1 - \frac{2}{2-\gamma^2} = \frac{\gamma^2}{\gamma^2-2}$ of them. □

Corollary 16. *When $\gamma > 1.59$, the optimal Steiner tree for a γ-stable instance does not have Steiner nodes.*

Proof. This happens when the min degree imposed by stability is larger than the max degree imposed by the packing bound. By Lemma 3 and Corollary 15, this happens when we have the following:

$$\frac{\gamma^2}{\gamma^2 - 2} \leq \frac{2}{2 - \gamma}$$

By solving the above equation for γ we get $\gamma \geq 2^{2/3}$, which is bounded from above by 1.59. □

This geometric property implies that for 1.59-stable instances, Steiner points will not be used in the optimal solution. Hence, an MST algorithm on just the terminal points will give the answer in polynomial time.

Finally, we point to the existence of Gilbert and Pollak's the Steiner ratio conjecture [12], which states that in the Euclidean plane, there always exists an MST within a cost of $2/\sqrt{3}$ of the minimum Steiner tree,and the behavior of this ratio for higher dimensions is yet unknown. Assuming this conjecture, in certain cases it may imply some limitations on the stability of Euclidean instances, especially in low dimensions, using the idea that even if the Steiner tree distances are "blown up" by more than the Steiner ratio, one could instead use the MST instead and get a cheaper solution. Unfortunately, because the MST may overlap with the Steiner tree, we cannot give a concrete statement.

5 Using approximation algorithms to solve stable instances

In this section we give a general argument about how strong approximation algorithms for Steiner tree problems give stability guarantees. We note that it is known that an FPTAS for the Steiner tree would imply P=NP [10], so there is no hope to use the result below in the general metric case. But if at some future point an FPTAS for the Euclidean variant of the Steiner tree problem is developed (currently, only a PTAS is known to exist [3]), then this would immediately imply the existence of polynomial-time algorithms for stable instances for any constant $\gamma > 1$.

Theorem 17. *An FPTAS for the Steiner tree problem gives a polynomial time algorithm for optimally solving any γ-stable Steiner tree problem in time poly$(n, (\gamma - 1)^{-1})$. In particular, this gives a polynomial-time algorithm for any constant $\gamma > 1$.*

Proof. Assume we are given an FPTAS for the Steiner tree problem. This means that we have an algorithm that runs in time poly$(n, 1/\epsilon)$ on instances of size n to give $(1+\epsilon)$-approximations to the optimum Steiner tree. Now consider a γ-stable instance for constant $\gamma > 1$. We run our FPTAS on that instance with $\epsilon = \frac{\gamma-1}{2n}$ to get a Steiner tree S' with weight within $\mathrm{OPT}(1+(\gamma-1)/2n)$. We now claim that every edge in the optimal solution whose weight is at least $\frac{\mathrm{OPT}}{n}$ must be in S'. Suppose it isn't – then we could perturb such an edge by γ and increase the weight of the optimal solution to $\mathrm{OPT}(1 + (\gamma - 1)/n)$ without increasing the weight of S', and S' would become cheaper than OPT, thereby violating γ-stability.

By the fractional pigeonhole principle, the most expensive edge of the FPTAS satisfies the desired property above and is therefore in OPT. Hence, we can contract this edge into a new vertex and get a new instance with $n - 1$ vertices at γ-stability. We can continue this process, getting one new edge of the optimal in each iteration, until we have a constant-size problem that we can brute-force. □

We note that the above technique could be used to convert even slightly weaker (than FPTAS) approximation algorithms to nontrivial stability guarantees.

Acknowledgments

We thank Xing Gao for useful discussions, and in particular, for pointing out an error in a previous version of Lemma 14.

References

[1] M. Ackerman and S. Ben-David. Clusterability: A theoretical study. In D. A. V. Dyk and M. Welling,

editors, *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, AISTATS 2009, Clearwater Beach, Florida, USA, April 16-18, 2009*, volume 5 of *JMLR Proceedings*, pages 1–8. JMLR.org, 2009.

[2] I. M. Alabdulmohsin. Algorithmic stability and uniform generalization. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 19–27, 2015.

[3] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998.

[4] P. Awasthi, A. Blum, and O. Sheffet. Center-based clustering under perturbation stability. *Inf. Process. Lett.*, 112(1-2):49–54, 2012.

[5] M. Balcan, A. Blum, and A. Gupta. Approximate clustering without the approximation. In C. Mathieu, editor, *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 1068–1077. SIAM, 2009.

[6] M. Balcan and Y. Liang. Clustering under perturbation resilience. *SIAM J. Comput.*, 45(1):102–155, 2016.

[7] S. Ben-David and L. Reyzin. Data stability in clustering: A closer look. *Theor. Comput. Sci.*, 558:51–61, 2014.

[8] S. Ben-David, U. von Luxburg, and D. Pál. A sober look at clustering stability. In G. Lugosi and H. U. Simon, editors, *Learning Theory, 19th Annual Conference on Learning Theory, COLT 2006, Pittsburgh, PA, USA, June 22-25, 2006, Proceedings*, volume 4005 of *Lecture Notes in Computer Science*, pages 5–19. Springer, 2006.

[9] Y. Bilu and N. Linial. Are stable instances easy? *Comb. Probab. Comput.*, 21(5):643–660, 2012.

[10] M. Chlebík and J. Chlebíková. The steiner tree problem on graphs: Inapproximability results. *Theor. Comput. Sci.*, 406(3):207–214, 2008.

[11] B. Fan, D. Ihara, N. Mohammadi, F. Sgherzi, A. Sidiropoulos, and M. Valizadeh. Learning lines with ordinal constraints. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[12] E. N. Gilbert and H. O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.

[13] R. M. Karp. *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA, 1972.

[14] T. Liu, G. Lugosi, G. Neu, and D. Tao. Algorithmic stability and hypothesis complexity. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2159–2167. PMLR, 2017.

[15] K. Makarychev, Y. Makarychev, and A. Vijayaraghavan. Bilu-linial stable instances of max cut and minimum multiway cut. In C. Chekuri, editor, *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2014, Portland, Oregon, USA, January 5-7, 2014*, pages 890–906. SIAM, 2014.

[16] W. Meulemans, B. Speckmann, K. Verbeek, and J. Wulms. A framework for algorithm stability and its application to kinetic euclidean msts. In M. A. Bender, M. Farach-Colton, and M. A. Mosteiro, editors, *LATIN 2018: Theoretical Informatics - 13th Latin American Symposium, Buenos Aires, Argentina, April 16-19, 2018, Proceedings*, volume 10807 of *Lecture Notes in Computer Science*, pages 805–819. Springer, 2018.

[17] M. Mihalák, M. Schöngens, R. Srámek, and P. Widmayer. On the complexity of the metric TSP under stability considerations. In I. Cerná, T. Gyimóthy, J. Hromkovic, K. G. Jeffery, R. Královic, M. Vukolic, and S. Wolf, editors, *SOFSEM 2011: Theory and Practice of Computer Science - 37th Conference on Current Trends in Theory and Practice of Computer Science, Nový Smokovec, Slovakia, January 22-28, 2011. Proceedings*, volume 6543 of *Lecture Notes in Computer Science*, pages 382–393. Springer, 2011.

[18] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, Berlin, Heidelberg, 2001.

Globally linked pairs in braced maximal outerplanar graphs

Dániel Garamvölgyi* Tibor Jordán†

Abstract

We say that a graph G is a braced MOP graph if it contains a maximal outerplanar graph as a spanning subgraph. We show that a pair $\{u, v\}$ of vertices of a braced MOP graph is globally linked in $\mathbb{R}^2$ in every generic realization of G if and only if uv is an edge of G or G contains three pairwise openly disjoint u-v paths. It follows that a braced MOP graph is globally rigid if and only if it is 3-vertex connected or isomorphic to K_3.

The former result verifies the conjectured characterization of global linkedness in the plane in a special case. The latter result leads to a linear time algorithm for testing global rigidity of braced MOP graphs.

Our proof is based on new structural results about maximal outerplanar graphs.

1 Introduction

Given a set $V = \{v_1, v_2, \ldots, v_n\}$ of n labeled points, a map $p : V \to \mathbb{R}^d$ defines a point configuration $P = \{p_1, p_2, \ldots, p_n\}$ of V in $\mathbb{R}^d$, where $p_i = p(v_i)$, for $1 \leq i \leq n$. For a subset E of the pairs of points in V, a basic geometric problem is whether every d-dimensional point configuration Q of V for which the pairwise distances in P and in Q are the same for all pairs in E, is congruent with P. In the local version of this question we focus on a pair v_i, v_j and ask whether the distance between the points corresponding to this pair is the same as in P in every point configuration Q that agrees with P on E. It is known that these decision problems are NP-hard, even in $\mathbb{R}^1$ [17].

If we restrict ourselves to generic point configurations, then global uniqueness, up to congruence, depends only on E (for fixed d). A combinatorial characterization of those pairs (V, E) that give rise to globally unique generic point configurations is known for $d = 1, 2$. The higher dimensional cases are still open. We also have a similar combinatorial question concerning the local version, which is open for all $d \geq 2$.

In this paper we consider the latter problem in $\mathbb{R}^2$ and characterize local uniqueness for a new family of graphs. Our results verify a conjectured characterization (Conjecture 1 below) in a special case. These problems are best described and studied by using the notions and tools of rigidity theory. In the rest of this section we introduce these notions as well as previous results of this area.

1.1 Globally linked pairs in frameworks and graphs

A d-dimensional *framework* is a pair (G, p), where $G = (V, E)$ is a graph and p is a map from V to $\mathbb{R}^d$. We also say that (G, p) is a *realization* of G in $\mathbb{R}^d$. Intuitively, we can think of a framework (G, p) as a collection of bars and joints where each vertex v of G corresponds to a joint located at $p(v)$ and each edge to a rigid (that is, fixed length) bar joining its endpoints. Two frameworks (G, p) and (G, q) are *equivalent* if $||p(u) - p(v)|| = ||q(u) - q(v)||$ holds for all pairs u, v with $uv \in E$, where $||.||$ denotes the Euclidean norm in $\mathbb{R}^d$. A pair of frameworks (G, p), (G, q) are *congruent* if $||p(u) - p(v)|| = ||q(u) - q(v)||$ holds for all pairs u, v with $u, v \in V$. This is the same as saying that (G, q) can be obtained from (G, p) by an isometry of $\mathbb{R}^d$. A framework (G, p) is said to be *generic* if the set of its $d|V(G)|$ vertex coordinates is algebraically independent over $\mathbb{Q}$.

A d-dimensional framework (G, p) is called *globally rigid* if every equivalent d-dimensional framework (G, q) is congruent to (G, p). The framework (G, p) is *rigid* if there exists some $\varepsilon > 0$ such that, if (G, q) is equivalent to (G, p) and $||p(v) - q(v)|| < \varepsilon$ for all $v \in V$, then (G, q) is congruent to (G, p). This is equivalent to requiring that every continuous motion of the vertices of (G, p) in $\mathbb{R}^d$ that preserves the edge lengths takes the framework to a congruent realization of G. It is known that for generic frameworks the rigidity (resp. global rigidity) in a given dimension depends only on G: either every generic realization of G in $\mathbb{R}^d$ is (globally) rigid, or none of them are [1, 4]. Thus, we say that a graph G is *rigid* (resp. *globally rigid*) in $\mathbb{R}^d$ if every (or equivalently, if some) d-dimensional generic realization of G is globally rigid in $\mathbb{R}^d$.

For $d = 1, 2$, combinatorial characterizations and corresponding deterministic polynomial time algorithms

*Department of Operations Research, ELTE Eötvös Loránd University, and the MTA-ELTE Egerváry Research Group on Combinatorial Optimization, Eötvös Loránd Research Network (ELKH), Pázmány Péter sétány 1/C, 1117 Budapest, Hungary. e-mail: daniel.garamvolgyi@ttk.elte.hu

†Department of Operations Research, ELTE Eötvös Loránd University, and the MTA-ELTE Egerváry Research Group on Combinatorial Optimization, Eötvös Loránd Research Network (ELKH), Pázmány Péter sétány 1/C, 1117 Budapest, Hungary. e-mail: tibor.jordan@ttk.elte.hu

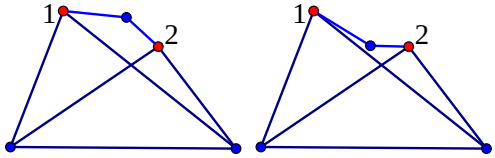


Figure 1: Two equivalent generic realizations of a graph G in $\mathbb{R}^2$. The graph is not globally rigid, but the pair $\{1, 2\}$ is globally linked in G.

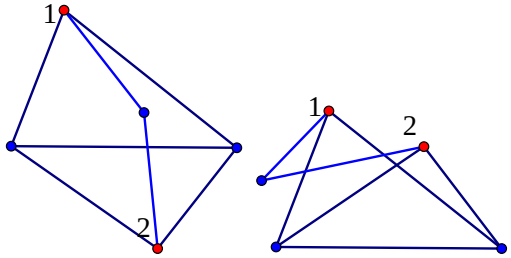


Figure 2: Two equivalent generic realizations of the same graph G in which $\{1, 2\}$ is not globally linked. The lengths of the edges incident with 1 and 2 permit a partial reflection in the rest of the graph.

are known for (testing) rigidity in $\mathbb{R}^d$ as well as global rigidity in $\mathbb{R}^d$ [7, 14, 16]. The existence of such a characterization (or algorithm) for $d \geq 3$ is a major open question in both cases. For more details on (globally) rigid graphs and frameworks see e.g. [12, 18].

A pair of vertices $\{u, v\}$ in a framework (G, p) is *globally linked in* (G, p) if for every equivalent framework (G, q) we have $||p(u) - p(v)|| = ||q(u) - q(v)||$. The pair $\{u, v\}$ is *globally linked in* G in $\mathbb{R}^d$ if it is globally linked in all generic d-dimensional frameworks (G, p). It is immediate from the definitions that G is globally rigid in $\mathbb{R}^d$ if and only if all pairs of vertices of G are globally linked in G in $\mathbb{R}^d$. Global linkedness in $\mathbb{R}^d$ is not a generic property (for $d \geq 2$): a vertex pair may be globally linked in some generic d-dimensional realization of G without being globally linked in all generic realizations. See Figures 1, 2.

The case $d = 1$ is exceptional and well-understood: a pair is globally linked in G in $\mathbb{R}^1$ if and only if there is a cycle in G that contains both vertices. In higher dimensions, no combinatorial (or efficiently testable) characterization is known for globally linked pairs in graphs.

1.2 The rigidity matroid

The rigidity matroid of a graph G is a matroid defined on the edge set of G which reflects the rigidity properties of all generic realizations of G. For a general introduction to matroid theory we refer the reader to [15]. For a detailed treatment of the 2-dimensional rigidity matroid, see [11].

Let (G, p) be a realization of a graph $G = (V, E)$ in $\mathbb{R}^d$. The *rigidity matrix* of the framework (G, p) is the matrix $R(G, p)$ of size $|E| \times d|V|$, where, for each edge $uv \in E$, in the row corresponding to uv, the entries in the d columns corresponding to vertices u and v contain the d coordinates of $(p(u) - p(v))$ and $(p(v) - p(u))$, respectively, and the remaining entries are zeros. The rigidity matrix of (G, p) defines the *rigidity matroid* of (G, p) on the ground set E by linear independence of the rows. It is known that any pair of generic frameworks (G, p) and (G, q) have the same rigidity matroid. We call this the d-dimensional *rigidity matroid* $\mathcal{R}_d(G) = (E, r_d)$ of the graph G.

We denote the rank of $\mathcal{R}_d(G)$ by $r_d(G)$. A graph $G = (V, E)$ is *$\mathcal{R}_d$-independent* if $r_d(G) = |E|$ and it is an *$\mathcal{R}_d$-circuit* if it is not $\mathcal{R}_d$-independent but every proper subgraph G' of G is $\mathcal{R}_d$-independent. We note that in the literature such graphs are sometimes called M-independent in $\mathbb{R}^d$ and M-circuits in $\mathbb{R}^d$, respectively. An edge e of G is an *$\mathcal{R}_d$-bridge in G* if $r_d(G - e) = r_d(G) - 1$ holds. Equivalently, e is an $\mathcal{R}_d$-bridge in G if it is not contained in any subgraph of G that is an $\mathcal{R}_d$-circuit.

The following characterization of rigid graphs is due to Gluck.

Theorem 1 *[3] Let $G = (V, E)$ be a graph with $|V| \geq d + 1$. Then G is rigid in $\mathbb{R}^d$ if and only if $r_d(G) = d|V| - \binom{d+1}{2}$.*

A graph is *minimally rigid* in $\mathbb{R}^d$ if it is rigid in $\mathbb{R}^d$ but deleting any edge results in a flexible graph. By Theorem 1, minimally rigid graphs in $\mathbb{R}^d$ on at least $d + 1$ vertices have exactly $d|V| - \binom{d+1}{2}$ edges.

Let $\mathcal{M}$ be a matroid on ground set E. We can define a relation on the pairs of elements of E by saying that $e, f \in E$ are equivalent if $e = f$ or there is a circuit C of $\mathcal{M}$ with $\{e, f\} \subseteq C$. This defines an equivalence relation. The equivalence classes are the *connected components* of $\mathcal{M}$. The matroid is *connected* if there is only one equivalence class, and *separable* otherwise. A graph $G = (V, E)$ is *$\mathcal{R}_d$-connected* if $\mathcal{R}_d(G)$ is connected. The subgraphs induced by the edges of the connected components of $\mathcal{R}_d(G)$ are the *$\mathcal{R}_d$-components* of G. Thus an edge $e \in E$ is an $\mathcal{R}_d$-bridge if $\{e\}$ is a trivial $\mathcal{R}_d$-component of G.

In the next section we shall see that $\mathcal{R}_2$-connected graphs play an important role in the characterization of globally rigid graphs in $\mathbb{R}^2$ (see Theorem 2 below) as well as in the conjectured characterization of globally linked pairs in $\mathbb{R}^2$.

2 Previous work

In the rest of the paper, we focus on the $d = 2$ case. Thus, we shall occasionally write that a graph is (globally) rigid to mean that it is (globally) rigid in $\mathbb{R}^2$, and

we may similarly omit the dimension when referring to global linkedness of vertex pairs in graphs. The following characterization of globally rigid graphs in $\mathbb{R}^2$ (the equivalence of (i) and (ii) below) is from [7]. As it was noted in [9], (ii) is in fact equivalent to (iii).

Theorem 2 *[7] Let G be a graph on at least four vertices. The following assertions are equivalent.*

(i) G is globally rigid in $\mathbb{R}^2$,
(ii) G is 3-connected and $\mathcal{R}_2$-connected,
(iii) G is 3-connected and contains no $\mathcal{R}_2$-bridges.

The complete characterization of globally linked pairs of vertices in a graph in $\mathbb{R}^2$ is not known. The truth of the following conjecture would imply a complete answer and an efficient algorithm for testing global linkedness.

Let $H = (V, E)$ be a graph and $x, y \in V$. We use $\kappa_H(x, y)$ to denote the maximum number of pairwise internally disjoint xy-paths in H. Note that if $xy \notin E$ then, by Menger's theorem, $\kappa_H(x, y)$ is equal to the size of a smallest set $S \subseteq V(H) - \{x, y\}$ for which there is no xy-path in $H - S$. It is easy to see that if $\kappa_G(x, y) \leq 2$ and $xy \notin E(G)$ then $\{x, y\}$ is not globally linked in G in $\mathbb{R}^2$ [9].

Conjecture 1 *[9, Conjecture 5.9] The pair $\{x, y\}$ is globally linked in a graph $G = (V, E)$ in $\mathbb{R}^2$ if and only if either $xy \in E$ or there is an $\mathcal{R}_2$-component H of G with $\{x, y\} \subseteq V(H)$ and $\kappa_H(x, y) \geq 3$.*

The following partial results are known. The first characterizes globally linked pairs in $\mathcal{R}_2$-connected graphs and implies the "if" direction of Conjecture 1.

Theorem 3 *[9, Theorem 5.7] Suppose that G is $\mathcal{R}_2$-connected and let u, v be a pair of vertices in G. Then $\{u, v\}$ is globally linked in G in $\mathbb{R}^2$ if and only if $\kappa_G(x, y) \geq 3$.*

In the case of minimally rigid graphs (in which every $\mathcal{R}_2$-component is trivial, i.e. isomorphic to K_2) it has been shown that there are no non-adjacent globally linked pairs.

Theorem 4 *[10] Let $G = (V, E)$ be a minimally rigid graph in $\mathbb{R}^2$ and $u, v \in V$. Then $\{u, v\}$ is globally linked in G in $\mathbb{R}^2$ if and only if $uv \in E$.*

3 Maximal outerplanar graphs

A (simple) graph $G = (V, E)$ on $n \geq 3$ vertices is said to be *outerplanar* if it has a planar embedding in which all vertices of G lie on the boundary of the outer face. It is well-known that if G is a 2-connected outerplanar graph, then G has a unique Hamiltonian cycle C, which bounds the outer face in any in such planar embedding of G. We shall consider 2-connected outerplanar graphs and call this cycle C the *boundary cycle* of G. The edges in $E - E(C)$ are called the *diagonals* of G.

A *maximal outerplanar graph*, or *MOP graph*, for short, is an outerplanar graph G for which $G + uv$ is not outerplanar for all nonadjacent vertex pairs $u, v \in V$. Equivalently, G is the graph of the triangulation of a polygon. Note that every MOP graph on at least three vertices is 2-vertex connected: indeed, if v is a cut-vertex in the outerplanar graph G, then G necessarily has some pair u_1, u_2 of non-adjacent neighbours for which $G + u_1u_2$ is outerplanar.

It is well-known that every 2-connected outerplanar graph has at least two vertices of degree two. For a degree two vertex v of a MOP graph on at least four vertices $G - v$ is also a MOP graph. Since K_3 is minimally rigid and adding a new vertex of degree two preserves minimal rigidity (see e.g. [11]), we have the following observation.

Proposition 5 *Every MOP graph is minimally rigid in $\mathbb{R}^2$.*

We shall mainly be concerned with the following larger family of graphs. A graph G' is a *braced MOP graph* if it contains a spanning MOP subgraph. If we fix such a subgraph $G = (V, E)$, so that $G' = (V, E \cup B)$, then we say that the edges in B are the *braces* of the graph. If $|B| = 1$ (i.e. if G' has $2|V| - 2$ edges), then we say that G' is a *uni-braced MOP graph*.

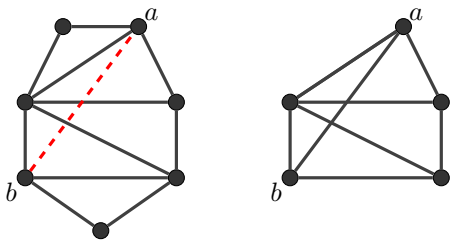
In order to identify the globally linked pairs in braced MOP graphs we first obtain new structural results concerning the connectivity properties of MOP graphs.

3.1 Separating pairs and 3-connected subgraphs in (braced) MOP graphs

A pair $\{u, v\}$ of vertices of G is a *separating pair* if $G - \{u, v\}$ is disconnected. The structure of the separating pairs in a MOP graph is rather special.

Lemma 6 *Let $G = (V, E)$ be a MOP graph and let $\{u, v\} \subset V$. Then $\{u, v\}$ is a separating pair if and only if uv is a diagonal of G.*

Proof. The "if" part follows from planarity (and holds for all 2-connected outerplanar graphs). To prove the "only if" direction suppose that $\{u, v\}$ is a separating pair in G. Let C denote the boundary cycle of G. Since C is a 2-connected spanning subgraph of G, the vertices u and v cannot be consecutive on C. Thus if $uv \in E$ then uv is a diagonal and we are done. Let us suppose, for a contradiction, that $uv \notin E$. It is clear that $G - \{u, v\}$ has exactly two connected components A_1, A_2 whose vertex sets coincide with the vertex sets of the two paths P_1, P_2 obtained from C by deleting u and v. The fact that G is outerplanar and there are no



(a) (b)

Figure 3: (a) A uni-braced MOP graph G' with a spanning MOP subgraph G (indicated by the solid edges) and a bracing edge ab. (b) the graph $H_G[a,b]+ab$.

edges in G between P_1 and P_2 imply that $G+uv$ is also outerplanar, contradicting the maximality of G. □

Consider a MOP graph $G=(V,E)$ and two non-adjacent vertices $a,b\in V$. We say that a diagonal uv is (a,b)-*separating* if a and b are in different connected components of $G-\{u,v\}$.

Proposition 7 *Let $G=(V,E)$ be a MOP graph and let $a,b\in V$ be two non-adjacent vertices. Then*
(i) for every (a,b)-separating diagonal v_iv_j and two internally disjoint a-b-paths P_1,P_2 we have (after relabeling the paths, if necessary) $v_i\in V(P_1)$, $v_j\in V(P_2)$,
(ii) for every diagonal v_kv_l which is not (a,b)-separating and for the component D of $G-\{v_k,v_l\}$ disjoint from $\{a,b\}$ we have that $G-V(D)$ is a MOP graph. In particular, $\kappa_{G-V(D)}(a,b)\geq 2$.

Proof. Lemma 6 implies (i). Part (ii) follows from the observation that for every diagonal v_iv_j and component K of $G-\{v_i,v_j\}$, $G-V(K)$ is a MOP graph. □

By using Proposition 7(ii) to "peel off" vertex sets from G, preserving 2-connectivity, we can deduce the following.

Lemma 8 *Let $G=(V,E)$ be a MOP graph and let $a,b\in V$ be two non-adjacent vertices. Then there is a unique smallest 2-connected subgraph H of G that contains a,b. This subgraph H is a MOP graph, induced by the vertex set consisting of the end-vertices of the (a,b)-separating diagonals, plus a,b.*

Proof. Let X be the set of the end-vertices of the (a,b)-separating diagonals in G. By Proposition 7(i) the vertex set of every 2-connected subgraph of G that contains a,b must also contain X. Let us fix an outerplanar embedding of G and consider a minimal (with respect to inclusion) subgraph of G which induces a MOP subgraph in this embedding and contains a,b. It follows from Proposition 7(ii) that every diagonal v_kv_l in H is (a,b)-separating, for otherwise we can delete the vertices of the component D of $H-\{v_k,v_l\}$ which is disjoint from $\{a,b\}$ to obtain a smaller MOP subgraph. The key observation is that such a deletion does not create new (a,b)-separating diagonals. Hence the vertex set of H must be equal to $X\cup\{a,b\}$. This completes the proof. □

The unique subgraph H in Lemma 8 belonging to a,b in G is denoted by $H_G[a,b]$. Note that $H_G[a,b]$ has exactly two vertices of degree two: a and b.

We close this subsection with a lemma which may be of independent interest. Since we shall not use it later, we omit the proof. (Necessity follows from Lemma 6.)

Lemma 9 *Let $G=(V,E)$ be a 2-connected graph. Then G is a MOP graph if and only if for all non-adjacent vertex pairs $\{a,b\}\subset V$ there is a separating pair $\{u,v\}$ with $uv\in E$ for which $G-\{u,v\}$ has exactly two connected components, each of which contains exactly one of a and b.*

3.2 Globally rigid braced MOP graphs

Lemma 10 *Let $G=(V,E)$ be a MOP graph and let $a,b\in V$ be two non-adjacent vertices. Then $H_G[a,b]+ab$ is a 3-connected $\mathcal{R}_2$-circuit.*

Proof. Since $H_G[a,b]$ is a MOP graph, every separating pair consists of the end-vertices of a diagonal by Lemma 6. Moreover, every diagonal in $H_G[a,b]$ is (a,b)-separating by Lemma 8. Hence $H_G[a,b]+ab$ is 3-connected.

By Proposition 5, $H_G[a,b]$ is minimally rigid. Thus, by basic matroid theory, $H_G[a,b]+ab$ contains a unique $\mathcal{R}_2$-circuit W with $ab\in E(W)$. It is well-known that deleting any edge of an $\mathcal{R}^2$-circuit results in a minimally rigid graph (see e.g. [7, Lemma 2.15]), so in particular $W-ab$ is minimally rigid. Since minimally rigid graphs on at least three vertices are 2-connected, the minimality of $H_G[a,b]$ implies that $W-ab=H_G[a,b]$ holds. Hence $H_G[a,b]+ab$ is an $\mathcal{R}_2$-circuit, as claimed. □

The above discussion leads to the following result.

Lemma 11 *Let G' be a uni-braced MOP graph obtained from the MOP graph G by adding the brace ab. Then $H_G[a,b]+ab$ is a maximal globally rigid subgraph of G'.*

Proof. The global rigidity of $H_G[a,b]+ab$ follows from Lemma 10 and Theorem 2. Maximality is due to the fact that every vertex v of G' which is disjoint from $H_G[a,b]+ab$ is separated from $H_G[a,b]+ab$ by a diagonal. □

Our first main result is as follows.

Theorem 12 *Let G' be a braced MOP graph. Then G' is globally rigid if and only if G' is 3-connected (or $G=K_3$).*

Proof. The only (braced) MOP graph and the only globally rigid graph on three vertices is K_3, so we may assume that $|V| \geq 4$. Necessity follows from Theorem 2. To prove sufficiency, we shall verify that every edge of G' belongs to an $\mathcal{R}_2$-circuit. Let us fix a spanning MOP subgraph $G = (V, E)$ and let B denote the set of braces of G'. For a brace $ab \in B$, Lemma 10 implies that $H_G[a, b] + ab$ is an $\mathcal{R}_2$-circuit that contains ab.

Next, consider a diagonal uv of G. Since G' is 3-connected, there is a brace $a'b' \in B$ that connects the two connected components of $G - \{u, v\}$. Then $\{u, v\}$ is (a', b')-separating, so $H[a', b'] + a'b'$ is an $\mathcal{R}_2$-circuit that contains uv by Lemmas 8 and 10.

Finally, consider an edge $e = v_i v_{i+1}$ of the boundary cycle C of G. The 3-connectivity of G' implies that the degree of v_i and v_{i+1} in G' is at least three. Hence there is a diagonal $v_i v_k$ incident with v_i. Let us suppose that v_k is as close to v_{i+1} as possible on the path $C - \{v_i\}$. By using 3-connectivity again, we can see that there is a brace cd that connects the two connected components of $G - \{v_i, v_k\}$. We may suppose that d and v_{i+1} are in the same components. Now the choice of v_k and the MOP structure implies that the $\mathcal{R}_2$-circuit $H_G[c, d] + cd$ contains the edge e.

Since G' is 3-connected and contains no $\mathcal{R}_2$-bridges, it is globally rigid by Theorem 2. □

3.3 Globally linked pairs in braced MOP graphs

We next consider the globally linked pairs. The next general lemma is easy to verify. Let H_1, H_2 be two disjoint graphs on at least three vertices with two designated edges $e_i = u_i v_i$, $e_i \in E(H_i)$, $i = 1, 2$. We say that the graph G obtained from H_1, H_2 by identifying u_1 with u_2 and v_1 with v_2 is the 2-*merge* of H_1 and H_2 along the edges e_i, $i = 1, 2$.

Lemma 13 *Let $G = (V, E)$ be the 2-merge of H_1 and H_2 and let $x, y \in V$. Then $\{x, y\}$ is globally linked in G if and only if $\{x, y\} \subseteq V(H_i)$ and $\{x, y\}$ is globally linked in H_i for some $i \in \{1, 2\}$.*

Our second main result is as follows.

Theorem 14 *Let G' be a braced MOP graph, obtained from the MOP graph $G = (V, E)$ by adding a set B of braces, and let $x, y \in V$. Then $\{x, y\}$ is globally linked in G' if and only if either $xy \in E(G')$ or $\kappa_{G'}(x, y) \geq 3$.*

Proof. We may assume that G' has at least four vertices. Suppose that $xy \notin E(G')$. Then $\kappa_{G'}(x, y) \geq 3$ is a necessary condition for the global linkedness of $\{x, y\}$ in G' (for otherwise applying a suitable partial reflection to a generic realization of G' shows that x and y are not globally linked). To prove sufficiency suppose that we have $\kappa_{G'}(x, y) \geq 3$. Every separating pair of G' is a separating pair of G. Thus every separating pair in G' consists of the end-vertices of a diagonal of G by Lemma 8. Hence we may assume, by Lemma 13, that G' is a 3-connected braced MOP graph. Then the theorem follows from Theorem 12. □

From the proof it also follows that if $\{x, y\}$ is globally linked in the braced MOP graph G', then there is an $\mathcal{R}_2$-component H of G with $x, y \in V(H)$. Indeed, the proof shows that there is even a globally rigid subgraph H' with $x, y \in V(H')$, and globally rigid graphs in $\mathbb{R}^2$ are $\mathcal{R}_2$-connected, c.f. Theorem 2. Thus, Theorem 14 also implies Conjecture 1 in the case of braced MOP graphs.

We note that there exist $\mathcal{R}_2$-connected graphs which are not braced MOP graphs (for example the complete bipartite graph $K_{3,4}$), as well as braced MOP graphs which are not $\mathcal{R}_2$-connected (for example, K_4 plus a degree-two vertex). Thus neither of Theorem 3 and Theorem 14 is implied by the other.

We can also deduce the following results on braced MOP graphs. For a 2-dimensional realization (G, p), let $r(G, p)$ denote the number of pairwise non-congruent realizations of G that are equivalent to (G, p). A *globally rigid cluster* of G is a maximal vertex set of G in which each vertex pair is globally linked in G.

Theorem 15 *Let G be a braced MOP graph. Then*

(i) global linkedness is a generic property of G, that is, for each vertex pair $\{x, y\}$ either the pair is globally linked in every generic realization of G, or not globally linked in every generic realization of G,

(ii) for every generic realization (G, p) of G in $\mathbb{R}^2$ we have $r(G, p) = 2^{c(G)}$, where $c(G)$ is the number of diagonals uv of G for which $\{u, v\}$ is a separating pair in G',

(iii) a vertex set X in G' is a globally rigid cluster if and only if $G[X]$ is a maximal globally rigid subgraph of G' if and only if $G[X]$ is a maximal 3-connected subgraph of G' (or a K_3).

We close this section with an inductive construction of globally rigid uni-braced MOP graphs. A result in [7] shows that every globally rigid graph can be obtained from K_4 by a sequence of so-called 1-extensions and edge additions. Our construction uses the vertex splitting operation, which is one of the frequently used tools in the theory of rigid (planar) graphs, see e.g. [2]. Let uv be an edge of the graph G and let F_1, F_2 be a partition of the edges incident with v in $G - uv$. The *vertex splitting* operation at vertex v on the edge uv consists of deleting the vertex v from G and then adding two new vertices v_1, v_2 and new edges uv_1, uv_2, v_1v_2, as well as the edges xv_i for every edge xv with $xv \in F_i$, $i = 1, 2$. See Figure 4. The vertex splitting operation is said to be *non-trivial* if F_1, F_2 are both non-empty.

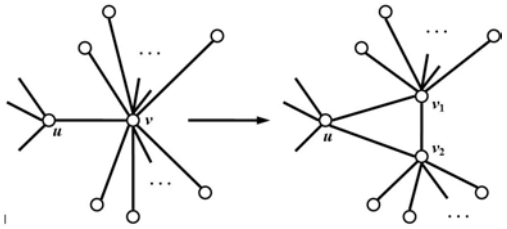


Figure 4: The vertex splitting operation at vertex v on edge uv.

Let v be a vertex of a MOP graph G, uv a diagonal incident with v, a let C_1, C_2 be the connected components of $G-\{u,v\}$. Then an *outerplanar vertex splitting* operation at v on uv is a vertex splitting operation in which F_i consists of the edges that connect v to C_i, $i = 1, 2$. It is not difficult to see that a graph obtained by an outerplanar vertex splitting from a MOP graph is again a MOP graph. In a uni-braced MOP graph G' with spanning MOP subgraph G and brace ab, we can define an outerplanar vertex splitting operation at v on edge uv in a similar manner, provided ab is disjoint from v.

With these definitions in place, we can show the following:

Theorem 16 *A graph is a 3-connected uni-braced MOP graph if and only if it can be obtained from K_4 by a sequence of non-trivial outerplanar vertex splitting operations.*

Proof. It is easy to see that non-trivial vertex splitting preserves 3-connectivity. Thus all graphs obtained from K_4 by non-trivial outerplanar vertex splitting operations are 3-connected uni-braced MOP graphs. To see the other direction, we show that if G' is such a graph on at least five vertices then we can perform the inverse operation (i.e. the contraction of some edge vv' of the boundary cycle which is disjoint from the brace and for which v and v' have a common neighbour) so that we obtain a smaller 3-connected uni-braced MOP graph. Let ab be the brace in G' and let $G = (V, E)$ be the underlying MOP graph. Every diagonal of G is (a, b)-separating, since otherwise the end-vertices of the diagonal would form a separating pair in G' by Lemma 6. Since G has at least five vertices, the boundary cycle of G must contain an edge $e = vv'$ that is disjoint from $\{a, b\}$. By the MOP property v and v' have a common neighbour. Thus e is contractible in the above sense and hence the resulting graph G'' obtained from G' by contracting e is a uni-braced MOP graph with brace ab. Moreover, G'' is 3-connected, since every diagonal of G is (a, b)-separating. The theorem follows by induction on the number of vertices. □

4 Questions and conjectures

Although we have extended the family of those graphs for which global linkedness is well characterized and computationally tractable in $\mathbb{R}^2$, Conjecture 1 remains open. On the other hand, our results motivate the following new questions:

(1) For which family of (rigid) graphs is global linkedness in $\mathbb{R}^2$ a generic property?

Examples of families that satisfy this property are $\mathcal{R}_2$-connected graphs (Theorem 3), graphs with a spanning MOP graph (Theorem 15), so-called special graphs (i.e. minimally rigid graphs without proper non-complete rigid subgraphs) [9] and 2-trees (graphs that can be obtained from a sequence of 2-merge operations on triangles).

(2) For which family of (rigid) graphs is global linkedness in $\mathbb{R}^2$ characterized by the $\kappa \geq 3$ condition (for non-adjacent vertices)?

The next question may also be interesting.

(3) For which family of (rigid) graphs does global linkedness in $\mathbb{R}^2$ hold only for adjacent vertex pairs?

Here we offer the following conjecture, which would extend Theorem 4, and which would follow from Conjecture 1.

Conjecture 2 *A graph $G = (V, E)$ has no globally linked pairs $\{u, v\}$ in $\mathbb{R}^2$ with $uv \notin E$ if and only if every $\mathcal{R}_2$-connected component of G is either a complete graph or can be obtained from a sequence of 2-merge operations on complete graphs of size at least four.*

Necessity follows from Theorem 3.

5 Concluding remarks

We have characterized the globally linked pairs in braced MOP graphs in $\mathbb{R}^2$, extending the family of graphs for which the statement of Conjecture 1 holds. As a by-product we have obtained a simpler characterization for the two-dimensional global rigidity of braced MOP graphs which makes it possible to test their global rigidity in linear time, by using fast algorithms that can check if a graph is 3-connected, see e.g. [6]. We note that the problem of testing, in polynomial time, whether a given graph is a braced MOP graph seems to be open.

The theory of globally rigid graphs and globally linked pairs has found numerous applications, including wireless sensor network localization, see [8]. Our result can also be used in this context. For example, the characterization of globally linked pairs (Theorem 14) enables one to identify the uniquely localizable vertices in $\mathbb{R}^2$ with respect to any designated set of so-called anchor

vertices in a sensor network, provided the graph of the network contains a spanning MOP graph. It also implies an affirmative answer to another conjecture of [9] concerning unique localizability in $\mathbb{R}^2$, when the graph is a braced MOP.

A recent research direction is to find efficient algorithms that can add a smallest set of new edges to a graph so that it becomes globally rigid, see [13]. By Theorem 12 this augmentation problem can be reduced to the well-studied 3-connectivity augmentation problem of graphs, provided the input is a (braced) MOP graph.

Finally, we remark that we have been able to substantially generalize our characterization of globally linked pairs in MOP graphs in $\mathbb{R}^2$ in several ways. In particular, the analogues of Theorems 12 and 14 turn out to also hold, in $d \geq 1$ dimensions, for so-called "braced d-trees", and, more generally, "braced d-connected chordal graphs." These new results will be part of a forthcoming manuscript.

6 Acknowledgements

This work was supported by the Hungarian Scientific Research Fund grant no. K135421 and the project Application Domain Specific Highly Reliable IT Solutions which has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme.

References

[1] R. Connelly, Generic global rigidity, *Discrete Comput. Geom.*, 33 (2005) 549–563.

[2] Z. Fekete, T. Jordán, and W. Whiteley, An inductive construction for plane Laman graphs via vertex splitting, Proc. 12th Annual European Symposium on Algorithms (ESA) 2004, (S. Albers and T. Radzik, eds) Springer Lecture Notes in Computer Science 3221, pp. 299–310, 2004.

[3] H. Gluck, Almost all simply connected closed surfaces are rigid, in Geometric topology, Lecture Notes in Math., Vol. 438, Springer, Berlin, 1975, 225–239.

[4] S. Gortler, A. Healy, and D. Thurston, Characterizing generic global rigidity, *American Journal of Mathematics*, Volume 132, Number 4, August 2010, pp. 897–939.

[5] B. Hendrickson, Conditions for unique graph realizations, *SIAM J. Comput* 21 (1992), pp 65–84.

[6] J.E. Hopcroft and R.E. Tarjan, Dividing a graph into triconnected components, *SIAM J. Comput.* 2 (1973), 135–158.

[7] B. Jackson and T. Jordán, Connected rigidity matroids and unique realizations of graphs, *J. Combin. Theory*, Ser. B, 94, 1–29, 2005.

[8] B. Jackson and T. Jordán, Graph theoretic techniques in the analysis of uniquely localizable sensor networks, in: G. Mao, B. Fidan (eds), Localization algorithms and strategies for wireless sensor networks, IGI Global, 2009, pp. 146–173.

[9] B. Jackson, T. Jordán, and Z. Szabadka, Globally linked pairs of vertices in equivalent realizations of graphs, *Disc. Comput. Geom.*, Vol. 35, 493–512, 2006.

[10] B. Jackson, T. Jordán, and Z. Szabadka, Globally linked pairs of vertices in rigid frameworks, in: Rigidity and Symmetry (R. Connelly, W. Whiteley and A. Ivic Weiss, eds.), Fields Institute Communications, 70, pp. 177–203 (2014).

[11] T. Jordán, Combinatorial rigidity: graphs and matroids in the theory of rigid frameworks, in: Discrete Geometric Analysis, *MSJ Memoirs*, vol. 34, pp. 33–112, 2016.

[12] T. Jordán and W. Whiteley, Global rigidity, in: Handbook of Discrete and Computational Geometry, 3rd edition, J.E. Goodman, J. O'Rourke, C. Tóth eds, CRC Press, 2018.

[13] C. Király and A. Mihálykó, Globally Rigid Augmentation of Minimally Rigid Graphs in $\mathbb{R}^2$, In: Calamoneri, Tiziana; Coro, Federico (eds.) Algorithms and Complexity : 12th International Conference, CIAC 2021, Proceedings, Springer Nature Switzerland AG (2021), pp. 326–339.

[14] G. Laman, On graphs and rigidity of plane skeletal structures, *J. Engineering Math.* 4 (1970), 331–340.

[15] J. G. Oxley, *Matroid Theory*, Oxford University Press, New York, 1992.

[16] H. Pollaczek-Geiringer, Über die Gliederung ebener Fachwerke, *Z. angew. Math. Mech.* 7 (1927), 58–72.

[17] J.B. Saxe, Embeddability of weighted graphs in k-space is strongly NP-hard, Tech. Report, Computer Science Department, Carnegie-Mellon University, Pittsburgh, PA, 1979.

[18] B. Schulze and W. Whiteley, in: Rigidity and scene analysis, Handbook of Discrete and Computational Geometry, 3rd edition, J.E. Goodman, J. O'Rourke, C.D. Tóth eds, CRC Press, 2018.

Unstacking Slabs Safely in Megalit is NP-Hard

Kirby Gordon* Jacob Lezberg† Aaron Williams‡

Abstract

We consider the problem of safely unstacking rectangular boxes through the lens of *Megalit* (ASCII, 1991). In this Game Boy game, the player is confronted with a pile of 1-by-k and k-by-1 megaliths (slabs). The goal is to pull and push the slabs until they reach the floor, without falling more than one cell at a time. We prove that an associated problem is NP-hard. Along the way, we introduce the drop-ladders problem, and prove that the Game Gear game *Popils* (Tengen, 1991) is NP-hard.

1 Introduction

In the late-1990s, the quintessential box pushing game *Sokoban* (倉庫番) was shown to be NP-hard independently by Fryers and Greene [11], Dor and Zwick [8], and Uehara [21]. In this top-down game, the player controls an agent who must push m boxes onto m locations. Many papers have been written under the Push[Push]-1/k/*-[X] banner (e.g., [15, 7]), where the goal is to reach a location under various physical models. Pulling [2], pushing rows [13], and rotation [12] have been studied, and *Sokoban* was proven PSPACE-complete [6].

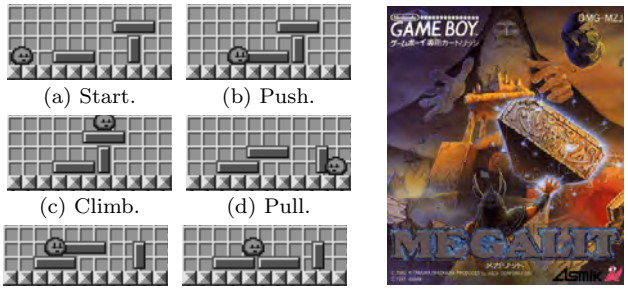
The 1990s also saw the establishment of NP-hardness for *Blocks World* [5]. In this grid-based problem, 1-by-1 blocks are stacked in columns and the goal is to unstack and restack the blocks to be in a particular state, and this led to ample subsequent research [14, 20, 16, 19].

In this paper, we consider a decision problem that has ingredients of both box pushing and *Blocks World*.

- *Side-view.* The two-dimensional world has gravity and a side perspective [18, 10] like *Block Dude* [4, 3].
- *Agent-based.* The player controls an agent who is vulnerable to falling objects, but not falling.
- *Unstacking.* Unlike *Sokoban* and *Blocks World*, the goal is to safely bring the boxes to the ground floor.
- *Fragility.* Unlike *Sokoban* and *Blocks World*, the blocks break when dropped more than one unit.
- *Push and Pull.* The agent can push and pull [17].

1.1 Inspiration and Outline

We were inspired by another 1990s artifact: *Megalit* (ASCII, 1991) for Nintendo's Game Boy (see Figure 1).



(a) Start. (b) Push. (c) Climb. (d) Pull. (e) Climb. (f) Push. (g) Box art (🇯🇵).

Figure 1: Solving a *Megalit* level (a)–(f).

- Section 2 formalizes our *Megalit* decision problem.
- Section 3 describes how any configuration of slabs can be flattened with the help of additional slabs.
- Section 4 introduces a toy problem called Ladders.
- Section 5 provides our reduction from Ladders.
- Section 6 proves that our reduction is correct.
- Section 7 concludes with final remarks.

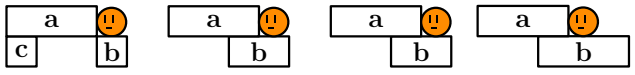
2 Megalit: Gameplay and Decision Problem

Megalit puzzles involve *slabs* and the following rules:

1. The playfield is a grid with bottom-left cell at $(0, 0)$.
2. The player may move left, move right, or jump. A jump is 3 units high. While jumping, the player may travel up to 2 units left or right.
3. If the player is horizontally adjacent to a slab and on stable footing (i.e. not in midair), they may grab the slab and push or pull it with them as they move.
   (a) If the player moves and falls off a slab while clutching a slab, then their grip is released.
   (b) Only the slab being grabbed is pushed or pulled; only one slab moves at a time.
   (c) Slabs will fall down due to gravity when unsupported. They cannot move upwards.
4. Slabs are horizontal 1-by-k or vertical k-by-1.
5. A slab $\mathbf{x}$ is *supported* by slab $\mathbf{y}$ when any cell of $\mathbf{x}$ is directly above a cell of $\mathbf{y}$ (see Figure 1d).
6. A level is failed if a slab falls ≥ 2 units or the player is crushed under a falling slab or trapped.
7. A cleared level has every slab touching the ground.

In the actual game, the slabs and player move horizontally in $\frac{1}{2}$-unit increments. In our decision problem we ignore this complication and use 1-unit moves. We also disallow pull and push moves that result in the player falling, which affects rule 3a as shown in Figure 2.

*Williams College, jacob.lezberg@gmail.com
†Williams College, kirbyjgordon@gmail.com
‡Williams College, aaron.williams@williams.edu



Can't push **a**. Can push **a**. Can't pull **a**. Can pull **a**.

Figure 2: In our Megalit decision problem the player must step onto a solid cell when performing a pull or push.

Our decision problem Megalit$(\mathcal{C}, p)$ asks if a configuration of supported slabs $\mathcal{C}$ can be flattened when the player starts at position p. Our main result is below.

Theorem 1 *The decision problem* Megalit *is NP-hard.*

We will specify a slab $\mathbf{s}$ by a 4-tuple, (x_1, y_1, x_2, y_2), where (x_1, y_1) and (x_2, y_2) are the bottom-left and top-right grid cells in $\mathbf{s}$, respectively. We also let cells($\mathbf{s}$) be the set of all cells in slab $\mathbf{s}$, and cells($\mathcal{C}$) be the union of cells($\mathbf{s}$) for $\mathbf{s} \in \mathcal{C}$. Finally, we normalize our co-ordinate system so that the minimum x and y coordinates of slabs in $\mathcal{C}$ are both zero. In other words, $(0, 0)$ is the bottom-left cell in the initial bounding box of the slabs.

3 Ramunto's Extraction Algorithm

In this section, we show that a configuration of slabs $\mathcal{C}$ can be safely flattened, so long as it is surrounded by other slabs. Our flattening process resembles pulling pizzas out of an oven using long wooden pizza trays known as *pizza peels*. Thus, we name our approach after our local pizza joint: *Ramunto's Brick House Pizza* [23].

3.1 Extractable Slabs

We define a slab $\mathbf{s} = (x_1, y_1, x_2, y_2)$ to be *extractable* if it has the following three properties:

X_1: There is no slab cell to the right of $\mathbf{s}$. That is, $\nexists \mathbf{t} \in \mathcal{C}$ with $(x, y) \in$ cells($\mathbf{t}$) and $x > x_2$ and $y_1 \leq y \leq y_2$.

X_2: There is no slab cell above $\mathbf{s}$. That is, $\nexists \mathbf{t} \in \mathcal{C}$ with $(x, y) \in$ cells($\mathbf{t}$) and $y > y_2$ and $x_1 \leq x \leq x_2$.

X_3: There is no gap to the right of $\mathbf{s}$ in the row below it. That is, if $\exists \mathbf{t} \in \mathcal{C}$ with $(x_2 + i, y_1 - 1) \in$ cells($\mathbf{t}$) for $i \geq 2$, then $\exists \mathbf{t}' \in \mathcal{C}$ with $(x_2+i-1, y_1-1) \in$ cells($\mathbf{t}'$).

For insight into this definition, the reader may skip ahead to Figures 4–5. Property X_3 ensures that $\mathbf{s}$ can be pulled along a series of slabs until it is transferred to a pizza peel; X_2 ensures that no slab $\mathbf{t}$ gets in the way when $\mathbf{s}$ is pulled to the right; X_1 ensures that $\mathbf{s}$ does not support any other slab $\mathbf{t}$ that could fall and break.

Lemma 2 *Every non-empty configuration of slabs $\mathcal{C}$ contains at least one extractable slab.*

Proof. We find an initial candidate slab, and then prove that it is extractable, or find a new candidate. During this process, we maintain a rectangular *search region* from (x_*, y_*) to (x^*, y^*), which are initially the bottom-left and top-right cells in cells($\mathcal{C}$), respectively.

If $\exists \mathbf{s} \in \mathcal{C}$ with $(x^*, y^*) \in$ cells($\mathbf{s}$), then $\mathbf{s}$ is extractable since all three properties are vacuously true. Otherwise, we let the first candidate be $\mathbf{s} = (x_1, y_1, x_2, y_2) \in \mathcal{C}$ that maximizes the minimum of $x^* - x_2$ and $y^* - y_2$, breaking ties by maximizing y_2. In other words, $\mathbf{s}$ is the first slab found by searching along down-right lines originating from the top row proceeding from right-to-left. After identifying the candidate, we reduce the search region.

- If $\mathbf{s}$ is horizontal, then we set $(x_*, y_*) = (x_1, y_1 + 1$ and $(x^*, y^*) = (x_2, y^*)$ (i.e., the cells above $\mathbf{s}$.
- If $\mathbf{s}$ is vertical, then we set $(x_*, y_*) = (x_1 + 1, y_1$ and $(x^*, y^*) = (x^*, y_2)$ (i.e., the cells right of $\mathbf{s}$).

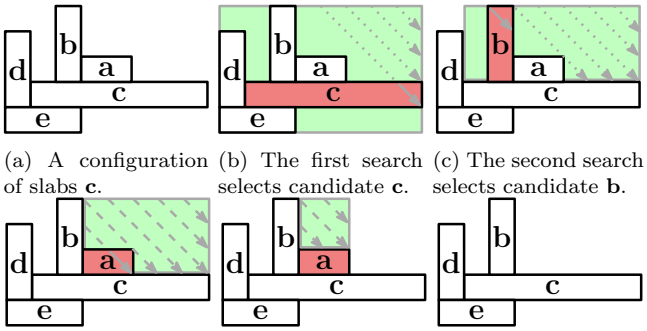
If a slab is found in the new search region, then it is the new candidate, and we repeat the process. Otherwise, $\mathbf{s}$ is our finalized candidate. Finally, we need to prove that $\mathbf{s}$ is extractable. We first assume that $\mathbf{s}$ is horizontal.

X_1: There is no slab $\mathbf{t}$ with a cell above $\mathbf{s}$ since the search that made $\mathbf{s}$ a candidate would have found $\mathbf{t}$.

X_2: There is no slab $\mathbf{t}$ with a cell to the right of $\mathbf{s}$ since nothing was found in the final search area.

X_3: There is no gap immediately below and right of $\mathbf{s}$ since the search that made $\mathbf{s}$ a candidate would find a slab $\mathbf{t}$ in the row immediately below $\mathbf{s}$.

When $\mathbf{s}$ is vertical the same points hold, but with the first two arguments interchanged. □



(a) A configuration of slabs **c**. (b) The first search selects candidate **c**. (c) The second search selects candidate **b**.

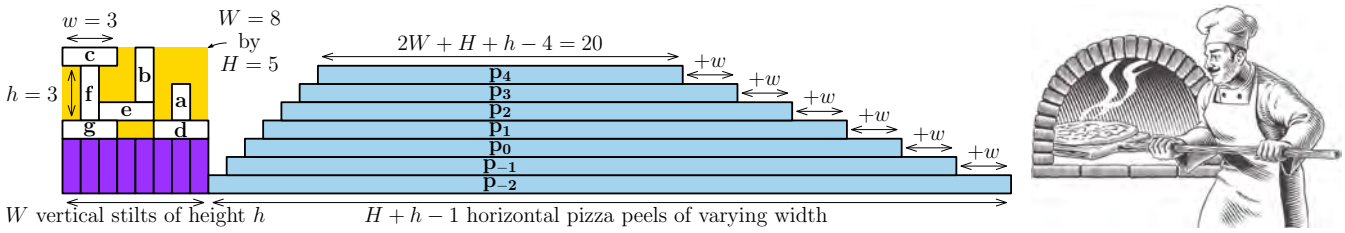
(d) The third search selects candidate **a**. (e) The fourth search confirms candidate **a**. (f) Ramunto's algorithm will extract **a**.

Figure 3: Lemma 2 considers successively smaller search regions (green) along arrows (gray) from the top-right. The confirmed candidate is extractable. The slabs are labeled by their extraction order during Ramunto's algorithm.

3.2 Pizza Oven Template

Let $\mathcal{C}$ be a configuration of slabs of width W and height H, with the maximum width and height of slabs being w and h, respectively. We surround this configuration slabs with additional slabs as follows.

- Vertical slabs of height h below $\mathcal{C}$.
- Horizontal slabs $\mathbf{p_i}$ for $-W + 1 \leq i \leq H - 1$. Each slab is $w + 1$ units wider than the one above, with the top slab $\mathbf{p_{H-1}}$ having width $2W + H + h - 4$. These slabs are organized into a left staircase immediately to the right of $\mathcal{C}$.



(a) The configuration $\mathcal{C}$ in the gold region, with purple vertical slabs and blue horizontal slabs (pizza peels) added, forms $\mathcal{C}' = \mathsf{pizza}(\mathcal{C})$. The peel lengths ensure support for each extraction.

(b) Slabs in $\mathcal{C}$ are extracted like pizzas, and (gently) dropped.

Figure 4: The demolition layout for our example configuration $\mathcal{C}$. The total width and height of $\mathcal{C}$ are $W = 8$ and $H = 5$, respectively, while the maximum width and height of a slab in $\mathcal{C}$ are $w = 3$ and $h = 3$, respectively.

We denote the resulting configuration $\mathcal{C}' = \mathsf{pizza}(\mathcal{C})$. This is illustrated in Figure 4, where the initial $\mathcal{C}$ form the slabs inside of the pizza oven.

3.3 Extraction Algorithm

This section's main result is illustrated in Figure 5.

Theorem 3 *If $\mathcal{C}$ is a configuration of slabs, then*

$$\mathsf{Megalit}(\mathcal{C}', p) = \mathit{yes},$$

where $\mathcal{C}' = \mathsf{pizza}(\mathcal{C})$ and p is the unique standing position immediately to the right of $\mathcal{C}$.

Proof. Algorithm 1 flattens $\mathcal{C}'$[1]. □

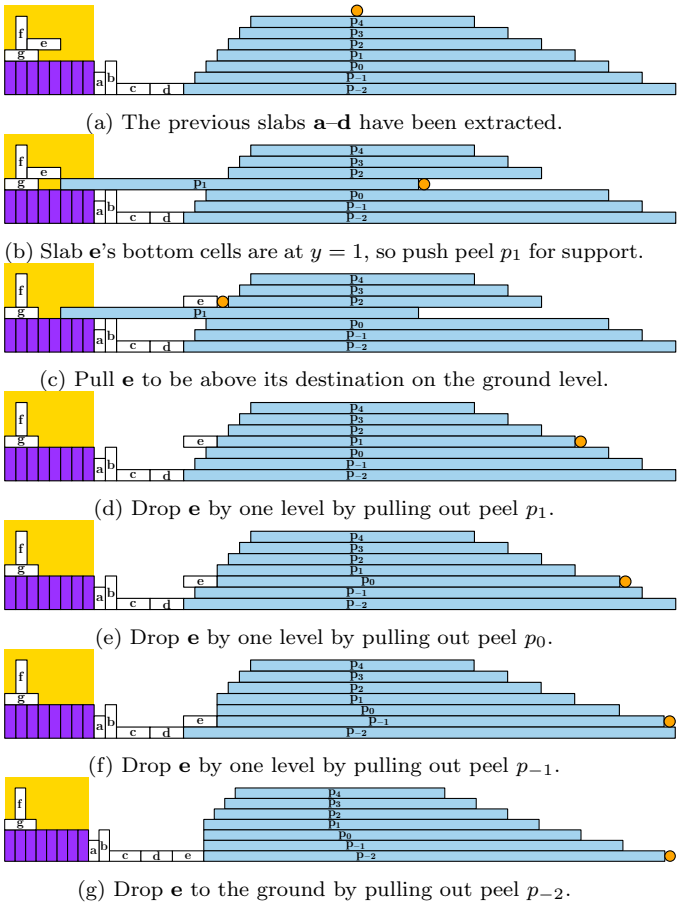
Algorithm 1 Ramunto's algorithm for flattening a configuration of slabs $\mathcal{C}' = \mathsf{pizza}(\mathcal{C})$ where p is the unique standing position immediately to the right of $\mathcal{C}$.

```
procedure RAMUNTOS(C')
  while |C| > 0 do
    C ← C − {s} for an extractable s = (x1, y1, x2, y2)
    push peel p_{x1} as close to s as possible
    pull s to be above the bottom peel p_{−W+1}
    pull peels p_{x1}, p_{x1−1}, ..., p_{−W+1} to lower s
    pull peels p_{−W+1}, p_{−W+2} ..., p_{H−1} to alignment
  end while
  pull peels p_{−W+1}, p_{−W+2}, ..., p_{H−1} to flatten them
end procedure
```

3.4 Flattening Goal to Target Location Goal

Theorem 3 helps us reduce the problem of flattening a configuration of slabs to reaching a particular location. For example, the player can complete Figure 4 so long as they can exit the initial gold region to the right, since Ramunto's algorithm will work regardless of where the slabs in $\mathcal{C}$ are located. We'll further refine this idea by surrounding the gold region with tall vertical walls, which ensures that the player can flatten the level if, and only if, they can reach the top-right cell in the region. In other words, we change the flattening goal into target location goal, and then a climbing goal. Climbing is further discussed in the following toy problem.

[1]Several temptingly simple greedy algorithms do not work.



(a) The previous slabs **a**–**d** have been extracted.

(b) Slab **e**'s bottom cells are at $y = 1$, so push peel p_1 for support.

(c) Pull **e** to be above its destination on the ground level.

(d) Drop **e** by one level by pulling out peel p_1.

(e) Drop **e** by one level by pulling out peel p_0.

(f) Drop **e** by one level by pulling out peel p_{-1}.

(g) Drop **e** to the ground by pulling out peel p_{-2}.

Figure 5: A snapshot of Ramunto's algorithm extracting slab **e** from configuration $\mathcal{C}$. After **e** reaches the ground in (g), the player realigns the peels as in (a) in order to extract **f**.

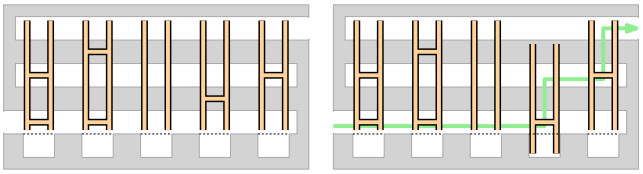
4 A Toy Problem

In this section, we introduce an NP-complete toy problem that resembles several well-established metatheorems [9] [22] [1]. The problem involves drop-ladders and we'll illustrate it by proving that *Popils* is NP-hard.

4.1 Drop-Ladders

A *drop-ladders* problem consists of ℓ ladders and $f + 1$ floors. Each *ladder* extends from the ground floor up

to the top floor and consists of some number of *rungs*. More specifically, there are $2f$ possible locations for a rung: on floor i and between floor i and $i+1$, for $1 \leq i \leq f$ (where $i = 1$ denotes the bottom floor). If a ladder contains a rung on floor i, then the player can *climb* from floor i to floor $i+1$ using this ladder; the rungs between floors cannot (immediately) be used to climb. In addition, when the player is on the bottom floor, they have the ability to lower any ladder by one unit. Lowering a ladder moves all of its rungs down one position, so a rung that was between floor i and $i+1$ moves to floor i, thus allowing the player to climb upward. Similarly, a rung on floor i moves between floor $i-1$ and floor i, eliminating its ability to help the player climb upward. The player starts on the bottom floor, and their goal is reach the top floor. Figure 6 provides an illustration.



(a) A Ladders problem. (b) A solution.

Figure 6: (a) A Ladders problem with $\ell = 5$ ladders and $f+1 = 3$ floors. It is a yes instance since the player can climb from the ground floor to the top floor using the rungs on the first ladder. Alternatively, they can drop the fourth ladder, then climb the fourth ladder and fifth ladder, as in (b). This level illustrates our reduction from ϕ in (1), with ladders for variables v_1–v_5 from left-to-right, and floors for $C_1 = (v_1 \vee v_2 \vee \neg v_4)$ and $C_2 = (v_1 \vee \neg v_2 \vee v_5)$ from bottom-to-top. The solution in (b) corresponds to the satisfying assignment $v_1 = v_2 = v_3 = v_5 = True$ and $v_4 = False$, with C_1 satisfied by $v_4 = False$ and C_2 satisfied by $v_5 = True$.

4.2 Ladders is NP-Complete

We now prove that Ladders is NP-complete.

Lemma 4 Ladders *is NP-complete.*

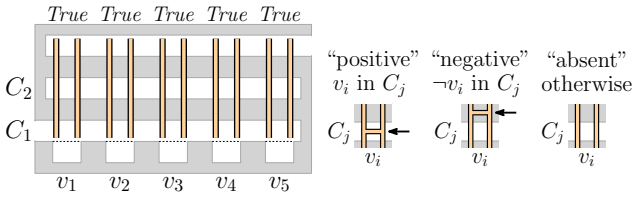
Proof. NP-hardness follows from a simple 3-SAT reduction, as illustrated in Figures 6–7 for the following:

$$\phi = C_1 \wedge C_2 = (v_1 \vee v_2 \vee \neg v_4) \wedge (v_1 \vee \neg v_2 \vee v_5). \quad (1)$$

In particular, lowering a ladder corresponds to changing the assignment of a variable from $True$ to $False$. Ladders is in NP since a suitable certificate specifies which ladders to drop and to climb. □

The following observation strengthens Lemma 4.

Observation 1 *It is possible to climb to floor i if and only if the variable assignment associated with the dropped ladders satisfies clauses $C_1, C_2, \ldots, C_i$.*



(a) Template for $n = 5$ variables and $m = 2$ clauses. (b) Rungs are added based on literal-clause membership.

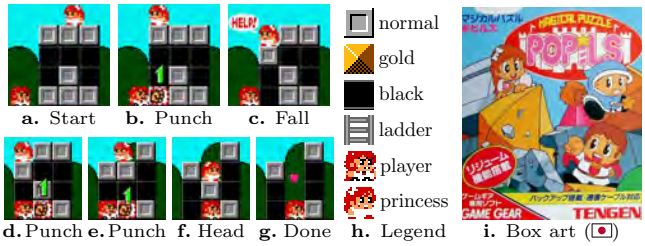
Figure 7: Reducing 3-SAT to Ladders using (a) a template, with (b) added rungs. Specifically, if v_i is in clause C_j, then ladder i has a rung on floor j; if $\neg v_i$ is in clause C_j, then ladder i has a rung between floor j and floor $j+1$.

4.3 Application: Popils is NP-Hard

Another 1990s handheld puzzler is Tengen's *Magical Puzzle Popils* (🇯🇵 1991) / *Popils* (🇺🇸 1992) for Sega's Game Gear. Each round follows the save-the-princess trope. Its mechanics and elements include the following:

- *Normal blocks.* Breakable by punching left or right, headbutting up, or kicking down.
- *Gold blocks.* Unbreakable and can be stood on.
- *Black blocks.* Empty and cannot be stood on.
- *Ladders.* The player can walk across or on top.

When the player breaks a normal block, all of the blocks stacked above in the same column will fall down one cell. The princess paces horizontally and never intentionally moves vertically; both characters are subject to gravity. The Popils decision problem generalizes the single-screen rounds to be arbitrarily large. See Figure 8.



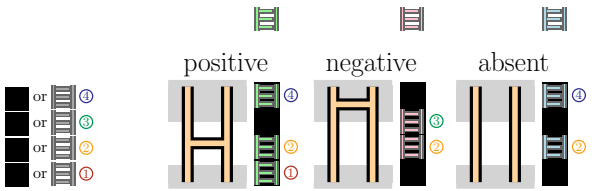
a. Start **b.** Punch **c.** Fall **d.** Punch **e.** Punch **f.** Head **g.** Done **h.** Legend **i.** Box art (🇯🇵)

Figure 8: Round 1 in *Popils* with partial legend and box art.

We now illustrate Ladders with two reductions to Popils. The first uses the elements in Figure 8h and has simpler player movements, while the second omits gold blocks and has simpler *rung gadgets*. See Figures 9–10 and 11–12 for details, where ladders cells in the rung gadgets are tinted green, pink, or blue for readability.

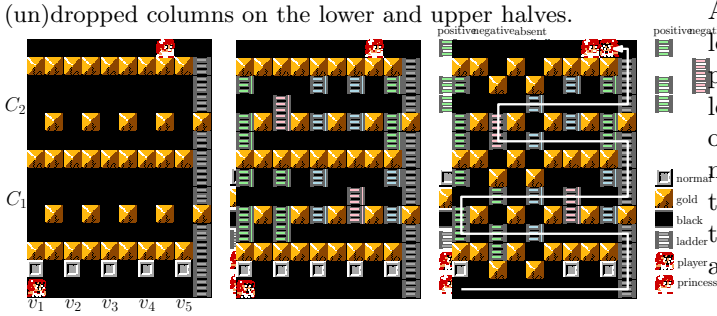
In both reductions, a drop-ladder occupies one column with a normal block at its base. The player starts in a *cellar* that appears below the first clause. The cellar is used to set the value of the variables. More specifically, the player can drop a drop-ladder by headbutting its normal block. Additional ladders on the right allow the player to exit a clause and enter the next clause, with the Princess pacing above the last clause. Each clause has a *lower-half* and an *upper-half*. To traverse a

clause, the player walks right-to-left on the lower-half, climbs a ladder associated with a satisfying literal, and then walks left-to-right on the upper-half.



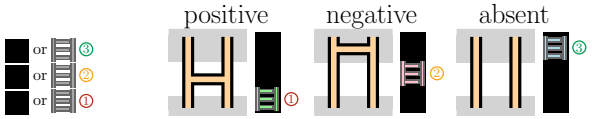
(a) Four cells. (b) Implementing the three gadgets.

Figure 9: Rung gadgets using four ladder or black square cells. To understand (b) note that ① and ② allow climbing undropped columns, while ② and ③ allow climbing dropped columns. The remaining ladder cells allow walking past (un)dropped columns on the lower and upper halves.



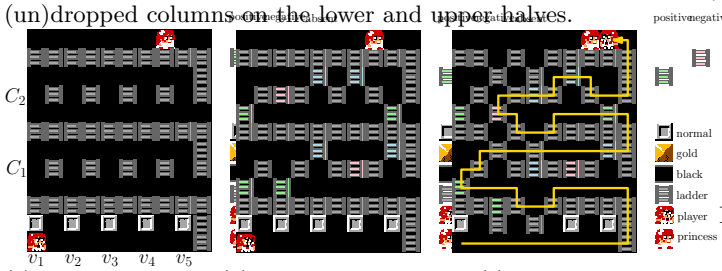
(a) Template. (b) Popils level for ϕ. (c) Solution.

Figure 10: Reducing Popils to Ladders for formula ϕ in (1) with $n = 5$ variables and $m = 2$ clauses. The provided solution involves headbutting and dropping the v_2 and v_3 columns, and then using $v_1 = T$ to satisfy $C_1 = (v_1 \vee v_2 \vee \neg v_4)$ and $v_2 = F$ to satisfy $C_2 = (v_1 \vee \neg v_2 \vee v_3)$.



(a) Three cells. (b) Implementing the three gadgets.

Figure 11: Rung gadgets using three ladder or black square cells. To understand (b) note that ① allows climbing undropped columns, while ② allows climbing dropped columns. The ladder in the absent case allows walking past (un)dropped columns on the lower and upper halves.



(a) $n = 5$ and $m = 2$. (b) Popils level for ϕ. (c) Solution.

Figure 12: Reducing Popils to Ladders for formula ϕ in (1). The provided solution follows the same format as Figure 10 but requires a more complicated path.

The second reduction gives the following theorem.

Theorem 5 Popils *is NP-hard with only ladders, normal blocks, and black blocks, even if the player can't fall.*

5 Reduction from 3-SAT

Now we describe our reduction from 3-SAT to Megalit. More specifically, we describe the configurations $\mathcal{C}$ that will be placed in the 'pizza oven' as per Section 3. We refer to these configurations as *haunted houses*. A sample is given in Figure 17. We define two new terms:

- A *cornerstone* is a lower corner cell of a slab. Vertical slabs have 1, while horizontal slabs have 2.
- A slab is *mobile* if the player can stand next to a cornerstone of the slab and push or pull it.

5.1 Primitive Gadgets: Tables and Chunks

A *table*[2] consists of a tabletop supported by a pair of legs which are 2 or 3 units in height. This gadget appears in many variations, as seen in Figure 13. Only one leg is needed to support the tabletop, which allows the other leg to be pulled away. The player may pass underneath a table, by pulling and pushing the legs to reset them as needed, or over it (given that nothing on the tabletop obstructs passage). They may also re-purpose a leg as a climbing aid elsewhere in the level.

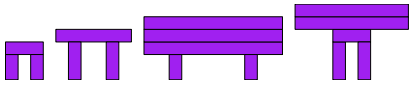
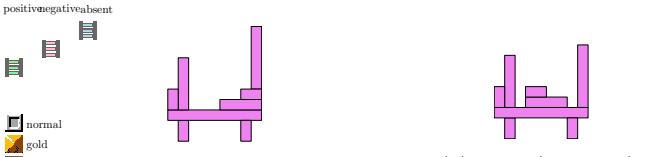


Figure 13: Tables come in many different sizes.

A *chunk* is a row of adjacent vertical slabs in any plural quantity. Notably, a chunk is extremely stable because only its two exterior slabs have exposed cornerstones. It follows that a chunk is completely immobile if the player cannot access either of these cornerstones.

5.2 Variable Towers

Variables are modeled by vertical structures called *variable towers* (see Figure 16). The base of each variable tower is a *pit of truth*, as shown in Figure 14a.



(a) True (default). (b) False (modified).

Figure 14: A pit of truth in its initial and modified state.

By pulling out the left leg of the underlying table, the player can jump up into the pit. To escape the pit, they must move the two horizontal slabs, as shown in Figure 14b. The net effect is that the tower is lowered by 2 units. This is analogous to dropping a ladder in the toy problem, or setting the associated variable to False. We note that a horizontally-mirrored version of the pit is used at the base of the last variable tower.

[2]Historically, this stone table structure is known as a *dolmen*.

Above a pit we stack literal gadgets for that variable. These gadgets come in three flavors (see Figure 15).

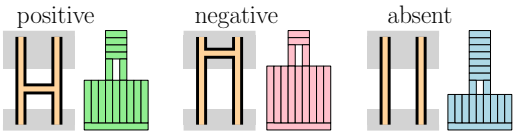


Figure 15: The three types of variable literal gadgets.

Each literal has a horizontal 9x1 slab at the base, then a chunk covering its full width, and finally a small table with multiple tabletop layers. Each gadget is 14 units tall, but the relative placement (and height) of the legs change based on the type of literal. As in our toy problem, the positive literals will be useful for climbing, unless the associated variable is negated, which causes its negative literals to become useful.

5.3 Scaffolding

Scaffolding towers between the variable towers enable climbing. A scaffolding tower is supported by a pair of *support gadgets* (yellow), which consist of a table with a tall vertical slab on top. The base of a scaffolding tower also includes a *pyramid gadget* (yellow), which allows the player to climb up to the rest of the scaffolding. See the bottom-middle of Figure 16.

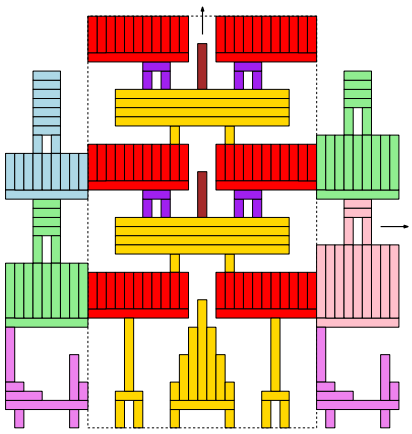


Figure 16: Illustrating two variable towers (left and right) with scaffolding (middle) between them.

A *scaffolding block* has two horizontal slabs supporting chunks of height 4 (red), each of which supports one leg of a large table (yellow). This table has 4 tabletop layers, and on top of it are two small tables (purple) on either side of a 5-unit vertical slab (maroon). This scaffolding tower can be seen in the center of Figure 16. If the player enters a scaffolding block from the bottom, they can access either side of the gadget, but cannot jump to the upper surface without a climbing aid (analogous to rungs from the toy problem). The mapping of these jumps to the satisfiability of the given problem is the discussion of Section 6.

6 Proof that Megalit is NP-Hard

The player must complete a constructed level in two stages – ascend to the top of the "haunted house," then flatten it using "pizza peels." We complete our proof of Theorem 1 by proving that the first stage can be completed if, and only if, the 3-SAT instance is satisfiable. More specifically, Observation 1 holds. The following lemmas prove the two directions.

Lemma 6 *Given a satisfiable instance of 3-SAT, the level generated by the reduction rules is climbable.*

Proof. Suppose we are given a satisfiable instance of 3-SAT and build a *Megalit* level by the procedure in Section 5. Next, we let the player dislodge the truth-setting slabs for each variable as would correspond to a satisfying assignment for the instance of 3-SAT. Since the given instance of the problem is satisfiable, each clause has at least one positive literal set to "true" or one negative literal set to "false."

In both of these cases, the corresponding gadget representing the variable literal is offset by exactly one unit above its neighboring scaffolding. We can see that the truth-assignment created this useful offset, just as it made the ladder rungs accessible in Lemma 4. As a result, the player is able to borrow an extra table leg from the literal and bring it into the scaffolding region close enough to the center that they can use it as a "ladder rung" to overcome the tall jump from the lower scaffolding surface to the upper one. See Figure 18.

Since this occurs for each clause, and therefore for each layer of scaffolding and literals, the player can jump all the way to the top of the house by moving laterally until they reach the satisfying literal, using its table leg to climb to the next layer, and repeating this process for each layer. In accordance with Observation 1, this climb is possible because the satisfying variable assignment provided a climbing aid on each "floor." □

Observation 2 *It is impossible to move a tabletop on an isolated table; see our physical model in Section 2.*

Observation 3 *Even when two tables are adjacent on the same surface, it is impossible to move either tabletop, provided the leg-heights of the tables are different.*

Lemma 7 *Given an unsatisfiable instance of 3-SAT, the level generated from the reduction has no escape routes from the haunted house.*

Proof. Suppose that we are given an unsatisfiable instance of 3-SAT. We must show that no alternative exits from the haunted house exist for the player. Referencing Figure 17, we will proceed with an examination of each gadget and then each zone where gadgets may interact with one another.

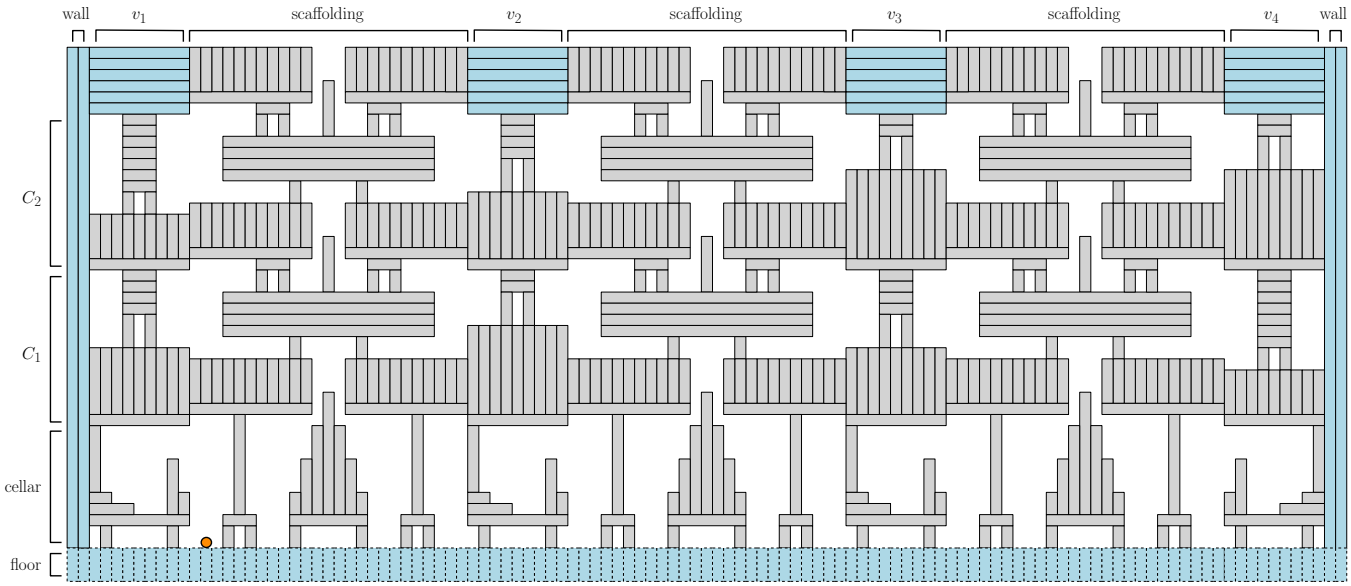


Figure 17: A full haunted house, with surrounding structure (walls and floor) and labeled regions, generated from the 3-SAT instance $\phi = C_1 \wedge C_2$ with $C_1 = (v_1 \vee \neg v_2 \vee v_3)$ (middle layer) and $C_2 = (v_2 \vee \neg v_3 \vee \neg v_4)$ (top layer). The player is shown at their starting location (orange circle) and the additions in blue ensure that the player may only exit the house via its top level. The added slabs on top of each scaffolding tower enable traversal along the "roof" and pairs of tall vertical slabs bookend the house as per Section 3.4. Note that the vertical "floor" slabs are much taller than shown here, and a set of pizza peels is present immediately to the right of this structure.

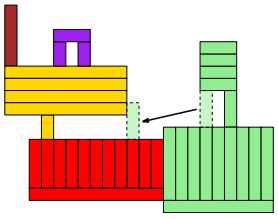


Figure 18: A +1 offset allows a table leg from the tower to be brought into the scaffolding region. The player cannot climb the scaffolding further without this aid.

The player has free lateral movement in the cellar and may encounter four notable structures:

1. *The Pit.* Due to the manner in which it traps the player, the pit has an isolating effect in all respects besides the intended 2-unit tower drop, and thus cannot affect the rest of the haunted house.
2. *Scaffolding Support.* Per Observation 3, the only way for the player to interact with the support's table is to pass beneath it. The mobility of the long vertical support slab is severely limited, as it cannot be safely dropped from the table and has only 1 unit of available movement left or right.
3. *Pyramid.* The aforementioned constraints on tables apply to the pyramid's base as well. The vertical slabs covering the tabletop's surface form a chunk, preventing access to each other's cornerstones.
4. *Thick Walls.* Double-slab walls blockade both sides of the house and are immobile.

None of these can interact with the larger structure except for the pit's intended transformation. Proceeding upwards, the player reaches a series of scaffolding/tower combinations. By itself, the scaffolding offers only two surfaces on which the player can stand and no mobile parts except the pairs of table legs. These legs cannot be dropped between scaffolding layers or surfaces within a layer because they would shatter. In an isolated tower, a variable literal offers the player only one surface on which to stand, from which the player can do nothing but pass underneath the table. At the edge of the house, the double-slab wall is again present, and equally immovable.

Finally, we consider interactions between the scaffolding and variable literals, characterized by the height of the literal relative to the scaffolding. For example, the surface of a true/negative literal is +3 units above the lower surface of the adjacent scaffolding.

- +3 (True/Negative): This offset is too high to safely drop a table leg from the tower's literal onto the scaffolding region. It is possible to use a scaffolding leg as a step down, creating an effective offset of +2, but there is no way to move it any closer to the scaffolding's center or cross the wide gap from there to the scaffolding's upper surface.
- +1 (False/Negative & True/Positive): As described in Lemma 6, this offset lets the player safely bring a table leg from the tower onto the scaffolding's lower surface, and use it as a ladder rung to reach the upper surface and continue climbing.
- −1 (False/Positive & True/Absent): Since slabs

may never move upwards (rule 3c), this table leg is stuck on the tower's surface. It is possible to drop a scaffolding leg into the tower section, but this offers no further productive moves. Note that the scaffolding leg *can* be used to replace the leg of a variable literal and lower the tower, but doing so will trap the player.

- -3 (False/Absent): The absent literal once again has no impact on the player's climbing ability.

Since no manipulations of slabs within a layer of scaffolding allow the player to climb higher without a positive evaluation of a variable, and there is no way to cause a collapse except for the 2-unit drop of a tower that occurs when escaping a pit, the player cannot escape the haunted house unless they create a positively-evaluated variable in each layer of scaffolding. This is equivalent to satisfying each clause in an instance of 3-SAT, but the given instance was unsatisfiable. Hence, the player cannot exit the haunted house. □

7 Open Problems

Sharpness. Is Megalit NP-complete or PSPACE-complete? Membership in NP is unclear since slabs can move back-and-forth and downward, but not upward.
Restrictions. Toward NP-completeness, one could consider slabs of constant size.
Generalizations. Toward PSPACE-completeness, one could add rectangular slabs, or immovable walls.
Physics. Megalit is not faithful to the physics of *Megalit*. In fact, $\frac{1}{2}$-unit moves and rule 3a invalidate Lemma 7. One could also consider center of gravity physics.
Fragility. Block pushing games and problems seldom consider the fragility of the objects being moved.
Unstacking. Other unstacking games could provide inspiration like *QBillion* (SETA, 1990) for Game Boy.

The authors would like to thank the anonymous referees whose feedback helped improve this paper.

References

[1] G. Aloupis, E. D. Demaine, A. Guo, and G. Viglietta. Classic nintendo games are (computationally) hard. *Theoretical Computer Science*, 586:135–160, 2015.

[2] J. Ani, S. Asif, E. D. Demaine, Y. Diomidov, D. H. Hendrickson, J. Lynch, S. Scheffler, and A. Suhl. PSPACE-completeness of pulling blocks to reach a goal. *J. Inf. Process.*, 28:929–941, 2020.

[3] J. Ani, L. Chung, E. D. Demaine, Y. Diomidov, D. Hendrickson, and J. Lynch. Pushing blocks via checkable gadgets: PSPACE-completeness of Push-1F and Block/Box dude. In *Proceedings of the 11th International Conference on Fun with Algorithm (FUN 2022)*, volume 226 of *LIPIcs*, page 3:1–3:30, 2022.

[4] A. Barr, C. Chang, and A. Williams. Block Dude puzzles are NP-hard (and the rugs really tie the reductions together). In *Proceedings of the 33rd Canadian Conference on Computational Geometry, Dalhousie University, Halifax, Canada, August 10-12, 2021*, 2021.

[5] S. V. Chenoweth. On the NP-hardness of blocks world. In *AAAI*, pages 623–628, 1991.

[6] J. Culberson. Sokoban is PSPACE-complete. In *Proceedings of the 1st International Conference on Fun with Algorithm*, pages 65–76, 1998.

[7] E. D. Demaine and M. Hoffmann. Pushing blocks is NP-complete for noncrossing solution paths. In *Proc. 13th Canad. Conf. Comput. Geom*, pages 65–68, 2001.

[8] D. Dor and U. Zwick. Sokoban and other motion planning problems. *Computational Geometry*, 13(4):215 – 228, 1999.

[9] M. Forišek. Computational complexity of two-dimensional platform games. In *FUN*, 2010.

[10] E. Friedman. Pushing blocks in gravity is NP-hard. *Unpublished manuscript, March*, 2002.

[11] M. Fryers and M. T. Greene. Sokoban, 1995.

[12] A. Greenblatt, O. Hernandez, R. A. Hearn, Y. Hou, H. Ito, M. J. Kang, A. Williams, and A. Winslo. Turning around and around: Motion planning through thick and thin turnstiles. In *CCCG*, 2021.

[13] A. Greenblatt, J. Kopinsky, B. North, M. Tyrrell, and A. Williams. Mazezam levels with exponentially long solutions. In *20th Japan Conference on Discrete and Computational Geometry, Graphs, and Games (JCDCGGG 2017)*, page 2, 2017.

[14] N. Gupta and D. S. Nau. On the complexity of blocks-world planning. *Artificial Intelligence*, 56(2-3):223–254, 1992.

[15] M. Hoffmann. Push-* is NP-hard. In *CCCG*, 2000.

[16] M. Johnson, C. Jonker, B. v. Riemsdijk, P. J. Feltovich, and J. M. Bradshaw. Joint activity testbed: Blocks world for teams (BW4T). In *International Workshop on Engineering Societies in the Agents World*, pages 254–256. Springer, 2009.

[17] A. G. Pereira, M. Ritt, and L. S. Buriol. Pull and pushpull are pspace-complete. *Theoretical Computer Science*, 628:50–61, 2016.

[18] M. Ritt. Motion planning with pull moves. *arXiv preprint arXiv:1008.2952*, 2010.

[19] L. She, S. Yang, Y. Cheng, Y. Jia, J. Chai, and N. Xi. Back to the blocks world: Learning new actions through situated human-robot dialogue. In *Proceedings of the 15th annual meeting of the special interest group on discourse and dialogue (SIGDIAL)*, pages 89–97, 2014.

[20] J. Slaney and S. Thiébaux. Blocks world revisited. *Artificial Intelligence*, 125(1-2):119–153, 2001.

[21] R. Uehara. 日の目を見なかった問題たち その2 [Problems that didn't see the light of day Part 2]. www.jaist.ac.jp/ uehara/etc/la/99/index.html, 1999.

[22] G. Viglietta. Gaming is a hard job, but someone has to do it! *Theory of Computing Systems*, 54:595–621, 2014.

[23] M. Willey. Ramunto's brick house pizza. . [Online; accessed 1-May-2022].

Computational Complexity of One-Dimensional Origami and Its Application to Digital Signature

Junnosuke Hoshido* Tonan Kamata* Tsutomu Ansai† Ryuhei Uehara*

Abstract

We investigate the computational complexity of a simple one-dimensional origami problem. We are given a paper strip P of length $n+1$ and fold it into unit length by creasing at unit intervals. Consequently, we have several paper layers at each crease in general. The number of paper layers at each crease is called the crease width at the crease. For a given mountain-valley assignment of P, in general, there are exponentially many ways of folding the paper into unit length consistent with the assignment. It is known that the problem of finding a way of folding P to minimize the maximum crease width of the folded state is NP-complete. In this study, we investigate a related paper-folding problem. For any given folded state of P, each crease has its mountain–valley assignment and crease-width assignment. Then, can we restore the folded state uniquely when only partial information about these assignments is given? We introduce this natural problem as the crease-restore problem, for which there are a number of variants depending on the information given about the assignments. In this paper, we show that some cases are polynomial-time solvable and that some cases are strongly NP-complete. As an application of the problem, we also propose a digital signature system based on the hardness of the crease-restore problem.

1 Introduction

Recently, computational origami has attracted the interest of theoretical computer scientists. In this paper, we focus on one of the simplest origami models: one-dimensional origami. This origami model involves a long rectangular strip of paper, which can be abstracted by a line segment and is uniformly subdivided by creases. At each crease, we fold the paper strip by degree π in either one of two choices for the direction of folding: a mountain fold, or a valley fold. Finding the number of feasible (i.e., without self-crossing) ways of folding a paper strip is known as a stamp-folding problem, for which the exact value remains open [5]: Experimentally, a paper strip of length $n+1$ has a total of $\Omega(3.06^n)$ feasible ways of folding and, on average, $\Omega(1.53^n)$ ways of folding for a given random mountain–valley assignment ("MV assignment," for short) of length n.

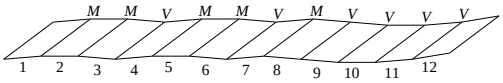


Figure 1: Example of MV assignment $MMVMMVMVVVV$ for paper strip of length 12.

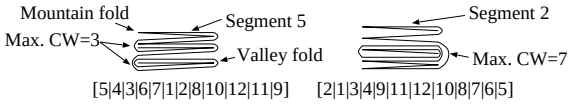


Figure 2: Side views of two folded states for MV assignment $MMVMMVMVVVV$. [5|4|3|6|7|1|2|8|10|12|11|9] and [2|1|3|4|9|11|12|10|8|7|6|5] describe the orders of paper segments from the top. The first folded state has the maximum crease width of 3, whereas the second has the maximum crease width of 7.

However, even when an MV assignment is given for the creases, the problem remains counterintuitive. In general, there are exponentially many ways of folding a paper strip with a given MV assignment. For example, a paper strip of length 12 with the MV assignment $MMVMMVMVVVV$, shown in Figure 1, has 100 different feasible folded states (as verified by a computer program), among which some are easy, while some are difficult, to fold flat. The main reason behind these differences in difficulty is the number of paper layers between two paper segments at each crease. For example, in the first folded state shown in Figure 2, the maximum number of layers at a crease is 3, whereas in the second folded state, the maximum number of layers is 7. From this viewpoint, an optimization problem was proposed and investigated in [6]. That paper introduced a new concept known as the "crease width" of a crease, which is defined by the number of paper layers at a crease in a folded state. Therein, it was proved

*School of Information Science, Japan Advanced Institute of Science and Technology, {s2110150,kamata,uehara}@jaist.ac.jp

†National Institute of Technology (KOSEN), Ibaraki College, ansai@gm.ibaraki-ct.ac.jp

that the decision problem for the maximum crease width of a given MV assignment is NP-complete. (In fact, among the 100 feasible folded states for the MV assignment $MMVMMVMVVVV$ shown in Figure 1, the first folded state is the only one with a maximum crease width of 3, which is optimal.)

Now, we consider the information necessary for specifying a folded state. We will observe that given both an MV assignment and a crease-width assignment for every crease ("CW assignment," for short), the folded state is uniquely determined if it is feasible. Then, what happens if we are given partial information about these assignments? This natural question leads us to our new computational origami problem, which is named the crease-restore problem. In this paper, we first show that the crease-restore problem is strongly NP-complete in general. More specifically, when we are given part of the MV assignment and CW assignment, the decision problem that asks whether there exists a feasible folded state is strongly NP-complete. We also show that even if the entire MV assignment is given, the crease-restore problem is still strongly NP-complete when only a part of the CW assignment is given.

Based on the hardness, we propose a digital signature system. In this system, an MV assignment is fixed as the ID of a user, and a CW assignment is used as its corresponding private key. Then, a pair consisting of the user ID and a partial CW assignment is used as a public key. The security of the signature system is based on the hardness of the strong NP completeness of the crease-restore problem.

2 Preliminaries

Herein, a *paper strip* refers to a one-dimensional line segment with creases at every integer position. (In other words, we ignore the thickness and width of the paper.) The paper strip is rigid except at the creases; that is, we are allowed to fold only along these creases at integer positions. We are given a paper strip of length $n+1$ placed in the interval $[0, n+1]$. (We will refer to this state as an *initial state*.) We call each paper segment between i and $i+1$ at the initial state the *segment* $i+1$. We assume that the top and bottom sides of the 1st segment are fixed. The paper strip is in a *folded state* if each crease is folded by a degree π or $-\pi$, and the folded strip is placed in the interval $[0, 1]$. The paper strip is *mountain (valley)-folded* at a crease i when the ith segment and the $(i+1)$st segment are folded in the direction such that their bottom sides (top sides, respectively) are close to touching (although they may not necessarily touch if they have some other paper layers between them). For a given paper strip, an *MV assignment* at crease i is either M or V, where M refers to a "mountain fold," and V refers to a "valley fold." A folded state is *feasible* if the paper strip does not penetrate itself in the given state.

We then provide formal definitions of feasibility and MV assignment for the sake of precision. When we obtain a folded state of P placed in the interval $[0, 1]$, the segments $1, 2, \ldots, n, n+1$ are positioned in this interval in some proper order. We define an ordering function f such that $f(i) = j$ denotes that the segment i is the jth layer in the folded state with $1 \leq i, j \leq n+1$. (That is, for the first folded state [5|4|3|6|7|1|2|8|10|12|11|9] shown in Figure 2, we have $f(1) = 6, f(2) = 7, f(3) = 3, f(4) = 2, f(5) = 1$, and so on.) Then, for each i with $1 \leq i \leq n$, the crease i (between segment i and $i+1$) is mountain-folded in the folded state if and only if (1) i is odd, and $f(i) < f(i+1)$, or (2) i is even, and $f(i) > f(i+1)$. Inversely, the crease i is valley-folded if and only if (3) i is odd, and $f(i) > f(i+1)$, or (4) i is even, and $f(i) < f(i+1)$. When the paper strip does not penetrate itself, the creases form a nest structure. Precisely, a folded state is feasible if and only if for any pair of integers i and j $(i \neq j)$ with the same parity,[1] we have either

- $\max\{f(i), f(i+1)\} < \min\{f(j), f(j+1)\}$ (crease i is over j),
- $\max\{f(j), f(j+1)\} < \min\{f(i), f(i+1)\}$ (crease j is over i),
- $f(i) < f(j) < f(j+1) < f(i+1)$, $f(i) < f(j+1) < f(j) < f(i+1)$, $f(i+1) < f(j) < f(j+1) < f(i)$, $f(i+1) < f(j) < f(j+1) < f(i)$ (crease i pinches j), or
- $f(j) < f(i) < f(i+1) < f(j+1)$, $f(j) < f(i+1) < f(i) < f(j+1)$, $f(j+1) < f(i) < f(i+1) < f(j)$, or $f(j+1) < f(i) < f(i+1) < f(j)$ (crease j pinches i).

(Consequently, the ith and jth creases should cross when we have $f(i) < f(j) < f(i+1) < f(j+1)$ or its symmetric cases, which denotes that the paper strip penetrates itself.)

For a given paper strip P of length $n+1$, we consider a feasible folded state. Then, the *crease width* at crease i is defined by $|f(i) - f(i+1)| - 1$, which gives the number of paper layers between the ith segment and the $(i+1)$st segment joined at the crease i.

On the other hand, for a folded state, the *CW assignment* is the assignment of crease widths to the creases.

In this study, we introduce the following *crease-restore problem*. We are given partial information on the MV and CW assignments of the creases of a folded state of P. Then, the solution to the problem is a folded state of P that satisfies these assignments. Precisely, the input of the crease-restore problem is composed of two functions: $\text{As} : [1, n] \to \{M, V, *\}$, and

[1] They satisfy the parenthesis theorem.

$\mathrm{Cw} : [1, n] \to \{0, 1, \ldots, n-1, *\}$. (Note that we have $0 \le \mathrm{Cw}(i) \le n-1$ for any $1 \le i \le n$.) The problem asks if there exists a feasible folded state of P consistent with these two functions. Precisely, a folded state satisfies these two functions if and only if for each crease i with $1 \le i \le n$, (1) it is mountain-folded if $\mathrm{As}(i) = M$ or $\mathrm{As}(i) = *$, (2) it is valley-folded if $\mathrm{As}(i) = V$ or $\mathrm{As}(i) = *$, and (3) the crease width at i is equal to $\mathrm{Cw}(i)$ or $\mathrm{Cw}(i) = *$.

Subsequently, we propose a digital signature system. A *digital signature* is a mathematical or computational scheme for verifying the authenticity of digital messages or documents. The scheme typically consists of three algorithms. A *key generation* algorithm selects a *private key* uniformly at random from a set of possible private keys. The algorithm outputs the private key and a corresponding *public key*. A *signing algorithm* then produces a *signature* for given a message and a private key. Finally, a *signature verifying algorithm* accepts or rejects the message's claim to authenticity given the message, public key, and signature. See, e.g., [3] for further details.

3 Computational Complexity of Crease Restore Problem

In this part of the study, we consider a number of variants of the crease-restore problem. We first consider a few trivial cases:

Observation 1 ([5, Proposition 1]) *All instances of the crease-restore problem are yes instances when* $\mathrm{As}(i) \in \{M, V\}$ *and* $\mathrm{Cw}(i) = *$ *for every* i *in* $\{1, 2, \ldots, n\}$.

Proof. Intuitively, we can repeat "end folding" for each $i = 1, 2, \ldots, n$ following $\mathrm{As}(i)$.[2] □

Observation 2 *We can solve the crease-restore problem in linear time when* $\mathrm{Cw}(i) \in \{0, 1, \ldots, n-1\}$ *and* $\mathrm{As}(i) \in \{M, V\}$ *for every* i *in* $\{1, 2, \ldots, n\}$.

Proof. We first fix segment 1 of *height* 0, where the height indicates the order of each paper segment in $[0, 1]$ in the final folded state. (We denote the height of segment 1 by $h(1) = 0$.) Then, for each $i = 1, \ldots, n$, we can compute the height of the segment $i+1$ from the height of the segment i by adding or subtracting $\mathrm{Cw}(i)$. The addition or subtraction is determined by the parity of i and $\mathrm{As}(i)$. Precisely, (1) $h(i) = h(i-1) + \mathrm{Cw}(i) + 1$ if i is odd and $\mathrm{As}(i) = V$, (2) $h(i) = h(i-1) + \mathrm{Cw}(i) + 1$ if i is even and $\mathrm{As}(i) = M$, (3) $h(i) = h(i-1) - (\mathrm{Cw}(i) + 1)$ if i is odd and $\mathrm{As}(i) = M$, or (4) $h(i) = h(i-1) - (\mathrm{Cw}(i) + 1)$ if i is even and $\mathrm{As}(i) = V$. After computation of the heights, we check if the folded state is feasible, and if the heights have no gaps. The folded state has no gap if and only if there is an integer j with $j \le 0$ such that there exists exactly one paper segment of height j' for every $j' = j, j+1, \ldots, j+n$. This consecutiveness check of heights can be done in linear time in the same technique as in bucket sort. The feasibility can be confirmed through checks of the nest structure. It is discussed in [4, Sect. 3.2.3] in the context of recognition of valid linear orderings in 2D map folding. Using the technique in [4, Sect. 3.2.3], it can be confirmed in linear time. □

Now, we turn to the main theorem in this section.

Theorem 1 *The crease-retrieve problem is strongly NP-complete when* $\mathrm{Cw}(i) \in \{0, 1, \ldots, n-1, *\}$ *and* $\mathrm{As}(i) \in \{M, V\}$ *for every* i *in* $\{1, 2, \ldots, n\}$.

Proof. It is easy to see that the problem is in NP. We prove the hardness via a reduction from the following problem 3-Partition, which is known to be strongly NP-complete even if B is bounded from above by some polynomial in m [2].

3-Partition
Input: Positive integers $a_1, a_2, a_3, \ldots, a_{3m}$ such that $\sum_{j=1}^{3m} a_j = mB$ for some positive integer B and $B/4 < a_j < B/2$ for $1 \le j \le 3m$.
Question: Is there a partition of $\{1, 2, \ldots, 3m\}$ into m subsets $A_1, A_2, \ldots, A_m$ such that $\sum_{j \in A_k} a_j = B$ for $1 \le k \le m$?

To begin with, we describe a construction of a paper strip P for a given instance $a_1, \ldots, a_{3m}$ and B of 3-Partition. The basic idea is slightly similar to the one in [6].

The strip P consists of a *folder part* and $3m$ *gadget parts* (Figure 3). The folder part consists of creases in $[1, 2m+3]$, and each of the $3m$ gadget parts corresponds to a_j $(1 \le j \le 3m)$, which contains $4m + 28m^2 a_j$ consecutive points on the strip. That is, the total length of P is $2m + 3 + \sum_{j=1}^{3m}(4m + 28m^2 a_j) = 3 + 2m + 12m^2 + 28m^3 B$. In the folder part, creases i with $1 \le i \le 2m+3$ form a zig-zag pattern via the MV assignment $VMVM \cdots MV$, as shown in Figure 3. Precisely, $\mathrm{As}(i) = V$ for odd i, and $\mathrm{As}(i) = M$ for even i. For even i, we let $\mathrm{Cw}(i) = 0$; that is, we cannot have any paper layers in the folded state at this crease (assigned M). For $i = 1$ and $i = 2m+3$, we set $\mathrm{Cw}(i) = *$; that is, we can have any number of paper layers in the folded state at these creases. These two creases 1 and $2m+3$ are called *trash folders*, where we will put useless paper layers. For each i with $i = 3, 5, 7, \ldots, 2m+1$, we set $\mathrm{Cw}(i) = 14m^2 B + 6m$. We call these m creases "unit folders."

Now, we move to the gadget part (Figure 4). For each integer a_j, we let $b_j = 14m^2 a_j$. We first consider the

[2] See [1] for the definition of the end folding. In our context, we just repeat folding along the leftmost crease line.

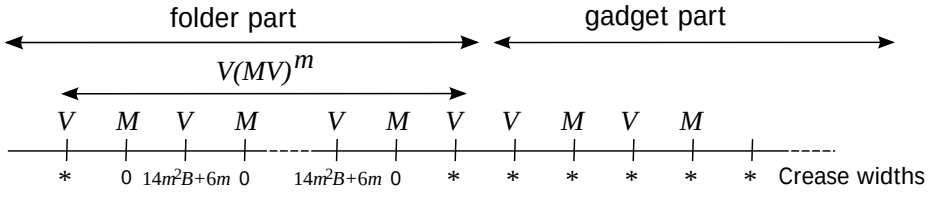


Figure 3: Construction of paper strip.

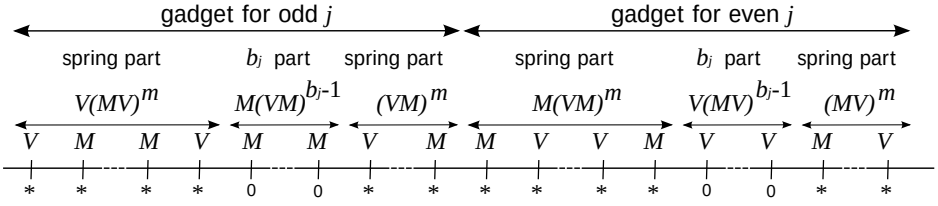


Figure 4: Construction of gadget part.

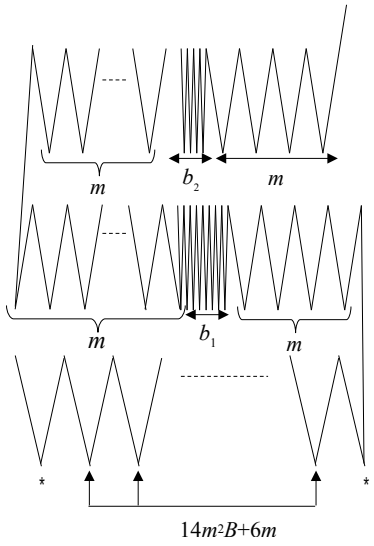


Figure 5: Overview of folding.

case that j is an odd number. Then, the jth gadget part consists of a zig-zag pattern of length $2m + b_j$ (which can be represented by $(VM)^{b_j+2m}$ in a standard representation of string). Let s_j be the first crease of the jth gadget part (which depends on $a_{j'}$ with all $j' < j$). Then, $\text{As}(i) = V$ for even $i = s_j + 2k$, and $\text{As}(i) = M$ for odd $i = s_j + 2k + 1$, with $0 \le k \le m + b_j/2$ (we note b_j is even). This zig-zag pattern contains three parts. We set their crease widths as follows: (1) $\text{Cw}(i) = *$ for $i = s_j + k$ for $0 \le k \le 2m$, (2) $\text{Cw}(i) = 0$ for $i = s_j + k$ for $2m < k < 2m + b_j$, and (3) $\text{Cw}(i) = *$ for $i = s_j + k$ for $2m + b_j \le k < 2m + b_j + 2m$. We call the first and third parts *spring parts* and the second part b_j *part*. Based on the requirement in (2), we cannot put any paper layers at the creases in the b_j part. Intuitively, this part can be considered as "glued," and this thickness of b_j should be put into some folder. On the other hand, each of the spring parts can be split in any way, and they can be put into any folders, including trash folders.

We next consider the case that j is an even number. The zig-zag pattern $(MV)^{b_j+2m}$ is obtained via flipping of the M and V used in the odd case. The crease widths are identical: (1) $\text{Cw}(i) = *$ for $i = s_j + k$ for $0 \le k \le 2m$, (2) $\text{Cw}(i) = 0$ for $i = s_j + k$ for $2m < k < 2m + 2b_j$, and (3) $\text{Cw}(i) = *$ for $i = s_j + k$ for $2m + 2b_j \le k < 2m + b_j + 2m$.

The construction of the paper strip P can be done in polynomial time. Therefore, it is sufficient to show that P can be folded into a unit length without penetration such that each crease i satisfies the condition for the crease width $\text{Cw}(i)$ if and only if the instance of 3-Partition is a yes instance.

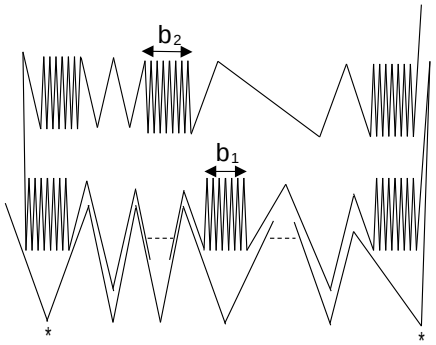


Figure 6: One feasible way of folding.

We first observe that most parts of P are in *pleat folding* $MVMV\cdots$ or $VMVM\cdots$. As shown in Figure 5, the folder part consists of m unit folders of crease width $14m^2B+6m$ between two trash folders, and each gadget corresponding to a_j consists of a "glued" part of width $2b_j$ between two springs of width $2m$.[3] Therefore, we consider putting gadget parts into unit folders to fill up each folder by exactly $14m^2B + 6m$ layers of paper.

We first assume that the instance of 3-Partition is a yes instance and show that P can be folded into unit length. Because the instance is a yes instance, the positive integers $a_1, a_2, a_3, \ldots, a_{3m}$ can be partitioned into m subsets $A_1, A_2, \ldots, A_m$ such that $\sum_{j \in A_k} a_j = B$ for $1 \le k \le m$. Then, we fill the unit folders as follows (Figure 6). We assume that a_1 is put into a subset $A_{k'}$ for some k'. Then, we put the b_1 gadget into the k'th unit folder, and two paper layers for each unit folder, as

[3]The *width* here refers to the number of layers.

shown in Figure 6. The other remaining segments in the two springs are put into trash folders on both sides. We can observe that these springs also act as unit folders after putting the b_1 gadget into $A_{k'}$. Therefore, we can repeat the same process for each $a_2, a_3, \ldots, a_{3m}$. Then, by the assumption with $b_j = 14m^2 a_j$, each unit folder $A_{k'}$ has $14m^2B + 6m$ paper layers at its corresponding crease. Thus, we obtain the required folded state of P.

Next, we assume that the paper strip P is folded, and we construct a solution for 3-Partition from it. We first observe that the total number of paper layers in the spring parts is $3m \cdot 4m = 12m^2$, which is much less than $14m^2$. Therefore, because each $b_j = 14m^2 a_j$ and $B/4 < a_j < B/2$, if a unit folder contains $14m^2B + 6m$ paper layers, it is easy to see that each unit folder contains exactly three b_j parts for some b_j, $b_{j'}$ and $b_{j''}$. Then, these parts together make $14m^2B$ paper layers because $6m$ is excessively small compared to each of b_j, $b_{j'}$, and $b_{j''}$. Therefore, we have $a_j + a_{j'} + a_{j''} = B$ for this unit. We can use the same argument for each unit folder, and we can construct a solution for 3-Partition, which completes the proof. □

In fact, if the proof of Theorem 1 is considered carefully, it can be inferred that the MV assignment in the proof is not necessary.

Corollary 2 *The crease-retrieve problem is strongly NP-complete when* $\textsc{Cw}(i) \in \{0, 1, \ldots, n-1, *\}$ *and* $\textsc{As}(i) = *$ *for every* i *in* $\{1, 2, \ldots, n\}$.

Proof. The reduction is identical to one given in the proof of Theorem 1, but we provide no MV assignment to P. When the instance of 3-Partition is a yes instance, we can use the same method as that used in the proof, and thus P can be folded into unit length in a way that satisfies the two functions. Therefore, we assume that the paper strip P is folded, and we construct a solution for 3-Partition from it.

We first focus on the folder part. We have $\textsc{Cw}(i) = 0$ for each even i, and $\textsc{Cw}(i)$ has the same value for each $i = 3, 5, 7, \ldots, 2m+1$. If we valley-fold at some even i, two consecutive unit folders have to have the same crease width, which is impossible. On the other hand, if we mountain-fold at some odd i, we cannot have $\textsc{Cw}(i-1) = \textsc{Cw}(i+1) = 0$. Therefore, the folder part should make a pleat folding.

Next, we focus on the gadget part for a_j. In this part, we have consecutive $b_j + 1$ creases i with $\textsc{Cw}(i) = 0$. For the same reason as for the folder part, we can observe that this part should make a pleat folding to satisfy the condition. Then, to satisfy the crease-width conditions in all unit folders, this part has to be put into some unit folder to contribute to its crease width by b_j.

Therefore, we can use the same argument as that applied in the proof of Theorem 1, and obtain the claim. □

4 Application to Digital Signature System

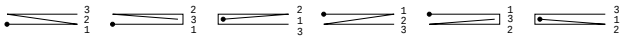


Figure 7: Six ways of folding a strip of length three.

In this section, we propose a digital signature system. The security of this system relies on the computational complexity of the crease-retrieve problem. We first observe the complexity of the stamp-folding problem in [5]. It is easy to see that one folded state can be represented by a permutation of $[1, n+1]$. For example, a strip of length three has $3! = 6$ ways of folding (Figure 7), which can be represented by [1|2|3], [1|3|2], [3|1|2], [3|2|1], [2|3|1], and [2|1|3]. However, when n is large, some of the permutations will cause penetrations. In general, the following is known.

Theorem 3 ([5]) *For a random MV assignment (of length* n*) for a paper strip of length* $n+1$*, the expected number of ways of folding is* $\Omega(1.53^n)$.

We note that $\Omega(1.53^n)$ is the theoretical lower bound; by contrast, it is $\Theta(1.65^n)$ experimentally.

Therefore, when we generate a random MV assignment of length n, there are $\Omega(1.53^n)$ permutations of $[1, n+1]$ corresponding to feasible folded states that satisfy the MV assignment. When we give a proper sequence of crease widths of length n, the folded state of the paper strip can be reconstructed in linear time by Observation 2. On the other hand, when a part of the sequence of crease widths is given, finding the folded state is NP-complete because its decision problem is NP-complete by Theorem 1. Based on the aforementioned observation, we can propose the following digital signature system with a public key cryptosystem.

As a preparation, every user first fixes a unique ID from a random MV assignment A of length n. This ID is a part of the public key.

4.1 Key generation algorithm G

From the MV assignment A, the algorithm G first generates a feasible folded state $F(A)$, which can be represented by a permutation p_n of $[1, n+1]$. An efficient algorithm that generates $F(A)$ from A can be obtained via modification of an algorithm in [6].

In [6], the authors show an algorithm for finding the folded state that achieves the minimum total crease width, which is defined by $\sum_{i=1}^{n} \textsc{Cw}(i)$, for a given MV assignment. The algorithm in [6] enumerates all feasible folded states for a given MV assignment, and it is proved that this algorithm shows that finding the minimum total crease width is fixed parameter tractable. That is, for a given MV assignment, the algorithm finds a feasible folded state in polynomial time if its minimum total

crease width is a constant. Therefore, we modify this algorithm and construct a feasible folded state $F(A)$ for an MV assignment A, as follows:
(0) We first initialize the folded state $F(A)$ by a segment $[0, 1]$.
(1) We then add the last line segment at the last crease i such that $\mathrm{As}(i) = R$, where R is M or V specified by the ith assignment in A.
(2) We put the last line segment in the interval $[0, 1]$. The height of the last segment is chosen at random from the feasible positions. Go to step (1) if the length of the paper strip is not exhausted.

Intuitively, we fold the last line segment according to A and put it in one of the feasible places at random in the current (partial) folded state. In the last step (2), we have to check the nest structure of the current folded paper strip to find the feasible positions. Using the technique in [4, Sect. 3.2.3], it can be done in linear time. Thus our algorithm for key generation runs in $O(n^2)$ time, where $n+1$ is the length of the paper strip P.

The folded state $F(A)$ of P can be represented by the corresponding permutation p_n. Because $n! \sim \sqrt{2\pi n}(n/e)^n$ by the Stirling Formula, p_n requires $O(n \log n)$ bits in a binary string.

Now, we turn to the generation of the public key. From the permutation p_n, we can generate the CW assignment $C(A) = (c_1, c_2, \ldots, c_n)$, where c_i is an integer in $[0, n-1]$. We then randomly replace some of these integers by $*$ and obtain a sequence $C^*(A) = (c_1^*, c_2^*, \ldots, c_n^*)$, where c_i^* is an integer in $[0, n-1]$ or a symbol $*$.[4]

Then, we make a pair $(A, C^*(A))$ the public key of this user. We note that A is a binary number of n bits that is fixed for each user and that $C^*(A)$ will be used once and then thrown away. It is easy to see that $C^*(A)$ can be encoded by a binary string of length $O(n \log n)$. (Because the number of ways of folding is $\Theta(3.3^n)$, which is much less than $n!$, we can theoretically reduce it to $O(n)$ bits.) We will use the CW assignment $C(A)$ as the signature key.

That is, for an MV assignment A, the corresponding public key is $(A, C^*(A))$, where $C^*(A)$ is partial information about the CW assignment $C(A)$ of a folded state $F(A)$ for A. By Theorem 1, reconstruction of the folded state $F(A)$ (and thus $C(A)$) from $(A, C^*(A))$ is strongly NP-complete in general.

4.2 Signature protocol

We suppose Alice is sending a message T to Bob. Let $(A, C^*(A))$ be the public key of Alice, which Bob knows. Alice first gives notice of sending a message to Bob, and then updates $C(A)$ by $C'(A)$ (to prevent spoofing by Bob). Then, Alice sends the message $(T, C(A))$. Bob can confirm the reliability of the message T by checking $C^*(A)$, which is partial information about $C(A)$ because it is NP-complete to restore $C(A)$ from $C^*(A)$ by Theorem 1. Once Bob has received and confirmed the message T, the $C(A)$ is discarded.

4.3 Discussions

For a given random MV assignment A, the expected number of folded states $F(A)$ (and thus $C(A)$) is $\Omega(1.53^n)$. Therefore, each user has exponentially many candidates for $C(A)$. We also note that no pair of distinct MV assignments A and B produces the same folded state $F(A) = F(B)$; the same is true for CW assignments. Therefore, we never have $C(A) = C(B)$ unless they share the same ID.

By Corollary 2, we can use the same system even if we remove A from the public key $(A, C^*(A))$. In this case, the public key is just $C^*(A)$, and only Bob can know that Alice is the person who has the public key $C^*(A)$, which is made from $C(A)$. This system can be used for some kinds of anonymous communication.

5 Concluding Remarks

In this study, we introduce the crease-restore problem and investigate its computational complexity. As investigated in [5], an MV assignment is not sufficient for determining the folded state of a strip of a paper. On the other hand, an MV assignment and a CW assignment are sufficient for determining the folded state. When we provide partial information on the CW assignment, the decision problem is NP-complete, whether we provide a full MV assignment or provide no MV assignment. One interesting open question is whether we can determine the folded state of a strip of paper when only a (full) CW assignment is given.

From the viewpoint of the proposed digital signature system, some specific MV assignment A has a few folded states, although there exists at least one folded state $F(A)$. It is known that A is a pleat folding (i.e., $MVMV\cdots$ or $VMVMV\cdots$) if and only if A has only one folded state. The characterization of the number of folded states for a given MV assignment remains open in the context of the stamp-folding problem.

In our framework, for a given CW assignment $C(A) = (c_1, c_2, \ldots, c_n)$, which is a secret key, the method for generating the public key $C^*(A) = (c_1^*, c_2^*, \ldots, c_n^*)$ is another problem that needs to be resolved. If we mask a few numbers in $C(A)$, it can be restored from $C^*(A)$ by brute force. On the other hand, when we mask too many numbers in $C(A)$, we may have some risk that $C^*(A) = C^*(A')$ for different MV assignments A and A'.

[4] This random part is crucial for the security in this system. The details are discussed in Concluding Remarks.

Finding a reasonable method (based on experiments) for masking $C(A)$ will be pursued in future research.

Acknowledgement

A part of this research is supported by JSPS KAKENHI Grant Numbers JP18H04091, JP20H05961, JP20H05964, and JP20K11673.

References

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

[2] M. R. Garey and D. S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[3] J. Katz. *Digital Signatures*. Springer, 2010.

[4] R. I. Nishat. *Map Folding*. Master thesis, University of Victoria, Department of Computer Science, 2013.

[5] R. Uehara. *Origami*^{5}, chapter Stamp foldings with a given mountain-valley assignment, pages 585–597. CRC Press, 2011.

[6] T. Umesato, T. Saitoh, R. Uehara, H. Ito, and Y. Okamoto. The complexity of the stamp folding problem. *Theoretical Computer Science*, 497:13–19, 2013.

Quantitative Helly-type Theorems via Hypergraph Chains

Attila Jung*

Abstract

We propose a combinatorial framework to analyze quantitative Helly-type questions. Using this framework, we prove a Quantitative Fractional Helly Theorem with Fractional Helly Number $3d$ and a stability version of the Quantitative Helly Theorem of Bárány, Katchalski and Pach.

1 Introduction

Two directions in the study of Helly-type Theorems are *quantitative* and *abstract* questions. *Quantitative* results concern intersection patterns of convex sets in some specific space, originally $\mathbb{R}^d$, where instead of finding points in the intersection, one bounds the size, for example the volume or the diameter of the intersection. *Abstract* results, on the other hand, study more general structures, e.g. hypergraphs, with certain properties that capture some essential aspects of the behavior of convex sets. In this note, we connect the two.

First, consider the *Quantitative Volume Theorem.*

Theorem 1 *[Bárány, Katchalski and Pach [3]] Assume that the intersection of any $2d$ members of a finite family of convex sets in $\mathbb{R}^d$ is of volume at least one. Then the volume of the intersection of all members of the family is of volume at least $c(d)$, a constant depending on d only.*

In [3], it is proved that one can take $c(d) = d^{-2d^2}$ and conjectured that it should hold with $c(d) = d^{-cd}$ for an absolute constant $c > 0$. Theorem 1 was confirmed with $c(d) \approx d^{-2d}$ by Naszódi [14], whose argument was refined by Brazitikos [4], who showed that one may take $c(d) \approx d^{-3d/2}$. For more on quantitative Helly-type results, see the surveys [6, 8].

Helly's theorem may be stated in the language of hypergraphs as follows. Let V be a finite family of convex sets in $\mathbb{R}^d$, and call a subset of V an edge of our hypergraph, if the intersection of the corresponding convex sets is not empty. Helly's theorem states that if all $(d+1)$-tuples of a subset S of V are edges of the hypergraph, then so is S. Observe that Theorem 1 cannot be translated to the same language, as two hypergaphs are involved: in one, the edges correspond to families of convex sets whose intersection is of volume at least one, and in the other, this volume is at least $c(d)$. The goal of this note is to provide a combinatorial framework in which Theorem 1, and other quantitative results can be translated.

The *Colorful Helly Theorem* found by Lovász [12] (and with the first published proof by Bárány [2]) states the following. *If $\mathcal{C}_1, \dots, \mathcal{C}_{d+1}$ are finite families (color classes) of convex sets in $\mathbb{R}^d$, such that for any colorful selection $C_1 \in \mathcal{C}_1, \dots, C_{d+1} \in \mathcal{C}_{d+1}$, the intersection $\bigcap_{i=1}^{d+1} C_i$ is non-empty, then for some j, the intersection $\bigcap_{C \in \mathcal{C}_j} C$ is also non-empty.*

In [5], the following quantitative variant is shown.

Theorem 2 *[Damásdi, Földvári and Naszódi [5]] Let $\mathcal{C}_1, \dots, \mathcal{C}_{3d}$ be finite families of convex sets in $\mathbb{R}^d$. Assume that for any colorful selection $C_1 \in \mathcal{C}_1, \dots, C_{3d} \in \mathcal{C}_{3d}$, the intersection $\bigcap_{i=1}^{3d} C_i$ is of volume at least one. Then, there is a j with $1 \le j \le 3d$ such that $\operatorname{vol}\left(\bigcap_{C \in \mathcal{C}_j} C\right) \ge d^{-cd^2}$ with a universal constant $c > 0$.*

The *Fractional Helly Theorem* due to Katchalski and Liu [11] (see also [13, Chapter 8]) is another classical Helly-type result, which states the following. *Fix a dimension d, and an $\alpha \in (0,1)$, and let $\mathcal{C}$ be a finite family of convex sets in $\mathbb{R}^d$ with the property that among the subfamilies of $\mathcal{C}$ of size $d+1$, there are at least $\alpha\binom{|\mathcal{C}|}{d+1}$ for whom the intersection of the $d+1$ members is nonempty. Then, there is a subfamily $\mathcal{C}' \subset \mathcal{C}$ of size $|\mathcal{C}'| \ge \frac{\alpha}{d+1}|\mathcal{C}|$ such that the intersection of all members of $\mathcal{C}'$ is nonempty.*

In [10], the following quantitative variant of the Fractional Helly Theorem is shown.

Theorem 3 *[Jung and Naszódi [10]] For every dimension $d \ge 1$ and every $\alpha \in (0,1)$, there is a $\beta \in (0,1)$ such that the following holds.*
Let $\mathcal{C}$ be a finite family of convex sets in $\mathbb{R}^d$. Assume that among all subfamilies of size $3d+1$, there are at least $\alpha\binom{|\mathcal{C}|}{3d+1}$ for whom the intersection of the $3d+1$ members is of volume at least one.
Then, there is a subfamily $\mathcal{C}' \subset \mathcal{C}$ of size at least $\beta|\mathcal{C}|$ such that $\operatorname{vol}\left(\bigcap_{C \in \mathcal{C}'} C\right) \ge d^{-cd^2}$ with a universal constant $c > 0$.

*Institute of Mathematics, Loránd Eötvös University, jungattila@gmail.com

In Theorems 1, 2 and 3, the cardinalities $2d$, $3d$ and $3d+1$ appear, respectively. It is easy to verify (cf. [3]) that Theorem 1 does not hold with any number below $2d$, which implies the same lower bound for the other two theorems. No better lower bounds are known.

Turning to abstract results, we describe Helly's Theorem and the Fractional and Colorful Helly Theorems in the language of hypergraphs. Let V be a (possibly infinite) set. A *hypergraph* on the base set V is any family of its subsets, $\mathcal{H} \subset 2^V$. A hypergraph is downwards closed, if $H \in \mathcal{H}$ and $G \subset H$ implies $G \in \mathcal{H}$. A downwards closed hypergraph $\mathcal{H}$ has *Helly Number* h, if for every finite subset $S \subset V$ the relation $\binom{S}{h} \subset \mathcal{H}$ implies $S \in \mathcal{H}$. Now let us denote the family of convex sets of $\mathbb{R}^d$ as $\mathrm{Cvx}(d)$ and the hypergraph which contains the subfamilies of convex sets with nonempty intersection by $\mathcal{K}_d = \{\mathcal{C} \subset \mathrm{Cvx}(d) : \cap_{C\in\mathcal{C}} C \neq \emptyset\}$. Helly's Theorem says that $\mathcal{K}_d$ has Helly-number $d+1$.

A downwards closed hypergraph $\mathcal{H}$ over a base set V has *Fractional Helly Number* k, if there exists a function $\beta : (0,1) \to (0,1)$ such that whenever $S \subset V$ is a finite subset such that $\left|\mathcal{H} \cap \binom{S}{k}\right|$, the number of edges of $\mathcal{H}$ of size k in S is at least $\alpha\binom{|S|}{k}$ with an $\alpha \in (0,1)$, then there exists a subset $S' \subset S$ of size at least $\beta|S|$ such that $S' \in \mathcal{H}$. The Fractional Helly Theorem says, that $\mathcal{K}_d$ has Fractional Helly Number $d+1$.

We turn to phrasing the Colorful Helly Theorem in an abstract setting. Let $S_1, \ldots, S_k \subset V$ be (not necessarily disjoint) subsets of a base set V, which we will call color classes. We call a set $F \subset V$ a colorful selection from these color classes, if F contains one element from each color class. Very formally, to clarify how elements belonging to multiple color classes are handled, we say that $F \subset V$ is a *colorful selection*, if there is a surjective map $\phi : [k] \longrightarrow F$ with $\phi(i) \in S_i$ for all $i \in [k]$. We denote the set of colorful selections by $S_1 \otimes \ldots \otimes S_k$.

A downwards closed hypergraph $\mathcal{H}$ over a base set V has *Colorful Helly Number* k, if for every k finite subset $S_1, \ldots, S_k \subset V$ such that $(S_1 \otimes \ldots \otimes S_k) \subseteq \mathcal{H}$, there exists a color class S_j with $S_j \in \mathcal{H}$. The Colorful Helly Theorem says that $\mathcal{K}_d$ has Colorful Helly Number $d+1$.

Alon, Kalai, Matoušek and Meshulam [1] considered Helly-type results in the abstract setting. They showed, that if a hypergraph has bounded Fractional Helly Number, then it also has the so called (p,q) property (see the definition in [1]). Holmsen [9] showed that if a hypergraph has Colorful Helly Number k, then it has Fractional Helly Number at most k. In this sense, the Fractional Helly Theorem can be deduced from the Colorful Helly Theorem with a purely combinatorial proof. Note that Holmsen's result does not immediately imply a similar relationship between Theorem 2 and Theorem 3, because there are two different kinds of intersection of convex sets (sets intersecting in volume one and sets intersecting in volume d^{-cd^2}).

We introduce the notion of hypergraph chains, and our first main result, Theorem 8 states that Holmsen's result extends to hypergraph chains, and, as a result, it can be applied in the context of quantitative Helly-type questions. It follows (see Corollary 12) that in Theorem 3 we can decrease the number $3d+1$ to $3d$ (at the expense of a bigger loss of volume). Our second main result, Theorem 9 states that Theorem 1 is stable: one does not need to check that all $2d$-tuples of the given convex sets have intersection of volume at least one. Instead, it is sufficient to verify it for almost all of them to obtain that almost all have an intersection of some positive volume.

Quantitative Helly-type theorems are considered in [7, 15, 16] with the focus on convex sets in $\mathbb{R}^d$, or the lattice $\mathbb{Z}^d$, or sets in topological spaces with particular topological properties. To our knowledge, ours is the first attempt to address quantitative Helly-type questions in the general context of hypergraphs in the spirit of the results of [1] and [9].

2 Hypergraph Chains

Definition 4 *Let V be a (possibly infinite) set. The infinite sequence $(\mathcal{H}_\ell)_{\ell\in\mathbb{Z}}$ of hypergraphs over the base set V is a* hypergraph chain*, if every $\mathcal{H}_\ell$ is downwards closed and for all $\ell \in \mathbb{Z}$, $\mathcal{H}_\ell \subset \mathcal{H}_{\ell+1}$.*

If $V = \mathrm{Cvx}(d)$ and $\mathcal{H}_\ell = \mathcal{K}_d$ for all ℓ, then $(\mathcal{H}_\ell)_{\ell\in\mathbb{Z}}$ is a hypergraph chain. A more interesting example is when $V = \mathrm{Cvx}(d)$, $v \in (0,1)$ a real number and for an $\ell \in \mathbb{Z}$, a family of convex sets from $\mathbb{R}^d$ is an edge in $\mathcal{H}_\ell$, if and only if their intersection is of volume at least v^ℓ. We will denote this hypergraph by $\mathcal{Q}_d\left(v^\ell\right)$.

Definition 5 *A hypergraph chain $(\mathcal{H}_\ell)_{\ell\in\mathbb{Z}}$ over a base set V has* Helly Number h*, if for every $S \subseteq V$, $\binom{S}{h} \subset \mathcal{H}_\ell$ implies $S \in \mathcal{H}_{\ell+1}$.*

According to this definition, $(\mathcal{K}_d)_{\ell\in\mathbb{Z}}$ has Helly Number $d+1$.

More interestingly, Theorem 1 states that if $v \approx d^{-3d/2}$, then $\left(\mathcal{Q}_d\left(v^\ell\right)\right)_{\ell\in\mathbb{Z}}$ has Helly Number $2d$.

Definition 6 *A hypergraph chain $(\mathcal{H}_\ell)_{\ell\in\mathbb{Z}}$ over a base set V has* Colorful Helly Number k*, if whenever $S_1, \ldots, S_k$ are finite subsets (color classes) of V and $S_1 \otimes \ldots \otimes S_k \subset \mathcal{H}_\ell$, then there is a color class S_j with $S_j \in \mathcal{H}_{\ell+1}$.*

Note that by taking $S_1 = S_2 = ... = S_k = S$, a hypergraph chain with Colorful Helly Number k has Helly Number $h \leq k$.

According to the definition, $(\mathcal{K}_d)_{\ell\in\mathbb{Z}}$ has Colorful Helly Number $d+1$.

More interestingly, the Quantitative Colorful Helly Theorem, Theorem 2 may be stated as follows. If $v =$

d^{-cd^2} from Theorem 2, then $\left(\mathcal{Q}_d\left(v^\ell\right)\right)_{\ell\in\mathbb{Z}}$ has Colorful Helly Number $3d$.

Definition 7 *A hypergraph chain $(\mathcal{H}_\ell)_{\ell\in\mathbb{Z}}$ over a base set V has* Fractional Helly Number k*, if there exists a function $\beta:(0,1)\to(0,1)$ such that for every finite set $S\subset V$, if $|\mathcal{H}_\ell\cap\binom{S}{k}|\geq\alpha\binom{|S|}{k}$ with some $\alpha\in(0,1)$, then there exists an $S'\subset S$ with $|S'|\geq\beta(\alpha)|S|$ and $S'\in\mathcal{H}_{\ell+1}$.*

As in the previous two cases, $(\mathcal{K}_d)_{\ell\in\mathbb{Z}}$ has Fractional Helly Number $d+1$ and Theorem 3 states, that if $v=d^{-cd^2}$ from Theorem 3, then $\left(\mathcal{Q}_d\left(v^\ell\right)\right)_{\ell\in\mathbb{Z}}$ has Fractional Helly Number $3d+1$.

Now we are ready to state our main result, which is a quantitative analogue of Theorem 3 from [9].

Theorem 8 *If the hypergraph chain $(\mathcal{H}_\ell)_{\ell\in\mathbb{Z}}$ has Colorful Helly Number k, then $(\mathcal{H}_{(k+1)\ell})_{\ell\in\mathbb{Z}}$ has Fractional Helly Number k.*

Here, the obtained Fractional Helly Number is the same as the assumed Colorful Helly Number, but not for the exact same hypergraph chain: we can only take every $(k+1)$th element from the original chain. Can the Fractional Helly number go below the Colorful Helly number? If for a hypergraph chain the Helly Number is smaller than the Colorful Helly Number, the answer is a partial yes.

Theorem 9 *If the hypergraph chain $(\mathcal{H}_\ell)_{\ell\in\mathbb{Z}}$ has Helly Number h and Colorful Helly Number $k\geq h$, then there exists a function $\beta:(0,1)\to[0,1)$ with $\lim_{\alpha\to1}\beta(\alpha)=1$ such that for every finite set $S\subset V$, if $|\mathcal{H}_\ell\cap\binom{S}{h}|\geq\alpha\binom{|S|}{h}$ with some $\alpha\in(0,1)$, then there exists an $S'\subset S$ with $|S'|\geq\beta(\alpha)|S|$ and $S'\in\mathcal{H}_{\ell+3}$.*

We can interpret this result as a stability version of the Helly property (under some additional assumptions), since $\lim_{\alpha\to1}\beta(\alpha)=1$.

As far as we know, the best possible β here might assign 0 to a large fraction of αs from $(0,1)$, this is the difference from hypergraph chains with Fractional Helly Number h, where this is not possible. But at least, if α is very close to 1, then $\beta(\alpha)$ is also close to 1.

3 Proof of Theorems 8 and 9

Let us begin with an analogue of Lemma 3.1 from [9]. We denote by $\omega_h(\mathcal{H}_\ell|_S)$ the size of the largest h-clique of S, ie. the size of the largest subset $K\subset S$ such that $\binom{K}{h}\subset\mathcal{H}_\ell$.

Lemma 10 *Let $(\mathcal{H}_\ell)_{\ell\in\mathbb{Z}}$ be a hypergraph chain with Helly Number h and Colorful Helly Number k over a base set V. Then for every finite subset $S\subset V$, we have*

(a) $\left|\binom{S}{k}\setminus\mathcal{H}_\ell\right|\geq\binom{\frac{1}{k}(|S|-\omega_k(\mathcal{H}_{\ell+1}|_S))}{k}$*, and*

(b) $\left|\binom{S}{h}\setminus\mathcal{H}_\ell\right|\geq\binom{k}{h}^{-1}\binom{\frac{1}{h}(|S|-\omega_h(\mathcal{H}_{\ell+2}|_S))}{h}$.

Proof. Note that $h\leq k$ holds for every hypergraph chain of Helly Number h and Colorful Helly Number k. Fix $\ell\in\mathbb{Z}$.

For the proof of part (a), let $\{M_1,\dots,M_t\}\subset\binom{S}{k}\setminus\mathcal{H}_{\ell+1}$ be a maximal size family of disjoint missing edges from $\mathcal{H}_{\ell+1}$, each of size k. By the maximality of this family, we have $\binom{S\setminus(M_1\cup\ldots\cup M_t)}{k}\subset\mathcal{H}_{\ell+1}$, and thus, $\omega_k(\mathcal{H}_{\ell+1}|_S)\geq|S\setminus(M_1\cup\ldots\cup M_t)|=|S|-tk$ or, equivalently,

$$t\geq\frac{1}{k}((|S|-\omega_k(\mathcal{H}_{\ell+1}|_S))). \tag{1}$$

Consider a selection $I\in\binom{[t]}{k}$ of k indices. Since each M_i is a missing edge from $\mathcal{H}_{\ell+1}$, we have that $\{M_i\,:\,i\in I\}$ is a family of k color classes, such that neither one is contained in $\mathcal{H}_{\ell+1}$. Since $(\mathcal{H}_\ell)_{\ell\in\mathbb{Z}}$ has Colorful Helly Number k, there is a colorful selection $\{v_i\,:\,i\in I\}\subset V$ of vertices (that is, $v_i\in M_i$ for all $i\in I$) such that $\{v_i\,:\,i\in I\}$ is not an edge in $\mathcal{H}_\ell$.

Observe that if $I_1,I_2\in\binom{[t]}{k}$ are distinct selections of indices, then, by the disjointness of the M_j, we have that $\{v_i\,:\,i\in I_1\}\neq\{v_i\,:\,i\in I_2\}$. Thus, we found $\binom{t}{k}$ members of $\binom{S}{k}\setminus\mathcal{H}_\ell$, completing the proof of part (a).

For the proof of part (b), let $\{M_1,\dots,M_t\}\subset\binom{S}{h}\setminus\mathcal{H}_{\ell+2}$ be a maximal size family of disjoint missing edges from $\mathcal{H}_{\ell+2}$, each of size h. Similarly to the argument in part (a), we have

$$t\geq\frac{1}{h}((|S|-\omega_h(\mathcal{H}_{\ell+2}|_S))). \tag{2}$$

Consider a selection $I\in\binom{[t]}{k}$ of k indices. Again, as in the proof of part (a), since $(\mathcal{H}_\ell)_{\ell\in\mathbb{Z}}$ has Colorful Helly Number k, there is a colorful selection $\{v_i\,:\,i\in I\}\subset V$ of vertices from the color classes $\{M_i\,:\,i\in I\}$ such that $\{v_i\,:\,i\in I\}$ is not an edge in $\mathcal{H}_{\ell+1}$. By the Helly property, there is a $J\in\binom{I}{h}$ and an $F\in\binom{S}{h}\setminus\mathcal{H}_\ell$ such that $|F\cap M_j|=1$ for every $j\in J$. Any fixed $J\in\binom{[t]}{h}$ can appear at most $\binom{t-h}{k-h}$ times in this way. Moreover, any fixed $F\in\binom{S}{h}\setminus\mathcal{H}_\ell$ may appear for only one J, so there are at least $\binom{t}{k}/\binom{t-h}{k-h}=\binom{t}{h}/\binom{k}{h}$ missing edges $F\in\binom{S}{h}\setminus\mathcal{H}_\ell$, which combined with (2) completes the proof of part (b) of Lemma 10. □

Proof. [of Theorem 9] Fix $\ell\in\mathbb{Z}$ and assume that the largest edge of $\mathcal{H}_{\ell+3}$ in S is of size at most $(1-\varepsilon)|S|$ for some $\varepsilon>0$. Since $(\mathcal{H}_\ell)_{\ell\in\mathbb{Z}}$ has Helly Number h, this implies $\omega_h(\mathcal{H}_{\ell+2}|_S)\leq(1-\varepsilon)|S|$. Part (b) of Lemma 10 yields $\left|\binom{S}{h}\setminus\mathcal{H}_\ell\right|\geq\binom{k}{h}^{-1}\binom{\varepsilon|S|/h}{h}\geq\delta\cdot\binom{|S|}{h}$ with some $\delta=\delta(\varepsilon,k,h)>0$. Thus, if $\beta(\alpha)\leq1-\varepsilon$, then $\alpha\leq1-\delta$, proving Theorem 9. □

In order to prove Theorem 8, we need the following technical lemma, which is an analogue of Lemma 3.2 from [9] and can be proved using part (a) of Lemma 10.

Lemma 11 *Let $(\mathcal{H}_\ell)_{\ell\in\mathbb{Z}}$ be a hypergraph chain over a base set V with Colorful Helly Number k. Let $S \subset V$ be a finite subset with $|S| = n$ large enough. If for a $t \in \mathbb{Z}$ and $c \in (0,1)$ the inequality $\omega_k(\mathcal{H}_{t+1}|_S) \leq cn/2$ holds, then given any $i \in [k]$ and a family $\mathcal{F}_i \subset \binom{S}{i}$ with $|\mathcal{F}_i| \geq c\binom{n}{i}$ there exists another family $\mathcal{F}_{i-1} \subset \binom{S}{i-1}$ and an $M \in \binom{S}{k} \setminus \mathcal{H}_t$ such that $|\mathcal{F}_{i-1}| \geq \left(\frac{c}{12k^2}\right)^k \binom{n}{n-1}$ and $A \cup \{v\} \in \mathcal{F}_i$ for all $A \in \mathcal{F}_{i-1}$ and $v \in M$.*

Proof. For every $A \in \binom{S}{i-1}$ let $\Gamma_A = \{v \in S : (A \cup \{v\}) \in \mathcal{F}_i\}$ and let

$$\mathcal{P} = \left\{(A, M) : A \in \binom{S}{i-1}, M \in \binom{\Gamma_A}{k} \setminus \mathcal{H}_t\right\}.$$

We want to lower bound $|\mathcal{P}|$. By part (a) of Lemma 10, for a fixed $A \in \binom{S}{i-1}$ there are at least $\binom{\frac{1}{k}(|\Gamma_A| - (c/2)n)}{k}$ distinct $M \in \binom{\Gamma_A}{k} \setminus \mathcal{H}_t$ such that $(A, M) \in \mathcal{P}$. Jensen's inequality gives

$$\begin{aligned}|\mathcal{P}| &\geq \sum_{A\in\binom{S}{i-1}} \binom{\frac{1}{k}\left(|\Gamma_A| - (c/2)n\right)}{k} \\ &\geq \binom{n}{i-1}\binom{\binom{n}{i-1}^{-1}\frac{1}{k}\sum_{A\in\binom{S}{i-1}}\left(|\Gamma_A| - (c/2)n\right)}{k}.\end{aligned}$$

Since

$$\sum_{A\in\binom{S}{i-1}} |\Gamma_A| = i|\mathcal{F}_i| \geq ic\binom{n}{i} > (n-i)c\binom{n}{i-1},$$

we get

$$\sum_{A\in\binom{S}{i-1}} (|\Gamma_A| - (c/2)n) > (n-i)c\binom{n}{i-1} - (c/2)n\binom{n}{i-1},$$

and thus

$$|\mathcal{P}| \geq \binom{n}{i-1}\binom{\frac{nc}{2k} - \frac{ci}{k}}{k}.$$

If n is large enough compared to i and k, then

$$|\mathcal{P}| \geq \left(\frac{c}{12k^2}\right)^k \binom{n}{i-1}\binom{n}{k}.$$

Since there are $\binom{n}{k}$ possible $M \in \binom{S}{k}$, there is an M with at least $\left(\frac{c}{12k^2}\right)^k \binom{n}{i-1}$ different $A \in \binom{S}{i-1}$ such that $(A, M) \in \mathcal{P}$. These A will form $\mathcal{F}_{i-1}$. □

Proof. [of Theorem 8] We are given $\alpha \in (0,1)$, and our goal is to find the corresponding $\beta \in (0,1)$ satisfying Definition 7. Let $f(x) = \left(\frac{x}{12k^2}\right)^k$, $\alpha_0 = \alpha$, $\alpha_{i+1} = f(\alpha_i)$. We will show that $\beta = \alpha_{k-1}$ is a good choice. Fix $\ell \in \mathbb{Z}$ and suppose for a contradiction that $\left|\mathcal{H}_\ell \cap \binom{S}{k}\right| \geq \alpha\binom{n}{k}$, but $\mathcal{H}_{\ell+k+1}$ has no edge of size at least βn inside S. Since $(\mathcal{H}_\ell)_{\ell\in\mathbb{Z}}$ has Colorful Helly Number k, it has Helly Number at most k, so $\mathcal{H}_{\ell+k+1}$ having no edge of size at least βn implies $\omega_k(\mathcal{H}_{\ell+k}|_S) < \beta n$.

Set $\mathcal{F}_k = \mathcal{H}_\ell \cap \binom{S}{k}$. Since $(\mathcal{H}_\ell)_{\ell\in\mathbb{Z}}$ is a hypergraph chain, $\mathcal{F}_k \subset \mathcal{H}_{\ell+i}$ for all $i \geq 0$, in particular, $\mathcal{F}_k \subset \mathcal{H}_{\ell+k}$. We have $|\mathcal{F}_k| \geq \alpha\binom{n}{k}$ and $\omega_k(\mathcal{H}_{\ell+k}|_S) < \beta n \leq (\alpha/2)n$, so we can apply Lemma 11 with $t = \ell+k-1$ and $c = \alpha$ to obtain an $\mathcal{F}_{k-1} \subset \binom{S}{k-1}$ with $|\mathcal{F}_{k-1}| \geq \alpha_1\binom{n}{k-1}$ and an $M_1 \in \binom{S}{k} \setminus \mathcal{H}_{\ell+k-1}$ such that $A \cup \{v\} \in \mathcal{F}_k$ for all $A \in \mathcal{F}_{k-1}$ and $v \in M_1$. Now, we have $|\mathcal{F}_{k-1}| \geq \alpha_1\binom{n}{k-1}$ and $\omega_k(\mathcal{H}_{\ell+k-1}|_S) \leq \omega_k(\mathcal{H}_{\ell+k}|_S) < \beta n \leq (\alpha_1/2)n$ and we can apply Lemma 11 again, this time with $t = \ell + k - 2$ and $c = \alpha_1$, to obtain an $\mathcal{F}_{k-2} \subset \binom{S}{k-2}$ with $|\mathcal{F}_{k-2}| \geq \alpha_2\binom{n}{k-2}$ and an $M_2 \in \binom{S}{k} \setminus \mathcal{H}_{\ell+k-2}$ such that $(A \cup \{v\}) \in \mathcal{F}_{k-1}$ for all $A \in \mathcal{F}_{k-2}$ and $v \in M_2$. Note that $(A \cup \{v_1, v_2\}) \in \mathcal{F}_k = \mathcal{H}_\ell \cap \binom{S}{k}$ for all $A \in \mathcal{F}_{k-2}$, $v_1 \in M_1$, $v_2 \in M_2$.

After repeating this process $k-1$ times, we obtain an $\mathcal{F}_1 \subset \binom{S}{1}$ with $|\mathcal{F}_1| \geq \alpha_{k-1}n = \beta n$ and $M_1, \ldots, M_{k-1} \in \binom{S}{k} \setminus \mathcal{H}_{\ell+1}$ such that $A \cup \{v_1, \ldots, v_{k-1}\} \in \mathcal{H}_\ell \cap \binom{S}{k}$ for all $A \in \mathcal{F}_1$, $v_1 \in M_1, \ldots, v_{k-1} \in M_{k-1}$. Since $\omega_k(\mathcal{H}_{\ell+1}|_S) < \beta n$, there must be an $M_k \in \binom{V(\mathcal{F}_1)}{k} \setminus \mathcal{H}_{\ell+1}$. But regarding $M_1, \ldots, M_k$ as color classes, $(\mathcal{H}_\ell)_{\ell\in\mathbb{Z}}$ having Colorful Helly-number k yields a contradiction, since $M_1 \otimes \ldots \otimes M_k \subset \mathcal{H}_\ell$, but there is no color class $M_i \in \mathcal{H}_{\ell+1}$. □

4 Consequences for Quantitative Theorems

If $v = d^{-cd^2}$ from Theorem 2, then $\left(\mathcal{Q}_d\left(v^\ell\right)\right)_{\ell\in\mathbb{Z}}$ has Colorful Helly Number $3d$ by Theorem 2, so the following Corollary follows from Theorem 8.

Corollary 12 *For every dimension $d \geq 1$ and every $\alpha \in (0,1)$, there is a $\beta \in (0,1)$ such that the following holds.*
Let $\mathcal{C}$ be a finite family of convex sets in $\mathbb{R}^d$. Assume that among all subfamilies of size $3d$, there are at least $\alpha\binom{|\mathcal{C}|}{3d}$ for whom the intersection of the $3d$ members is of volume at least one.
Then, there is a subfamily $\mathcal{C}' \subset \mathcal{C}$ of size at least $\beta|\mathcal{C}|$ such that $\operatorname{vol}\left(\bigcap_{C\in\mathcal{C}'} C\right) \geq d^{-cd^3}$ *with a universal constant $c > 0$.*

Proof. The above claim is equivalent to saying that $\left(\mathcal{Q}_d\left(v^\ell\right)\right)_{\ell\in\mathbb{Z}}$ has Fractional Helly Number $3d$, if $v = d^{-c'd^3}$ with a universal constant c'. Theorem 2 states that $\left(\mathcal{Q}_d\left(v^\ell\right)\right)_{\ell\in\mathbb{Z}}$ has Colorful Helly Number $3d$, if $v = d^{-cd^2}$ as in Theorem 2. By applying Theorem 8 to the latter Hypergraph Chain, we can conclude, that

$\left(\mathbb{Q}_d\left(v^{(3d+1)\ell}\right)\right)_{\ell\in\mathbb{Z}}$ has Fractional Helly Number $3d$ and $v = d^{-cd^2}$. But this is equvivalent to $\left(\mathcal{Q}_d\left(v^{\ell}\right)\right)_{\ell\in\mathbb{Z}}$ having Fractional Helly Number $3d$ if $v = d^{-c'd^3}$. □

This is a slight improvement on the Fractional Helly Number, which was $3d+1$ in Theorem 3. Can we go below $3d$? Theorem 9 implies at least a stability version of the Quantitative Helly Theorem with Helly Number $2d$ as follows.

Corollary 13 *For every positive integer d there exists a function $\beta : (0,1) \to [0,1)$ with $\lim_{\alpha\to 1}\beta(\alpha) = 1$ such that the following holds.*
Let $\mathcal{C}$ be a finite family of convex sets in $\mathbb{R}^d$. Assume that among all subfamilies of size $2d$, there are at least $\alpha\binom{|\mathcal{C}|}{2d}$ for whom the intersection of the $2d$ members is of volume at least one.
Then, there is a subfamily $\mathcal{C}' \subset \mathcal{C}$ of size at least $\beta|\mathcal{C}|$ such that $\operatorname{vol}\left(\bigcap_{C\in\mathcal{C}'} C\right) \geq d^{-cd^2}$ with a universal constant $c > 0$.

Proof. Since $\left(\mathcal{Q}_d\left(v^{\ell}\right)\right)_{\ell\in\mathbb{Z}}$, with $v = d^{-cd^2}$ from Theorem 2, has Helly Number $2d$ by Theorem 1 and Colorful Helly Number $3d$ by Theorem 2, we can apply Theorem 9. The assumption of Corollary 13 states that for a finite subset of convex sets $\mathcal{C}$, the inequality $\left|\mathcal{Q}_d(v^0)\cap\binom{\mathcal{C}}{2d}\right| \geq \alpha\binom{|\mathcal{C}|}{2d}$ holds with some $\alpha\in(0,1)$, where v can be $v = d^{-cd^2}$ from Theorem 2. Theorem 9 yields a subfamily $\mathcal{C}' \subset \mathcal{C}$ with $\mathcal{C}' \in \mathcal{Q}_d(v^3)$ and $|\mathcal{C}'| \geq \beta(\alpha)|\mathcal{C}|$, where β is the function from Theorem 9. For $\mathcal{C}'$, the inequality $\operatorname{vol}\left(\bigcap_{C\in\mathcal{C}'} C\right) \geq \left(d^{-cd^2}\right)^3 = d^{-3cd^2}$ holds. □

5 Remarks

The following questions are left open.

Conjecture 1 *For every dimension d, there is a $v = v(d) \in (0,1)$, such that $\left(\mathcal{Q}_d\left(v^{\ell}\right)\right)_{\ell\in\mathbb{Z}}$ has Fractional Helly Number $2d$.*

Conjecture 2 *For every dimension d, there is a $v = v(d) \in (0,1)$, such that $\left(\mathcal{Q}_d\left(v^{\ell}\right)\right)_{\ell\in\mathbb{Z}}$ has Colorful Helly Number $2d$.*

Our Theorem 8 shows that proving Conjecture 2 would also confirm Conjecture 1.

References

[1] N. Alon, G. Kalai, J. Matoušek, and R. Meshulam. Transversal numbers for hypergraphs arising in geometry. *Advances in Applied Mathematics*, 29(1):79 – 101, 2002.

[2] I. Bárány. A generalization of Carathéodory's theorem. *Discrete Math.*, 40(2-3):141–152, 1982.

[3] I. Bárány, M. Katchalski, and J. Pach. Quantitative Helly-type theorems. *Proc. Amer. Math. Soc.*, 86(1):109–114, 1982.

[4] S. Brazitikos. Brascamp-Lieb inequality and quantitative versions of Helly's theorem. *Mathematika*, 63(1):272–291, 2017.

[5] G. Damásdi, V. Földvári, and M. Naszódi. Colorful Helly-type theorems for the volume of intersections of convex bodies, 2019.

[6] J. A. De Loera, X. Goaoc, F. Meunier, and N. H. Mustafa. The discrete yet ubiquitous theorems of Carathéodory, Helly, Sperner, Tucker, and Tverberg. *Bull. Amer. Math. Soc. (N.S.)*, 56(3):415–511, 2019.

[7] J. A. De Loera, R. N. La Haye, D. Rolnick, and P. Soberón. Quantitative combinatorial geometry for continuous parameters. *Discrete Comput. Geom.*, 57(2):318–334, 2017.

[8] A. Holmsen and R. Wenger. Helly-type theorems and geometric transversals. In J. E. Goodman, J. O'Rourke, and C. D. Tóth, editors, *Handbook of discrete and computational geometry*, pages 91–123. CRC Press, Boca Raton, FL, 2018. Third edition.

[9] A. F. Holmsen. Large cliques in hypergraphs with forbidden substructures. *Combinatorica*, 40(4): 527-537, 2020.

[10] A. Jung and M. Naszódi. Quantitative Fractional Helly and (p, q)-theorems. *European Journal of Combinatorics*, 99:103424, 2022.

[11] M. Katchalski and A. Liu. A problem of geometry in $\mathbf{R}^n$. *Proc. Amer. Math. Soc.*, 75(2):284–288, 1979.

[12] L. Lovász. Exercise 206. *Mat. Lapok*, 25(3-4):p. 370, 1974. (in Hungarian).

[13] J. Matoušek. *Lectures on discrete geometry*, volume 212 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 2002.

[14] M. Naszódi. Proof of a conjecture of Bárány, Katchalski and Pach. *Discrete Comput. Geom.*, 55(1):243–248, 2016.

[15] D. Rolnick and P. Soberón. Quantitative (p,q) theorems in combinatorial geometry. *Discrete Mathematics*, 340(10):2516 – 2527, 2017.

[16] S. Sarkar, A. Xue, and P. Soberón. Quantitative combinatorial geometry for concave functions. *arXiv e-prints*, page arXiv:1908.04438, Aug 2019.

Online Square Packing with Rotation

Shahin Kamali* Pooya Nikbakht†

Abstract

We consider the square packing problem, where the goal is to place a multiset of square items of different side-lengths in $(0, 1]$ into a minimum number of square bins of uniform side-length 1. We study the problem under the online setting, where the multiset of items forms a sequence revealed in an online and sequential manner. An online algorithm must place each item into a square bin without prior knowledge of the forthcoming items. Most existing results assume square items are placed orthogonally to the square bins (that is, parallel to the sides of the bins). In the presence of rotation, Kamali and Nikbakht [COCOA 2020] proved that the offline problem is NP-hard and admits an APTAS in an augmented setting. This paper investigates the online problem when item rotation is allowed. We introduce a linear-time algorithm that achieves an asymptotic competitive ratio of 2.306 when square-items have any size $x \in (0, 1]$, and a better asymptotic competitive ratio of 1.732 when $x \in (0, 1/2]$. We also study another problem where items, instead of squares, are isosceles right triangles (half-squares) and present a linear-time online algorithm with an asymptotic competitive ratio of at most 1.897.

1 Introduction

An instance of the *square packing problem* is defined with a multiset of *squares-items* of different sizes in the range $(0, 1]$. The goal is to place these squares into a minimum number of unit *square-bins* in a way that two square items placed in the same square bin do not intersect. At the same time, they can still "touch" each other. The problem is a generalization of the classical bin packing problem into two dimensions, and we sometimes refer to the squares-items simply as "items" and square-bins as "bins." A square item can be recognized by its side-length, which we refer to as the *size* of the square.

In the offline setting, all square-items are given in advance, and the algorithm can process them as a whole before placing any item into a bin. In particular, the algorithm can sort squares in decreasing order of their sizes, which comes in handy in designing algorithms. In the online setting, the multi-set of items forms a *sequence* which is revealed online and sequentially. When an item is revealed, an online algorithm must place it into a square bin without prior knowledge of forthcoming items. The decisions of an online algorithm are irrevocable.

Square packing has many applications in practice. One application is cutting stock where bins represent stocks (e.g., wood boards) and items are requests to squares of specific sizes. When requests arrive, an algorithm must cut the stock to provide the pieces that match the requests. This cutting process is equivalent to placing items into bins. Note that cutting stock aims to minimize the number of stocks, which also matches a square packing goal. We note that in many practical applications, requests arrive in an online manner, and the stock should be cut without prior knowledge about future requests. It is needless to say that the cutting process is irrevocable, which gives an inherently online nature to these applications of square packing.

There has been a rich body of research around square packing. All existing results except for recent work on the offline problem by Kamali and Nikbakht [20] assume that squares are not allowed to rotate; that is, the sides of square-items should be parallel to the square-bins. While this assumption makes the combinatorial analysis of the problem more straightforward, it comes at a cost. For example, consider an instance of the problem formed by n items of size 0.36. If we do not allow rotation, any bin can include at most four items, giving any algorithm a total cost of $n/4$. Allowing rotation, however, five items fit in each bin, and we can reduce the cost to $n/5$ (see Figure 1). As a result, the number of required bins is decreased by $n/20$, which is a notable saving in practice, e.g., for cutting stock applications.

In [20], it was proved that the offline problem is NP-

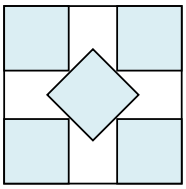


Figure 1: If all items have a length 0.36, allowing rotation helps pack 5 items per bin instead of 4.

*Department of Electrical Engineering and Computer Science, York University, Toronto, Canada, kamalis@yorku.ca

†Department of Computer Science, University of Manitoba, Winnipeg, Canada, nikbakhp@myumanitoba.ca

complete in the presence of rotation and admits an APTAS in an augmented setting, where the bins of the online algorithm are slightly larger than those of the optimal offline algorithm. In this paper, we consider the online square packing problem with rotation:

Definition 1 *In the online square packing with rotation, the input is a sequence $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ which is revealed in an online manner. At time-step t, the value of a_t is revealed, and an online algorithm has to place a square of size a_t into a bin, using any degree of transition and rotation, such that no two items in the same bin intersect (they can touch). The algorithm's decisions are irrevocable and are made without knowing the values of $a_{t'}$ for $t' > t$. The goal is to pack all square bins into a minimum number of squares of unit size.*

The asymptotic competitive ratio is the standard method for analyzing online packing problems. An algorithm A has a competitive ratio of c if there exists a constant $c_0 \geq 0$ such that, for all n and for all input sequences σ of length n, we have $A(\sigma) \leq c \cdot \textsc{Opt}(\sigma) + c_0$ where $A(\sigma)$ and $\textsc{Opt}(\sigma)$ denote the costs of A and an optimal offline algorithm Opt (with unbounded computational power) for processing σ, respectively.

1.1 Related Work

The 1-dimensional bin packing has been studied extensively in offline and online settings (e.g., [13, 12, 6, 7, 18, 1]). In the 1-dimensional setting, each item has a size in $(0, 1]$, and each bin has a capacity of 1. The offline, 1-dimensional bin packing problem is NP-hard [13], and the best existing result is an algorithm that opens at most $\textsc{Opt}(\sigma) + O(\log \textsc{Opt}(\sigma))$ bins σ [16]. In the online setting, the best existing algorithm has a competitive ratio of 1.578 [2], while no online algorithm has a competitive ratio better than 1.54278 [3].

There are many ways to extend bin packing to higher dimensions (see [5] for a survey). Packing axis-aligned square items into square bins is perhaps the most straightforward extension. In the offline setting, the problem is NP-hard [21], and there exists an APTAS for the problem [4]. In the online setting, the upper and lower bounds have been improved a few times [22, 10]. The best existing algorithm has a competitive ratio of 2.1187 [14] while the best existing lower bound is 1.6707 [15]. Almost-online square (and triangle) packing, where an online algorithm receives some "advice" about the input sequence, is studied in [17, 19].

Another generalization of bin packing into two dimensions assumes items are axis-aligned rectangles. This problem is also studied extensively (see [5] for details). In particular, a variant of this problem assumes rectangles can be rotated by exactly ninety degrees (see, e.g., [8]). We note that rotation by ninety degrees is not relevant for square packing and is quite restrictive compared to the rotations considered in this paper.

1.2 Contribution

We study the online square packing problem and present an online algorithm that achieves a competitive ratio of 2.306 for the square packing problem. Our algorithm is based on classifying squares based on their sizes and placing squares of similar sizes tightly, possibly using rotations, in the same bins. This approach was previously used to introduce different families of Harmonic algorithms for the classic bin packing in both one dimension and higher dimensions. However, the presence of rotations makes our classification and analysis different from the previous work. While analyzing the algorithm, we also consider sequences where the item sizes are at most $1/2$ and show that our algorithm has a competitive ratio of at most 1.732 in this case. We also study another problem where items, instead of squares, are isosceles right triangles (half-squares), also called "tans", and present a linear-time online algorithm with an asymptotic competitive ratio of at most 1.897.

2 Online Square Packing

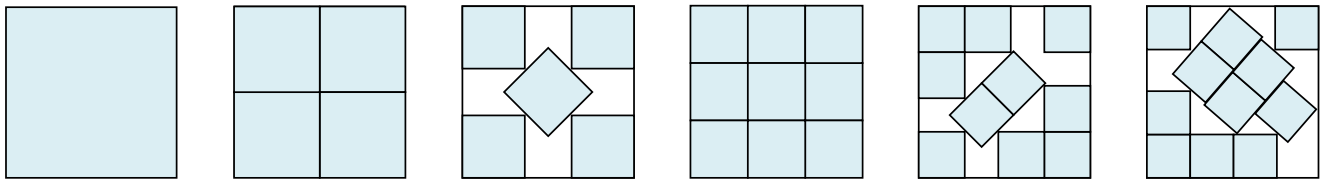
In this section we introduce our square packing algorithm called Square-Rotate.

2.1 Item Classification

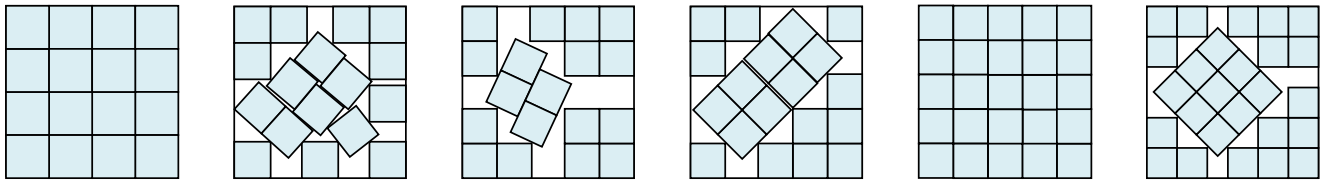
We classify squares by their side lengths, which we refer to as the "size" of the items. Square-Rotate packs squares of each class separately from other classes.

In total, there are 13 classes of squares. Square items with sizes in the range $(0, 0.1752]$ are in class 13; we refer to this class as the tiny class, and items that belong to it are called tiny items. We refer to items that belong to class $i \in [1, 12]$ as regular items. For each class $i \in [1, 12]$, the range of items in the class is specified as $(x_i, x_{i-1}]$ (for convenience, we define $x_0 = 1$). The values of x_i's are defined so that a certain number of items, denoted by S_i, of class i can fit in the same bin. The specific range of item sizes for each class $i \in [1, 12]$ and values of S_i are derived from the best-known or optimal results on the *congruent square packing problem* [11]. This problem asks for a square's minimum size $c(j)$ that can contain j unit-sized squares. A scaling argument, where the container size is fixed to be 1, gives values of $u(j)$'s when the goal is to pack j identical squares of maximum size $u(j)$ into a unit square. Table 1 provides the scaled best-known/optimal $u(j)$ values for $1 \leq j \leq 36$. These scaled numbers give the specific ranges that we use for classifying items.

Items of class 1 have sizes in the range $(1/2, 1]$, and we have $x_1 = 1/2$. Note that exactly $S_1 = 1$ item of



(a) Class 1: $x \in (0.5000, 1.0000]$ (b) Class 2: $x \in (0.3694, 0.5000]$ (c) Class 3: $x \in (0.3333, 0.3694]$ (d) Class 4: $x \in (0.2697, 0.3333]$ (e) Class 5: $x \in (0.2579, 0.2697]$ (f) Class 6: $x \in (0.2500, 0.2579]$



(g) Class 7: $x \in (0.2139, 0.2500]$ (h) Class 8: $x \in (0.2073, 0.2139]$ (i) Class 9: $x \in (0.2047, 0.2073]$ (j) Class 10: $x \in (0.2000, 0.2047]$ (k) Class 11: $x \in (0.1779, 0.2000]$ (l) Class 12: $x \in (0.1752, 0.1779]$

Figure 2: Placement of regular square items of class $i \in [1, 12]$ in their respective bin. It is possible to pack i square items of class i into a single square bin [11].

j	$u(j)$	opt./b.k.	j	$u(j)$	opt./b.k.
1	1.0	opt.	19	0.2047	b.k.
2-4	0.5	opt.	20-22	$= 0.2$	b.k.
5	0.3694	opt.	23-25	$= 0.2$	opt.
6-9	0.3333	opt.	26	0.1779	b.k.
10	0.2697	opt.	27	0.1752	b.k.
11	0.2579	b.k.	28	0.1716	b.k.
12-13	$= 0.25$	b.k.	29	0.1685	b.k.
14-16	$= 0.25$	opt.	30-33	≈ 0.1667	b.k.
17	≈ 0.2139	b.k.	34-36	≈ 0.1667	opt.
18	≈ 0.2073	b.k.			

Table 1: Optimal (opt.) or best-known (b.k.) values of $u(j)$ for $1 \leq j \leq 36$ when the goal is to pack j identical squares of the largest size $u(j)$ into a unit square. Values of $u(j)$ are the scaled values of the known results on congruent square packing [11].

class 1 can fit in the same bin. For $i \in [2, 12]$, S_i is the largest number of items of size x_{i-1} that fit in the same bin. For example, for $i = 2$, we have $S_2 = 4$ because $x_1 = 1/2$, and at most 4 items of size $1/2$ fit in the same bin. Moreover, x_i is defined as the largest value so that $S_i + 1$ items of size x_i do not fit in the same bin. For example, we have $x_2 = 0.3694$ because, according to Table 1, $S_2 + 1 = 5$ squares of size larger than 0.3694 do not fit in the same bin (with respect to the best known results).

The respective range of items for each class, as well as the values of S_i, is presented in Table 2. For example, a square is in class 1, 2, or 12 if its side size is in the interval $(0.5, 1]$, $(0.3694, .5]$, or $(0.1752, 0.1779]$, respectively. In Figure 2, it is specified how S_i items of the largest size in class i can fit into a square bin. We refer to [11] for details on the unit square packing problem.

2.2 Packing Regular Items

For each class i ($1 \leq i \leq 12$), the algorithm has at most one active bin of type i. When a bin of type i is opened, it is declared as the active bin of the class, and S_i square "spots", each of which having a size equal to the largest square of class i, are reserved in the bin. Upon the arrival of an item of class i, it is placed in one of the S_i spots of the active bin. If all these spots are occupied, a new bin of type i is opened. This ensures that all bins of type i, except potentially the current active bin, include S_i items.

2.3 Packing Tiny Items

For packing tiny items, the algorithm uses a different approach, proposed by Epstein and van Stee [9]. Briefly, it maintains at most one active bin for placing tiny items. The algorithm maintains a partitioning of the active bin into sub-bins whose sizes are 2^{-i} for non-negative, integer values of i. Upon the arrival of a tiny item of size x, the smallest sub-bin of size $y > x$ is repeatedly partitioned into four sub-bins of size $y/2$, up to the point that further partitioning results in sub-bins of size less than x. At this point, x is placed in one of the resulting partitions. Note that if there is no sub-bin of size $y \geq x$, the active bin is closed, and a new one is opened.

Lemma 1 *[9] Consider the square packing problem (without rotation) in which all items are of size at most $1/M$ for some integer $M \geq 2$. There is an online algorithm (as described above) that creates a packing in which all bins, except possibly one, have an occupied area of size at least $(M^2 - 1)/(M + 1)^2$.*

Class	Side length x	S_i	Occupied Area	Weight	Density
1	$(0.5000, 1.0000]$	1	$> 1(0.250)$=0.250	1	< 4.000
2	$(0.3694, 0.5000]$	4	$> 4(0.136)$=0.544	1/4	< 1.838
3	$(0.3333, 0.3694]$	5	$> 5(0.111)$=0.555	1/5	< 1.801
4	$(0.2697, 0.3333]$	9	$> 9(0.072)$=0.648	1/9	< 1.543
5	$(0.2579, 0.2697]$	10	$> 10(0.066)$=0.660	1/10	< 1.515
6	$(0.2500, 0.2579]$	11	$> 11(0.062)$=0.682	1/11	< 1.466
7	$(0.2139, 0.2500]$	16	$> 16(0.045)$=0.720	1/16	< 1.388
8	$(0.2073, 0.2139]$	17	$> 17(0.042)$=0.714	1/17	< 1.400
9	$(0.2047, 0.2073]$	18	$> 18(0.041)$=0.738	1/18	< 1.355
10	$(0.2000, 0.2047]$	19	$> 19(0.040)$=0.760	1/19	< 1.315
11	$(0.1779, 0.2000]$	25	$> 25(0.031)$=0.775	1/20	< 1.290
12	$(0.1752, 0.1779]$	26	$> 26(0.030)$=0.780	1/26	< 1.282
13	$(0, 0.1752]$		> 0.702	$1.425x^2$	≈ 1.425

Table 2: A summary of item classification, weights, and densities, as used in the definition and analysis of Square-Rotate.

2.4 Analysis

In this section, we prove a competitive ratio of at most 2.306 for Square-Rotate. We use a *weighting function argument*. For each item of size x, we define a weight $w(x) \geq x$ and prove that: (1) the total weight of square items in each bin of the algorithm, except potentially a constant number of them, is at least 1, and (2) the total weight of items in each bin of an optimal packing is at most 2.306. If $w(\sigma)$ denotes the total weight of items in an input sequence σ, then (1) implies that the number of bins opened by the algorithm is at most $w(\sigma) + c$, for some constant value of c, and (2) implies that the number of bins in an optimal packing is at least $w(\sigma)/2.306$. Therefore, the (asymptotic) competitive ratio of the algorithm would be at most 2.306.

Weight assignment. Recall that all bins opened for squares of class i ($1 \leq i \leq 12$), except possibly the last active bin, include S_i squares. We define the weight of items of class i to be $1/S_i$. This way, the total weight of items in bins opened for all squares of classes 1 to 12, except possibly 12 of them (the last bin from each class), is exactly 1. Therefore, (1) holds for bins opened for regular items.

We define the weight of a tiny item of size x as $x^2/0.701 (\approx 1.42x^2)$. All tiny items are of size at most 0.1752. Therefore, by Lemma 1, the occupied area of all bins opened for tiny items (except possibly one of them) will be at least 0.701. This implies their total weight is at least $0.701/0.701 = 1$.

Table 2 gives a summary of the weights of items in different classes. From the above argument, we conclude the following lemma.

Lemma 2 *The total weight of squares in each bin opened by* Square-Rotate*, except possibly 13 of them, is at least 1.*

Next, we provide an upper bound for the total weight of items in a bin of the optimal offline algorithm (Opt). Define the *density* of an item of size x as the ratio between its weight and area, i.e., $w(x)/x^2$. Given the lower bound for the size of each square belonging to class i ($1 \leq i \leq 12$), we can calculate an upper bound for the density of items in each class. For tiny items, the density is simply $1.425x^2/x^2 = 1.425$. Density upper bounds for all classes are reported in Table 2. Defining densities comes in handy in a case analysis used to prove the following lemma.

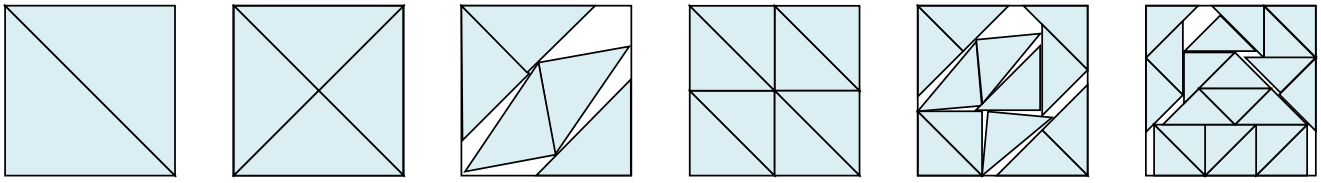
Lemma 3 (Appendix A) *The total weight of items in a bin of* Opt *is less than* 2.306*.*

Proof. (sketch) We consider a bin B of Opt and use case analysis to find an upper bound for the total weight of items in B. The case analysis considers the number of items of classes $1, 2$, and 3 in B. In each case, the upper bounds for densities yield an upper bound for the total weight. In the simplest case, when there is no item of class 1 in B, the density of all items will be at most 1.838, and so will be the total weight of all items in B. The presence of an item of class 1 restricts the number and class of other items in B, as captured by 14 sub-cases in the proof of the lemma. Nevertheless, the maximum weight in all cases is at most 2.306, which happens when B contains one item of class 1, three items of class 2, and one item of class 3. □

Provided with the above two lemmas, we can derive the main result of this section.

Theorem 1 *There is an algorithm* Square-Rotate *for the online square packing problem with rotation which achieves a competitive ratio of at most* 2.306*.*

Proof. For an input σ, let $SR(\sigma)$ and $\text{Opt}(\sigma)$ denotes the cost of Square-Rotate and Opt, respectively. Let



(a) Class 1: $x \in (0.7072, 1.0000]$ $t = 1.000$ (b) Class 2: $x \in (0.5593, 0.7072]$ $t = 0.7072$ (c) Class 3: $x \in (0.5000, 0.5593]$ $t = 0.5593$ (d) Class 4: $x \in (0.3828, 0.5000]$ $t = 0.5000$ (e) Class 5: $x \in (0.3056, 0.3828]$ $t = 0.3828$ (f) Class 6: $x \in (0.2843, 0.3056]$ $t = 0.3056$

Figure 3: Placement of regular tan items of class $i \in [1, 6]$ in their respective bin. It is possible to pack i tans of class i into a square bin [11].

$w(\sigma)$ denote the total weight of items of σ. Lemma 2 implies that $SR(\sigma) \leq w(\sigma) + 13$. Meanwhile, Lemma 3 implies that $Opt(\sigma) \geq w(\sigma)/2.306$. From these inequalities, we conclude $SR(\sigma) \leq 2.306\ \textsc{Opt}(\sigma) + c$, where c is a constant independent of the length of σ. □

It is possible to analyze Square-Rotate when all items are of size at most 1/2. In particular, we can establish the following result using the same weighting argument as before and a case analysis slightly different from that of Lemma 3, to get an upper bound for the total weight of items in a bin of optimal packing.

Theorem 2 *[Appendix B] When all items are of size at most 1/2,* Square-Rotate *achieves a competitive ratio of at most* 1.732.

3 Online Tan Packing

This section studies a problem similar to the online square packing problem, called the online tan packing problem, where the sequence of items is formed by isosceles right triangles (half-squares), which we refer to as "tans".

Definition 2 *The input to the* online tan packing with rotation *problem is a sequence* $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$*, where* $a_t \in (0, 1]$ *denote the leg sizes of right isosceles triangles (half-square triangles or tans), that need to be packed into a minimum number of unit bins. The decisions of the algorithm at any time* t *are irrevocable and are made without knowing the values of* $a_{t'}$ *for* $t' > t$*.*

3.1 Half-Square-Rotate Algorithm

We will introduce an online algorithm, called Half-Square-Rotate, that classifies tans by their leg sizes and packs tans of each class separately from other classes. There are seven classes, as presented in Table 3. We refer to items that belong to classes $i \in [1, 6]$ as regular items and those in class 7 as tiny items. Tiny items have sizes in the range $(0, 0.2843]$. For each class $i \in [1, 6]$, the range of items in class i is specified as $(y_i, y_{i-1}]$ (for convenience, we define $y_0 = 1$). The values of y_i are defined so that a certain number T_i of tans of class i can fit in the same bin.

The specific range of item sizes for each class $i \in [1, 6]$ and values of T_i is derived from the best-known or optimal results on the congruent tan packing problem [11], which asks for the minimum size $s(j)$ of a square that can contain j tans of unit leg size. A scaling argument, where the container size is fixed to be 1, gives $t(j)$ values when the goal is to pack j identical tans of maximum leg size $t(j)$ into a unit square. Table 4 provides the scaled best-known/optimal $t(j)$ values for $1 \leq j \leq 20$. These scaled numbers give the specific ranges that we used for classifying items as follows: Tans of class 1 have sizes in the range $(0.7072, 1]$, and we have $y_1 = 0.7072$. Note that exactly $T_1 = 1$ item of class 1 can fit in the same bin (for tans of size ≤ 0.7072, it is possible to pack at least two tans in the bin). For $i \in [2, 6]$, T_i is the number of items of size y_{i-1} that fit in the same bin. For example, for $i = 2$, we have $T_2 = 4$ because $y_1 = 0.7072$, and up to 4 items of size 0.7072 fit in the same bin. Moreover, y_i is defined as the largest value so that $T_i + 1$ items of size y_i cannot fit in the same bin. For example, we have $y_2 = 0.5593$ because, according to Table 4, $T_2 + 1 = 5$ tans of size 0.5593 do not fit in the same bin (with respect to the best known results). The respective range of items for each class, as well as the values of T_i, are presented in Table 3. Figure 3 shows how T_i items of the largest size in class i can fit into a square bin.

3.2 Packing Regular Items

For each class i ($1 \leq i \leq 6$), the algorithm has at most one active bin of type i. When a bin of type i is opened, it is declared as the active bin of the class, and T_i tan "spots", each of which has a size equal to the largest tan of class i, are reserved in the bin. Upon the arrival of an item of class i, it is placed in one of the T_i spots of the active bin. If all these spots are occupied by previous items, a new bin of type i is opened. This ensures that all bins of type i, except potentially the current active bin, include S_i items.

Class	Side length y	T_i	Occupied Area	Weight	Density
1	$(0.7072, 1.0000]$	2	$> 2(0.250)$=0.500	1/2	< 2.000
2	$(0.5593, 0.7072]$	4	$> 4(0.156)$=0.626	1/4	< 1.599
3	$(0.5000, 0.5593]$	5	$> 5(0.125)$=0.625	1/5	< 1.600
4	$(0.3828, 0.5000]$	8	$> 8(0.073)$=0.586	1/8	< 1.706
5	$(0.3056, 0.3828]$	12	$> 12(0.047)$=0.560	1/12	< 1.785
6	$(0.2843, 0.3056]$	20	$> 20(0.040)$=0.808	1/20	< 1.237
Tiny	$(0, 0.2843]$		> 0.557	$1.795(y^2/2)$	1.795

Table 3: A summary of item classification, weights and densities, as used in the definition and analysis of Half-Square-Rotate.

3.3 Packing Tiny Item

To pack tiny items, Half-Square-Rotate uses the same approach as Square-Rotate. Namely, the algorithm maintains partitioning any tiny bin into sub-bins formed by tans of various sizes. The only difference, compared to the algorithm of Epstein and van Stee [9], is that the square bin is divided initially into two tans (of side length 1) instead of four sub-bins. Subsequently, instead of partitioning larger square sub-bins into four sub-squares, we partition large tan sub-bins into *two* smaller tan sub-bins. One crucial observation is that it is possible to partition a tan into two sub-tans, which allows using the same approach as in [9]. Using a similar proof to the one in [9], we can show this adapted algorithm almost entirely packs each bin.

Lemma 4 *[9] Consider the tan packing problem in which all items are of size at most $1/M$ for some integer $M \geq 2$. There is an online algorithm that creates a packing in which all bins, except possibly one, have an occupied area of size at least $(M^2-1)/(M+1)^2$.*

3.4 Analysis

We use a weighting argument to prove the competitive ratio of Square-Rotate is at most 1.897. For each tan of size x, we define a weight $w(x)$ as follows. Recall that all bins opened for tans of class i ($1 \leq i \leq 6$), except possibly the last active bin of each class, include T_i tans. We define the weight of items of class i to be $1/T_i$. For a tiny tan of leg size x, we define its weight as $1.795(x^2/2)$ where $x^2/2$ is the area of the tan. Table 3 gives a summary of the weights of the items in different classes. Our definition of weights ensures that all bins opened for regular items, except potentially the last six active bins, have a total weight of 1 since they include i items of weight i. Similarly, our definition of weight for tiny items, paired with Lemma 4, ensures that all tiny bins, except potentially the active one, will have a weight of 1. We can conclude the following lemma.

n	t	opt./b.k.	n	t	opt./b.k.
1	$= 1.0$	opt	11	≈ 0.4143	b.k.
2	$= 1.0$	opt	12	≈ 0.3828	b.k.
3	≈ 0.7072	opt.	13	≈ 0.3720	b.k.
4	≈ 0.7072	opt.	14	≈ 0.3614	b.k.
5	≈ 0.5593	b.k.	15	≈ 0.3536	b.k.
6	≈ 0.5163	b.k.	16	≈ 0.3536	opt.
7	≈ 0.5003	b.k.	17	≈ 0.3367	b.k.
8	$= 0.5$	opt.	18	≈ 0.3333	opt.
9	≈ 0.4531	b.k.	19	≈ 0.3124	b.k.
10	≈ 0.4179	b.k.	20	≈ 0.3056	b.k.

Table 4: Optimal (opt.) or best-known (b.k.) $t(j)$ values for $1 \leq j \leq 20$ when the goal is to pack j identical tans of the largest leg size $t(j)$ into a unit square. Values of $t(j)$ are the scaled values of the known results on congruent tan packing [11].

Lemma 5 *[Appendix C] The total weight of tans in each bin opened by* Half-Square-Rotate*, except possibly 7 of them, is at least 1.*

Using a case analysis, which is similar to that of Lemma 3 and is based on investigating the density of items in a bin of Opt, we can find an upper bound for the total weight of items in any bin of the optimal offline algorithm Opt.

Lemma 6 (Appendix D) *The total weight of items in a bin of* Opt *is less than* 1.897*.*

Theorem 3 *The* Half-Square-Rotate *algorithm for the online tan packing with rotation achieves a competitive ratio of at most* 1.897*.*

Proof. For an input σ, let $HR(\sigma)$ and $\textsc{Opt}(\sigma)$ denote the cost of Half-Square-Rotate and Opt, respectively. Let $w(\sigma)$ denote the total weight of items of σ. Lemma 5 implies that $HR(\sigma) \leq w(\sigma) + 7$. Meanwhile, Lemma 6 implies that $\textsc{Opt}(\sigma) \geq w(\sigma)/1.897$. From these inequalities, we conclude $HR(\sigma) \leq 1.897\ \textsc{Opt}(\sigma) + c$, for some constant $c \in O(1)$ which proves an upper bound 2.306 for the asymptotic competitive ratio of Square-Rotate. □

Acknowledgements

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) [funding reference number DGECR-2018-00059].

References

[1] S. Angelopoulos, S. Kamali, and K. Shadkami. Online bin packing with predictions. In *Proceedings of the 31st Internation Joint Conference on Artificial Intelligence (IJCAI)*, 2022.

[2] J. Balogh, J. Békési, G. Dósa, L. Epstein, and A. Levin. A new and improved algorithm for online bin packing. In *Proceedings of the 26th Annual European Symposium on Algorithms (ESA)*, volume 112, pages 5:1–5:14, 2018.

[3] J. Balogh, J. Békési, G. Dósa, L. Epstein, and A. Levin. A new lower bound for classic online bin packing. *Algorithmica*, 83(7):2047–2062, 2021.

[4] N. Bansal, J. R. Correa, C. Kenyon, and M. Sviridenko. Bin packing in multiple dimensions: Inapproximability results and approximation schemes. *Math. Oper. Res.*, 31(1):31–49, 2006.

[5] H. I. Christensen, A. Khan, S. Pokutta, and P. Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24:63–79, 2017.

[6] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In *Approximation algorithms for NP-hard Problems*. 1997.

[7] E. G. Coffman Jr., J. Csirik, G. Galambos, S. Martello, and D. Vigo. Bin packing approximation algorithms: survey and classification. In *Handbook of Combinatorial Optimization*, pages 455–531. 2013.

[8] L. Epstein. Two-dimensional online bin packing with rotation. *Theor. Comput. Sci.*, 411(31-33):2899–2911, 2010.

[9] L. Epstein and R. van Stee. Optimal online bounded space multidimensional packing. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 214–223, 2004.

[10] L. Epstein and R. van Stee. Online square and cube packing. *Acta Inf.*, 41(9):595–606, 2005.

[11] E. Friedman. Packing unit squares in squares: A survey and new results. *The Electronic Journal of Combinatorics*, pages 1–24, 2000.

[12] G. Galambos and G. J. Woeginger. On-line bin packing - A restricted survey. *Math. Methods Oper. Res.*, 42(1):25–45, 1995.

[13] M. R. Garey and D. S. Johnson. Approximation algorithms for bin packing problems - a survey. In G. Ausiello and M. Lucertini, editors, *Analysis and Design of Algorithms in Combinatorial Optimization*, pages 147–172. Springer, 1981.

[14] X. Han, D. Ye, and Y. Zhou. A note on online hypercube packing. *Central European Journal of Operations Research*, 18(2):221–239, 2010.

[15] S. Heydrich and R. van Stee. Beating the harmonic lower bound for online bin packing. In *Proceedings of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 41:1–41:14, 2016.

[16] R. Hoberg and T. Rothvoss. A logarithmic additive integrality gap for bin packing. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2616–2625, 2017.

[17] S. Kamali and A. López-Ortiz. Almost online square packing. In *Proceedings of the 26th Canadian Conference on Computational Geometry (CCCG)*, 2014.

[18] S. Kamali and A. López-Ortiz. All-around near-optimal solutions for the online bin packing problem. In *Proceedings of the 26th International Symposium on Algorithms and Computation (ISAAC)*, volume 9472, pages 727–739, 2015.

[19] S. Kamali, A. López-Ortiz, and Z. Rahmati. Online packing of equilateral triangles. In *Proceedings of the 27th Canadian Conference on Computational Geometry (CCCG)*, 2015.

[20] S. Kamali and P. Nikbakht. Cutting stock with rotation: Packing square items into square bins. In *Proceedings of the 14th International Conference on Combinatorial Optimization and Applications (COCOA)*, pages 530–544, 2020.

[21] J. Y. T. Leung, T. W. Tam, C. S. Wong, G. H. Young, and F. Y. L. Chin. Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10(3):271–275, 1990.

[22] S. S. Seiden and R. van Stee. New bounds for multidimensional packing. *Algorithmica*, 36(3):261–293, 2003.

	C1	C2	C3	Sum of Weights (W)	Sum of Areas (A)	Remaining Area ($A_r = 1-A$)	Weight of Items in the Remaining Area ($W_r = A_r \times 1.543$)	Total Weight of Items in the Bin ($W_{max} = W + W_r$)
Number of items of each class in an OPT bin	1	0	0	1.00	> 0.250	< 0.750	< 1.157	< 2.157
	1	0	1	1.20	> 0.361	< 0.639	< 0.986	< 2.186
	1	0	2	1.40	> 0.472	< 0.528	< 0.815	< 2.215
	1	0	3	1.60	> 0.583	< 0.417	< 0.644	< 2.244
	1	0	4	1.80	> 0.694	< 0.306	< 0.472	< 2.272
	1	1	0	1.25	> 0.386	< 0.614	< 0.948	< 2.198
	1	1	1	1.45	> 0.497	< 0.503	< 0.776	< 2.226
	1	1	2	1.65	> 0.608	< 0.392	< 0.605	< 2.255
	1	1	3	1.85	> 0.719	< 0.281	< 0.434	< 2.284
	1	2	0	1.50	> 0.522	< 0.478	< 0.738	< 2.238
	1	2	1	1.70	> 0.633	< 0.367	< 0.566	< 2.266
	1	2	2	1.90	> 0.744	< 0.256	< 0.395	< 2.295
	1	3	0	1.75	> 0.658	< 0.342	< 0.528	< 2.278
	1	3	1	1.95	> 0.769	< 0.231	< 0.356	< 2.306

Table 5: The fourteen possible cases for a combination of items of class 2 (C2) and 3 (C3) together with an item x of class 1 (C1) in a bin B. Here, *sum of weights* (W) and *sum of areas* (A) indicate, respectively, the total weight and area of items of the first three classes in B. *Remaining area* is the area left in the bin that is used for packing items of class 4 or higher. *Weight of items in the remaining area* is an upper bound for the total weight of items of class 4 or higher in B (these items have a density of no more than 1.543). Finally, *total weight of items in the bin* indicates the sum of weights of all items (from all classes) in B.

Appendix

A Proof of Lemma 3

Proof. We use the following case analysis:

Case 1: Assume there is no item of class 1 in B. Since the density of items of other classes are less than 1.838, even if B is fully packed with items of the largest density, the total weight of items cannot be more than 1.838, which is less than 2.306.

Case 2: Assume there is one item x of class 1 (note that no two items of class 1 fit in the bin). Without loss of generality, we assume the size of x is $1/2+\epsilon$, where ϵ is a small positive value greater than zero. Clearly, a larger size for x does not increase the total weight of other items in B because it would leave less space to occupy more items in the bin (while the weight of x stays 1). Next, we consider all possible cases in which we have some items of class 2 and 3 together with x in B. As presented in Table 5, there will be 14 sub-cases to analyze. To see how we reach these 14 sub-cases, first note that it is not possible to accommodate four or more items of class 2 in addition to x in B (i.e., a total number of 5 or more items from these classes 1 and 2). This is because no five items with size larger than 0.3694 can fit in B [11]. A similar argument shows that we cannot have six or more items from classes 1, 2, and 3 together in a bin; otherwise, we could accommodate six identical squares of size strictly larger than 0.3333, which is a contradiction to the fact that no six items of size larger than 0.3333 can fit in the same bin [11]. In summary, the 14 sub-cases summarized in Table 5 cover all possibilities for items of the first three classes in Case 2.

According to Table 2, the density of items belonging to class i ($4 \leq i \leq 12$) as well as tiny items is at most 1.543 (which is the density of class-4 items). Using a similar argument made for Case 1, we suppose that, after placing a certain number of items of class 2 and 3 beside x in B, in each sub-case, we can fill the remaining space of B with the items of the maximum density 1.543. This allows us to calculate an upper bound for the maximum total weight of items in B for each of the sub-cases. The resulting bounds for each sub-case can be found in the last column of Table 5, where the maximum upper bound among all sub-cases is 2.306, which happens when we have one item of class 1 in B together with three items of class 2 and one item of class 3.

As a result, in both Case 1 and Case 2, the total weight of items in B cannot be more than 2.306. □

B Proof of Theorem 2

Proof. We employ the same approach and weighting function as the one we used to analyze the upper bound of Square-Rotate for the general setting, except that we exclude items of class 1, that is, items of size more than 1/2. By Lemma 2, the total weight of items in each bin of Square-Rotate, except for a possibly a constant number of them, is at least 1. Therefore, to prove the theorem, it suffices to show the total weight of items in any bin B of an optimal packing is at most 1.732. To study the maximum weight of items in B, we consider all possible combinations of items from classes 2 and 3 in B. For that, we consider the following two limitations: (i) we cannot have more than four items of class 2 in B; otherwise, we could accommodate five squares of size more than 0.3694, which is not possible [11]. (ii) we cannot have more than six items of class 2 or 3 in B. Otherwise, one could place six squares of size more than 0.3333, which is known to be impossible [11]. Altogether, we will have 19 cases to consider as presented in Table 6.

	C2	C3	Total Weight of Items in the Bin	C2	C3	Total Weight of Items in the Bin
Number of items of each class $c \in \{2,3\}$ in B.	0	0	< 1.543	1	4	< 1.698
	0	1	< 1.572	2	0	< 1.623
	0	2	< 1.601	2	1	< 1.652
	0	3	< 1.629	2	2	< 1.681
	0	4	< 1.658	3	0	< 1.664
	0	5	< 1.687	3	1	< 1.692
	1	0	< 1.583	3	2	< 1.721
	1	1	< 1.612	4	0	< 1.704
	1	2	< 1.641	4	1	< 1.732
	1	3	< 1.669			

Table 6: Maximum total weight of items of size $\in (0, 1/2)$ in a bin B of an optimal packing in all nineteen possible cases in which there is no item of class 1 but a combination of items of class 2 (C2) and 3 (C3) in B.

We have used the same method as in Lemma 3 to calculate the upper bound for the total weight of items in each case. Table 6 summarizes the final results for all cases. The maximum weight, 1.732, happens when we have four items of class 2 together with one item of class 3 packed in B. □

C Proof of Lemma 5

Proof. In every bin of class i ($1 \leq i \leq 6$), except possibly the last bin, there are T_i items, each having a weight of $1/T_i$. As a result, the total weight of the items in each bin of such classes, except possibly 6 of them, is exactly $T_i \times 1/T_i = 1$. For bins of tiny items, we know, by Lemma 4 and considering $1/M = 0.2843$, that the occupied area of all bins, except possibly the last one, will be at least 0.557. Therefore, the total weight of items in such a bin is at least $W = 1.795 \times 0.557 = 1$. □

D Proof of Lemma 6

Proof. We first define the *density* of a tan item of leg size x as the ratio between its weight and its area, i.e., $w(x)/(x^2/2)$. Given the lower bound for the leg size of items in each class i ($1 \leq i \leq 6$), and, hence, their area, we can calculate an upper bound for the density of each item in the class. For tiny items, the density is simply equal to $1.795(x^2/2)/(x^2/2) = 1.795$. The densities of items from different classes are reported in Table 3. In what follows, we use a case analysis approach to prove that the total weight of items in a bin B of an optimal packing is at most 1.897. There are three cases to consider: either 0, 1, or 2 tans of class 1 exist in the bin. Note that it is not possible to accommodate 3 or more items of class 1 in a bin because their total area would exceed 1.

Case 1: No class-1 item in B: Since the density of items of class 2 or larger is at most 1.795, even if all area of B is filled with items of the largest density, the total weight of items cannot exceed 1×1.795 which is less than 1.897.

Case 2: Exactly one class-1 item in B: If there is exactly one item of class 1 (with weight 1/2 and the area of at least $(0.7072 \times 0.7072)/2 = 0.25$) in B, the remaining area in the bin will be at most 0.75. Items of classes other than class 1 have a density of at most 1.795. Even if we fill the remaining area with such items of the highest density, the total weight of items in B will not exceed 1.346. As a result, the total weight of items in B cannot be more than $1/2 + 1.346 = 1.846$, which is less than 1.897.

Case 3: Exactly two class-1 items in B: If there exists exactly two items of class 1 with weight 1/2 (and the total weight of 1) and total area of at least $2 \times (0.7072 \times 0.7072)/2 = 0.50$, the remaining area in B will be at most 0.50. If this remaining area is filled by items of the highest density, which are items of the last class with a density of at most 1.795, the total weight of items in B will not be more than $1 + 0.50 \times 1.795 = 1.897$.

In conclusion, the total weight of items in a bin B of an optimal packing cannot be more than 1.897 in all cases. □

A Randomized Algorithm for Non-crossing Matching of Online Points

Shahin Kamali* Pooya Nikbakht† Arezoo Sajadpour‡

Abstract

We study randomized algorithms for the online non-crossing matching problem. Given an online sequence of n online points in general position, the goal is to create a matching of maximum size so that the line segments connecting pairs of matched points do not cross. In previous work, Bose et al. [CCCG 2020] showed that a simple greedy algorithm matches at least $\lceil 2n/3 - 1/3 \rceil \approx 0.6\bar{6}n$ points, and it is the best that any deterministic algorithm can achieve. In this paper, we show that randomization helps achieve a better competitive ratio; that is, we present a randomized algorithm that is expected to match at least $235n/351 - 202/351 \approx 0.6695n$ points.

1 Introduction

In the geometric matching problems, the input is a set of geometric objects, and the goal is to create a pairwise matching of these objects under different restrictions and objectives. In the bottleneck matching problem, for example, the goal is to create a perfect matching of n points, assuming n is even, to minimize the maximum length of the line segments that connect matched pairs [8]. Using the same terminology as in graph theory, we refer to the line segments that connect pairs of matched vertices as the *edges* of the matching. Other variants of the geometric matching problems ask for perfect matchings that minimize the total length of edges [4] or maximize the length of the shortest edge [6]. Matching objects other than points are also studied (e.g., [1, 2]).

In the non-crossing matching problem, the input is a set of points in general position, and the goal is to match points so that the edges between the matched pairs do not cross. It is relatively easy to solve the problem in the offline setting: one can sort all points by their x-coordinate and match pairs of consecutive points. All points, except possibly the last one, will be matched. The running time of this algorithm is $O(n \log n)$, which is asymptotically optimal [5]. Other variants of non-crossing matching have been studied in the offline setting; see [7]. For example, Aloupis et al. [1] considered the computational complexity of finding the non-crossing matching of a set of points with a set of geometric objects that can be a line, a line segment, or a convex polygon.

Bose et al. [3] studied the online variant of the non-crossing matching. Under this setting, the input is a set of n points in the general position that appears sequentially. When a point x arrives, an online algorithm can match it with an existing unmatched point y, provided that the edge between them does not cross previous edges in the matching. Alternatively, the algorithm can leave the point unmatched. In making these decisions, the algorithm has no information about the forthcoming points or the input length. The algorithm's decisions are irrevocable in the sense that once a pair of points is matched, that pair cannot subsequently be removed from the matching. The objective is to find a matching of maximum size.

Under a worst-case analysis, where an adversary generates the online sequence, it is not possible to match all points. For example, consider an input that starts with two points x and y. If an online algorithm leaves the two points unmatched, the adversary ends the sequence, and the matching is already sub-optimal. If the algorithm matches x and y, then the adversary generates the following two points on the opposite sides of the line between x and y, and the matching will be sub-optimal for this input of length 4. Bose et al. [3] extended this argument to show that no deterministic algorithm can match more than $\lceil 2n/3 - 1/3 \rceil$ points in a worst-case input of length n. Meanwhile, they showed that any greedy algorithm matches at least $\lceil 2n/3 - 1/3 \rceil$ points, and hence is optimal. An algorithm is greedy if it does not leave a point x unmatched if there is a suitable unmatched point y that x can be matched to (that is, the edge between x and y does not cross existing edges in the matching).

1.1 Contribution

We study randomized algorithms for the non-crossing matching problem. As in [3], we investigate worst-case scenarios where the input is adversarially generated. We assume the adversary is *oblivious* to the random choices made by the algorithm, but it is aware of how the algorithm works (that is, the "code" of the algorithm).

*Department of Electrical Engineering and Computer Science, York University, Toronto, Canada, kamalis@yorku.ca

†Department of Computer Science, University of Manitoba, Winnipeg, Canada, nikbakhp@myumanitoba.ca

‡Department of Computer Science, University of Manitoba, Winnipeg, Canada, sajadpoa@myumanitoba.ca

We present a randomized algorithm that matches at least $\lfloor 235n/351 - 202/351 \rfloor \approx 0.6695n$ points on expectation for any input of size n. This result establishes the advantage of randomized algorithms over the best deterministic algorithm, which matches roughly $0.6\bar{6}n$ points in the worst case [3].

There are two main components in our randomized algorithm. First, the algorithm maintains a convex partitioning of the plane and matches two points only if they appear in the same partition. The matching is followed by updating the partitioning by extending the edge between the matched pair. This partitioning enables us to use a simple inductive argument to analyze the algorithm. Second, the algorithm deviates from the greedy strategy and gives a chance to an incoming point x to stay unmatched even if there are one or two points in the same convex region that x can be matched to. As we will see, this will be essential for any improvement over deterministic algorithms.

2 A Randomized Online Algorithm

We present and analyze a randomized online algorithm for the non-crossing matching problem. In what follows, for any $a \neq b$ we use L_{ab} to denote the line passing through a and b, and S_{ab} to denote the line segment between a and b.

2.1 Algorithm's description

The algorithm maintains a partitioning of the plane into convex regions and matches points only if they belong to the same region. In the beginning, only one region is formed by the entire plane. After four points appear inside a convex region, one or two pairs of points are matched, and the convex region is partitioned into two or three convex regions by extending the line segments passing through the matched pairs.

Let x, y, z, and w be the first four points that appear (in the same order) inside a convex region C. In what follows, we describe how these four points are treated.

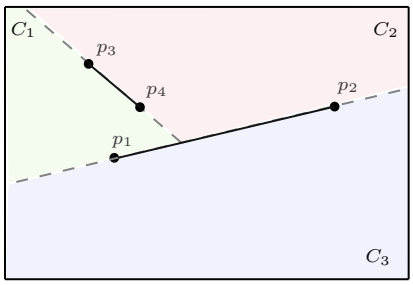
- Upon the arrival of x, there is no decision to make, given that there is no point inside C to be matched with x.
- Upon the arrival of y, it is matched with x with a probability of $1/2$ and stays unmatched with a probability of $1/2$.
- Upon the arrival of z, if the pair (x, y) is already matched, then there is no decision to make. Otherwise, z is matched with x with a probability of $1/3$, with y with a probability of $1/3$, and stays unmatched with a probability of $1/3$.
- Upon the arrival of w, there are two possibilities to consider:
  - First, suppose a pair of points $a, b \in \{x, y, z\}$ is already matched, while a third point $c \in \{x, y, z\}/\{a, b\}$ is unmatched. If it is possible to match w with c (that is, S_{wc} does not cross S_{ab}), then w is matched with c; otherwise, when S_{wc} and S_{ab} cross, there is no decision to make.
  - Second, suppose no pair of the first three points are matched. Then w is matched with a point $a \in \{x, y, z\}$ so that the two points $b, c \in \{x, y, z\}/\{a\}$ appear on different sides of the line L_{aw} (if there is more than one such point, w is matched with z).

After the arrival of four points inside C, either all points are matched into two pairs, in which case we say a "double-pair is realized", or only two points are matched while the other two appear on different sides of the matched pair, in which case we say a "single-pair is realized." If a single-pair is realized, the algorithm extends the line segment between the matched pair until it hits the boundary of C; in this case, C is partitioned into two convex regions. If a double-pair is realized, the line segment between the first matched pairs is extended until it hits the boundary of C or the (non-extended) segment between the second matched pair. This is followed by extending the line segment between the second pair until it hits the boundary of C or an extended line that passes through the first matched pair. In the case of a double-pair, C is partitioned into three convex regions when a double-pair is realized.

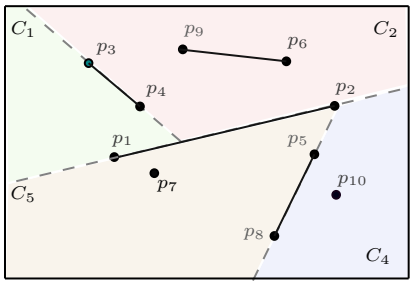
Assume $n \geq 8$. A single-pair is "good" iff, after all the n points appear, each of the two regions resulting from extending the line segment of the matching contains at least 2 points (discounting their potential further partitionings in the future), and it is "bad" otherwise. Similarly, a double-pair is said to be "good" if, after all the n points appear, one of the three regions formed by extending the line segments of the two matched pairs is empty; otherwise, it is "bad." The presence of 2 or more points or no points in a region leaves a possibility of matching all pairs; hence we assert that a single/double pair is "good" or "bad" as specified above.

When a double-pair is realized, the ordering at which we extend the line segments between matched pairs will impact whether the double-pair is good or bad. But ultimately, our worst-case analysis still holds if we change the algorithm to extend the line segment between the second matched pair before the first one.

The following example illustrates the algorithm's steps. Consider an input formed by 10 points labeled from p_1 to p_{10} in the order of their appearance, as depicted in Figure 1. The convex regions maintained by the algorithm are highlighted in different colours. Initially, the entire plane is a convex region C_0, where point p_1 appears. Upon the arrival of p_2, the algorithm



(a) The state of the algorithm after processing $p_1, \ldots, p_4$.



(b) The state of the algorithm after processing $p_1, \ldots, p_{10}$.

Figure 1: One possible output of the algorithm when the input is a sequence of 10 points labelled as $p_1, \ldots, p_{10}$ in the order of their appearance.

matches it with p_1 with a probability of 1/2. Suppose (p_1, p_2) are matched. Then, there is no decision to be made for p_3. Upon the arrival of p_4, the line segments $S_{p_1p_2}$ and $S_{p_3p_4}$ do not cross. Therefore, p_4 is matched with p_3. At this point, four points have appeared in C_0 and a double-pair (p_1, p_2) and (p_3, p_4) has been realized. Therefore, C_0 is partitioned into three smaller convex regions C_1, C_2, and C_3 by extending S_{p_1,p_2} and S_{p_3,p_4} (Figure 1a). Points p_5 and p_6 appear respectively in C_3 and C_2. Since these are the first points in their respective regions, there is no decision to be made for them, and they stay unmatched. Subsequently, p_7 appears in C_3 and the algorithm matches with p_5 with a probability of 1/2. Suppose these two points are not matched. Upon the arrival of p_8 in C_3, it is matched with p_5 or p_7, each with a probability of 1/3, and is left unmatched with a probability of 1/3. Suppose (p_5, p_8) are matched. Next, point p_9 appears in C_2 and is matched with p_6 with a probability of 1/2, and stays unmatched with a probability of 1/2. Suppose (p_6, p_9) are matched. Finally, point p_{10} appears on C_3. Given that the $S_{p_7p_{10}}$ crosses $S_{p_5p_8}$, there is no decision to be made, and p_{10} stays unmatched. At this point, four points have appeared in C_3, and a single-pair (p_5, p_8) has been realized. Therefore, C_3 is partitioned into two smaller convex regions C_4 and C_5 by extending S_{p_5,p_8} (Figure 1b).

2.2 Algorithm's analysis

Let $f(n)$ denote the expected number of unmatched points by the algorithm when input is formed by n items. We use an inductive argument to find an upper bound for $f(n)$. First, we prove the following lemma, which is used when establishing the base of the induction.

Lemma 1 *After four points arrive in a convex region C, with a probability of at least 1/3, a double-pair is realized, and with a probability of at most 2/3, a single-pair is realized.*

Proof. Let x, y, z, and w denote the four points in the same order they appear. There are two cases to consider:

- Suppose S_{xy} crosses S_{zw}. In this case, a double-pair is realized iff (i) x and y are not matched, and (ii) z is matched with either x or y (in which case w will be matched to the other point). The chance of (i) is 1/2, and the chance of (ii) is 2/3; therefore, a double-pair is realized with a chance of 1/3.
- Suppose S_{xy} does not cross S_{wz}. Then, (x, y) are matched with a probability of 1/2, and after that, (w, z) are matched, and a double-pair is realized with a chance of 1/2.

□

Using Lemma 1, we can prove the following lemma, which serves as the base case for the inductive proof of the main result.

Lemma 2 *We have $f(0) = 0$, $f(1) = 1$, $f(2) = 1$, $f(3) = 4/3$, $f(4) \leq 4/3, f(5) \leq 5/3, f(6) \leq 20/9$, and $f(7) \leq 26/9$.*

Proof. Suppose n items appear in a convex region C. The proof is trivial for $n \leq 2$. In what follows, we prove the lemma for other values of n.

- For $n = 3$, it is possible that all points stay unmatched, which happens when the second point is not matched with the first one (with a probability of 1/2), and then the third point is not matched with any of the first two points (with a probability of 1/3). Therefore, with a probability of 1/6, all three points stay unmatched, and one point stays unmatched with a probability of 5/6. We can write $f(3) = 1/6 \cdot 3 + 5/6 \cdot 1 = 4/3$.
- For $n = 4$, using Lemma 1, we can write $f(4) \leq 1/3 \cdot 0 + 2/3 \cdot 2 = 4/3$.
- For $n = 5$, after the first four points appeared, either a single-pair or a double-pair is realized:

– Suppose a single-pair is realized. Then, C is partitioned into two regions, one containing one point and the other one containing two points. Therefore, it is expected that $f(1) + f(2) = 2$ points stay unmatched.

– Suppose a double-pair is realized. Then, the first four points are matched, and only the fifth point stays unmatched.

By Lemma 1, with a probability of at least $1/3$, a double-pair is realized, and with a probability of at most $2/3$, a single-pair is realized. Therefore, we can write $f(5) \leq 1/3 \cdot 1 + 2/3 \cdot 2 = 5/3$.

- For $n = 6$, after the first four points appeared, either a single-pair or a double-pair is realized:

  – Suppose a single-pair is realized. Then, C is partitioned into two regions. Either (i) the fifth or the sixth points appear in the same region, in which case one region will have one point, and the other one will have three points, or (ii) the fifth and the sixth points appear in different regions, in which case each region contains two points. Therefore, it is expected that at most $\max\{f(1) + f(3), f(2) + f(2)\} = 7/3$ points stay unmatched.

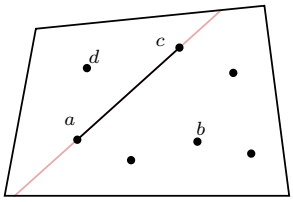
  – Suppose a double-pair is realized. Then, at most 2 points (the last two points) stay unmatched.

  By Lemma 1, with a probability of at least $1/3$, a double-pair is realized, and with a probability of at most $2/3$, a single-pair is realized. Therefore, we can write $f(6) \leq 1/3 \cdot 2 + 2/3 \cdot 7/3 = 20/9$.

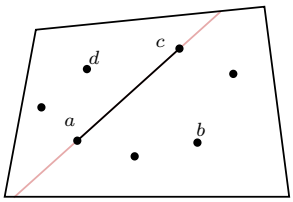
- For $n = 7$, after the first four points appeared, either a single-pair or a double-pair is realized:

  – Suppose a single-pair is realized. Then, C is partitioned into two regions. Either (i) the fifth, the sixth, and the seventh points all appear in the same region, in which case one region has one point, and the other one has four points (Figure 2a), or (ii) one of these points appear in one region, and the other two appear in the other region, in which case one region contains two points, and the other region contains three points (Figure 2b). Therefore, at most $\max\{f(1) + f(4), f(2) + f(3)\} \leq \max\{1 + 4/3, 1 + 4/3\} = 7/3$ points stay unmatched.

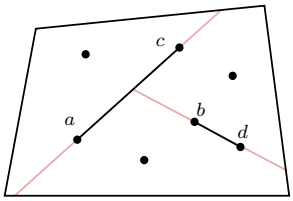
  – Suppose a double-pair is realized. Then, at most three points stay unmatched, which happens when any of the three regions formed extending the line segments between the matched pairs includes a point (see Figure 2c).



(a) The case where a single-pair is realized, and the last three points appear in different regions.



(b) The case where a single-pair is realized, and the last three points appear in different regions.



(c) The case where a double-pair is realized, and the last three points appear in different regions.

Figure 2: The cases used in the calculation of $f(7)$; $a, b, c, d \in \{x, y, z, w\}$ where x, y, z, and w are the first four points in the same order of their appearance.

Unlike other cases, for $n = 7$, the upper bound for the expected number of unmatched points is larger when a double-pair is realized compared to when a single-pair is realized; hence we cannot use Lemma 1. Instead, we note that the probability of a single-pair being realized is at least $1/6$. This is because the first three points stay unmatched with a probability of $1/2 \cdot 1/3 = 1/6$, and then the fourth point gets matched to the point that bisects the unmatched points (by the definition of the algorithm). Therefore, we can write $f(7) \leq 5/6 \cdot 3 + 1/6 \cdot 7/3 = 26/9$. □

We use an inductive argument to prove $f(n) \leq cn + d$ where $c = 116/351 \approx 0.3304$ and $d = 32c - 10 = 202/351 \approx 0.5754$. First, we apply Lemma 2 to establish the base of induction in the following theorem.

Lemma 3 *For $n \in [2, 7]$, it holds that $f(n) \leq cn + d$ where $c = 116/351$ and $d = 202/351$.*

Proof. The proof follows from Lemma 2. For $n = 2$, we have $f(2) = 1 < 2c + d$ (since $2c + d > 1.2364$). For

$n = 3$, we have $f(3) = 4/3 = 3c + d$ (since $3c + d > 1.5669$). For $n = 4$, we have $f(4) \leq 4/3 < 4c + d$ (since $4c + d > 1.8974$). For $n = 5$, we have $f(5) \leq 5/3 < 5c + d$ (since $5c + d > 2.2279$). For $n = 6$, we have $f(6) \leq 20/9 < 6c + d$ (since $6c + d > 2.5584$). For $n = 7$, we have $f(7) \leq 26/9 = 7c + d$ (note that $7c + d = 26/9$). □

Lemma 4 *Consider an input sequence with $n \geq 8$ points. Suppose at least four points appear in some convex region C maintained by the algorithm. At least one of the following statements holds with respect to the first four points in C, regardless of how an adversary generates the input:*

- *A good single-pair is realized in C with a probability of at least 1/6.*
- *A good double-pair is realized in C with a probability of at least 1/6.*

Proof. Let x, y, z, and w denote the first four points in the same order that they appear in C.

First, suppose the convex hull formed by the four points is a triangle Δ which includes the fourth point inside it. We consider the following two cases:

- Assume w is the point that is inside Δ. Then the pairs (x, y) and (w, z) form a double-pair that is realized with a probability of 1/2. This is because the pair (x, y) is matched with a probability of 1/2, and then the pair (w, z) is matched with a probability of 1. Meanwhile, (w, z) is a single-pair which is realized with a probability of 1/6. This is because, with a chance of 1/6, the first three points stay unmatched, and then the algorithm matches w to z with a chance of 1. Now, if the double pair formed by the pairs (x, y) and (w, z) is bad, then there should be at least one future point on each side of the line passing through (w, z), which means (w, z) is a good single-pair (see Figure 3a).
- Assume w is a vertex of Δ and another point $c \in \{x, y, z\}$ is inside Δ. Let a, b be the other two points in $\{x, y, z\}$. Then, the pairs (a, b) and (c, w) form a double-pair which is realized with a probability of at least 1/6. This is because the pair (a, b) is matched with a probability of at least 1/6 (the pair (a, b) is matched with a probability of 1/2 if $z \notin \{a, b\}$, and with a probability of 1/6 if $z \in \{a, b\}$), and then w is matched with c with a probability of 1. Meanwhile, the pair (c, w) is a single-pair which is realized with a probability of 1/6. Similar to the previous case, if the double pair formed by the pairs (a, b) and (c, w) is bad, then there should be at least one future point on each side of (a, b), which means (c, w) is a good single-pair (see Figure 3b).

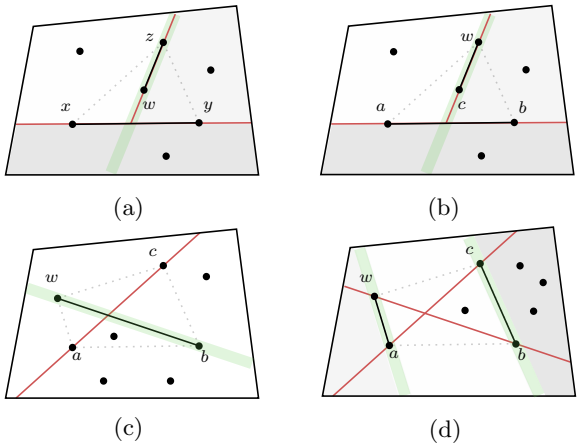


Figure 3: An illustration of the proof of Lemma 4. (a) when w is inside the triangle Δ, either the single-pair formed by (w, z) is a good single-pair, or the double-pair formed by $(x, y), (w, z)$ is a good double-pair. (b) when $c \in \{x, y, z\}$ is inside the triangle Δ, either the double pair formed by $(a, b), (w, c)$ is a good double-pair, or the single-pair formed by (w, c) is a good single-pair. (c) the case when at least one of the diagonals of the convex hull formed by the four points (here (w, b)) forms a good single-pair (d) when none of the single-pairs formed by the diagonals of the convex hull are good, all remaining points appear in one of the quarter-planes formed by extending these diagonals; therefore, the pair of points on the boundary of the quarter-plane (here (b, c)) and the pair of points outside the quarter-planes (here (w, a)) form a good double-pair.

Next, suppose the convex hull formed by the four points is a quadrilateral and includes all of them. Consider the two single-pairs formed by the diagonals of the convex hull. Any of these pairs can be realized with a probability of at least 1/6. Specifically, the diagonal involving w is realized when no pair of points from $\{x, y, z\}$ are matched, which takes place with a probability of 1/6. The other diagonal is either between x and y, which is realized with a probability of 1/2 or between z and $a \in \{x, y\}$, which is realized with a probability of 1/6. Therefore, if any of the two diagonals form a good single-pair, the statement of the lemma holds, and we are done (see Figure 3c). If none of the two diagonals is good, then all the remaining points in the input sequence should appear in one of the quarter-planes formed by extending these diagonals (see Figure 3d). Then, the double-pair formed by the pair of points on the boundary of the quarter-plane (points b and c in Figure 3d) and the pair of points outside of the quarter-plain (points w and a in Figure 3d) is a good double-pair. The probability of such a good double-pair being realized is at least 1/6. This is because one of the pairs in the double-pair involves two of the first three

points. If these points are (x, y), the double-pair is realized with a probability of 1/2; otherwise, it is realized with a probability of 1/6. □

We are now ready to prove the main result.

Theorem 5 *There is a randomized algorithm that, for any input formed by $n \geq 2$ points, leaves at most $cn + d$ points unmatched on expectation, where $c = 116/351$ and $d = 202/351$.*

Proof. We use an inductive argument to show that our algorithm satisfies the conditions specified in the theorem. For $n \leq 7$, the claim holds by Lemma 3. Suppose $n \geq 8$, and assume that for any $m < n$, it holds that $f(m) \leq cm + d$.

First, we claim that the number of unmatched points is at most $cn + d + (2 - 6c)$ when a bad single-pair is realized or a bad double-pair is realized after the first four points of the input sequence appear. If a bad single-pair is realized, then either (I) there is one point on one side of the matched pair and $n - 3 > 2$ points on the other side, or (II) there is no point on one side of the matched pair and $n - 2 > 2$ points on the other side. For (I), by the induction hypothesis, the number of unmatched points on the side with $n-3$ points will be at most $f(n-3) \leq cn-3c+d$. Therefore, the number of unmatched points is at most $f(n-3)+1 \leq cn-3c+d+1 < cn+d+(2-6c)$. The last inequality holds because $c < 1/3$. For (II), the number of unmatched points will be at most $f(n-2) \leq cn + d - 2c < cn + d + (2 - 6c)$.

If a bad double-pair is realized, then one of the following cases holds for the three regions formed by extending the line segments between the matched pairs (regardless of the ordering at which we extend the line segments):

i) One region contains $n-6$ points, and the other two regions each contains one point. Note that $n-6 \geq 2$ since $n \geq 8$. By the induction hypothesis, the expected number of unmatched points is at most $2 + f(n-6) = cn + d + (2 - 6c)$.

ii) One region contains $m \geq 2$ points, another region contains one point, and the third region contains $n - m - 5 \geq 2$ points. The expected number of unmatched points is at most $f(m)+f(n-m-5)+1 \leq cn - 5c + 2d + 1 < cn + d + (2 - 6c)$. The last inequality holds because $c + d < 1$.

iii) One region contains $m_1 \geq 2$ points, one region contains $m_2 \geq 2$ points, and the third region contains $m_3 = n - m_1 - m_2 - 4 \geq 2$ points. The expected number of unmatched points is at most $f(m_1) + f(m_2) + f(m_3) \leq cn - 4c + 3d < cn + d + (2 - 6c)$. The last inequality holds because $c + d < 1$.

In summary, if a bad single-pair or a bad double-pair is realized, the expected number of unmatched points is at most $cn + d + (2 - 6c)$, and the claim holds.

By Lemma 4, after the appearance of the first four points, either a) a good single-pair or b) a good double-pair can be realized with a probability of at least 1/6.

Suppose case a) holds, that is, a good single-pair is realized with a probability of at least 1/6, which implies a bad single-pair or double-pair is realized with a probability of at most 5/6. In case the good single-pair is realized, there will be $m \geq 2$ points on one side of the line segment connecting matched pair, and $n - m - 2 \geq 2$ points on the other side. Therefore, the expected number of unmatched points will be at most $f(m)+f(n-m-2) \leq cn+2d-2c = (cn+d)+(d-2c)$. On expectation, the number of unmatched points will be at most $1/6((cn+d)+(d-2c))+5/6(cn+d+(2-6c)) = cn + d + 1/6(d - 32c + 10) = cn + d$. The last equality holds because $d = 32c - 10$.

Next, suppose case b) holds, that is, a good double-pair is realized with a probability of at least 1/6, which implies a bad single-pair or double-pair is realized with a probability of at most 5/6. If the good double-pair is realized, by definition, at least one of the three convex regions formed by extending the double-pair will be empty. For the other two regions, we have the following cases:

i) One region is empty, and the other contains $n - 4 \geq 2$ points, in which case the expected number of unmatched points becomes $f(n-4) \leq cn+d-4c < cn+d+(1-5c)$. The last inequality holds because $c < 1$.

ii) One region contains a single point, and the other one contains $n - 5 \geq 2$ points. The expected number of unmatched points will be at most $f(n-5)+1 \leq cn + d + (1 - 5c)$.

iii) Both regions include $m \geq 2$ and $n - m - 4 \geq 2$ points. In this case, the expected number of unmatched points will be at most $f(m) + f(n - m - 4) \leq cn + d + (d - 4c) < cn + d + (1 - 5c)$. The last inequality holds because $c + d < 1$.

Therefore, as long as the good double-pair is realized, the expected number of unmatched points will be at most $cn + d + (1 - 5c)$. Then we can write $f(n) \leq 1/6((cn + d) + (1 - 5c)) + 5/6((cn + d) + (2 - 6c)) = cn + d + 1/6(11 - 35c) < cn + d$. The last inequality holds since $c > 11/35$. □

Acknowledgements

The authors thank to the anonymous referees for their constructive comments and recommendations. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) [funding reference number DGECR-2018-00059].

References

[1] G. Aloupis, J. Cardinal, S. Collette, E. D. Demaine, M. L. Demaine, M. Dulieu, R. F. Monroy, V. Hart, F. Hurtado, S. Langerman, M. Saumell, C. Seara, and P. Taslakian. Non-crossing matchings of points with geometric objects. *Comput. Geom.*, 46(1):78–92, 2013.

[2] A. Banik, M. J. Katz, and M. Simakov. Bottleneck segment matching. In *Proceedings of the 27th Canadian Conference on Computational Geometry CCCG*, 2015.

[3] P. Bose, P. Carmi, S. Durocher, S. Kamali, and A. Sajadpour. Non-crossing matching of online points. In *Proc. 32nd Canadian Conference on Computational Geometry (CCCG)*, 2020.

[4] A. Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.

[5] J. Erickson. https://mathoverflow.net/questions/86906. https://mathoverflow.net/questions/86906. Accessed: 2020-05-03.

[6] A. Kaneko and M. Kano. Discrete geometry on red and blue points in the plane—a survey. *Discrete & Computational Geometry*, 25:551–570, 2003.

[7] L. Lovász and M. Plummer. *Matching Theory*. AMS Chelsea Publishing Series. North-Holland, 2009.

[8] P. M. Vaidya. Geometry helps in matching. *SIAM J. Comput.*, 18(6):1201–1225, 1989.

Burning Number for the Points in the Plane*

J. Mark Keil† Debajyoti Mondal‡ Ehsan Moradi§

Abstract

The burning process on a graph G starts with a single burnt vertex, and at each subsequent step, burns the neighbors of the currently burnt vertices, as well as one other unburnt vertex. The burning number of G is the smallest number of steps required to burn all the vertices of the graph. In this paper, we examine the problem of computing the burning number in a geometric setting. The input is a set of points P in the Euclidean plane. The burning process starts with a single burnt point of P, and at each subsequent step, burns all the points that are within a distance of one unit from the currently burnt points and one other unburnt point of P. The burning number of P is the smallest number of steps required to burn all the points of P. We call this variant *point burning*. We consider another variant called *anywhere burning*, where we are allowed to burn any point of the plane. We show that point burning and anywhere burning problems are both NP-complete, but $(2+\varepsilon)$ approximable for every $\varepsilon > 0$. Moreover, if we put a restriction on the number of burning sources that can be used, then the anywhere burning problem becomes NP-hard to approximate within a factor of $\frac{2}{\sqrt{3}} - \varepsilon$.

1 Introduction

Graph burning is a discrete process that propagates fire to burn all the nodes in a graph. In particular, the fire is initiated at a vertex of the graph and at each subsequent step, the fire propagates to the neighbors of the currently burnt vertices and a new unburnt vertex is chosen to initiate a fire. The vertices where we initiate fire are called the *burning sources*. The burning process continues until all the vertices are burnt. The *burning number* of a graph G is the minimum number of steps to burn all its vertices. Bonato et al. [4] introduced graph burning as a model of social contagion. The problem is NP-Complete even for simple graphs such as a spider or forest of paths [1].

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

†Department of Computer Science, University of Saskatchewan, mark.keil@usask.ca

‡Department of Computer Science, University of Saskatchewan d.mondal@usask.ca

§Department of Computer Science, University of Saskatchewan ehsan.moradi@usask.ca

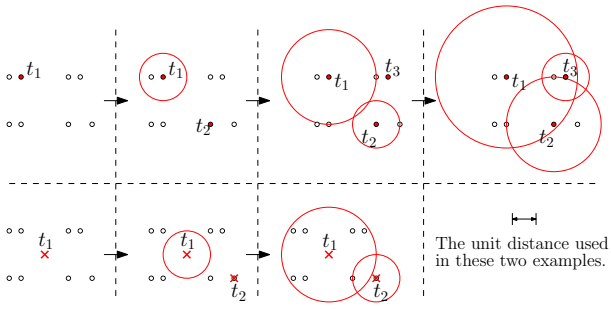


Figure 1: Illustration for (top) point burning and (bottom) anywhere burning. The burning sources are illustrated in labelled dots and cross marks, respectively.

In this paper we introduce burning number for the points in the plane. We consider two methods for burning: *point burning* and *anywhere burning*. Both problems take a set of points P as an input, and seek for the minimum number of steps to burn all points of P.

In the point burning model, we can initiate fire only at the given points. The burning process starts by burning one given point, and then at each subsequent step, the fire propagates to all unburnt points of the plane that are within one unit of any burnt point of the plane and a new unburnt given point is chosen to initiate the fire. Figure 1(top) illustrates this model. Note that we may not have an unburnt vertex at the last step.

In the anywhere burning model, we can start a fire anywhere on the plane, and at each subsequent step, the fire propagates to all unburnt points of the plane that are within one unit of any burnt point of the plane, and a new unburnt point is chosen to initiate the fire. Figure 1(bottom) illustrates this model.

In addition to being a natural generalization of graph burning, our proposed burning processes may potentially be used to model supply chain systems. A hypothetical example of how a burning process may model a supply chain management system is as follows. Consider a business that needs to maintain a continuous supply of perishable goods to a set of P locations. Each day it can manage to send one large shipment to a hub location that distributes the goods further to the nearby locations over time. The point burning considers only the points of P as potential hubs, whereas anywhere burning allows to create a hub at any point in the plane. The burning number indicates the minimum number of days needed to distribute the goods to all locations. For

example, in Figure 1(top), the hubs are t_1, t_2, and t_3, and the business can keep sending the shipments to the hubs after every three days in the same order.

1.1 Related Results

Finding the graph burning number is NP-Hard [1], but approximable within a factor of 3 [6]. These results have been improved very recently. García-Díaz et al. [10] have given a $(3 - 2/b)$-approximation algorithm where b is the burning number of the input graph. Mondal et al. [17] have shown the graph burning problem to be APX-hard, even in a generalized setting where $k = O(1)$ vertices can be chosen to initiate the fire at each step. They gave a 3-approximation algorithm for this generalized version [17]. Since the introduction of the graph burning problem [5], a rich body of literature examines the upper and lower bound on the graph burning number for various classes of graphs [20, 14, 8] as well as the parameterized complexity of computing the burning number [13]. We refer the reader to [2] for a survey on graph burning.

Researchers have also explored burning number for geometric graphs. Gupta et al. [11] examined square grid graphs and gave a 2-approximation algorithm for burning square grids. They also showed the burning number to be NP-Complete for connected interval graphs. Bonato et al. [3] considered the burning process on dynamic graphs, which are growing grids in the Cartesian plane with the center at the origin. They explore the proportion or density of burned vertices relative to the growth speed of the grid. Recently, Evans and Lin [9] have introduced polygon burning, where given a polygonal domain and an integer k, the problem seeks for k vertices such that the polygonal domain is burned as quickly as possible when burned simultaneously and uniformly from those k vertices. They gave a 3-approximation algorithm for polygon burning.

The anywhere burning problem that we introduced can be seen to be related to the nonuniform version of the *k-center* problem. Given a set of points, the goal of the k-center problem is to find the minimum radius R and a placement of k disks of radius R to cover all the given points. In the *nonuniform k-center* problem [7], given a set of points and a set of k numbers $r_0 \geq \ldots \geq r_{k-1}$, the goal is to find a minimum dilation α and a placement of k disks where the ith disk, $1 \leq i \leq k$, has radius αr_i and all the given points are covered. If the anywhere burning number of a set of points is k, then the nonuniform k-center problem with $r_i = i$ admits a solution with $\alpha = 1$.

1.2 Our Contribution

We introduce two discrete-time processes (i.e., point burning and anywhere burning) to burn the points in the plane, which naturally extend the graph burning model to the geometric setting. We prove that in both models, computing the burning number is NP-hard, and give polynomial-time $(2+\varepsilon)$-approximation algorithms. We then show that if we put a restriction on the number of burning sources that can be used, then the anywhere burning problem becomes NP-hard to approximate within a factor of $\frac{2}{\sqrt{3}} - \varepsilon$.

2 Approximating Burning Number

2.1 Point Burning

The *burning sources at the ith step* are all the vertices that we choose to initiate the fire from the beginning of the burning process to the ith step (including the ith step). We refer to the number of burning sources as B_i. The *maximum burning radius* R_i at the ith step of the burning process is the maximum radius over all burning sources. After the ith step of the burning process, the maximum burning radius is exactly $(i-1)$ and the number of burning sources is exactly i (except possibly for the last step). Therefore, if δ^* is the number of steps in the optimal solution, then the number of burning sources is at most δ^*, and the maximum burning radius is exactly $(\delta^* - 1)$. Hence for the ith step, we have the following.

$$\delta^* \geq i \geq B_i. \tag{1}$$

Theorem 1 *Given a set P of points in $\mathbb{R}^2$ and an $\varepsilon > 0$, one can compute a point burning sequence for P in polynomial time such that the length of the sequence is at most $(2+\varepsilon)$ times the point burning number of P.*

Proof. Let G_k be a unit disk graph where $k/2$ equals one unit, i.e., each vertex of G_k corresponds to a disk of radius $k/2$ in $\mathbb{R}^2$, and there is an edge between two vertices of G_k if their corresponding disks intersect. Consider the graph G_k on P where P represents the centers of the disks. We denote by D_k a *minimum dominating set* of G_k, i.e, the smallest set of vertices such that each vertex of G_k is either in D_k or a neighbor of a vertex in D_k. There exists PTAS to approximate D_k [19], i.e., D_k is approximable within a factor of $(1+\varepsilon)$ for every fixed $\varepsilon > 0$.

Let δ^* be the burning number for P. We now claim that δ^* must be at least $|D_{\delta^*-1}|$. Suppose for a contradiction that the burning number is strictly smaller than $|D_{\delta^*-1}|$ and let S be the corresponding burning sources. Since the maximum burning radius over S is at most $(\delta^* - 1)$, we could use $|S|$ disks, each of radius $(\delta^* - 1)$ to burn all the points. Hence, we could choose the disks corresponding to S as a dominating set for G_{δ^*-1}. This contradicts that D_{δ^*-1} is a minimum dominating set. Hence we have $\delta^* \geq |D_{\delta^*-1}|$.

We now iteratively guess the burning number δ from 1 to n, where $|P| = n$. For each δ, we construct $G_{\delta-1}$, and compute a $(1+\varepsilon)$ approximation $D'_{\delta-1}$ for $D_{\delta-1}$. If $\frac{|D'_{\delta-1}|}{(1+\varepsilon)}$, i.e., the lower bound on the burning sources, is strictly larger than δ, then it violates Equation 1 and our guess can be increased. We stop as soon as we have $\frac{|D'_{\delta-1}|}{(1+\varepsilon)} \leq \delta$. Since none of the previous guesses were successful, here we know that $\delta^* \geq \delta$.

To burn P, we first choose $D'_{\delta-1}$ as the burning sources and burn them in arbitrary order. We then keep burning another $(\delta-1)$ steps (or, stop early if all points are burnt). Since all the points are within the distance $(\delta-1)$ from some point in $D'_{\delta-1}$, all the points will be burnt. Since $|D'_{\delta-1}| \leq (1+\varepsilon)\delta$, and since $\delta^* \geq \delta$, the length of the burning sequence we compute is $|D'_{\delta-1}| + (\delta-1) \leq (1+\varepsilon)\delta^* + \delta^* = (2+\varepsilon)\delta^*$. □

2.2 Anywhere Burning

We leverage the discrete unit disk cover problem to obtain a $(2+\varepsilon)$-approximation for anywhere burning. The input of a *discrete unit disk cover* problem is a set of points P and a set of unit disks $\mathcal{U}$ in $\mathbb{R}^2$, and the task is to choose the smallest set $U \subseteq \mathcal{U}$ that covers all the points of P. There exists a PTAS for the discrete unit disk cover problem [18].

We relate the discrete unit disk cover problem to anywhere burning using the observation that there exists an optimal anywhere burning sequence where each burning source either coincides with a given point or lies at the center of some circle determined by two or three given points. More specifically, consider a burning source q with a burning radius r in an optimal anywhere burning process. Let S be the set of points burned by q. Let C be the smallest circle that covers all the points of S. Then we could choose a burning source at the center of C instead of at q and burn all points of S.

A $(1+\varepsilon)$-approximation for anywhere burning problem can now be obtained by iteratively guessing the anywhere burning number using the same technique as in Section 2.1 but using an approximation to the discrete unit set cover problem.

Theorem 2 *Given a set P of points in $\mathbb{R}^2$ and an $\varepsilon>0$, one can compute an anywhere burning sequence in polynomial time such that the length of the sequence is at most $(2+\varepsilon)$ times the anywhere burning number of P.*

Proof. Let P be the input to the anywhere burning problem. Note that Equation 1 holds also for anywhere burning. We now iteratively guess the anywhere burning number δ from 1 to n, where $|P| = n$. For each δ, we construct a set of $\binom{n}{3} + \binom{n}{2}$ disks, where each disk is of radius δ and is centered at the center of a circle determined by either two or three points of P. We compute a $(1+\varepsilon)$-approximation U'_δ for the discrete unit disk cover U_δ. If $\frac{|U'_\delta|}{(1+\varepsilon)}$, i.e., the lower bound on the burning sources, is strictly larger than δ, then it violates Equation 1 and our guess can be increased. We stop as soon as we have $\frac{|U'_\delta|}{(1+\varepsilon)} \leq \delta$. Here we know that $\delta^* \geq \delta$.

To burn all the points of P, we first choose U'_δ as the burning sources and burn them in arbitrary order. We then keep burning another $(\delta-1)$ steps (or, stop early if all points are burnt). Since all the points are within the distance $(\delta-1)$ from some point in U'_δ, all the points will be burnt. Since $|U'_\delta| \leq (1+\varepsilon)\delta$, and since $\delta^* \geq \delta$, the length of the burning sequence we compute is $|U'_\delta| + (\delta-1) \leq (1+\varepsilon)\delta^* + \delta^* = (2+\varepsilon)\delta^*$. □

3 NP-hardness

In this section we present the hardness results.

3.1 Point Burning

Consider a decision version of the point burning problem where given a set of points and an integer b, the task is to decide whether there is a burning sequence that burns all the points in at most b steps. This decision version of the point burning problem is in NP because given a sequence of burning sources, in polynomial time one can simulate the burning process to check whether all the points are burnt. We now consider the hardness.

The graph burning number problem is NP-hard even for a forest of paths [4]. To prove the NP-hardness of the point burning one can easily reduce the path forest burning problem into the point burning problem as follows.

Let I be an instance of the path forest burning problem and let $L_1, \ldots, L_t$ be the paths in I. We draw the vertices of each path L_i, $1 \leq i \leq t$, along the x-axis in the (left-to-right) order they appear on the path with unit length distance between consecutive vertices. We ensure a gap of $(2n+1)$ units between consecutive paths, where n is the number of vertices in the forest. The point burning number for the vertices of the paths is at most n. Since we can only burn the points (equivalently, vertices) in the point burning model, any point burning process can be seen as a graph burning and vice versa. Hence we have the following theorem.

Theorem 3 *The point burning problem is NP-complete.*

3.2 Anywhere Burning

Similar to point burning, the decision version of the anywhere burning problem is in NP because given a sequence of burning sources, in polynomial time one can simulate the burning process to check whether all the points are burnt.

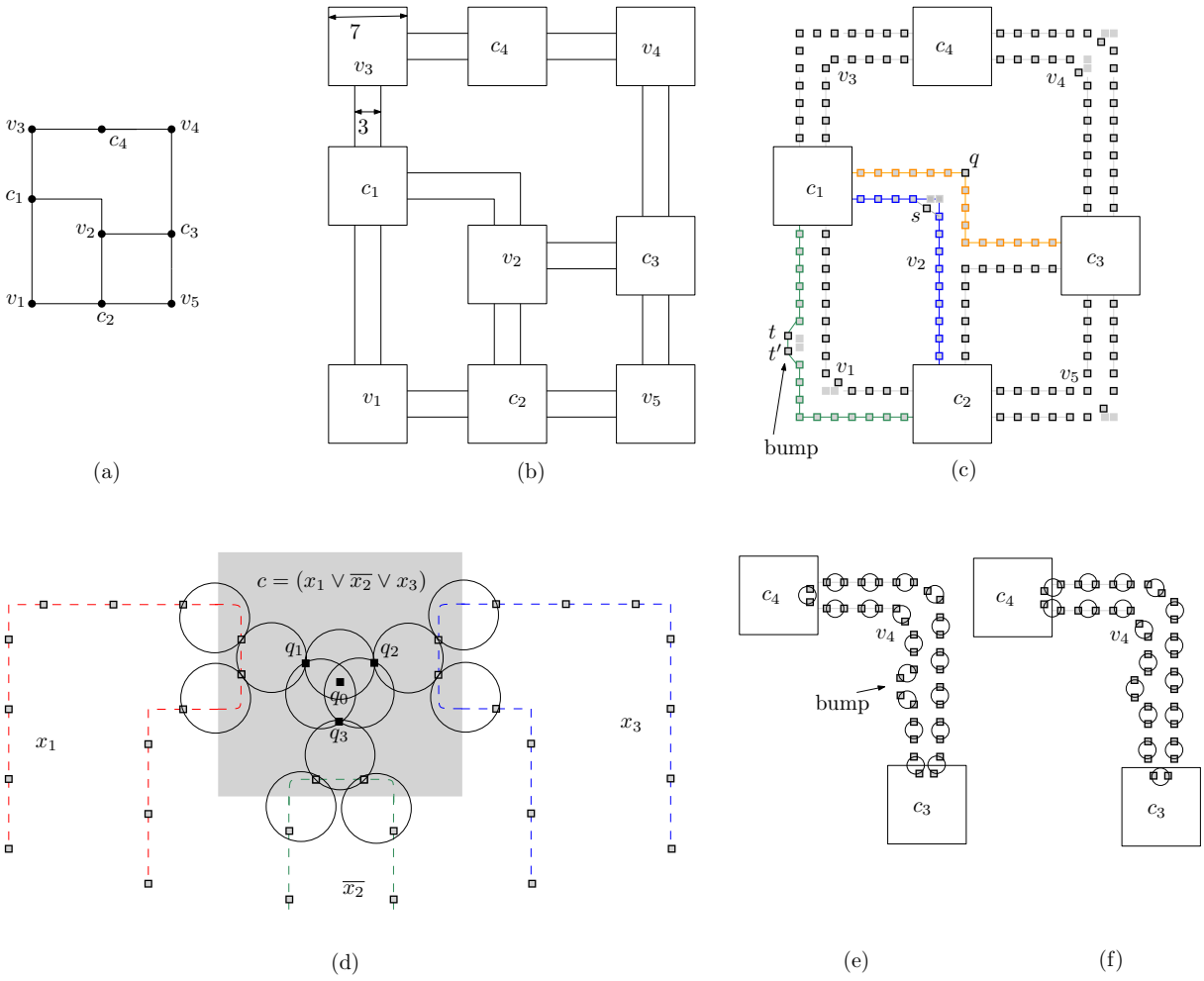


Figure 2: Illustration for the construction of the point set P.

To show the hardness we can use almost the same hardness reduction that we used for point burning. Let I be an instance of the path forest burning problem and let P be the corresponding point set we constructed in Section 3.1. If the burning number for I is b, then we can simulate the same burning process to burn all points of P in b steps. If P admits an anywhere burning within b steps, then for each burning source q, we choose the nearest point q' of P as the burning source. By the construction of P, the distance between q and q' is at most 0.5. Since the burning radius of q is an integer, a burning source with the same radius at q' will burn the same set of points as that of q. Therefore, if we now burn the chosen points of P in the order corresponding to the anywhere burning sequence, this simulates a graph burning process on I and burns all the vertices within b steps. We thus have the following theorem.

Theorem 4 *The anywhere burning problem is NP-complete.*

Hardness of Approximation with Bounded Burning Sources: If we put a restriction on the number of burning sources that can be used, then we can modify the above hardness proof to derive an inapproximability result on the number of burning steps.

To show the hardness of approximation, we first give a different NP-hardness proof for anywhere burning. Here we reduce the NP-hard problem planar exactly 3-bounded 3-SAT [16]. The input of the problem is a 3-CNF formula where each variable appears in exactly 3 clauses, each clause contains at least two and at most three literals, and the corresponding SAT graph (a graph with clauses and variables as vertices, and clause-variable incidences as edges) is planar. The task is to decide whether there exists a truth value assignment for the variables that satisfies all the clauses.

Let I be an instance of the planar exactly 3-bounded 3-SAT and let G be the corresponding SAT graph. We now show how to compute a point set P and an integer b such that P can be burned within b steps if and only if I admits an affirmative solution. Our construction is inspired by an NP-hardness reduction for the k-center

problem [15], but contains nontrivial details due to variable sizes for the burning radii. We will use the concept of β-*disk*, which is a disk of radius β.

We first compute an orthogonal planar drawing D of G where the vertices are represented as grid points and edges as orthogonal polylines (Figure 2(a)). Every planar graph with maximum degree three has such an orthogonal planar grid drawing inside a square of side length $\lfloor n/2 \rfloor$ [12]. We then scale up the drawing and replace the vertices with squares and edges with parallel orthogonal lines (Figure 2(b)). We will refer to this new representation for an edge as a *tunnel*. We ensure that each square is of side length 7 units and the pair of parallel line segments of a tunnel are 3 units apart. We then replace the square for each variable by joining the tunnels incident to it (Figure 2(c)).

Creating Points for Variables and Clauses: We now add some points along the boundary of the tunnels as follows. Let L be a polygonal line (determining a side of the tunnel) from a clause to another clause (e.g., the orange line Figure 2(c)). We place points from both ends such that no three points can be covered by a 1.10-disk. The first point is placed at one unit distance from the boundary of the square representing the clause, and then each subsequent point is placed two units apart from the previous one. If the two sequences of points from the two ends of L meet at a common point (e.g., the point q in Figure 2(c)), then nothing else needs to be done. If the two sequences does not meet at a common point and a bend point is available, then we create a new point instead of creating two points that are one unit apart (e.g., the point s in Figure 2(c)). This ensures the property that no three points can be covered by a 1.10-disk. Note that instead of modifying a bend, one can also create a 'bump' on L to ensure this property, as illustrated with the points t, t' in Figure 2(c).

For each square representing a clause, we add 2 points for each variable incident to it and an additional 4 points q_0, q_1, q_2, q_3, as illustrated in Figure 2(d). We refer to the points q_0, q_1, q_2, q_3 as the *clause points*. Some black unit disks are drawn to illustrate the configuration of these points. The key property here is that no 1.10-disk can cover all clause points, but if we exclude one clause point among $\{q_1, q_2, q_3\}$, then the remaining clause points can be covered using a unit disk.

Note that each variable now corresponds to a sequence of points forming a *loop*. We create some more bumps to ensure that each variable contains an even number of points. This allows us to have two ways of covering the loop by using only unit disks by taking alternating pairs, as illustrated in Figures 2(e)–(f). Later, we will relate such covering to burning and if both variable-loop points inside the clause gadget are covered by the same unit disk, then will set that literal to true. Therefore, we add some more points to ensure consistency. For example, assume that in Figures 2(e)–(f), the clauses c_4 and c_3 contain the literals $\overline{v_4}$ and v_4, respectively. We create a bump so that if both variable-loop points inside the gadget of c_4 are covered by a single unit disk, then the two variable-loop points inside the gadget of c_3 will be covered by two different unit disks, and vice versa.

Since the width and height of the drawing is of size $O(n)$, the total number of points is $O(n^2)$. We will denote by N_v and N_c the points that we created for the variables and clauses, respectively.

Creating Points to Accommodate Burning Process: We now scale up the drawing by r units, where we set r to be $10\left(\frac{|N_v|}{2} + \frac{|N_c|}{4}\right)$. Let the resulting drawing be D'. Consequently, all the above covering properties for unit disks and 1.10-disks now hold for r-disks and $1.10r$-disks, respectively.

We now create r points w_i, where $1 \leq i \leq r$, along a horizontal line such that each point is far from the rest of the points by at least $3r$ units.

We will refer to the points created in this step as the *outlier points* and denote them by N_t. Note that the points of N_t lie outside of D'.

From 3-SAT to Burning Number: We now show that if the 3-SAT instance I admits an affirmative solution, then the point set $(N_v \cup N_c \cup N_t)$ can be burned in $1.10r$ steps.

In the first $0.10r$ steps we initiate $0.10r$ burning sources inside D' and then initiate r burning sources at the *outlier points*. After this, the minimum radius of the burned area for any burning source started within D' is at least r and the maximum radius for such sources is $(1.10r-1)$. The burning sources inside D' can be seen as β-disks where $\beta \in [r, 1.10r]$.

For each true literal, we cover the corresponding two variable points and the nearest clause point by initiating a single burning source (e.g., Figures 3(a)–(c)). We then burn the variable loops by initiating burning sources for alternating pairs of points. This takes $|N_v|/2$ burning sources. Since all clauses are satisfied, for each clause, at least one of the clause points from $\{q_1, q_2, q_3\}$ will be allocated to burn along with a pair of variable-loop points. Therefore, each clause now requires one burning source to ensure the burning of all its clause points. Hence the total number of burning sources we use within D' is $0.10r = \left(\frac{|N_v|}{2} + \frac{|N_c|}{4}\right)$. The set N_t contains r points where no two of them can be covered by a $1.10r$-disk. It is straightforward to burn them in r steps. Therefore, the total number of steps required is $1.10r$.

From Burning Number to 3-SAT: We now show that if the point set $(N_v \cup N_c \cup N_t)$ can be burned in $1.10r$ steps, then the 3-SAT instance I admits an affirmative solution.

Since the set N_t contains r points where no two of

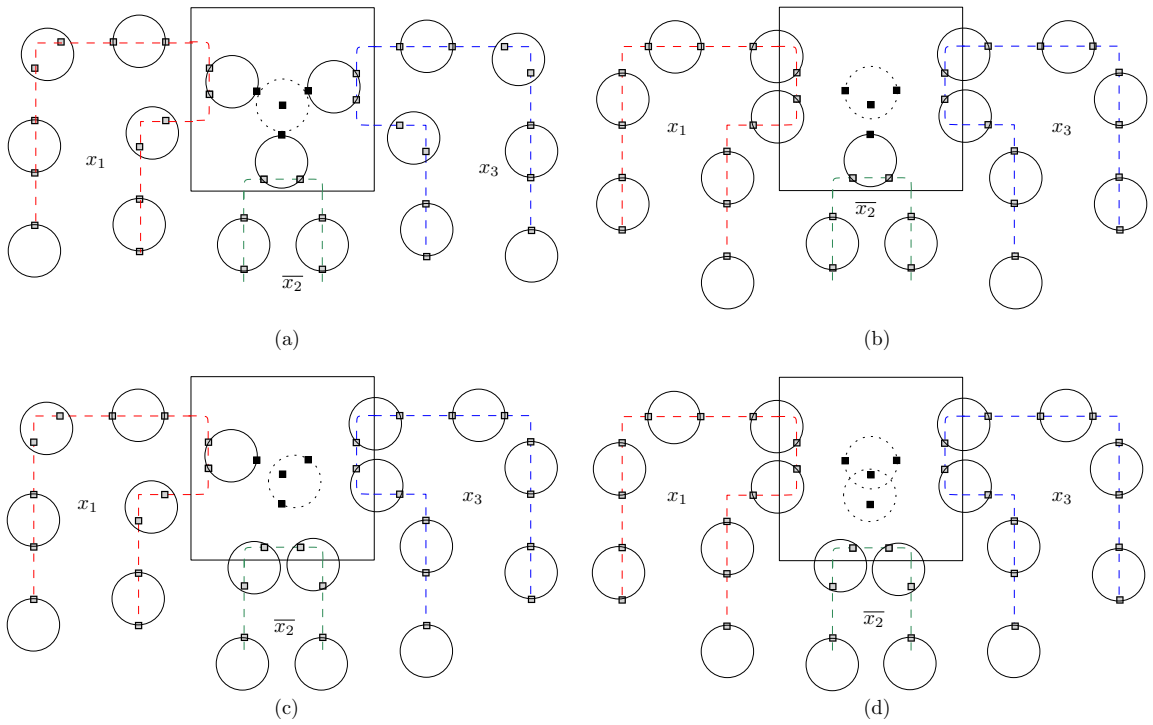


Figure 3: Illustration for the reduction. (a) $x_1 = True$, $x_2 = False$, $x_3 = True$. (b) $x_1 = False$, $x_2 = False$, $x_3 = False$. (c) $x_1 = True$, $x_2 = True$, $x_3 = False$. (d) $x_1 = False$, $x_2 = True$, $x_3 = False$.

them can be covered by a $1.10r$-disk, any burning sequence would need r burning sources outside of D'. Since there are at most $1.10r$ steps to burn all the points, we are left with at most $0.10r = \left(\frac{|N_v|}{2} + \frac{|N_c|}{4}\right)$ burning sources inside D'. Note that none of these burning sources can have a radius larger than $(1.10r - 1)$. By the construction of the variable-loop points, no three points can be covered by a $1.10r$-disk. Therefore, the variable-loop requires at least $|N_v|/2$ burning sources. If none of q_1, q_2, q_3 are burned along with the variable-loop points, then a clause gadget requires two burning sources (e.g., Figures 3(d)). Otherwise, each clause gadgets requires at least one burning source to ensure all clause points are burned even if q_1, q_2, q_3 are all burned along with the variable-loop points (e.g., Figures 3(a)). Since there are $\frac{|N_c|}{4}$ clause gadgets and exactly that many burning sources remaining, each clause gadget will have exactly one from the remaining burning sources. Since a $1.10r$-disk cannot cover all four clause points of a clause gadget, one of them must be burned together with a pair of variable-loop points. We set the corresponding literal to true. The construction of the variable-loop ensures the consistency of the truth value assignment for each variable at different clauses.

Inapproximability Factor (with Bounded Burning Sources): Assume that we are only allowed to initiate $\left(\frac{|N_v|}{2} + \frac{|N_c|}{4}\right) + |N_t|$ fires. We change r to be $10^{\delta}\left(\frac{|N_v|}{2} + \frac{|N_c|}{4}\right)$, where δ is a constant. We now can burn $(N_v \cup N_c \cup N_t)$ in $(1 + 10^{-\delta})r$ steps by first initiating $10^{-\delta}r = \left(\frac{|N_v|}{2} + \frac{|N_c|}{4}\right)$ burning sources inside D' and then r burning sources to burn the points in N_t. Since our reduction can be carried out with $1.10r$-disks, we can continue burning for $(1.10r - (1 + 10^{-\delta})r)$ more steps. Therefore, we obtain an inapproximability factor of $\frac{1.10}{(1+10^{-\delta})}$, i.e., $(1.10 - \varepsilon)$, where ε can be made arbitrarily small by choosing a large value for δ.

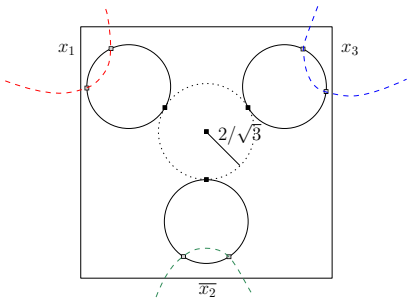


Figure 4: Illustration for the modified clause gadget.

Although for simplicity we used an orthogonal setting where variable loops enter a clause gadget either horizontally or vertically, we could slightly change the construction using curves (similar to [15]) such that they make 120° angles at the clause gadget (Figure 4). This

allows us to carry out the reduction using a $\frac{2}{\sqrt{3}}$-disks and thus to have an inapproximability factor of $\frac{2}{\sqrt{3}} - \varepsilon$.

Corollary 5 *The anywhere burning problem with a restriction on the number of burning sources that can be used is NP-hard to approximate with a factor of $\frac{2}{\sqrt{3}} - \varepsilon$, for every fixed $\varepsilon > 0$.*

4 Conclusion

In this paper, we introduced two burning processes — point burning and anywhere burning — to burn a set of points in the Euclidean plane. We proved that computing the burning number for these processes are NP-complete and gave approximation algorithms for them. We showed that inapproximability results can be derived for anywhere burning if only a restricted number of burning sources are allowed. Hence a natural future research direction to explore is to design efficient approximation algorithms for computing the burning number as well as to establish better inapproximability results.

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, K. Gunderson, and A. Shaw. Burning the plane. *Graphs Comb.*, 36(5):1311–1335, 2020.

[4] A. Bonato, J. C. M. Janssen, and E. Roshanbin. Burning a graph as a model of social contagion. In A. Bonato, F. C. Graham, and P. Pralat, editors, *Algorithms and Models for the Web Graph (WAW)*, volume 8882 of *LNCS*, pages 13–22. Springer, 2014.

[5] A. Bonato, J. C. M. Janssen, and E. Roshanbin. How to burn a graph. *Internet Mathematics*, 12(1-2):85–100, 2016.

[6] A. Bonato and S. Kamali. Approximation algorithms for graph burning. In *International Conference on Theory and Applications of Models of Computation*, pages 74–92. Springer, 2019.

[7] D. Chakrabarty, P. Goyal, and R. Krishnaswamy. The non-uniform k-center problem. In I. Chatzigiannakis, M. Mitzenmacher, Y. Rabani, and D. Sangiorgi, editors, *Proc. of the 43rd International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 55 of *LIPIcs*, pages 67:1–67:15, 2016.

[8] S. Das, S. R. Dev, A. Sadhukhan, U. kant Sahoo, and S. Sen. Burning spiders. In *Conference on Algorithms and Discrete Applied Mathematics*, pages 155–163. Springer, 2018.

[9] W. Evans and R. Lin. The polygon burning problem. In *International Conference and Workshops on Algorithms and Computation*, LNCS, pages 123–134. Springer, 2022.

[10] J. Garcia-Diaz, J. C. P. Sansalvador, L. M. Rodríguez-Henríquez, and J. A. Cornejo-Acosta. Burning graphs through farthest-first traversal. *IEEE Access*, 10:30395–30404, 2022.

[11] A. T. Gupta, S. A. Lokhande, and K. Mondal. Burning grids and intervals. In A. Mudgal and C. R. Subramanian, editors, *Proc. of the 7th International Conference on Algorithms and Discrete Applied Mathematics (CALDAM)*, volume 12601 of *LNCS*, pages 66–79. Springer, 2021.

[12] G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16(1):4–32, 1996.

[13] Y. Kobayashi and Y. Otachi. Parameterized complexity of graph burning. In Y. Cao and M. Pilipczuk, editors, *Proc. of the 15th International Symposium on Parameterized and Exact Computation (IPEC)*, volume 180 of *LIPIcs*, pages 21:1–21:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[14] H. Liu, X. Hu, and X. Hu. Burning numbers of path forests and spiders. *Bulletin of the Malaysian Mathematical Sciences Society*, 44(2):661–681, 2021.

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

[16] M. Middendorf and F. Pfeiffer. On the complexity of the disjoint paths problems. *Comb.*, 13(1):97–107, 1993.

[17] D. Mondal, N. Parthiban, V. Kavitha, and I. Rajasingh. Apx-hardness and approximation for the k-burning number problem. In R. Uehara, S. Hong, and S. C. Nandy, editors, *Proc. of the 15th International Conference and Workshops on Algorithms and Computation (WALCOM)*, volume 12635 of *LNCS*, pages 272–283. Springer, 2021.

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

[19] T. Nieberg and J. L. Hurink. A PTAS for the minimum dominating set problem in unit disk graphs. In T. Erlebach and G. Persiano, editors, *Proc. of Approximation and Online Algorithms (WAOA)*, volume 3879 of *LNCS*, pages 296–306. Springer, 2005.

[20] K. Sim, T. S. Tan, and K. Wong. On the burning number of generalized petersen graphs. *Bulletin of the Malaysian Mathematical Sciences Society*, 41, 11 2017.

Uniformly Monotone Partitioning of Polygons Revisited*

Hwi Kim† Jaegun Lee‡ Hee-Kap Ahn§

Abstract

Partitioning a polygon into simple pieces is a fundamental problem in computational geometry with a long history. In this paper, we revisit the problem of partitioning a simple polygon P with n vertices (including R reflex vertices) and no holes into a minimum number of uniformly monotone subpolygons using open line segments drawn inside P. We present an $O(nR \log n + R^5)$-time algorithm for the problem by adding diagonals between pairs of vertices of P. When Steiner points can be placed on the boundary of P and the subdivision is formed by adding diagonals between pairs of vertices of P including Steiner points, we present an $O(n + R^5)$-time algorithm. We present an $O(n + R^4)$-time algorithm when Steiner points can be placed anywhere in P. Our algorithms improve upon the previously best ones for polygons with a small number of reflex vertices relative to the total number of vertices. We also present simple and efficient 2-approximation algorithms.

1 Introduction

Partitioning a polygon into disjoint simple pieces, such as triangles, trapezoids, convex polygons, and star-shaped polygons, is an important and fundamental problem in computational geometry [2, 3, 7, 8, 11, 12]. A classic and typical example is the triangulation of a simple polygon in the plane [2].

A simple polygon is called *monotone with respect to a line* ℓ if for any line ℓ' perpendicular to ℓ the intersection of the polygon with ℓ' is connected. The problem of partitioning a polygon into *monotone* subpolygons has been well studied [7, 9]. Many geometric algorithms run faster asymptotically for monotone polygons than for more general ones, and it is often straightforward to implement algorithms for monotone polygons [12].

In this paper, we study the problem of partitioning a simple polygon with no holes into a minimum number of uniformly monotone subpolygons using diagonals (open line segments lying in the interior of the polygon) between pairs of vertices of the polygon. A partition is *uniformly monotone with respect to a line* ℓ if every subpolygon in the partition is monotone with respect to ℓ. Among all uniformly monotone partitions, we wish to compute one that minimizes the number of subpolygons in the partition. Below we define the problem formally.

Minimum Uniformly Monotone Partition: Given a simple polygon P with n vertices (including R reflex vertices) and no holes, find a pair (ℓ^*, P^*) of a line ℓ^* and a uniformly monotone partition P^* of P with respect to ℓ^* such that the number of subpolygons in P^* is the minimum among all pairs (ℓ, P) of lines ℓ and uniformly monotone partitions P with respect to ℓ.

We call such a pair (ℓ^*, P^*) a *minimum partition-pair*, and such a partition P^* a *minimum partition* of P. In the rest of the paper, we may simply refer to the minimum uniformly monotone partition problem as the minimum partition problem.

We also consider two variants of the minimum partition problem with additional vertices, called *Steiner points*. Steiner points are added as part of the partition to reduce further the number of subpolygons. There are two ways of adding Steiner points, either placing them only on the boundary of P or placing them anywhere in P. See Figure 1.

1.1 Previous works

Lee and Preparata [9] gave an $O(n \log n)$-time plane sweep algorithm to partition a simple polygon with n vertices into monotone pieces, but not necessarily into a minimum number of monotone pieces. Liu and Ntafos [10] studied the minimum partition problem and gave an $O(nR^2 \log n + nR^3 + R^5)$-time algorithm for the problem without using Steiner points. Using Steiner points lying on the boundary of the polygon, they gave an $O(nR^3 \log n + R^5)$-time algorithm to compute a minimum partition.

*This research was partly 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)).

†Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang, Korea. hwikim@postech.ac.kr

‡Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang, Korea. jagunlee@postech.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

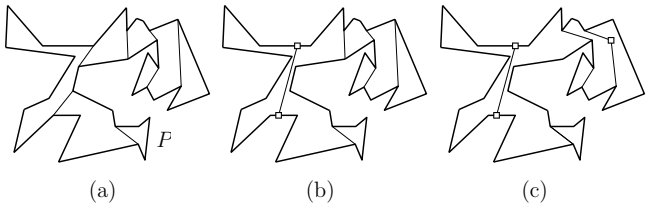


Figure 1: Uniformly monotone partitions of P with respect to the y-axis. (a) A minimum partition (8 pieces) with no Steiner points. (b) A minimum partition (7 pieces) with two Steiner points (white squares) on the boundary of P. (c) A minimum partition (6 pieces) with three Steiner points (white squares) in P.

Wei et al. [13] considered the minimum partition problem for a polygon with n vertices and h holes. Using Steiner points lying in the polygon, they presented an $O(K(n \log n + h \log^3 h))$-time algorithm to compute a minimum partition of the polygon, where K is the number of edges of the polygon's visibility graph.

For the problem of partitioning a polygon into a minimum number of subpolygons such that each subpolygon is monotone with respect to some line, Keil [6] gave an $O(n^4 R)$-time algorithm for polygons with n vertices and no holes without using Steiner points. They also showed that the decision version of the problem is NP-complete for polygons with holes but without using Steiner points.

1.2 Our results

Our results are fourfold. First, we present an $O(nR \log n + R^5)$-time algorithm for the minimum partition problem with no Steiner points. The algorithm by Liu and Ntafos [10] is claimed to take $O(nR^2 \log n + nR^3 + R^5)$ time. But this may not hold for simple polygons with vertices of interior angle π and the running time may increase to $O(n^2 + nR^3 + R^5)$. We give more details in Section 3.1. Our algorithm runs faster than their algorithms for $R = o(n^{1/2})$ while the running times are the same asymptotically for $R = \Omega(n^{1/2})$.

Second, we present an $O(n + R^5)$-time algorithm for the minimum partition problem when Steiner points can be placed on the boundary of P. Observe that the algorithm takes only $O(R^5)$ time in addition to the time linear to the input size. It runs faster than the $O(nR^3 \log n + R^5)$-time algorithm by Liu and Ntafos for $R = o(n^{1/2} \log^{1/2} n)$ while the running times are the same asymptotically for $R = \Omega(n^{1/2} \log^{1/2} n)$.

Third, we present an $O(n + R^4)$-time algorithm for the minimum partition problem when Steiner points can be placed anywhere in P. It takes only $O(R^4)$ time in addition to the time linear to the input size, and thus it runs fast for R small relative to n. The algorithm by Wei et al. runs in $O(Kn \log n)$ time for this problem, but K can be $\Theta(n^2)$.

Finally, we present simple factor-2 approximation algorithms for the minimum partition problem, that is, the number of subpolygons in the partition returned by our algorithm is at most twice the number of subpolygons in a minimum partition. We present an $O(n \log n)$-time algorithm with no Steiner points, and an $O(n + R \log n)$-time algorithm using Steiner points lying on the boundary of P. The solution returned by the latter algorithm is also a 2-approximation for the case that Steiner points can be placed anywhere in P.

Sketches of our algorithms. Our algorithms for the minimum partition problem are based on the work by Liu and Ntafos [10]. Given a line ℓ, their algorithm first computes *peaks*, each of which is a vertex of P and a source of local non-monotonicity with respect to ℓ. Then the algorithm removes the peaks using a minimum number of non-intersecting diagonals between vertices and obtains a minimum partition of P with respect to ℓ in $O(n \log n + nR + R^3)$ time.

To compute a minimum number of non-intersecting diagonals between vertices, they use a *circle graph* G. The vertices of G correspond to the peaks in order along the boundary of P. There is a *chord* between two vertices of G if and only if their corresponding peaks can be removed by adding the diagonal between them. Since each peak can be removed by a diagonal and each diagonal removes at most two peaks, a minimum partition of P can be obtained by computing a *maximum independent chord set (MICS)* of G in $O(m^3)$ time for m vertices in G. By running this algorithm for each of $O(R^2)$ distinct lines defined by pairs of reflex vertices, Liu and Ntafos compute a minimum partition-pair of P in $O(nR^2 \log n + nR^3 + R^5)$ time.

Our algorithms also construct circle graphs and compute their MICSs. Our algorithms are different to the ones by Liu and Ntafos in three aspects. The first difference is that our algorithms use a data structure for geodesic queries while the algorithms by Liu and Ntafos compute visibility polygons repeatedly for vertices and edges in peaks. From this, we reduce the time for computing diagonals.

The second difference is that our algorithm maintains and updates the vertices and the chords of the circle graph efficiently over $O(R^2)$ distinct lines. The algorithm by Liu and Ntafos constructs the circle graph from scratch for each of the lines. Our algorithm uses certain coherence between the circle graphs induced by two lines of consecutive orientations among the orientations defined by pairs of reflex vertices. Imagine that a line ℓ rotates. Then the circle graph G induced by ℓ changes at certain orientations: a vertex of G is removed, a new vertex is inserted to G, a chord of G is removed, or a new chord is inserted to G. By keeping

track of these changes in the order of orientations using a priority queue, our algorithm maintains and updates the circle graph and the minimum number of monotone subpolygons efficiently for $O(R^2)$ lines.

The third difference is that our algorithm computes MICSs without computing their corresponding partitions explicitly while the algorithm by Liu and Ntafos computes a partition of P explicitly for each of $O(R^2)$ distinct lines. Whenever the circle graph is updated, our algorithm computes an MICS, but it does not compute the corresponding partition explicitly. Instead, it maintains and updates the minimum number of monotone subpolygons for the MICS. Once our algorithm is done with $O(R^2)$ lines, it computes a minimum partition P^* using the line corresponding to the minimum number of monotone subpolygons. This leads to further improvements on the time complexity.

When Steiner points are allowed to be placed anywhere in the polygon, the cardinality of an MICS of the circle graph equals the cardinality of the maximum matching of the circle graph. We show that the circle graph is a *chordal bipartite graph* under updates, and thus its maximum matching can be computed efficiently.

Our approximation algorithms achieve factor 2 by computing the minimum number of peaks of P among all orientations. This is because the number of subpolygons in a minimum partition cannot be less than half of the number of peaks.

2 Preliminaries

We denote by P the input simple polygon with n vertices (including R reflex vertices) and no holes. We assume that P is given as a sequence of vertices in clockwise order along its boundary. For a compact set X, we use ∂X to denote the boundary of X. For a point p, we denote by $y(p)$ the y-coordinate of p.

A *diagonal* is an open line segment that connects two vertices of P and lies in the interior of P. A set of non-intersecting diagonals induces a *partition* of P into subpolygons.

For a partition P of P, we denote by $|\mathsf{P}|$ the number of subpolygons in P. A *minimum partition* of P with respect to a given line ℓ, called the *scan line*, is a uniformly monotone partition P^*_ℓ with respect to ℓ such that $|\mathsf{P}^*_\ell| \leq |\mathsf{P}_\ell|$ for any uniformly monotone partition P_ℓ of P with respect to ℓ. A minimum partition of P is a uniformly monotone partition P^* of P with respect to some scan line such that $|\mathsf{P}^*| \leq |\mathsf{P}|$ for any minimum partition P of P with respect to some scan line.

For two points p and q in the plane, we denote by pq the line segment connecting them. Two points $p, q \in P$ are *visible* to each other if pq is contained in P. Two edges e_1, e_2 of P are visible to each other if there are points $p \in e_1, q \in e_2$ such that p and q are visible to each other.

A *super-vertex* of P is a maximal chain of consecutive and collinear vertices of P. A super-vertex is *reflex* if both its endpoints are reflex vertices.

For a fixed scan line ℓ, a *peak* is a reflex vertex or a reflex super-vertex v of P such that both neighboring vertices of v lie in one side of the line through v and perpendicular to ℓ. A peak with respect to ℓ is called a *normal-peak* if it is a reflex vertex of P, and a *super-peak* if it is a reflex super-vertex of P. The peaks are sources of local non-monotonicity with respect to ℓ. See Figure 2(a).

A super-peak and a normal-peak are visible to each other if the super-peak has a vertex that is visible from the normal-peak. Two super-peaks are visible to each other if there are two vertices, one from each super-peak, that are visible to each other. See Figure 2(b).

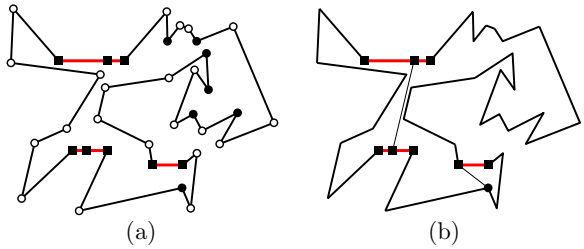


Figure 2: (a) The vertices of P are shown as white disks, black disks, and black squares. Black disks are normal peaks and red chains (connecting black squares) are super-peaks with respect to the y-axis. (b) Two super-peaks are visible to each other, and a normal-peak and a super-peak are visible to each other.

Missing proofs and details can be found in the full version of the paper.

3 Maximum independent chord set of a circle graph

In this section, we describe the approach of using a *circle graph* given by Liu and Ntafos [10], and give an outline of their algorithm using the approach.

We fix the scan line to be the y-axis throughout this section unless stated otherwise. We call a normal-peak v a *top normal-peak* if both neighboring vertices of v lie above v, and a *bottom normal-peak* otherwise. Similarly, we call a super-peak a *top super-peak* if both neighboring vertices of the chain (super-peak) lie above it, and a *bottom super-peak* otherwise. A diagonal is *full* if it connects a top peak u and a bottom peak v with $y(u) \geq y(v)$. Thus, by adding a full diagonal, two peaks are removed simultaneously.

A minimum partition of P with respect to a scan line can be obtained by adding a minimum number of non-intersecting diagonals that remove all peaks of P with

respect to the scan line. Observe that each peak can be removed by a diagonal and each diagonal removes at most two peaks. Thus, a minimum partition is obtained by finding a maximum number of full diagonals such that no peak is incident to more than one full diagonal. The algorithm by Liu and Ntafos computes a maximum number of full diagonals using a *circle graph* G. The vertices of G correspond to the peaks in one-to-one mapping in order along ∂P. There is a *chord* between two vertices of G if and only if there is a full diagonal between the peaks corresponding to the vertices. A minimum partition of P can be obtained by computing an MICS of G.

Lemma 1 ([10, 13]) *The number of subpolygons in a minimum partition of P with respect to a fixed scan line is $N_p - N_m + 1$, where N_p is the total number of peaks of P with respect to the scan line and N_m is the cardinality of an MICS of the circle graph induced by P and the scan line.*

3.1 Minimum partition algorithms by Liu and Ntafos

Using Lemma 1, Liu and Ntafos [10] gave an $O(n \log n + nR + R^3)$-time algorithm that computes a minimum partition of P with respect to a fixed scan line using no Steiner points. Their algorithm first reduces the minimum partition problem to the problem of computing an MICS of the circle graph G induced by P and the scan line. The algorithm finds all the top and bottom peaks as vertices of G. Then it computes the visibility polygon of each top peak using an $O(n)$-time algorithm for finding the visibility polygon of a point in a simple polygon with no holes. From the visibility polygon of a top peak u, it identifies all bottom peaks v such that the open line segment connecting u and v is a full diagonal. Each such full diagonal contributes a chord to G. Thus, G can be computed in $O(nR)$ time.

They gave an $O(m^3)$-time algorithm to compute an MICS of a circle graph with m vertices. Since there are $O(R)$ vertices in G, the algorithm computes an MICS of G in $O(R^3)$ time.

After the algorithm partitions P into subpolygons using the set of full diagonals corresponding to the MICS, no subpolygon in the partition has a full diagonal. The algorithm removes all the remaining peaks in the partition in $O(n \log n)$ time using the subdivision algorithm by Lee and Preparata [9]. Thus, in total it takes $O(n \log n + nR + R^3)$ time.

Liu and Ntafos extend their algorithm to find a minimum partition-pair of P. There are $O(R^2)$ disjoint intervals of orientation, which are defined by pairs of reflex vertices, such that the set of peaks and the set of full diagonals remain the same for any orientation in an interval. With this observation, Liu and Ntafos claim that a minimum partition-pair of the input polygon can be computed in $O(nR^2 \log n + nR^3 + R^5)$ time, by running the algorithm for a fixed scan line for $O(R^2)$ directions, once for each orientation interval.

However, their algorithms may take more time than what they claim for simple polygons with vertices of interior angle π. Their algorithm for a fixed scan line works as follows. For each vertex u in a super-peak, it computes the visibility polygon of u and identifies the vertices v such that there is a full diagonal connecting u and v. Since there can be $\Theta(n)$ non-reflex vertices in $O(R)$ super-peaks, the time for computing the visibility polygons increases to $O(n^2)$. Then their algorithm takes $O(n^2 + R^3)$ time for a fixed scan line, and $O(n^2 + nR^3 + R^5)$ time for finding a minimum partition-pair of P.

When Steiner points are allowed to be placed on ∂P, there is a full diagonal pq between a top super-peak u and a bottom super-peak v for the y-axis scan line and a point $p \in u$ and a point $q \in v$ if $y(p) \geq y(q)$, and uv is a diagonal. See Figure 1(b). The algorithm by Liu and Ntafos considers each super-peak as an edge of P and computes the visibility polygon of the edge. Using the visibility polygons, it computes pairs of super-peaks visible to each other and Steiner points lying on them together with full diagonals connecting them. The rest of the algorithm is the same with the algorithm for no Steiner points. Using an $O(n \log n)$-time algorithm for finding the visibility polygon of an edge in a simple polygon with no holes, their algorithm takes $O(nR \log n + R^3)$ time for a fixed scan line, and $O(nR^3 \log n + R^5)$ time for finding a minimum partition-pair.

4 Minimum uniformly monotone partition

We present our algorithms for the minimum partition problem. In Section 4.1, we describe our approach for maintaining the circle graph efficiently. In Sections 4.2-4.4, we present algorithms, one using no Steiner points, one using Steiner points lying on ∂P, and one using Steiner points lying anywhere in P.

4.1 Maintaining the circle graph

We show certain coherence between the circle graphs induced by two lines of consecutive orientations among the orientations defined by pairs of reflex vertices. We use this to compute a minimum partition of P efficiently, provided that we have an algorithm, denoted by fd-algo, for computing a minimum partition of P with respect to a fixed scan line.

Imagine that the scan line ℓ rotates around the origin in clockwise by π, starting from the y-axis. Let ℓ_θ denote the scan line of orientation $\theta \in [0, \pi)$. The set of peaks and the set of full diagonals with respect to ℓ_θ may change at certain discrete orientations θ at which one of the following occurs.

- *Diagonal event*: a diagonal connecting a top normal-peak and a bottom normal-peak becomes perpendicular to ℓ_θ.
- *Peak event*: an edge incident to a reflex vertex becomes perpendicular to ℓ_θ.

For super-vertices, it suffices to consider their peak events for capturing changes to the circle graph because a super-vertex induces no diagonal event. We construct a circle graph G and maintain it for these events at discrete orientations. Initially, we construct G as a complete graph with vertices corresponding to reflex vertices and reflex super-vertices of P in order along ∂P. The circle graph induced by P and ℓ_θ is a subgraph of G. We mark each vertex and each chord of G either as *used* or *unused*. At a diagonal event, we change the mark of the chord of G corresponding to the diagonal accordingly. At a peak event, we change the mark of the vertex and the chords of G corresponding to the peak accordingly.

Our algorithm works as follows. It constructs G as above and mark the vertices and chords of G such that the subgraph of G induced by used vertices and chords is the circle graph induced by P and the y-axis. It also computes the orientations of diagonal and peak events, and stores them in a priority queue with the orientations as keys. Then it processes the events one by one in order using the priority queue. For each event and its orientation θ, it updates the number of peaks with respect to ℓ_θ and updates the mark of the vertex or chord of G corresponding to the event accordingly. Then it computes an MICS of the subgraph of G induced by the used vertices and chords. The number of subpolygons in a minimum partition of P with respect to ℓ_θ is determined by the number of peaks and the cardinality of the MICS by Lemma 1. It updates the minimum number of subpolygons if the number with respect to ℓ_θ is smaller than the minimum number we have so far.

After processing all the events, we have the line ℓ^* such that the minimum partition P^* of P with respect to ℓ^* is a minimum partition of P. Finally, we compute a minimum partition P^* of P with respect to ℓ^*, and return (ℓ^*, P^*) as a minimum partition-pair.

We analyze the time complexity of the algorithm. We denote by T_c, T_m and T_r the times for constructing the circle graph induced by P and a fixed scan line, for computing an MICS of the circle graph, and for computing a uniformly monotone partition corresponding to the MICS, respectively.

Lemma 2 *We can compute a minimum partition of P in $O(n+R^2 \log R+T_c+R^2T_m+T_r)$ time using $O(n+R^2)$ space.*

4.2 With no Steiner points

We present an $O(nR \log n + R^5)$-time algorithm for the minimum partition problem with no Steiner points, improving upon the result by Liu and Ntafos. Our algorithm uses a geodesic query data structure for finding full diagonals between super-peaks efficiently. Here, the *geodesic* between any two points $p, q \in P$, denoted by $\pi(p, q)$, is the unique shortest path between p and q that is contained in P.

Lemma 3 *Given a top super-peak H_1 and a bottom super-peak H_2 of P consisting of n_1 and n_2 vertices, respectively, we can check if there is a full diagonal between a vertex of H_1 and a vertex of H_2 in $O(\min\{n_1, n_2\} \log n)$ time, after an $O(n)$-time preprocessing using $O(n)$ space.*

By efficiently computing the chords corresponding to pairs of super-peaks as in Lemma 3, and using the result in Section 4.1, we have our result for the minimum partition problem with no Steiner points.

Theorem 4 *Given a simple polygon P with n vertices (including R reflex vertices) and no holes, we can compute a minimum partition-pair of P in $O(nR \log n+R^5)$ time using $O(n+R^2)$ space, when no Steiner points are allowed.*

4.3 With boundary Steiner points

We present an $O(n + R^5)$-time algorithm for the minimum partition problem using Steiner points lying on the boundary of P, improving upon the result by Liu and Ntafos. Our algorithm first constructs the circle graph efficiently using geodesic queries for vertices in peaks.

Lemma 5 *We can compute full diagonals of P for the y-axis scan line in $O(n + R^2 \log n + R^3)$ time.*

After constructing the circle graph using Lemma 5, our algorithm computes an MICS of the graph. Then, it computes a minimum partition from the MICS efficiently using ray shooting queries, and uses the result in Section 4.1.

Theorem 6 *Given a simple polygon P with n vertices (including R reflex vertices) and no holes, we can compute a minimum partition-pair of P in $O(n + R^5)$ time using $O(n+R^2)$ space, using Steiner points lying on the boundary of P.*

4.4 With boundary and interior Steiner points

We give a sketch of an $O(n+R^4)$-time algorithm for the minimum partition problem using Steiner points lying on the boundary and interior of P.

The number of subpolygons in a minimum partition of P may be reduced if Steiner points are allowed to be placed in the boundary and interior of P. See Figure 1(b,c). In Figure 1(c), a y-monotone chain connects

a top peak and a bottom peak that are not visible to each other.

We refer to a chain that connects a pair of vertices of P including Steiner points on ∂P, and is contained in the interior of P, except at the end vertices, as a *diagonal chain* of P. For a scan line ℓ_θ, a diagonal chain is *full* if it connects a top peak u and a bottom peak v with $y_\theta(u) \geq y_\theta(v)$, and it is monotone with respect to ℓ_θ. Here, $y_\theta(p)$ for a peak p is the y-coordinate of the point obtained by rotating p around the origin in counterclockwise by θ. We use full diagonal chains, instead of full diagonals, for constructing chords of a circle graph. Then Lemma 1 holds for the circle graph.

We can show that there is a full diagonal chain between a pair of top and bottom peaks for scan line ℓ_θ if and only if there are a vertex u in the top peak and a vertex v in the bottom peak with $y_\theta(u) \geq y_\theta(v)$ such that the geodesic between u and v is monotone with respect to ℓ_θ. Thus, we can compute the chords of the circle graph G by computing for each pair of peaks, the geodesic between two vertices, one from each peak, and checking if it is monotone with respect to ℓ_θ.

After constructing G, we compute an MICS of G. Lemma 2 holds by modifying the definition of the diagonal event as follows.

- *Diagonal event*: the geodesic between a vertex in a top peak and a vertex in a bottom peak becomes monotone with respect to the scan line ℓ_θ.

It is known that the cardinality of an MICS of G equals the cardinality of a maximum matching of G [13]. We can show that G is a *chordal bipartite graph* for any fixed scan line, which is a bipartite graph such that every cycle C of length at least 6 in the graph has an edge not in C that connects two vertices in C. A maximum matching of a chordal bipartite graph can be computed efficiently [1, 14]. Thus, we can efficiently compute the cardinality of an MICS of G.

By computing the cardinalities of MICSs as above and using the result in Section 4.1, we compute the orientation θ^* corresponding to a minimum partition of P. Then, we explicitly compute an MICS of the circle graph induced by P and the scan line of orientation θ^*. We compute a partition of P corresponding to the MICS as follows. We place Steiner points on line segments lying in P and perpendicular to the scan line, each of which is incident to a reflex vertex of P. We assign each Steiner point to the corresponding chord in the MICS, and connect the assigned Steiner points for each chord in the MICS to obtain a full diagonal.

From each remaining peak in the partition induced by the full diagonals, we shoot a ray parallel to the scan line of orientation θ^*, and add the line segment obtained from the ray that lies in the subpolygon containing the peak. The resulting partition is a minimum partition of P. In total, it takes $O(n + R^4)$ time using $O(n + R^2)$ space to compute a minimum partition-pair of P.

5 Approximation algorithms

We give simple factor-2 approximation algorithms for the minimum partition problem. Observe that the number of subpolygons in a minimum partition cannot be less than half of the number of peaks. Thus, any partition of P with the scan line of the orientation, that minimizes the number of peaks among all orientations, induced by diagonals each removing at least one peak, is a 2-approximation for the minimum partition problem. We compute such a partition in $O(n \log n)$ time with no Steiner points, and in $O(n + R \log n)$ time using Steiner points lying on the boundary of P. The solution returned by the latter algorithm is also a 2-approximation for the case that Steiner points can be placed anywhere in P.

References

[1] M.-S. Chang. Algorithms for maximum matching and minimum fill-in on chordal bipartite graphs. *7th International Symposium on Algorithms and Computation (ISAAC)*, 146-155, 1996.

[2] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(3):485–524, 1991.

[3] B. Chazelle and D. P. Dobkin. Optimal convex decompositions. *Machine Intelligence and pattern recognition*, 2:63–133, 1985.

[4] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

[5] L. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989.

[6] J. M. Keil. Decomposing a polygon into simpler components. *SIAM Journal on Computing*, 14(4):799–817, 1985.

[7] J. M. Keil. Polygon Decomposition. In J.R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 11, pages 491–518. Elsevier, 2000.

[8] H. Kim, J. Lee and H.-K. Ahn. Rectangular Partitions of a Rectilinear Polygon. `arXiv:2111.01970 [cs.CG]`, 2021.

[9] D. T. Lee and F. P. Preparata. Location of a point in a planar subdivision and its applications. *SIAM Journal on Computing*, 6:594–606, 1977.

[10] R. Liu and S. Ntafos. On decomposing polygons into uniformly monotone parts. *Information Processing Letters*, 27:85–89, 1988.

[11] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer, 1985.

[12] M. van Kreveld, O. Schwarzkopf, M. de Berg and M. Overmars. *Computational Geometry: Algorithms and Applications.* Springer, 2000.

[13] X. Wei, A. Joneja and D. M. Mount. Optimal uniformly monotone partitioning of polygons with holes. *Computer-Aided Design*, 44(12):1235–1252, 2012.

[14] J. P. Spinrad. Doubly lexical ordering of dense 0–1 matrices. *Information Processing Letters*, 45(5):229–235, 1993.

Opposing Half Guards*

Erik Krohn† Bengt J. Nilsson‡ Christiane Schmidt§

Abstract

We study the art gallery problem for opposing half guards: guards that can either see to their left or to their right only. We present art gallery theorems, show that the problem is NP-hard in monotone polygons, present approximation algorithms for spiral and staircase polygons, and show that the location of half guards in 2-guardable polygons is not restricted to extensions.

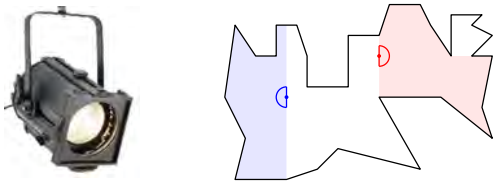
1 Introduction

The Art Gallery Problem (AGP), based on a question by Victor Klee, is one of the classical problems in Computational Geometry. Klee asked for the minimum number of stationary guards with 360° vision that we need to place to achieve complete visibility coverage of a polygon $\mathbf{P}$. Such a guard $g \in \mathbf{P}$ can see a point $p \in \mathbf{P}$ iff $\overline{gp}$ is fully contained in $\mathbf{P}$. Typical results can be classified in two categories:

1. Computational complexity and algorithmic results for the minimization of the number of star-shaped polygons (the *visibility polygons* (VPs) of guards) that cover a polygon—computation of $G(\mathbf{P})$.
2. "Art Gallery Theorems": Worst-case, combinatorial bounds on the number of VPs that are sometimes necessary and always sufficient to cover a class of polygons—bounds on the maximum value of $G(\mathbf{P})$ over all polygons of n vertices, $g(n)$.

Results on (2) are presented in, e.g., [4, 7, 11, 16]. Results settling the computational complexity, (1), are given in, e.g., [17, 13, 16, 6, 1], approximation algorithms are given in, e.g., [12, 3].

Here, the guards do not have 360° vision, but every guard can either see to the left or the right: imagine a spotlight as shown in Figure 1(a), for which the upper bow is fixedly mounted (and cannot rotate/yaw), so, the only degree of freedom for the spotlight is analogue to an aircraft pitch. If we fully utilize this movement, the spotlight illuminates one of the two halfplanes defined by the line that contains the upper bow. Formally, a left-looking (right-looking) half guard $g \in \mathbf{P}$ sees a point $p \in \mathbf{P}$ iff $\overline{gp}$ is fully contained in $\mathbf{P}$ and if p does not have larger (smaller) x-coordinate than g, see Figure 1(b) for an illustration. We call such guards *opposing* in contrast to half guards that can all only see in one direction. Half guards that all see in only one direction have been considered by Gibson et al. [9].



(a) (b)

Figure 1: (a) A spotlight (picture from flickr.com user Toby Simkin under licence CC BY-NC-SA 2.0.), (b) a polygon (black) with a left-looking half guard and its visibility polygon (blue) and a right-looking half guard and its visibility polygon (red).

Of course, if we found a feasible solution for the AGP with "ordinary" guards, placing a left- and a right-looking half guard for each guard yields a feasible solution also for the AGP with half guards—usually this will not be optimal, for monotone mountains (where we can compute an optimal solution for the AGP in polynomial time [5]) it would directly yield a 2-approximation. In this paper, we study questions of both types (1) and (2) for half guards.

2 Notation and Preliminaries

We let $\mathbf{P}$ denote a simple polygon, $|\mathbf{P}| = n$. We let r denote the number of reflex vertices of P. A simple polygon $\mathbf{P}$ is *x-monotone* if the intersection $\ell \cap \mathbf{P}$ of $\mathbf{P}$ with any vertical line ℓ is a connected set. Any x-monotone polygon decomposes into two x-monotone polygonal chains between the rightmost and leftmost point of $\mathbf{P}$. An x-monotone polygon is a *monotone mountain* or *uni-monotone*, if one of its two chains—w.l.o.g. the upper chain—is a single horizontal segment, $\mathcal{H}$. A polygon $\mathbf{P}$ is *orthogonal* (or rectilinear) if all of its edges are axis-parallel, that is, either horizontal or vertical. An orthogonal polygon $\mathbf{P}$ is a *staircase polygon* if it is x- and y-monotone. We assume a leftmost (rightmost) point of a staircase polygon to also be its

*B. J. N. and C. S. were supported by grants 2018-04001 (Nya paradigmer för autonom obemannad flygledning) and 2021-03810 (Illuminate: bevisbart goda algoritmer för bevakningsproblem) from the Swedish Research Council (Vetenskapsrådet). E. K. was supported by a sabbatical grant from the University of Wisconsin - Oshkosh.

†Department of Computer Science, University of Wisconsin - Oshkosh, Oshkosh, WI, USA, krohne@uwosh.edu

‡Department of Computer Science and Media Technology, Malmö University, Sweden, bengt.nilsson.TS@mau.se

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

lowest (highest) point and denote this point by p_ℓ (p_u). We have two polygonal chains connecting p_ℓ and p_u, the chain for which there exists a point on the other chain with the same x-coordinate but larger y-coordinate is denoted as the lower chain, the other chain is the upper chain.

Our half guards can either look to their left or their right, formally, we define left-looking and right-looking half guards. Let p_x and p_y define the x- and y-coordinate of a point p, respectively.

- A *left-looking half guard* $g \in \mathbf{P}$ can see a point $q \in \mathbf{P}$ iff ($\overline{gq}$ does not intersect $\mathbf{P}$'s boundary AND $g_x \geq p_x$); we say that g *half sees* q.
- A *right-looking half guard* $g \in \mathbf{P}$ can see a point $q \in \mathbf{P}$ iff ($\overline{gq}$ does not intersect $\mathbf{P}$'s boundary AND $g_x \leq p_x$); we say that g *half sees* q.

Because we sometimes compare with "normal" visibility, we also define when a (full) guard $g \in \mathbf{P}$ *sees* a point $p \in \mathbf{P}$: when the line segment $\overline{gp}$ does not intersect P's boundary. For a point p, we let $\mathbf{V}(p)$ denote the half-visibility polygon of p and $\mathcal{V}(p)$ denote the "normal" visibility polygon of p. In a polygon $\mathbf{P}$, a set of witnesses W is a set of points in $\mathbf{P}$, such that $\forall w_1, w_2 \in W : \mathcal{V}(w_1) \cap \mathcal{V}(w_2) = \emptyset$.

3 Art Gallery Theorems for Opposing Half Guards

In this section, we give Art Gallery Theorems, that is, statements of the type "$x(n)$ guards are always sufficient and sometimes necessary for polygons with n vertices", for different polygon classes.

Theorem 1 *In simple polyons with n vertices:*

- *For $r > \frac{n}{2}$: $2\lfloor \frac{n}{3} \rfloor$ half guards are always sufficient and sometimes necessary.*
- *For $r \leq \frac{n}{2}$: $r+1$ half guards are always sufficient and sometimes necessary.*

Proof. For $r > \frac{n}{2}$, the upper bound follows trivially from the $\lfloor \frac{n}{3} \rfloor$ upper bound by Fisk [7] for "normal" guards (triangulating the poylgon, three-coloring the vertices and using the least-frequently used color yields at most $\lfloor \frac{n}{3} \rfloor$): placing one right- and one left-looking half guard at each position of a "normal" guard results in $2\lfloor \frac{n}{3} \rfloor$ half guards.

For the lower bound, we construct a family of polygons $\mathbf{P}_n$, see Figure 2(a), that needs $2\lfloor \frac{n}{3} \rfloor$ half guards. $\mathbf{P}_n$ is an x-monotone polygon: the upper polygonal chain has reflex vertices only (except for the rightmost and leftmost vertex, which are convex), the lower chain has alternating reflex and convex vertices. The reflex vertices of the upper chain have the same x-coordinate as the convex vertices of the lower chain. The lower-chain vertices incident to the rightmost and leftmost vertex of $\mathbf{P}_n$ are either a reflex or convex vertex, such that we can define $\mathbf{P}_n$ also for $(n \mod 3) \neq 0$. For each

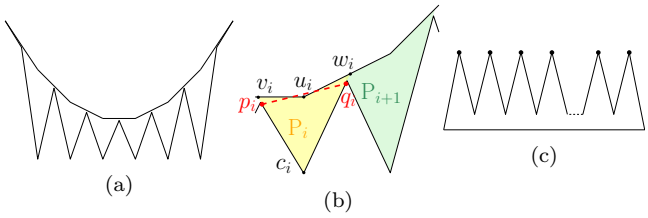


Figure 2: (a) Lower bound construction for simple (and monotone) polygons with $r > \frac{n}{2}$; (b) zoomed in on (a) showing $\mathbf{P}_i$ (yellow) and $\mathbf{P}_{i+1}$ (green). (c) Lower bound construction for simple (and monotone) polygons with $r \leq \frac{n}{2}$.

convex vertex c_i on the lower chain, we define a subpolygon $\mathbf{P}_i \subset \mathbf{P}_n$. We extend the two edges incident to c_i; let the two points where these extensions intersect with the upper chain be v_i and w_i. Let the reflex vertex of the upper chain with the same x-coordinate as c_i be u_i. The polygon $\mathbf{P}_i$ is defined by c_i, v_i, u_i and w_i, see Figure 2(b). Note that $\mathbf{P}_i \cap \mathbf{P}_{i+1} \neq \emptyset$. We claim that we need two half guards per $\mathbf{P}_i$. Let p_i and q_i be two points on the edges incident to c_i within distance ε from the reflex vertices (marked in red in Figure 2(b)). The point p_i can be seen from a right-looking half guard g only if $g_x \leq p_{i_x}$, however—as indicated by the red line segment—such a half guard cannot see q_i. A similar argument holds for a left-looking half guard seeing q_i. Both p_i and q_i can be seen from points in $\mathbf{P}_i$ only. However, we saw that $\mathbf{P}_i$ and $\mathbf{P}_{i+1}$ overlap: assume that we place a left-looking half guard g at w_i, it can see q_i and p_{i+1}. We still need two (more) half guards in $\mathbf{P}_{i+1}$: a right-looking half guard at w_i cannot see q_{i+1}, but a left-looking half guard that sees w_i cannot see all of $\mathbf{P}_{i+1}$. Hence, we need $2\lceil \frac{n}{3} \rceil$ half guards.

For the upper bound for $r \leq \frac{n}{2}$, we recursively partition the polygon into $r+1$ convex pieces: we pick any reflex vertex and extend one of its incident edges until we hit the boundary. Then, in both subpolygons we created, this vertex is no longer a reflex vertex. Because we end up with convex pieces, we can cover each piece with either a left-looking half guard at its rightmost vertex or a right-looking guard at its leftmost vertex. This yields in total $r+1$ half guards. For the lower bound, we construct a family of polygons $\mathbf{P}_n$, see Figure 2(c), that needs $r+1$ half guards. No two of the vertices marked with a point can be seen by a single half guard. Hence, we need $r+1$ half guards. □

The lower bound constructions for simple polygons are in fact both also monotone polygons, we yield:

Corollary 2 *In monotone polyons with n vertices:*

- *For $r > \frac{n}{2}$: $2\lfloor \frac{n}{3} \rfloor$ half guards are always sufficient and sometimes necessary.*
- *For $r \leq \frac{n}{2}$: $r+1$ half guards are always sufficient and sometimes necessary.*

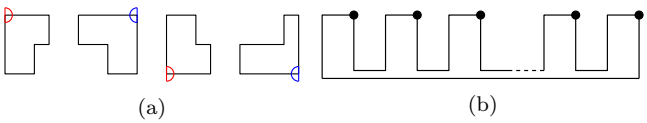


Figure 3: (a) Guarding L-shaped pieces with a single half guard; right-looking half guards are shown in red, left-looking half guards are shown in blue. (b) Lower bound construction for orthogonal polygons.

Theorem 3 *In simple orthogonal polyons with n vertices, $\lfloor \frac{n}{4} \rfloor$ half guards are always sufficient and sometimes necessary.*

Proof. An orthogonal polygon can be partitioned in $\lfloor \frac{n}{4} \rfloor$ L-shaped pieces (in linear time) [15, 16]. L-shaped pieces are orthogonal hexagons. Any L-shaped piece can be guarded by a single half guard placed at the only convex vertex, v, that can see all other vertices of the L-shaped piece (when the piece is considered as a simple polygon). Depending on whether the interior of the L-shaped piece lies in the left or right half plane of the vertical line through v, we use a left-looking or right-looking half guard, respectively. See Figure 3(a) for examples of guarding L-shaped pieces of the four possible orientations. This establishes that $\lfloor \frac{n}{4} \rfloor$ half guards are always sufficient.

Figure 3(b) shows a family of polygons $\mathbf{P}_n$ that needs $\lfloor \frac{n}{4} \rfloor$ half guards: no two of the vertices marked with a point can be seen by a single half guard. Hence, we need one half guard per four edges. □

Theorem 4 *In monotone mountains with n vertices:*

- *For $r < \frac{n}{2}$: $r+1$ half guards are always sufficient and sometimes necessary.*
- *For $\frac{n}{2} \leq r \leq \frac{3n}{4}$: $\lfloor \frac{n}{2} \rfloor$ half guards are always sufficient and sometimes necessary.*
- *For $r > \frac{3n}{4}$: $2 \cdot (n - r - 2) \leq \frac{n}{2}$ half guards are sometimes necessary.*

Proof. For $r \leq \frac{n}{2}$, the upper and lower bound follow as in the proof of Theorem 1 (i.e., the lower bound is shown in Figure 2(c) (where the lower horizontal chain is a horizontal segment)).

For the upper bound for $\frac{n}{2} \leq r \leq \frac{3n}{4}$, we consider the lower polygonal chain (the upper polygonal chain is a single horizontal segment $\mathcal{H}$). For guarding monotone mountains with "normal" guards, all guards can be placed on $\mathcal{H}$ and it is sufficient to guard all points of the lower polygonal chain to guard the complete polygon [5], the arguments used there also hold for half guards.

In between any pair of consecutive convex vertices, we have a reflex chain. One or two of the vertices in a reflex chain have a larger y-coordinate than the other vertices of that reflex chain. We split the reflex chain at one of these two vertices. Now, any convex vertex v is adjacent to a split reflex chain both on its right, $C_R(v)$, and on its left, $C_L(v)$. Let ℓ be the vertex with maximal y-coordinate in $C_L(v)$. We distinguish three cases:

1. Both $C_R(v)$ and $C_L(v)$ have more than two edges: Let $h \in \mathcal{H}$ be the point with $h_x = v_x$, we place a right-looking and a left-looking guard at h. These half guards monitor $C_R(v)$ and $C_L(v)$, and we use two half guards for at least four edges. The vertex v_1 in Figure 4 is an example for this case.
2. One of the two split reflex chains, w.l.o.g. $C_R(v)$, has more than two edges: let h be the point on $\mathcal{H}$ with $h_x = \ell_x$. We place a right-looking half guard at h. This half guard sees $C_R(v)$ and $C_L(v)$ ($C_L(v)$ has only one edge), and we use one half guard for at least three edges. The vertex v_2 in Figure 4 is an example for this case.
3. Both $C_R(v)$ and $C_L(v)$ have only one edge: let h be the point on $\mathcal{H}$ with $h_x = \ell_x$. We place a right-looking half guard at h. It monitors $C_R(v)$ and $C_L(v)$, and we use one half guard for two edges. The vertex v_3 in Figure 4 is an example for this case.

In all cases, each half guard monitors on average at least two edges, and the claim follows.

For the lower bound for $\frac{n}{2} \leq r \leq \frac{3n}{4}$, we use a similar construction as in Figure 2(c), however, in between two consecutive convex vertices, we now include one, two or three reflex vertices, see Figure 5 for an example. If $C_R(v)$ and $C_L(v)$ have both one edge, we can guard both of these edges with a single (left- or right-looking) half guard. If $C_R(v)$ and $C_L(v)$ have both two edges, these four edges cannot be seen by a single half guard, and no half guard from neighboring reflex chains can fully monitor these edges. Hence, we use one half guard per two edges.

For $r > \frac{3n}{4}$, we only use case 1 (that is, only what is depicted in dark red in Figure 4): we place

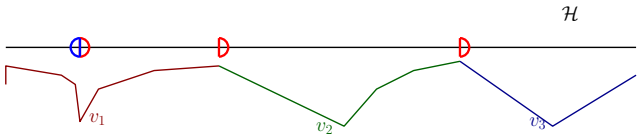


Figure 4: Upper bound construction for monotone mountains. The vertex v_1 and $C_R(v_1)$ and $C_L(v_1)$ are shown in dark red; v_2, $C_R(v_2)$, and $C_L(v_2)$ are shown in dark green; and v_3, $C_R(v_3)$, and $C_L(v_3)$ are shown in dark blue. Right-looking half guards are shown in red, left-looking half guards are shown in blue.

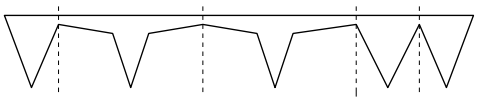


Figure 5: Lower bound construction for monotone mountains and $\frac{n}{2} \leq r \leq \frac{3n}{4}$.

a right-looking and a left-looking guard on $\mathcal{H}$ at the x-coordinates of convex vertices—except for the leftmost and rightmost convex vertex that are located on $\mathcal{H}$. These guards see the complete lower polygonal chain. We have $c = n - r$ convex vertices, thus, we place $2(c-2) = 2(n-r-2)$ half guards. For the lower bound, we insert reflex chains as the dark-red reflex chains from Figure 4 in the construction from Figure 5. □

4 Hardness Results for Opposing Half Guards

NP-hardness for point guarding a monotone polygon with half guards that only see to the right was claimed in [10]. The same reduction can be used for opposing half guards. In [10], the authors show an NP-hardness reduction from 3SAT. They show that certain vertices on the boundary represent truth values for variables in the original 3SAT instance. Clauses are represented by specific points on the boundary of the polygon. For example, if a clause $c = x_2 \vee x_5 \vee \overline{x_7}$ were in the original instance, then a vertex would exist on the boundary that would be seen by three vertices, namely the ones representing x_2, x_5 and $\overline{x_7}$. We briefly look at each pattern and show that if the polygon is guardable with k guards, then all k guards must be right-looking half guards.

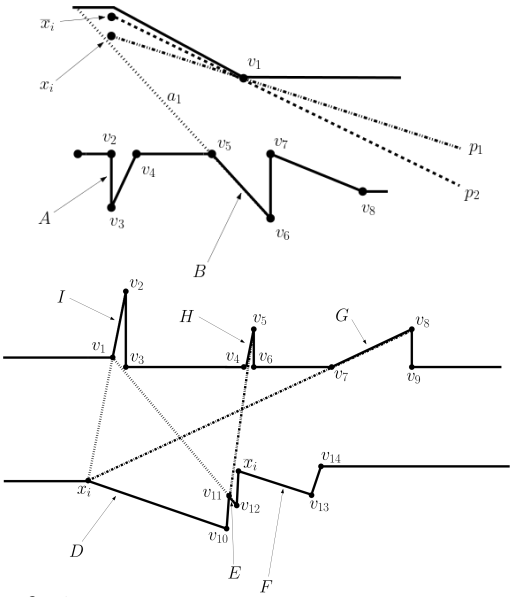


Figure 6: A starting pattern and a variable pattern for the NP-hardness reduction.

Starting Pattern: No left-looking half guard can see both v_3 and v_6, see Figure 6. No left-looking half guard placed outside of the starting pattern can see v_3 or v_6. Placing a left-looking half guard in the starting pattern for x_i would require at least two guards to be placed for the x_i starting pattern when one right-looking half guard is sufficient.

Variable Pattern: Distinguished vertices of a variable pattern that can be seen by guards outside of the variable pattern are vertices v_{10} and v_{13}, see Figure 6. Neither of these vertices, and none of the other distinguished vertices in this pattern, can be seen by a left-looking half guard outside of the variable pattern. Any left-looking half guard placed inside the variable pattern will require too many guards to be placed to guard the entire variable pattern. In the original reduction, two right-looking guards are required to guard the variable pattern. Even with left-looking guards being allowed, two guards are still required.

We note that only one of v_{10} or v_{13} will be seen by a guard to the left of the variable pattern. An incorrectly placed guard that sees both v_{10} and v_{13} will be an additional guard and will not reduce the number of guards needed in the current variable pattern. No guard (left or right-looking) can see both v_2 and v_5. Therefore, at least two guards are required to see v_2 and v_5 in each variable pattern. First, assume v_{10} is seen by a right-looking half guard to the left of of the variable pattern. This leaves the following vertices to be guarded: $\{v_2, v_5, v_8, v_{12}, v_{13}\}$. No left-looking half guard can see more than one distinguished vertex in that list. If a left-looking half guard sees v_2, then the only location to see both v_5 and v_{12} is a right-looking half guard at location v_{11}. If a left-looking guard is placed to see v_5, then the only location that sees v_2 and v_{12} is a right-looking guard at location v_1. In both cases, neither v_1 nor v_{11} sees v_8 and a third guard would be required.

Next, assume v_{13} is seen by a guard outside of the variable pattern. In this instance, no left-looking guard can see more than one of $\{v_2, v_5, v_8, v_{10}, v_{12}\}$. The same argument as above also applies here. Therefore, if a left-looking half guard is placed in this variable pattern, three guards are required when two right-looking half guards are sufficient. Those guard locations are $\{x_i, v_{11}\}$ or $\{\overline{x_i}, v_1\}$.

No extra guards are required to see any of the clause distinguished points. Any left-looking half guard that is placed to see a clause distinguished point will only see that particular clause distinguished point. No other distinguished points in any starting or variable pattern will be seen by such a guard. If k guards are sufficient to guard the entire polygon, then if a single left-looking half guard is placed in any pattern, an additional k guards are required to see the entire polygon. Thus, the reduction from [10] holds even if left-looking half guards are allowed to be placed inside the monotone polygon.

5 2-Guardable Polygons

In his Master thesis, Belleville [2] showed that if a polygon $\mathbf{P}$ is two-guardable (two guards can fully monitor $\mathbf{P}$), $\mathbf{P}$ is two-guardable by two guards that are located on edge extensions (including the edges themselves). We show that this statement does not hold for half guards:

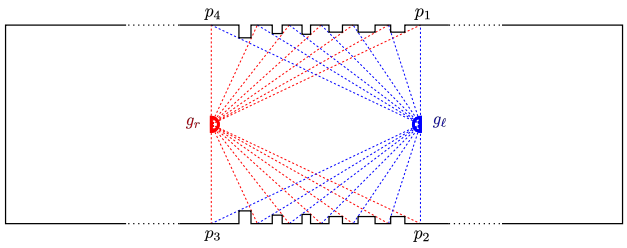


Figure 7: Polygon construction for Theorem 5.

Theorem 5 *Let* $\mathbf{P}$ *be a polygon for which the minimum half-guard cover has cardinality two, let these two guards be denoted as* g_1 *and* g_2*. Then neither* g_1 *nor* g_2 *must be located on edge extensions. Moreover, neither* g_1 *nor* g_2 *must be located on polygon diagonals.*

Proof. We construct a polygon $\mathbf{P}$ that can be covered by two half guards, but these guards do not lie on any edge extensions (or edges), that is, if we would restrict half-guard locations to edges and edge extensions, $\mathbf{P}$ cannot be covered by two half guards. The polygon $\mathbf{P}$ is shown in Figure 7. The shown half guards (one left-looking and one right-looking half guard) monitor $\mathbf{P}$, let the right-looking half guard be denoted as g_r and the left-looking half guard be denoted as g_ℓ. The lines of sight meeting in the niches on the top and bottom of $\mathbf{P}$ are shown in red and blue for g_r and g_ℓ, respectively. The dots in the polygon edges indicate very long edges.

Clearly, as $\mathbf{P}$ is an orthogonal polygon, neither g_r nor g_ℓ lie on any edge extensions, nor on any diagonals. It remains to show that no other pair of half guards monitors $\mathbf{P}$.

First, note that we cannot use two half guards that look into the same direction to monitor $\mathbf{P}$ with two half guards only: Then half guards cannot "share" seeing the niches, and we need one half guard per pair of mirrored niches. Hence, any minimum half-guard cover of $\mathbf{P}$ must contain a right-looking and a left-looking half guard.

Now assume that we try to move g_r and g_ℓ. Assume first that we only alter the y-coordinate of g_r. W.l.o.g.—the polygon is symmetric—we increase the y-coordinate of g_r and obtain guard g_r'. Then, the first point that g_r' sees on the top-right edge of $\mathbf{P}$ (the horizontal edge ending in the upper right vertex of $\mathbf{P}$) has a larger x-coordinate than p_1. Hence, we need to increase the x-coordinate of g_ℓ. If we only alter the x-coordinate of g_ℓ to obtain g_ℓ', points at distance ε from p_3 and p_4 on the same edges are not visible to g_ℓ', and we do no longer have a half-guard cover. Increasing also the y-coordinate of g_ℓ leaves part on the top unseen. Thus, assume that we increase the x-coordinate and decrease the y-coordinate of g_ℓ. Then, a point at distance ε from p_3 on its edge is not visible.

Similar arguments yield that changing the x-coordinate of g_r does not allow us to find a position for g_ℓ such that $\mathbf{P}$ is covered. □

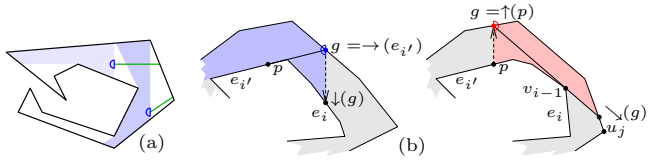


Figure 8: (a) Windows of a spiral polygon. (b) Illustrating the four operations.

Note that for "normal" guards, we may move the two guards to the x-coordinates of the left-most and right-most vertical edges of the niches.

6 An Approximation Algorithm for Spiral Polygons

A simple polygon $\mathbf{P}$ is *spiral* if it has two convex vertices u and u' such that a clockwise boundary walk from u to u' encounters only convex vertices and a counterclockwise boundary walk from u to u' encounters only reflex vertices. Nilsson and Wood [14] show a linear time greedy algorithm to compute the minimum number of "normal" guards for spiral polygons. We show a 3/2-approximation for half guards based on dynamic programming.

For a half visibility polygon $\mathbf{V}(p)$ of a point p in polygon $\mathbf{P}$, we call a *window* a boundary edge of $\mathbf{V}(p)$ that does not coincide with the boundary edges of $\mathbf{P}$; see Figure 8(a). We make the following claims without proof.

Lemma 6 *For a point* p *in a spiral polygon* $\mathbf{P}$*, the half visibility polygon* $\mathbf{V}(p)$ *has at most three windows.*

Lemma 7 *In a spiral polygon* $\mathbf{P}$*, there is an optimal set of half guards that all lie on the convex chain of* $\mathbf{P}$*.*

We let $n = n_c + n_r$ be the total number of vertices where n_c is the number of convex vertices including u and u' and n_r is the number of reflex vertices. We order the vertices from u to u' so that $u = u_1, \ldots, u_{n_c} = u'$ are the convex vertices and $v_1, \ldots, v_{n_r}$ are the reflex vertices in counterclockwise order starting from the vertex after u and ending at the vertex before u'. To simplify, we let $v_0 = u$. We also denote by $e_i = [v_{i-1}, v_i]$ the edge of the reflex boundary connecting v_{i-1} and v_i, $1 \le i \le n_r$.

We identify special vertices that we call *corners*. A convex vertex u_j in $\mathbf{P}$ is an *(outer) corner*, if the two incident edges to u_j both lie on the same side of a vertical line through u_j and assume for simplicity that $\mathbf{P}$ has no vertical edges. We say that a corner is a *left corner* if the incident edges lie to the right of the vertical line through the corner, otherwise it is a *right corner*. Similarly, we can define the *inner corners* as the vertices of the reflex boundary chain for which the adjacent edges lie on the same side of the vertical line through the vertex. As we follow the convex chain in clockwise order from a left corner to a right corner, we say that a left looking half guard is a *backward guard* and a right looking half guard

is a *forward guard*. Similarly following the convex chain in clockwise order from a right corner to a left corner, a left looking half guard is a *forward guard* and a right looking half guard is a *backward guard*.

In addition, we define some useful operations as follows. For an edge e of the reflex chain, the point $\rightarrow(e)$ is the point on the convex chain intersected by a ray exuded in the counterclockwise direction along e.

For a point p on the convex chain, the point $\downarrow(p)$ is the point on the reflex chain intersected by a ray exuded in the vertical direction from p towards the interior of the polygon, if it exists, otherwise $\downarrow(p)$ is undefined.

For a point p on the convex chain, the point $\searrow(p)$ is the point on the convex chain intersected by a ray exuded towards the last vertex on the reflex chain seen by a forward half guard at p.

For a point p on the reflex chain, the point $\uparrow(p)$ is the point on the convex chain intersected by a ray exuded in the vertical direction from p towards the interior of the polygon. If p is an inner corner, the ray is exuded in the vertical direction making $\uparrow(p)$ the furthest of the two possible points along the convex chain; see Figure 8(b).

Consider a set of half guards $\mathcal{G}$ on the convex chain and a half guard g placed at point p of the convex chain. Assume that the half guards in $\mathcal{G}$ completely see the edges $e_1, \ldots, e_{i'-1}$ and g completely sees the edges $e_{i'}, \ldots, e_{i-1}$, for indices $i' \leq i$, then operation $\mathrm{ix}_{\mathcal{G}}(p)$ evaluates to the index i. Our algorithm will always place guards so that they see a contiguous portion of the reflex chain starting at $v_0 = u$ and ending at some point p on e_i, $i \geq 1$. We can therefore assume that $\mathcal{G}$ is any set of half guards that see the edges before $e_{i'}$ and define $\mathrm{ix}(p) = i$ only based on the half guard g at p.

We specify our algorithm as a dynamic programming algorithm based on the following recursion. Let i be the index of the furthest reflex edge e_i not completely seen by the currently placed half guards, let p be the last point on e_i seen by the currently placed half guards, and let q be the last point on the convex chain seen by the currently placed half guards. The first call is $G(0, u, u)$.

$$G(i,p,q) = \min \begin{cases} G\big(\mathrm{ix}(u_j), v_{\mathrm{ix}(u_j)-1}, \searrow(u_j)\big) + 1, & \text{half guard at next corner } u_j, \text{ if } u_j \text{ sees } p \text{ and } q \\ G\big(\mathrm{ix}(g), \downarrow(g), g\big) + 1, & \text{backward guard at } g = \rightarrow(e_i), \text{ if } g \text{ sees } q,\ \downarrow(g) \text{ is defined, and } g \text{ is before next corner} \\ G\big(\mathrm{ix}(u_j), v_{\mathrm{ix}(u_j)-1}, \searrow(u_j)\big) + 2, & \text{backward guard at } g = \rightarrow(e_i) \text{ and half guard at next corner } u_j, \text{ if } g \text{ sees } q,\ \downarrow(g) \text{ is undefined, and } g \text{ is before } u_j \\ G\big(\mathrm{ix}(q), v_{\mathrm{ix}(q)-1}, \searrow(q)\big) + 1, & \text{backward guard at } g = \searrow(q) \text{ if } v_i \text{ does not see } q,\ \downarrow(g) \text{ is defined, and } g \text{ is before next corner} \\ G\big(\mathrm{ix}(u_j), v_{\mathrm{ix}(u_j)-1}, \searrow(u_j)\big) + 2, & \text{backward guard at } g = \searrow(q) \text{ and half guard at next corner } u_j, \text{ if } v_i \text{ does not see } q,\ \downarrow(g) \text{ is undefined, and } g \text{ is before } u_j \\ G\big(\mathrm{ix}(q), v_{\mathrm{ix}(q)-1}, \searrow(q)\big) + 1, & \text{forward guard at } q, \text{ if } q \text{ lies before } \uparrow(p) \text{ on convex chain} \\ G\big(\mathrm{ix}(g), v_{\mathrm{ix}(g)-1}, \searrow(g)\big) + 1, & \text{forward guard at } g = \uparrow(p), \text{ if } \uparrow(p) \text{ lies before } q \text{ on convex chain} \end{cases}$$

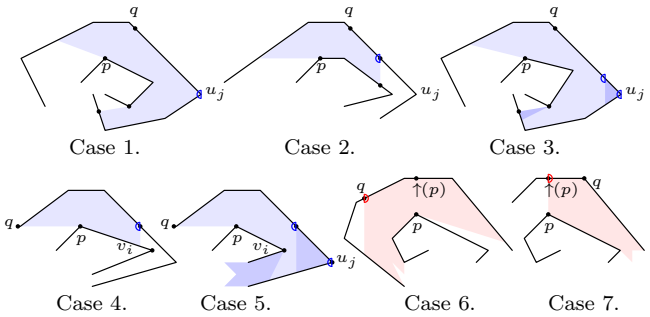


Figure 9: Illustrating the seven cases in the recursion.

We ignore the bottom of the recursion, when $i > n_r + 1$ as it follows the general description above without the recursive calls.

Each possible half guard position q can be precomputed as the (outer) corners, the intersection points $\rightarrow(e)$ on the convex chain issuing from each edge e of the reflex chain, and the intersection points $\uparrow(v)$ on the convex chain issuing vertically from each vertex v on the reflex chain, giving a linear number of possible positions. For each of these, we identify the linear sized sequence of continued intersection points $\searrow(q)$ of the supporting segments with the convex boundary, giving at most a quadratic number of positions. Each position can be acquired in constant amortized time by a traversal of the two boundary chains taking quadratic time in total. The dynamic programming thus fills out a table of size $O(n) \times O(n^2)$, each position in constant time.

We next prove the correctness and approximation ratio of the algorithm.

Lemma 8 *If $\mathcal{G}^*$ is a minimal set of half guards for a spiral polygon, then the algorithm covers the polygon and places at most $3|\mathcal{G}^*|/2 + 1$ half guards.*

Proof. The correctness of the algorithm follows by construction, since it ensures that each window created by a half guard is seen by the next half guard placed.

It remains to show the approximation ratio for the algorithm. To do so, let $\mathcal{B}$ be the set of prespecified and computed points on the convex boundary chain of $\mathbf{P}$ where the algorithm can place a half guard. We have $|\mathcal{B}| \in O(n^2)$ from the discussion above. Consider an optimum set of half guards $\mathcal{G}^*$ that we can assume by Lemma 7 all lie on the convex chain. Follow the convex chain from u to u' until a half guard g_1 in $\mathcal{G}^*$ is reached that does not lie on a point in $\mathcal{B}$. Let p be the last point on the reflex chain seen by the guards in $\mathcal{G}^*$ that lie before g_1 in $\mathcal{G}^*$. If g_1 is the first half guard, then $p = u$. We will show that we can exchange g_1 and the subsequent half guard g_2 in $\mathcal{G}^*$ for three half guards g, g', and g'' that indeed lie on the corresponding points in $\mathcal{B}$. Repeating the argument as the process follows the convex chain in clockwise order, proves our claim.

If g_1 can be moved to the subsequent point in $\mathcal{B}$, without losing visibility, we do so. This will not increase the

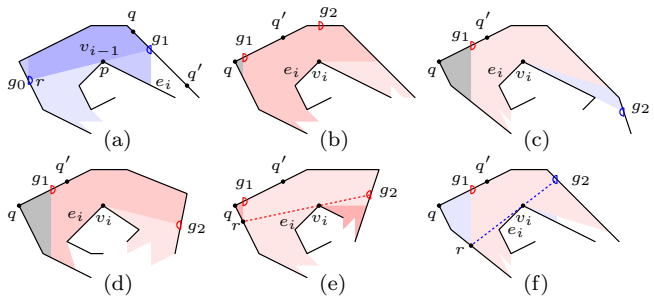


Figure 10: Illustrating the proof of Lemma 8. Grey regions are seen by previous half guards in $\mathcal{G}^*$.

number of guards.

Assume that g_1 is a backward guard between two subsequent points q and q' in $\mathcal{B}$ and that g_1 cannot be moved to q' without losing visibility. Let v_{i-1} be the first vertex on the reflex chain that g_1 sees and thus g_1 sees e_i. So, p is some point on e_i (except v_i). Since g_1 cannot be moved to q' and the reflex chain is seen by the previous guards until p on e_i, there must be points on the convex chain that are not seen if g_1 is moved towards q'. Let r be such a point on the convex chain and let g_0 be the guard in $\mathcal{G}^*$ that sees r. The guard g_0 must be a backward guard at r and r is by assumption in $\mathcal{B}$, hence, g_1 lies on the intersection of the convex chain and the supporting line from r through the reflex chain and this point lies in $\mathcal{B}$, a contradiction; see Figure 10(a).

Assume next that g_1 is a forward guard between two subsequent points q and q' in $\mathcal{B}$, that g_1 cannot be moved to q' without losing visibility, and that the region behind g_1 is seen by the previous guards in $\mathcal{G}^*$. Let v_i be the last vertex on the reflex chain that g_1 sees and assume furthermore that g_2 is a forward guard (or a backward guard with a corner between g_1 and g_2 along the convex chain). Without loss of generality, we can assume that g_2 lies on $\searrow(g_1)$ or on $\uparrow(v_i)$ depending on whether g_1 sees the next corner or not; see Figures 10(b), (c), and (d). If g_2 does not lie on any of these points, we can move it there without losing visibility. We can place two forward guards g and g' at q and q'. The half guard g' sees a vertex $v_{i'}$ with $i \leq i'$, hence we can replace g_2 by a half guard at $\searrow(q')$, if g' sees the next corner, or at $\uparrow(v_{i'})$. Note that if g_1 sees the next corner, so must g'. In both cases, g_1 and g_2 are replaced with three guards at positions in $\mathcal{B}$.

Finally, assume that g_1 is a forward guard, that the subsequent guard g_2 in $\mathcal{G}^*$ is backward (or forward with a corner between them) and the region behind g_1 is not seen by previous guards in $\mathcal{G}^*$. We can argue for g_2 as we did in the first case to obtain a point r on the convex chain not seen by g_2 if g_2 is moved forward; see Figure 10(e) and (f). The forward guard g_1 lies on the intersection between vertical line segment intersection through r and the convex chain (the vertical segment spans between r and g_1). Thus, the convex region defined by the convex chain from r to g_1 and limited by the vertical segment through these points is seen by g_2 and it contains a corner. We can replace g_1 and g_2 by a half guard g at the corner, a forward guard g' at $\uparrow(v_i)$, where v_i is the last reflex vertex seen be g, and either a backward guard at $\rightarrow(e_{i'})$, where $e_{i'}$ is the first unseen edge of the reflex boundary, or a half guard at the subsequent corner, whichever point comes first along a traversal of the convex chain; see Figures 10(e) and (f).

In each case, a pair of half guards from the optimum is replaced by a triple of half guards at a subset of our designated positions. Hence, there exists a feasible placement of half guards having size at most $\lceil 3|\mathcal{G}^*|/2 \rceil$. Since the dynamic programming algorithm computes the minimum such placement, the lemma follows. □

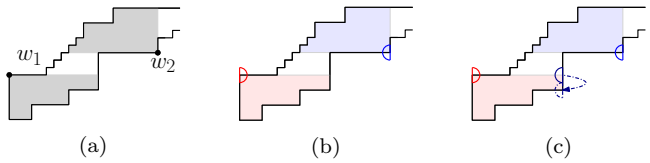


Figure 11: (a) A staircase polygon with CWS points w_1 and w_2, visibility polygons are shown in gray. (b) Placement of guards in G_{cw}. (c) Placement of guard in G_s (darkblue) and moving the guard to a feasible vertex (dash-dotted).

We have the following result.

Theorem 9 *The algorithm described computes a 3/2-approximate set of opposing half guards for spiral polygons in $O(n^3)$ time.*

7 2-Approximation for Staircase Polygons

Gibson et al. [8] show that staircase polygons allow for a 2-approximation for "normal" guards. Our algorithm is inspired by their CCCG algorithm (where CCCG stands for canonical convex corner guard).

Let G^* be an optimal opposing-half-guard set for a staircase polygon. We place a guard set G that is composed of two sets, that is, we have $G = G_{cw} \cup G_s$.

To construct G_{cw}, we place a set of witnesses on convex vertices, the *convex witness set* (CWS): We place witnesses alternatingly on (some) convex vertices of the lower and the upper chain. We place the first witness, w_1, on the first convex vertex of the upper chain that does not lie on P's lowest edge. We then define $\mathbf{P}_i = \mathbf{P} \setminus \mathcal{V}(w_{i-1})$, where $\mathcal{V}(p)$ denotes the visibility polygon of p under "normal" visibility. In $\mathbf{P}_i$ we place a witness on the first convex on the lower chain vertex that does not lie on $\mathbf{P}_i$s lowest edge, see Figure 11(a) for an examplary witness placement. We yield $W = \{w_1, w_2, \ldots\}$.

Lemma 10 *W is a set of witness points (and a CWS) and, hence, $|W| \leq |G^*|$.*

Proof. The topmost edge of $\mathcal{V}(w_i)$, for $i = 2k+1, k = 0, \ldots$, is a horizontal edge, e_i^h, and w_i and e_i^h have the same y-coordinate. By construction, the lowest edge of $\mathcal{V}(w_{i+1})$, e_{i+1}^h, is also horizontal, and its y-coordinate is larger than that of e_i^h. An analogous argument holds for the vertical edges limiting $\mathcal{V}(w_i)$ and $\mathcal{V}(w_{i+1})$ for $i = 2k, k = 0, \ldots$. Thus, the visibility polygons of the w_i are pairwise disjoint. □

We now place G_{cw} as follows: place a right-looking half guard on each w_i for which $i = 2k+1, k = 0, \ldots$, and a left-looking half guard on each w_i for which $i = 2k+1, k = 1, \ldots$, see Figure 11(b). Because the convex vertices are incident to vertical edges that limit their visibility to one half plane, the visibility polygons of the half guards coincide with the "normal" visibility polygons of our witnesses. We have $|G_{cw}| = |W| \leq |G^*|$.

For the construction of G_s, we consider the still unseen parts of $\mathbf{P}$: we can at most have $|G_{cw}|$ such polygon pieces (between each pair of witness visibility polygons and possibly one that includes either $\mathbf{P}$'s topmost or $\mathbf{P}$'s rightmost edge). We show that each such region is a staircase polygon for which either the upper or the lower chain has exactly two edges—a *stair*. Consider the placement of w_{2k+1} and $w_{2(k+1)}$: We place $w_{2(k+1)}$ on the first convex lower chain vertex that does not have the same y-coordinate as w_{2k+1}. Hence, the lower chain of the polygonal region between $\mathbf{V}(w_{2k+1})$ and $\mathbf{V}(w_{2(k+1)})$ consists of one horizontal edge (defined by the upper edge of w_{2k+1}'s visibility polygon) and one vertical edge (the upper end point of this edge has the same y-coordinate as $w_{2(k+1)}$). Thus, the polygonal region is a stair. We can place a left-looking half guard at its lowest-rightmost point that covers it completely, see Figure 11(c). Analgously, the polygonal region between $\mathbf{V}(w_{2k})$ and $\mathbf{V}(w_{2k+1})$ is a stair for which the upper chain has two edges, and we can guard it with a right-looking half guard at its highest-leftmost point. Thus, we obtain:

Theorem 11 *The set G covers all of $\mathbf{P}$ and $|G| = |G_{cw}| + |G_s| \leq 2 \cdot |G_{cw}| \leq 2 \cdot |G^*|$.*

In fact, the result holds for vertex half guards: all half guards in G_{cw} are already placed on vertices; we observe that we can slide each guard $g \in G_s$ between w_{2k+1} and $w_{2(k+1)}$ down along the vertical edge it resides on without loosing coverage of its stair, see Figure 11(c); analogously, a guard between w_{2k} and w_{2k+1} can be slided left along the horizontal edge it resides on.

References

[1] M. Abrahamsen, A. Adamaszek, and T. Miltzow. The art gallery problem is $\exists\mathbb{R}$-complete. *Journal of the ACM*, 69(1), 2021.

[2] P. Belleville. Computing two-covers of simple polygons. Master's thesis, McGill University, 1991.

[3] É. Bonnet and T. Miltzow. An Approximation Algorithm for the Art Gallery Problem. In *33rd International Symposium on Computational Geometry, SoCG'2017*, volume 77 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:15, 2017.

[4] V. Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory B*, 18:39–41, 1974.

[5] O. Daescu, S. Friedrichs, H. Malik, V. Polishchuk, and C. Schmidt. Altitude terrain guarding and guarding uni-monotone polygons. *Computational Geometry*, 84:22–35, 2019.

[6] S. Eidenbenz. Inapproximability results for guarding polygons without holes. In *Algorithms and Computation, ISAAC'1998*, pages 427–437, 1998.

[7] S. Fisk. A short proof of Chvátal's watchman theorem. *Journal of Combinatorial Theory B*, 24:374–375, 1978.

[8] M. Gibson, E. Krohn, B. J. Nilsson, M. Rayford, and P. Zylinski. A note on guarding staircase polygons. In *31st Canadian Conference on Computational Geometry, CCCG'2019*, pages 105–109, 2019.

[9] M. Gibson, E. Krohn, and M. Rayford. Guarding monotone polygons with half-guards. In *29th Canadian Conference on Computational Geometry, CCCG'2017*, pages 168–173, 2017.

[10] H. M. Hillberg, E. Krohn, and A. Pahlow. On the complexity of half-guarding monotone polygons. https://arxiv.org/abs/2204.13143, 2022.

[11] J. Kahn, M. Klawe, and D. Kleitman. Traditional art galleries require fewer watchmen. *SIAM Journal on Algebraic and Discrete Methods*, 4(2):194–206, 1983.

[12] J. King and D. Kirkpatrick. Improved approximation for guarding simple galleries from the perimeter. *Discrete & Computational Geometry*, 46(2):252–269, 2011.

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

[14] B. J. Nilsson and D. Wood. Watchmen routes in spiral polygons. In *2nd Canadian Conference on Computational Geometry, CCCG'1990*, pages 269–272, 1990.

[15] J. O'Rourke. An alternate proof of the rectilinear art gallery theorem. *Journal of Geometry*, 21(1):118–130, 1983.

[16] J. O'Rourke. *Art Gallery Theorems and Algorithms*. International Series of Monographs on Computer Science. Oxford University Press, New York, 1987.

[17] J. O'Rourke and K. Supowit. Some NP-hard polygon decomposition problems. *Information Theory, IEEE Transactions on*, 29(2):181–190, 1983.

Computing Realistic Terrains from Imprecise Elevations*

Anna Lubiw† Graeme Stroud†

Abstract

In the imprecise 2.5D terrain model, each vertex of a triangulated terrain has precise x- and y-coordinates, but the elevation (z-coordinate) is an imprecise value only known to lie within some interval. The goal is to choose elevation values from the intervals so that the resulting precise terrain is "realistic" as captured by some objective function.

We consider four objectives: #1 minimizing local extrema; #2 optimizing coplanar features; #3 minimizing surface area; #4 minimizing maximum steepness.

We also consider the problems down a dimension in 1.5D, where a terrain is a poly-line with precise x-coordinates and imprecise y-coordinate elevations. In 1.5D we reduce problems #1, #3, and #4 to a shortest path problem, and show that problem #2 can be 2-approximated via a minimum link path.

In 2.5D, problem #1 was proved NP-hard by Gray et al. [Computational Geometry, 2012]. We give a polynomial time algorithm for a triangulation of a polygon. We prove that problem #2 is strongly NP-complete, but give a constant-factor approximation when the triangles form a path and lie in a strip. We show that problems #3 and #4 can be solved efficiently via Second Order Cone Programming.

1 Introduction

A natural problem that arises in Geographic Information Systems is to compute a triangulated terrain in 3D space that is "nice" or "realistic". There is no single objective function to capture "niceness". In the study of erosion and hydrology, it is generally accepted that pits in a triangulated terrain are artifacts of imprecision, due to the unrealistic occurrence of water accumulation in flow simulations [9]. This motivates minimizing the number of extrema in the terrain. Since actual terrains tend to be smoothed by erosion, other natural objectives are to minimize the surface area, or to make the terrain as flat as possible.

A triangulated terrain is often computed from real elevation data. It is usually assumed that the data is accurate, however data acquisition can be complex and potentially prone to errors. It may be appropriate to model the input data as coming from a possible range of values to account for this uncertainty. Dealing with uncertainty or imprecision in the input data is a broad, well-studied area in computational geometry. Each input point may be represented by an uncertainty region, and the issue then is to find the best (or worst) placement of points, one in each region, for the problem at hand. For imprecise points in the plane, there is work on minimizing/maximizing the width, the area of the bounding box, or the diameter of the points [11, 13].

For the case of terrains, Gray and Evans [8], and Gray [7] formulated the ***imprecise 2.5D terrain model***. In this model, the x, y-coordinates of points are given as input, in addition to a triangulation defined on the points when projected to the xy plane, but the z-coordinate (elevation) of each point is only known "imprecisely" within some interval of possible values. We obtain a ***precise 2.5D terrain*** by choosing a precise elevation from each uncertainty interval and connecting the points together according to the input triangulation. Various "niceness" criteria for choosing a precise terrain have been considered in the past such as minimizing the number of local extrema [9], or minimizing the length of the shortest path along the terrain from one point to another [8, 12].

When these problems are NP-hard or have unknown computational complexity for 2.5D terrains, researchers (e.g., Gray et al. [10]) have considered ***imprecise 1.5D terrains***. Here, the x-coordinates are precise, and the elevations are the y-coordinates, each of which is given imprecisely via an interval.

We explore four objective functions that capture different "niceness" criteria for a terrain. To the best of our knowledge, only the first one (minimizing the number of extrema) has been considered before.

Objective #1: Minimizing local extrema. A local extremum is a local maximum or minimum compared to its neighbours in the triangulation. In a terrain these correspond to peaks or pits. To deal with equal elevations, define a ***plateau*** to be a maximal set of points of equal elevation that are connected by edges. A ***local minimum [maximum]*** is a plateau such that all neighbouring points have higher [lower] elevations. The problem of minimizing the number of local extrema was proved NP-hard in 2.5D by Gray et al. [9]. We give a polynomial time algorithm for the special case of a triangulation of a polygon, and solve the 1.5D version in

*Results from Master's thesis of the second author [16].

†School of Computer Science, University of Waterloo, alubiw@uwaterloo.ca, graeme.stroud@uwaterloo.ca

	#1: extrema	#2: coplanar	#3: area/length	#4: steepness
1.5D	$\boldsymbol{O(n)}$ **[§2]**	**2-approx [§2]**	$\boldsymbol{O(n)}$ **[§2]**	$\boldsymbol{O(n)}$ **[§2]**
2.5D special	$\boldsymbol{O(n^4)}$ **[§3] (polygon triangulation)**	**5-approx [§4.1] (strip triangulation)**		
2.5D general	NP-hard [9]	**NP-hard [§4.2]**	**SOCP [§5]**	**SOCP [§6]**

Table 1: Summary of results, with new results in bold.

linear time via a shortest path.

Objective #2: Optimizing coplanar features. To make a smooth terrain, we would like triangles to be coplanar with adjacent triangles if possible. This can be formalized as minimizing the number of ***patches***, where a patch is a maximal set of coplanar triangles that are connected edge-to-edge. An alternative is to minimize the number of ***bends***, where a bend is an edge whose two incident triangles are not coplanar. These objectives have different solutions in general, though they have the same solutions in 1.5D, where a patch is a maximal set of connected collinear edges (a "link") and a bend is a point whose two incident edges are not collinear. We show that both the patch and the bend versions are NP-complete in 2.5D. We give an easy 2-approximation in 1.5D and extend this to a 5-approximation for 2.5D in the special case where the triangles form a path in a strip (i.e., there are only two y-values).

Objective #3: Minimizing surface area/length. These are very natural objective functions. In 1.5D this becomes a shortest path problem. We formulate the 2.5D version as a Second Order Cone Program (SOCP). Second Order Cone Programming is a type of convex optimization problem that can be solved quite efficiently [15] (though not in polynomial time).

Objective #4: Minimizing maximum steepness. The ***steepness*** of a segment in 2D is the absolute value of its slope, and ***steepness*** of a triangle in 3D is the norm of its gradient. We consider minimizing the maximum steepness. Minimizing steepness gives a terrain that is as flat as possible, another reasonable objective.

We formulate the 2.5D version as a Second Order Cone Program, and show that the 1.5D version is solved via a shortest path—even for a lexicographic version where we minimize the maximum steepness, and subject to that, minimize the second maximum, etc.

Background. Gray [7] was the first to consider the imprecise terrain model. (See also Gray and Evans [8].) The problem they considered was finding the shortest path from one point to another over all precise realizations of the terrain. Various other objective functions have been explored for imprecise 1.5D and 2.5D terrains. The problem of minimizing the number of extrema was first explored by Gray et al. [9], and they show there is no $O(\log \log n)$ approximation algorithm unless P = NP. Driemel et al. [6] considered the problem of determining whether water can flow between two points of an imprecise 2.5D terrain. Here, the assumption is that water flows down the path of steepest descent. Gray et al. [10] considered a few objectives that result in "smooth" 1.5D terrains, such as minimizing [maximizing] the total turning angle, and minimizing [maximizing] the largest [smallest] turning angle.

2 1.5D Terrains

In this section we show that optimal 1.5D terrains for Objectives #1, #3, and #4 can be computed in linear time by finding a shortest path in an appropriate polygon, and that a minimum link path in the polygon provides a linear time 2-approximation for Objective #2.

Suppose the input to the problem has n points where, for $i = 1, \ldots, n$, point i must lie in segment ℓ_i at x-coordinate x_i, with $x_1 < x_2 < \cdots x_n$. Let P be the simple polygon whose vertices are the top and bottom endpoints of the segments, with a chain joining consecutive top endpoints, a chain joining consecutive bottom endpoints, plus the two edges ℓ_1 and ℓ_n. See Figure 1. We use a shortest path from ℓ_1 to ℓ_n, which is unique unless it is a straight horizontal path that can shift up/down.

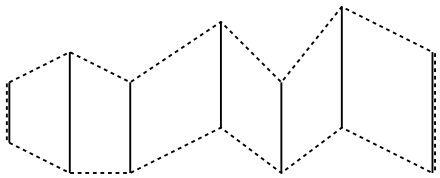


Figure 1: The input segments for the imprecise 1.5D terrain problem (solid) and the polygon P (dashed).

Theorem 1 *A shortest path from ℓ_1 to ℓ_n in polygon P can be found in linear time and provides an optimal 1.5D terrain for Objectives #1, #3, and #4.*

Proof. The shortest path from one segment to another in a simple polygon can be found in linear time [3]. This algorithm needs a triangulation of the polygon. Thankfully, we do not need Chazelle's impractical linear time algorithm [2], since P is composed of trapezoids each of which can be cut into two triangles.

Note that a shortest path in a polygon only bends at the polygon vertices. The vertices of polygon P are

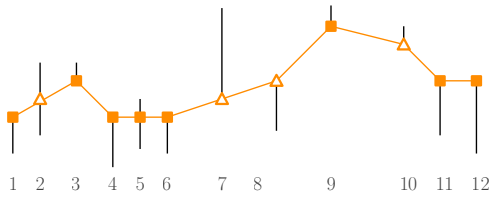


Figure 2: A shortest path terrain with five extreme plateaus (marked by squares).

endpoints of segments, and therefore a shortest path from ℓ_1 to ℓ_n provides a 1.5D terrain. (As discussed below, the fact that a minimum link path may bend at non-vertex points is why we can only achieve a 2-approximation for Objective #2.)

The theorem is obvious for Objective #3 (minimizing length). We next consider Objectives #1 and #4.

Objective #1. Suppose the shortest path has k extrema. We must prove that this is optimal, i.e., that any 1.5D terrain has at least k extrema. The plateaus of the leftmost point and rightmost point are extreme by definition. If $k = 1$ then there is a single plateau (the shortest path is horizontal) and this is clearly optimal. So suppose $k > 1$. Note that the extrema alternate between minima and maxima as we traverse the path. Let p_{i_j} be the rightmost point of the jth extreme plateau, lying on segment ℓ_{i_j} for $j = 1, \ldots, k-1$, and—since we want points where the path bends—let p_{i_k} be the leftmost point of the rightmost extreme plateau.

We will show that any 1.5D terrain must include at least k extrema, the leftmost and rightmost extrema plus at least $k-2$ others, one between segments $\ell_{i_{j-1}}$ and $\ell_{i_{j+1}}$ for each j, $2 \le j \le n-1$.

First, note that the points p_{i_j} zig-zag, i.e., if p_{i_j} is a minimum [maximum] then p_{i_j} is lower [higher] than $p_{i_{j-1}}$ and $p_{i_{j+1}}$. We see this in Figure 2, for instance, where the point on segment 4 is below the points on segments 3 and 9. Also, if p_{i_j} is part of a minimum [maximum] plateau, then the angle above [below] the path at p_{i_j} is strictly convex, so (because the path is shortest) p_{i_j} must be at the upper [lower] endpoint of its segment. Therefore, any point on segment ℓ_{i_j} is necessarily below [above] the points on segments $\ell_{i_{j-1}}$ and $\ell_{i_{j+1}}$ so there must always be a local minimum [maximum] between segments $\ell_{i_{j-1}}$ and $\ell_{i_{j+1}}$. Finally, note that since these extrema must alternate between minima and maxima, the extremum between $\ell_{i_{j-1}}$ and $\ell_{i_{j+1}}$ is distinct from the extremum between ℓ_{i_j} and $\ell_{i_{j+2}}$.

Objective #4. We prove that the shortest path provides something stronger: it minimizes the maximum steepness, and, subject to that, minimizes the second maximum steepness, and so on. We call this ***lexicographically minimizing the maximum steepness.*** In the full version [14] we prove the following.

Proposition 1 *A shortest path from ℓ_1 to ℓ_n in polygon P lexicographically minimizes the maximum steepness.*

This completes the proof of Theorem 1. □

We now turn to Objective #2, minimizing the number of links/bends. First note that the number of links is one more than the number of bends, so the two versions are equivalent (unlike in 2.5D). We make use of a minimum link path in polygon P from ℓ_1 to ℓ_n, which can be found in linear time using Suri's minimum link path algorithm [17]. (Suri's algorithm finds a minimum link path from a source point to a target point in a simple polygon, but, internally, it finds a minimum link path from a segment (a visibility window) to the target point, so it can easily be extended to deal with source and target segments.) This path may have bends that are not at the input line segments, but each such bend b can be replaced by two bends at the line segments just before and after b.

Theorem 2 *Let π be a minimum link path from ℓ_1 to ℓ_n in P. Then the points where π intersects the segments provide a 1.5D terrain with at most twice the minimum number of bends.*

Proof. The number of bends in π is clearly a lower bound, and each bend in π is replaced by at most two bends in the terrain. □

3 Local Extrema

We now turn to 2.5D terrains, as defined in the Introduction. Note that we allow input triangulations that do not include all the convex hull edges of the projected 2D points (to model triangulating a general shape). In this section we consider Objective #1, minimizing the number of local extrema.

Gray et al. [9] showed that this problem is NP-hard for 2.5D terrains. Therefore, we will examine a special case where we have a triangulation of a polygon, i.e., all points are on the boundary of the triangulation.

Theorem 3 *There is an $O(n^4)$ dynamic programming algorithm to minimize the number of extrema for imprecise 2.5D terrains when the triangulation is of a polygon.*

The following claim (proved in the full version [14]) shows that we can restrict to a discrete set of elevation values.

Claim 4 *Let $E = \{b_1, t_1, \ldots, b_n, t_n\}$ denote the set z-values of the bottom and top endpoints of the input intervals. Then there exists an optimal solution z^* so that $z^*_i \in E$ for all $i = 1, \ldots, n$.*

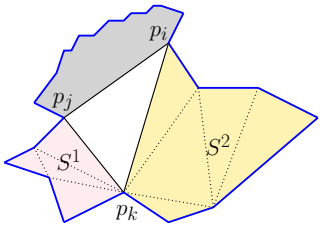


Figure 3: Splitting subproblem $S_{i,j}$ into $S^1 := S_{k,j}$ (pink) and $S^2 := S_{i,k}$ (yellow).

We will now describe the algorithm. Label the vertices around the polygon $p_1, \ldots, p_n$ in clockwise order. For each edge p_ip_j of the triangulation with $i < j$, we define a subproblem $S_{i,j}(z_i, \alpha_i, \beta_i, z_j, \alpha_j, \beta_j)$. This records the minimum number of *internal* extreme plateaus for the subpolygon $p_i, \ldots, p_j$, where $z_i \in E$ is the elevation for p_i, $\alpha_i \in \{T, F\}$ records whether there are above (higher) elevations connected to p_i's plateau, $\beta_i \in \{T, F\}$ records whether there are below (lower) elevations connected to p_i's plateau, and similar for j. Here "internal" means that we do not count the plateau(s) of p_i and p_j. It is easy to add those plateaus into the count, since p_i's plateau is a local extremum in $S_{i,j}$ iff $\neg\alpha_i \vee \neg\beta_i$ (i.e., there are no higher elevations connected to its plateau or there are no lower elevations connected to its plateau) and similarly for p_j. Furthermore, they are in the same plateau iff $z_i = z_j$.

The algorithm computes all $S_{i,j}$ entries using dynamic programming. Initialize by setting $S_{i,j}(z_i, \alpha_i, \beta_i, z_j, \alpha_j, \beta_j)$ to ∞ when the parameters are *incompatible*, meaning that a z value is outside its interval, or the α, β values contradict the z values, e.g., $\alpha_i = F$ but $z_j > z_i$, etc.

We solve for $S_{i,j}(z_i, \alpha_i, \beta_i, z_j, \alpha_j, \beta_j)$ for compatible parameter values, starting with smaller values of $j - i$ before larger values. When $j = i+1$, there are only two points (i.e., the subpolygon is an edge), and the number of internal extrema is zero.

For $j > i + 1$, there is a (unique) triangle p_i, p_k, p_j with $i < k < j$. Our goal is to combine solutions to the two subproblems $S_{i,k}$ and $S_{k,j}$ for various z, α, β values. See Figure 3. $S_{i,k}$ inherits z_i. $S_{k,j}$ inherits z_j. For z_k, we try all values in E (the same value in both subproblems). The above/below values are not simply inherited, since, for example, a T value for α_i in $S_{i,j}$ can come from a F value in $S_{i,k}$ if z_j provides the above elevation.

To simplify notation, let S^1 be $S_{i,k}$ and S^2 be $S_{k,j}$. Let α_i^1 be the α-value(s) of p_i in S^1, let α_k^1 be the α-value(s) of p_k in S^1, and etc. for the β values and for S^2. We have $\alpha_i \equiv \alpha_i^1 \vee (z_j > z_i)$. This tells us which values of α_i^1 to try. Similarly for β_i^1 and α_j^2, β_j^2.

We next specify which above/below values to try for p_k in the two subproblems. We will consider all possibilities for the final above/below values α_k, β_k of p_k in $S_{i,j}$. Namely, $(T,T), (T,F), (F,T), (F,F)$. We have $\alpha_k \equiv \alpha_k^1 \vee \alpha_k^2$, i.e., there are elevations above p_k's plateau in $S_{i,j}$ iff there are elevations above p_k's elevation in S^1 or in S^2. This tells us which values of α_k^1 and α_k^2 to try for a given choice of α_k. Similarly for β_k.

Finally, we set $S_{i,j}$ to be the minimum value, among all these choices, obtained as $S^1 + S^2 + \delta$ where $\delta \in \{0, 1\}$ is 1 iff p_k's plateau is an extremum distinct from the plateaus of p_i and p_j, i.e., iff $(\neg\alpha_k \vee \neg\beta_k) \wedge (z_k \neq z_i) \wedge (z_k \neq z_j)$.

The final minimum number of local extrema is obtained by taking the best of all the $S_{1,n}$ values, after adding 0, 1, or 2 extrema for p_1 and p_n as appropriate.

The algorithm is correct because we have considered all possibilities for the two subproblems.

Runtime. There are $O(n)$ edges p_ip_j in the triangulation, and for each, we try $O(n^2)$ elevations and above/below values, for a total of $O(n^3)$ subproblems. To solve a subproblem for $S_{i,j}$ we try $O(n)$ values for z_k and a constant number of combinations of above/below values. Thus the runtime of the algorithm is $O(n^4)$.

4 Coplanar Features

In this section, we explore the problem of minimizing the number of patches/bends. First, we give a 5-approximation algorithm for a triangulation in a strip (as shown in Figure 6). Then, we show that the general case is NP-complete for both objectives.

In general, these two objectives are not equivalent, see Figures 4 and 5. However, they are equivalent for a triangulation of a polygon—as we prove in the full version [14], the number of patches will be the number of bends plus one.

4.1 An algorithm for a strip triangulation

A strip triangulation is a special case of a triangulation of a polygon, so we can give an algorithm that works for both objectives.

Theorem 5 *There is a poly-time 5-approximation algorithm for the problem of minimizing the number of bends/patches when the input is restricted to a strip.*

Let the triangles along the strip be $T_1, \ldots, T_N$, where $N = n - 2$. We first greedily find the maximum index j such that triangles $T_1, \ldots, T_j$ can be coplanar. To test a given j, use a linear program whose variables are the z values of the imprecise points and the coefficients A, B, C of the plane $z = Ax + By + C$ that the triangles should lie in. Find the maximum j using binary search.

Note that any precise terrain for $T_1, \ldots, T_{j+1}$ must have at least one bend. Let $k > j$ be the minimum index such that triangle T_k shares no vertices with T_j. The

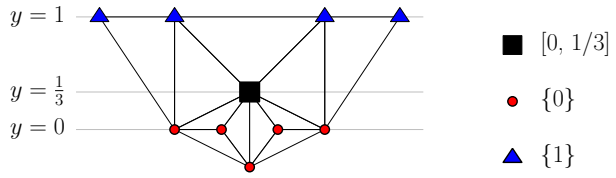


Figure 4: Proving that minimizing the number of bends is not always equivalent to minimizing the number of patches. The central point (drawn with a big black square) is the only one with a non-trivial z-interval, viz. $[0, 1/3]$.

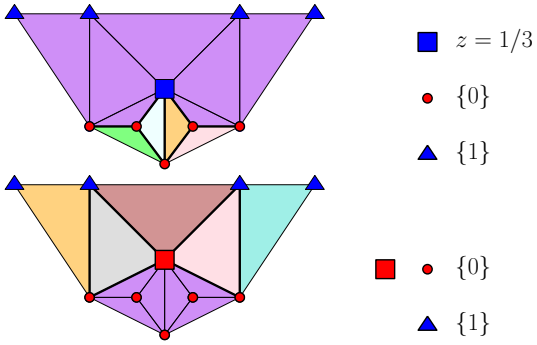


Figure 5: Solution (top) : If we use the top end of the black interval, we get 5 patches (optimal) and 7 bends. Solution (bottom) : If we use the bottom end of the black interval, we get 6 patches and 6 bends (optimal).

plan is to start the next greedy step from T_k. Observe that the situation is as shown in Figure 6: the last edge of T_j is ab; the first edge disjoint from ab is pq which is the first edge of T_k; and all intermediate triangles $T_{j+1}, \ldots, T_{k-1}$ include vertex a (without loss of generality, assume a and p lie on the top side of the strip). Note that the elevations of a and b have been fixed by the first greedy step, and the elevations of p and q will be fixed by the second greedy step. By induction, it suffices to choose elevations for the remaining vertices, the ones that lie strictly between b and q along the bottom of the strip, so that the resulting precise terrain on $T_1, \ldots, T_{k-1}$ is a 5-approximation of the optimum.

Observe that triangles $T_{j+2}, \ldots, T_{k-3}$ form a ***fan*** F between apex a (with fixed elevation) and base edges (with imprecise elevations) on the bottom of the strip. Two adjacent triangles in this fan are coplanar iff their

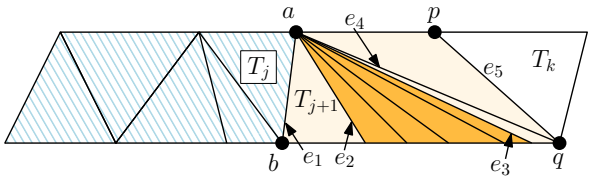


Figure 6: The first iteration of the algorithm. Fan F is colored in dark orange.

base edges are colinear. This reduces the problem to a 1.5D imprecise terrain problem in the xz-plane through the bottom of the strip. We use the linear time algorithm from Theorem 2 to find a 2-approximation for the minimum number of bends.

Let OPT be an optimum solution and let B^* be the number of bends in OPT on edges up to and including pq. Let B be the number of bends on these edges produced by the above algorithm. Let s^* be the minimum number of bends for internal edges of the fan F. Then we have: $B^* \geq 1+s^*$ since there is at least 1 bend before T_{j+1}, and s^* bends within F; and $B \leq 5 + 2s^*$, since there are five bends outside F (on the labelled edges in Figure 6) and at most $2s^*$ inside F by Theorem 2. Thus $5B^* \geq 5 + 5s^* \geq B$. Applying induction proves the approximation ratio is correct for the whole input.

4.2 NP-hardness for the general setting

We show that the objective of minimizing the number of patches is NP-complete for the case of a general triangulation without holes, using a reduction from Monotone Rectilinear Planar 3-SAT. The same reduction shows that minimizing the number of bends is NP-complete—see [16].

Theorem 6 *Minimizing the number of patches [or bends] is strongly NP-complete in the general setting.*

Containment in NP is proved in the full version [14]—a non-deterministic guess for the patches/bends can be verified in polynomial time using linear programming.

Reduction details. The reduction will be from the NP-complete problem Monotone Rectilinear Planar 3-SAT [4]. In this variant of 3-SAT, each clause has either three positive literals or three negative literals, and the input includes a planar representation where each variable v is represented by a thin vertical rectangle along the line $x = 0$, each positive [negative] clause is represented by a thin vertical rectangle at a positive [negative, resp.] x-coordinate, and there are horizontal line segments ("wires") joining each clause rectangle to the rectangles of the variables in the clause. We modify the representation by shrinking each clause rectangle to a square and adding vertical segments to the wires. See Figure 7. For n variables and m clauses, the representation can be on an $O(m) \times O(n + m)$ grid.

Given an instance of Monotone Rectilinear Planar 3-SAT Φ, we will construct an imprecise 2.5D terrain.

Variable gadget and component. The ***variable gadget*** for variable v, shown in Figure 8a, consists of four triangles: two ***selector*** triangles (in white); a *true* triangle (green striped), which we force onto the plane $x = z$; and a *false* triangle (checkered), which we force onto the horizontal plane $z = 0$. The z-interval of the

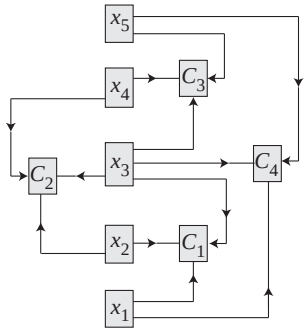


Figure 7: An instance of Monotone Rectilinear Planar 3-SAT, modified so the clauses have fixed height.

leftmost vertex of the gadget extends between the two planes, which permits the two selector triangles to be coplanar with the *true* triangle or with the *false* triangle. Thus, if the variable gadget is limited to two patches, the selector triangles "select" a true/false value for the variable. The gadget for variable v is placed inside v's input rectangle—see Figure 8a for the exact x, y-coordinates and z-intervals.

To model the wires in the input, we expand v's variable gadget to a ***variable component*** by constructing ***chains*** of ***path*** triangles as shown in Figure 8b. The associated z-intervals are large enough to permit all path triangles to be coplanar with v's *true* triangle (lying in the plane $x = z$) or with v's *false* triangle (lying in the horizontal plane $z = 0$). If the variable component is limited to two patches, then the choice made by the selector triangles is transmitted to all the path triangles.

Clause gadget. The gadget for clause c, shown in Figure 8c and 9, consists of three triangles sharing a ***centre vertex*** and joining the three final vertices of the chains corresponding to the variables in the clause. The z-interval of the central vertex is strictly above the $z = 0$ plane for a positive clause, and strictly above the $x = z$ plane for a negative clause. Therefore, for a positive [negative] clause, if all three chains are in the $z = 0$ plane [the $x = z$ plane] (corresponding to setting the literals false), then the three triangles of the clause gadget must form three patches. However, by making the z-interval of the central vertex large enough, we ensure that if at least one chain lies in the other plane (corresponding to setting the literal true), then the central vertex may be chosen to lie in the plane of the other three vertices, thus creating one coplanar patch out of the three clause triangles.

Completing the triangulation. The variable components and clause gadgets can be completed to a triangulation by filling in the holes with ***spike*** triangles, each of which has one new vertex that is off the xy-integer grid and that we force to a z-coordinate at least four times lower than anything constructed so far. By this choice of z, each spike triangle must form one patch. See Figure 10. Figure 11 illustrates the full reduction from the 3-SAT instance in Figure 7.

Lemma 7 *Let $k = 2n + m + s$, where s is the number of spike triangles. Then there is a satisfiable truth-value assignment for Φ if and only if there is a selection of elevations z that creates a terrain with at most k patches.*

Proof. (sketch) Suppose there is a satisfiable truth-value assignment for Φ. Choose elevations that put the variable component of each true variable in the $x = z$ plane and the variable component of each false variable in the $z = 0$ plane. This creates $2n$ patches. Since each clause has at least one true literal, we can choose the elevation of the centre vertex of each clause gadget so that the clause gadget uses one patch. This creates m patches. Finally, each spike triangle is one patch, so the total number of patches is $2n + m + s = k$.

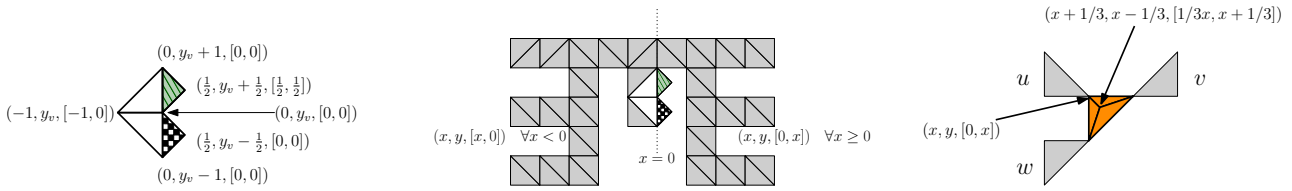
For the other direction, suppose there is a precise terrain with at most k patches. Each spike triangle forms one patch, each variable component forms at least two patches, and each clause gadget forms at least one patch (note that variable components do not share *edges* with clause gadgets). Thus each variable component must use two patches (thus forcing the three outer vertices of each clause gadget to respect the true/false choices), and each clause gadget must use one patch (thus requiring at least one of its literals to be true). □

5 Surface Area

We show that the surface area of an imprecise 2.5D terrain can be minimized using Second Order Cone Programming [1, Section 4.4.2] which is an extension of Linear Programming, with additional constraints of the form $\|Ax + b\| \leq c^\top x + d$, where $\|\cdot\|$ represents the Euclidean (L_2) norm. Second Order Cone Programs can be solved with additive error ε in time polynomial in the size of the input and $\log(\frac{1}{\varepsilon})$ using interior point methods. This is efficient, although not polynomial time.

We use variables $z_i, i = 1, \ldots, n$ for the elevations, and the linear constraints $b_i \leq z_i \leq t_i$ to ensure that each elevation value is within its interval. For each triangle T, a variable s_T will upper bound the area of T, via the constraint $\text{area}(T) \leq s_T$. Then minimizing the linear objective function $\sum_{T \in \mathcal{T}} s_T$ guarantees that the total surface area is minimized.

We only need to show that $\text{area}(T) \leq s_T$ is a valid SOCP constraint. If T has imprecise vertices p_1, p_2, p_3, then $\text{area}(T)$ is $\frac{1}{2}\|(p_2 - p_1) \times (p_3 - p_1)\|$, where $\times$ is cross product. Because x and y are fixed, $(p_2 - p_1) \times (p_3 - p_1)$ is a linear function of the z variables.



(a) Variable gadget (b) Variable component (c) Clause gadget (positive literals)

Figure 8: Gadgets for the NP-hardness reduction.

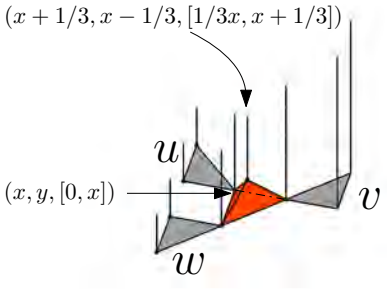


Figure 9: A 3D depiction of the clause gadget from Figure 8c.

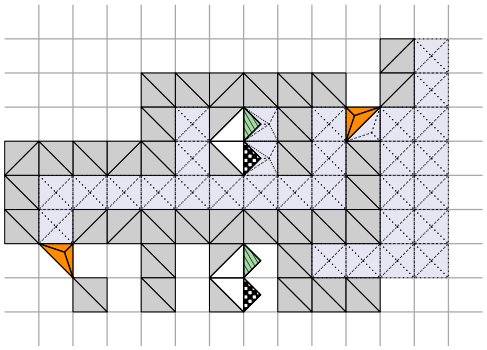


Figure 10: A portion of the final construction showing how spike triangles (in blue) fill in the triangulation.

6 Min Max Steepness

We show that minimizing the maximum steepness of an imprecise 2.5D terrain can be formulated as a Second Order Cone Program (as defined in the previous section). The ***steepness*** of triangle T lying on the plane $z = A_T x + B_T y + C_T$ is the L_2 norm of the gradient, i.e., $\|(A_T, B_T)\|$.

As above, we use variables z_i for the elevations, together with the linear constraints $b_i \leq z_i \leq t_i$. For each triangle T, we introduce variables A_T, B_T, C_T representing the coefficients of the plane containing T, as captured by the constraints $z_i = A_T x_i + B_T y_i + C_T$ for each vertex (x_i, y_i, z_i) of T. Finally, we add constraints $\|(A_T, B_T)\| \leq F$ for one new variable F. Then minimizing F will minimize the maximum steepness.

7 Conclusion

For imprecise 1.5D terrains, we gave linear time exact algorithms for three objectives, but could only achieve a 2-approximation for minimizing the number of bends. We believe that minimizing the number of bends for an imprecise 1.5D terrain is weakly NP-hard. Is the problem Fixed Parameter Tractable in the number of bends?

Another direction worth exploring is imprecise 2.5D terrains when the triangulation is not fixed, so the input consists only of imprecise points, and the problem is to find precise points and a triangulation for the given objective. Even if the points are given precisely, choosing the best triangulation can be NP-hard, as shown by De Kok et al. [5] for minimizing extrema. Are any of the other objectives NP-hard when the triangulation is not fixed, either for precise or imprecise points?

References

[1] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[2] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(3):485–524, 1991.

[3] Y.-J. Chiang and R. Tamassia. Optimal shortest path and minimum-link path queries between two convex polygons inside a simple polygonal obstacle. *International Journal of Computational Geometry & Applications*, 7:85–121, 1997.

[4] M. de Berg and A. Khosravi. Optimal binary space partitions in the plane. In M. T. Thai and S. Sahni, editors, *Computing and Combinatorics*, volume 6196 of *Lecture Notes in Computer Science*, pages 216–225, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[5] T. De Kok, M. Van Kreveld, and M. Löffler. Generating realistic terrains with higher-order delaunay triangulations. *Computational Geometry*, 36(1):52–65, 2007.

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

[7] C. Gray. Shortest paths on uncertain terrains. Master's thesis, University of British Columbia, 2004.

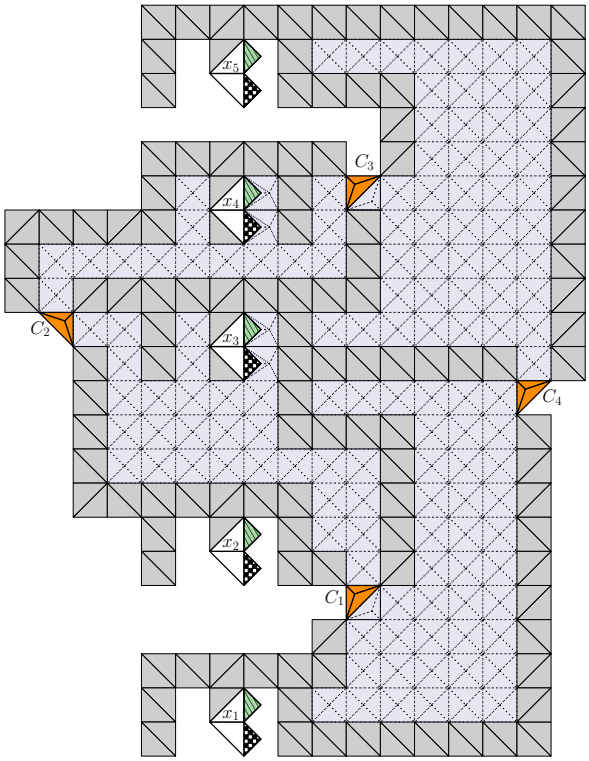


Figure 11: The resulting imprecise 2.5D terrain constructed from the instance of 3-SAT in Figure 7.

[8] C. Gray and W. Evans. Optimistic shortest paths on uncertain terrains. In *Proceedings of the 16th Canadian Conference on Computational Geometry (CCCG'04)*, pages 68–71, Montréal, Canada, 2004.

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

[10] C. Gray, M. Löffler, and R. I. Silveira. Smoothing imprecise 1.5D terrains. *International Journal of Computational Geometry & Applications*, 20(04):381–414, 2010.

[11] V. Keikha, M. Löffler, A. Mohades, and Z. Rahmati. Width and bounding box of imprecise points. In *Proceedings of the 30th Canadian Conference on Computational Geometry (CCCG'18)*, pages 142–148, Winnipeg, Canada, 2018.

[12] Y. Kholondyrev. Optimistic and pessimistic shortest paths on uncertain terrains. Master's thesis, University of British Columbia, 2007.

[13] M. Löffler and M. van Kreveld. Largest bounding box, smallest diameter, and related problems on imprecise points. *Computational Geometry*, 43(4):419 – 433, 2010.

[14] A. Lubiw and G. Stroud. Computing realistic terrains from imprecise elevations. *arxiv*, 2022. to appear.

[15] F. A. Potra and S. J. Wright. Interior-point methods. *Journal of Computational and Applied Mathematics*, 124(1):281–302, 2000. Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations.

[16] G. Stroud. Computing realistic terrains from imprecise elevations. Master's thesis, University of Waterloo, 2022.

[17] S. Suri. A linear time algorithm for minimum link paths inside a simple polygon. *Computer Vision, Graphics, and Image Processing*, 35(1):99–110, 1986.

Efficient Predicate Evaluation using Statistical Degeneracy Detection

Victor Milenkovic* Elisha Sacks†

Abstract

Computational geometry algorithms branch on the signs of predicates. Evaluating degenerate (zero sign) predicates is costly. Degeneracy is common for predicates whose arguments have common antecedents. Prior degeneracy detection techniques are slow, especially on predicates that involve algebraic numbers. We present statistical degeneracy detection (SDD) algorithms. Rational predicates are evaluated modulo randomly selected primes. Algebraic predicates are evaluated on randomly perturbed inputs. We analyze the failure rates under statistical assumptions. The algorithms are incorporated into an exact geometric computation library. Extensive testing shows that the library is reliable and fast. We also give an algorithm that reduces algebraic degeneracy detection to rational degeneracy detection without perturbation. This algorithm is much slower than the perturbation algorithm yet is far faster than prior work even when rational predicates are evaluated deterministically.

1 Introduction

We present research on the implementation of computational geometry algorithms. Implementations employ floating point arithmetic, whereas algorithms are expressed using real arithmetic. Although floating point is very accurate, even a tiny numerical error can cause a logical operator to return an incorrect Boolean value, which can cause a program crash or an unbounded error in the output. We follow the exact geometric computation (EGC) [25] strategy of ensuring accurate output by implementing logical operators that are correct despite numerical error.

A CG algorithm takes as input points or other geometric objects with rational parameters (coordinates or coefficients), expressed as ratios of integers or floating point numbers. The algorithm branches on the signs, > 0, $= 0$, or < 0 of polynomial functions of parameters, called predicates. For example, the sign of

$$\text{turn}(a, b, c) = (a - b) \times (b - c) \text{ with } u \times v = u_x v_y - u_y v_x$$

determine if abc turns left, goes straight, or turns right at b. Algorithms also generate new parameters using rational functions on (antecedent) parameters or the zeros of polynomials whose coefficients are rational functions of parameters. The former is rational, such as the coordinates of the intersection p of lines ab and cd, and the latter is algebraic, such as the intersections of the circle through b with center a with the line cd. Parameters and the predicates on them are rational if all functions in their derivation are rational; otherwise, they are algebraic.

In general, it is inexpensive to determine the sign of a nonzero predicate. For example, double precision floating point interval arithmetic usually results in an interval that does not contain zero, and hence the sign is known. Occasionally, additional precision is needed, such as using the MPFR library (`mpfr.org`). However, we find the additional cost is modest, up to 20%.

The situation for degenerate (zero) predicates is much direr. For rational predicates, it is necessary to use exact rational arithmetic using a library such as GMP (`gmplib.org`), and this can be much more expensive than double precision interval arithmetic. The general technique for algebraic predicates is root separation bounds, and these are very pessimistic, requiring many bits of precision. Exact rational arithmetic can sometimes be practical, but root separation bounds are almost never practical.

Degeneracy resulting from input in special position, such as collinear a, b, c can be eliminated by input perturbation: adding a small random quantity to each input parameter [12]. However, the cost of rational arithmetic for input degeneracies is not too high, and special position rarely results in algebraic degeneracy.

Derived parameters that are related by shared antecedents can also cause predicates to be degenerate. For example, consider line segments a_1b_1, a_2b_2, a_3b_3, and cd whose endpoint coordinates are input parameters. If cd intersects the other segments at p_1, p_2, and p_3, these points are collinear, $\text{turn}(p_1, p_2, p_3) = 0$, and this degeneracy is impervious to input perturbation. The coordinates of c and d are common antecedents of p_1, p_2, and p_3 in a manner that makes $\text{turn}(p_1, p_2, p_3)$ identically zero as a rational function of the coordinates of the eight input points. We call this type of degenerate predicate an *identity* because it is identically zero on an open set in the input parameter space, whereas a special position is zero on a measure zero set.

*Department of Computer Science, University of Miami, Coral Gables, FL 33124-4245, USA, `vjm@cs.miami.edu`. Supported by NSF CCF-1526335.

†Computer Science Department, Purdue University, West Lafayette, IN 47907-2066, USA, `eps@purdue.edu`. Supported by NSF CCF-1524455.

For a derivation depth of one or perhaps two, it is possible to analyze the identities and detect them by logic. For example, we know the p_1, p_2, p_3 in the above example will be collinear without exact rational evaluation. However, when multiple operations are cascaded, the number of types of identities rises exponentially with the derivation depth, and logic analysis becomes impractical. Unfortunately, the bit-complexity hence the cost of exact rational arithmetic or root separation also grows much greater.

We believe this effect often prevents the practical use of CG. Individual operations are efficient, but identities cause multiple consecutive operations to be very inefficient.

1.1 Contribution

We propose statistical degeneracy detection (SDD) to detect degenerate predicates without using exact rational arithmetic or separation bounds. We present three SDD algorithms and provide a library implementation at `https://github.com/Robust-Geometric-Computation`. Each algorithm outputs an estimated failure (false degeneracy) probability based on statistical assumptions. This approach allows efficient EGC on cascaded geometric operations. The estimated probabilities can be set so low as to make failure impossible in practice.

We provide the first probabilistic algorithms for degeneracy detection for both the rational and algebraic case, and we introduce the concept of statistical degeneracy detection. Prior work [10] uses a statistical assumption but does not provide an estimate of the probability of failure.

The first algorithm (Sec. 2) detects degenerate rational predicates by evaluating ambiguous (interval arithmetic interval contains zero) predicates modulo k random primes. We prove a worst-case bound on the probability of failure. However, this *probabilistic* bound requires having a bound b on the bit-complexity, but cancellation (of common factors of the numerators and denominators) greatly and unpredictably reduces b. Even given b, the probability bound is very pessimistic. We estimate the probability using a statistical assumption: nonzero predicates are zero modulo a random prime at the same rate as all nonzero expressions. In our tests, the estimated failure probability is always negligible for $k = 2$.

The second algorithm (Sec. 3) uses polynomial quotient rings to reduce an algebraic predicate to multiple rational predicates without the use of exact arithmetic or root separation bounds. The rational predicates can be evaluated using the first algorithm. It is much more efficient than root-separation-based methods but is limited to a small number of arguments.

The third algorithm (Sec. 4) uses the observation that identities remain zero after input perturbation, but all other expressions are likely to change their values. It perturbs its input to eliminate input (special position) degeneracies with high probability. It applies a second perturbation (provisionally) to predicates that remain ambiguous at h bits of precision. If an expression remains ambiguous, it reports an identity. It uses a measure on nonzero, nonidentity expressions to report a probability of failure, under the statistical assumption that this measure is the same for nonzero predicates. In our tests, the estimated failure probability is always negligible for $h = 265$.

1.2 Prior work

EGC comprises exact geometry kernels and number types. An exact geometry kernel supports a set of predicates for a class of objects, possibly with a limited capacity for defining new objects and predicates. The canonical examples are the CGAL [8] and Leda [15] kernels for rational operations on points. CGAL also provides an algebraic kernel for zeros of univariate polynomials. Two zeros can be ordered, but other predicates involving zeros are not supported. An exact number type supports a set of operations on a subset of the real numbers. The canonical example is the Leda real type for general expressions involving rational operations, radicals, and zeros of polynomials [7]. A number type is more flexible and easier to use than a kernel. But a kernel can model entire objects, rather than their parameters, and can exploit domain-specific algorithms.

Geometry kernels and number types are built from a common set of tools: interval arithmetic, arbitrary size integer arithmetic, arbitrary precision floating point arithmetic, and separation bounds.

Interval arithmetic [19] uses floating point arithmetic to compute an interval of floating point numbers that contains the value of an expression. EGC uses interval arithmetic in floating point filtering [5]: the sign of a predicate is determined when its interval excludes zero.

Arbitrary size integer arithmetic libraries, such as GMP (`gmplib.org`), are used to evaluate rational predicates. The complexity of a b-bit operation is $O(b \log b)$. The bit complexity of a rational predicate can be exponential in its derivation depth but is much lower in practice because common factors are canceled. Adaptive precision evaluation [22] is faster than GMP style arithmetic but is restricted to predicates in input parameters.

Rational predicates can also be evaluated using modular arithmetic. A predicate of bit complexity b requires b/k k-bit moduli. Degeneracy can be determined by verifying that all the residues are zero, at a cost of $\mathrm{O}(b)$. Computing the sign requires Chinese remaindering at a worst-case cost of $\mathrm{O}(b^2)$ and an expected cost of $\mathrm{O}(b)$ [6]. This paper proposes a probabilistic algorithm, but

it depends on on a lemma [10] (Lemma 3.1) that makes the statistical assumption that the value of a polynomial modulo an integer is uniformly distributed. Despite the log factor savings for modular arithmetic in theory, arbitrary size integer arithmetic is most used in practice.

Degenerate algebraic predicates can be detected using separation bounds. A separation bound β for a predicate e is a positive number such that $e \neq 0$ implies $|e| > \beta$. Li, Pion, and Yap [13] survey separation bound computation and Emiris, Mourrain, and Tsigaridas [9] present the state of the art. The LEDA exact number type [7] evaluates a predicate with interval arithmetic and increases the floating point precision until the interval excludes zero or the interval width is less than a separation bound. The separation bound technique is rarely practical because the bounds shrink rapidly as the number and degree of the algebraic numbers increase.

Some special cases are handled without separation bounds. Berberich et al [3] present an arrangement algorithm for plane algebraic curves using only symbolic methods. It includes univariate and bivariate polynomial support that is faster than the CGAL algebraic kernel. Masterjohn et al [14] present an arrangement algorithm that uses our fixed δ perturbation framework (below) to avoid degeneracies. Neither is subject to identities. Blomer [4] provides a probabilistic algorithm for rational expression whose leaves are roots of integers.

Halperin [12] pioneered input perturbation for preventing special position degeneracy. Each input parameter of an algorithm is perturbed by a value chosen uniformly in $[-\delta, \delta]$. The algorithm is run on the perturbed input. The δ is chosen so that floating point filtering succeeds with high probability. If every instance succeeds for a run of the algorithm, the output is correct for the perturbed input, hence is correct with backward error δ for the original input. Otherwise, the algorithm is rerun with a different perturbation. This approach does not address identities. Moreover, it can require values of δ that exceed the error bounds of applications.

We [18] developed a perturbation algorithm that uses a fixed, user-specified δ. We evaluate predicates using arbitrary precision interval arithmetic [11] and increase the precision until the interval excludes zero. We abort the algorithm when the precision reaches a threshold that is high enough that only identities reach it. When this happens, we devise ad hoc code for that identity.

We [17] present an identity detection algorithm for all predicates involving contacts between polyhedrons with four degrees of freedom. These predicates can be expressed as $g(r)$ where r if a zero of f, and both f and g are univariate polynomials whose coefficients are multivariate polynomials in the input parameters. The algorithm uses a precomputed table of all polynomials f and g, up to isomorphism, and their multivariate factorizations.

2 SDD algorithm for rational predicates

The probabilistic degeneracy detection algorithm for an ambiguous rational predicate evaluates it modulo a random 32-bit prime p. Each input parameter is converted from a double to the form $a \times 2^b$, with a and b integers, then this expression is evaluated modulo p. Modular addition and multiplication are 64-bit machine operations followed by evaluation modulo p. Modular division uses the extended Euclidean algorithm. If any divisor is divisible by p, the test is rerun with a new random prime. We repeat the test k times and declare the predicate degenerate if the residue is zero each time.

We bound the probability that the algorithm will report a false degeneracy for a predicate e whose bit complexity is bounded by b. Let n denote the number of 32-bit primes. At most $b/31$ primes divide e because they are larger than 2^{31}, so the probability that a given prime divides e is at most $t = b/(31n)$. The probability that k primes divide e is at most t^k and so a maximum failure probability of r is ensured by setting $k = \log r / \log t$. Unfortunately, there is no general method to determine a tight bound on b.

The SDD algorithm estimates the actual false degeneracy rate under the assumption that an ambiguous predicate and its subexpressions are equally likely to fail the test. To get this estimate, we set t equal to $1/k$ times the fraction of the unambiguous (hence nonzero) subexpressions that are zero modulo one of the k primes.

3 Extension to algebraic predicates using quotient rings

We extend rational degeneracy detection to algebraic predicates. Consider a predicate polynomial $e(x)$ evaluated at a simple zero r of $f(x)$, where e and f have rational coefficients. We compute $g = \gcd(e, f)$, using rational degeneracy detection to detect if the leading coefficient of a remainder is zero and hence the remainder has lower degree than the generic case. Since $f(r) = 0$, $x - r \mid f$ and so $e(r) = 0$ if and only if $g(r) = 0$. Let $h = f/g$. If $g(r) = 0$, $x - r \mid g$, so $x - r \nmid h$, since r is a simple zero of f, and $h(r) \neq 0$. If $h(r) = 0$, $x - r \mid h$, so $x - r \nmid g$ and $g(r) \neq 0$. Hence, either $g(r) = 0$ and $h(r) \neq 0$ or vice versa. We evaluate both expressions in interval arithmetic and increase the precision until one interval excludes zero. If $h(r) \neq 0$, $e(r)$ is degenerate. This technique detects an algebraic zero without the use of separation bounds or exact arithmetic.

We use quotient rings to extend this algorithm to predicates that have multiple algebraic parameters. Consider a predicate $e(x_1, x_2)$ evaluated on simple zeros r_1 and r_2 of $f_1(x_1)$ and $f_2(x_2)$. Let R_1 denote the quotient ring $\mathbb{Q}[x_1]/f_1$ of polynomials in x_1 modulo f_1. Convert e to $R_1[x_2]$ by expressing it as $e = \sum_k p_k(x_1)x_2^k$ then replacing each p_k by its remainder when divided

by f_1. Apply the above algorithm to e and f_2, which is trivially in $R_1[x_2]$, with $g = \gcd(e, f_2)$ and $h = f_2/g$ in $R_1[x_2]$. Either $h(r_1, r_2) = 0$ or $g(r_1, r_2) = 0$, and $e(r_1, r_2)$ is degenerate if $h(r_1, r_2) \neq 0$. In the general case, $e(x_1, \ldots, x_m)$ is evaluated on simple zeros $r_1, \ldots, r_m$ of $f_1(x_1), \ldots, f_m(x_m)$. Let $R_0 = \mathbb{Q}$ and $R_i = R_{i-1}[x_i]/f_i(x_i)$ for $i > 0$. Convert e and f_m to elements of $R_{m-1}[x_m]$ and apply the above algorithm, with $e(r_1, \ldots, r_m)$ degenerate if $h(r_1, \ldots, r_m) \neq 0$.

Implementing this algorithm requires zero detection and multiplicative inverse computation in R_i. These use rational degeneracy detection (Sec. 2) for $R_0 = \mathbb{Q}$. For $i > 0$, an element $a(x_i) \in R_i$ is represented by the remainder of a when divided by f_i in $R_{i-1}[x_i]$. Zero leading coefficients are detected in R_{i-1}, and a is zero when all the coefficients are zero. The inverse of a is computed with the extended Euclidean algorithm on $R_{i-1}[x_i]$. We detect a zero divisor (that has no inverse) when $g = \gcd(a, f_i)$ has nonzero degree. In that case, we calculate $h = f_i/g$, determine whether g or h is nonzero on $r_1, \ldots, r_i$, replace f_i with the other, and restart the degeneracy detection algorithm for e.

The number of restarts is at most total degree of f_i minus total degree of r_i, $i = 1, \ldots, m$. To prove correctness, we need to show that one of g and h is nonzero on $r_1, \ldots, r_i$ and the other is zero. Consider $f_i(r_1, \ldots, r_{i-1})(x_i) \in \mathbb{R}[x_i]$ that is the current $f_i(x_i)$ with $r_1, \ldots, r_{i-1}$ substituted in its coefficients. The invariant is that $f_i(r_1, \ldots, r_{i-1})(x_i)$ is divisible by $(x - r_i)$ but not $(x - r_i)^2$. This is true for the initial f_i. If $f_i(x_i) = g(x_i)h(x_i)$, then $f_i(r_1, \ldots, r_{i-1})(x_i) = g(r_1, \ldots, r_{i-1})(x_i)h(r_1, \ldots, r_{i-1})(x_i)$, one factor is divisible by $x - r_i$, and neither factor is divisible by $(x - r_i)^2$ due to the uniqueness of factorization. The factor that is not divisible by $x - r_i$ is nonzero on r_i.

The algorithm is impractical for large values of m because it requires d^{2m} operations for $f_1, \ldots, f_m$ of total degree d. It is difficult to interface with computational geometry algorithms because predicates must be expressed as polynomials in the m algebraic parameters.

4 Perturbation-based SDD algorithm for algebraic predicates

The exponential complexity of the quotient-ring-based SDD algorithm leads us to prefer a perturbation-based algorithm. This algorithm cannot detect special position degeneracy, so we prevent it with an input perturbation. We classify an ambiguous predicate as an identity if it remains ambiguous when the precision of the interval arithmetic is increased and when it is evaluated on a second perturbed input. A nondegenerate predicate is unlikely to remain ambiguous in either case.

The user selects the perturbation size δ and the identity detection control parameter h with default values $\delta = 2^{-27} \approx 10^{-8}$ and $h = 212$. The former is chosen based on the accuracy needed for the application, and the latter can be increased if the SDD failure probability estimate is too high. The algorithm employs two internal parameters: the perturbation precision b and the secondary perturbation size s, with initial values $b = 26$ and $s = \delta$. Each input parameter is perturbed by a b-bit number that is uniformly distributed in $[-\delta, \delta]$. For the initial b and the default δ, perturbing an input is equivalent to randomizing the lower half of its mantissa. Smaller δ or larger b would require expressing each input as a sum of doubles.

The algorithm runs the geometric computation on the perturbed input p. If an algebraic predicate $e(p)$ is ambiguous in floating point interval arithmetic, it is reevaluated in h-bit interval arithmetic. If it is still ambiguous, the algorithm selects a second perturbation $q = p + rv$. Each coordinate of the vector v is drawn uniformly from the set of b-bit numbers in $[-1, 1]$. The scalar r is initialized to s then is divided by ten until the isolating intervals of the algebraic numbers, which were computed at p, are also isolating at q. The algorithm reports an identity when $e(q)$ is ambiguous in h-bit interval arithmetic. Otherwise, it computes the sign of $e(p)$ using l-bit interval arithmetic, starting with $l = 2h$ and doubling l until the interval excludes zero.

This algorithm requires that the initial b-bit perturbation eliminates all degenerate non-identity predicates; otherwise, l would increase without bound. In practice, we put an upper bound $m = 424$ on l to prevent an infinite loop. If that bound were ever reached, we would restart with a different perturbation. If the bound were reached yet again, we would double the default values of b and m.

The algorithm cannot assign a nonzero sign to an identity. We bound the probability of a false identity under the assumption that $e(t)$ is analytic on $[0, r]$. This assumption can be guaranteed when the algebraic numbers in e are zeros of polynomials whose coefficients are rational parameters: when applying the Descartes rule of signs, verify that these coefficients have constant signs on $[0, r]$. We know of no practical test for polynomials with algebraic coefficients. We discuss this issue further in Sec. 6.

Suppose $e(p)$ is reported as an identity. Since $e(p)$ and $e(q)$ are ambiguous, $\Delta = e(q) - e(p)$ is ambiguous. A sufficient condition for ambiguity is $|\Delta| < w$ with w the width of the interval value of Δ in h-bit interval arithmetic. We call $|\Delta|/w$ the perturbation ratio of e. We approximate $e(q)$ by its linear Taylor series $e(p) + rg \cdot v$ with g the gradient of e with respect to p. The approximate perturbation ratio is $r|g \cdot v|/w$. Its maximum for a predicate that depends on d input parameters is $m = r||g||\sqrt{d}/w$ because the components of v are bounded by one.

Lemma 1 *The probability that $r|g \cdot v|/w < 1$ is less than $2\sqrt{2d}/m$.*

Proof. If $|g \cdot v| < w/r$, v lies in a slab of $(-1,1)^d$ of thickness at most $2w/(r||g||)$. The maximum area of a cross section is $\sqrt{2}$ by Ball's theorem. Hence, the volume of the slab is at most $2\sqrt{2}w/(r||g||) = 2\sqrt{2d}/m$. □

We estimate an upper bound on $2\sqrt{2d}/m$ using the maximum d and using the minimum perturbation ratio of all ambiguous predicates that are nondegenerate as an estimate for the lower bound of m. This false identity probability estimate neglects the truncation error of the linear Taylor series. One can estimate this error by comparing $e(p + rv)$ to $e(p + rv/2)$ and can control it by shrinking r.

5 Results

We tested SDD on eight computational geometry algorithms on polyhedrons. Table 1 lists the input to the tests, and subsequent tables list the results. The inputs are displayed in the papers cited below. We provide the software at `https://github.com/Robust-Geometric-Computation`.

5.1 Rational predicates

The first five tests use rational predicates. 1) We pack three polyhedrons into a box with an algorithm [2] that composes ten Minkowski sums and Boolean operations. 2) We compute a constrained Delaunay triangulation of a polyhedron with an algorithm [23] that places Steiner points on edges. 3) We apply the same triangulation algorithm to the Minkowski sum of two polyhedrons. 4) We compute a constrained Delaunay mesh of a polyhedron with an algorithm [24] that places Steiner points on edges and facets, and at the circumcenters of tetrahedrons. 5) We repeat a test of our algorithm [1] for approximating the free space of a four degree of freedom (4DOF) polyhedron that translates freely and rotates around its z axis.

The best prior degeneracy detection algorithm is exact rational evaluation with floating point filtering. Adaptive precision evaluation [22] is inapplicable because most of the predicates have derived parameters. For test 1, our prior work [21] eliminates all degeneracies by perturbing the vertices of the polyhedron output of each step. Topology changes are allowed. As indicated in Sec. 1.2, this approach is efficient yet lacks an error bound. For test 5, our prior work [1] uses a preliminary version of rational SDD.

Table 2 shows the test results using $k = 2$ primes. Columns p through c refer to SDD. The percentage of ambiguous predicates a ranges from 0% to 22% of which the degenerate percentage d is a large majority. The predicate evaluation time t is between 30% and 60% of the total CPU time c. At least 60% of t is for floating point interval arithmetic f, at most 25% is for modular arithmetic m, and at most 38% is for arbitrary precision interval arithmetic e. Using $k = 5$ primes increases m by median and maximum factors of 1.7 and 2.9. The next two columns compare SDD to exact evaluation of ambiguous predicates using GMP. The predicate evaluation time increases by a factor $\times t$ of up to 216 and the CPU time increases by a factor $\times c$ of up to 131. The last column lists the maximum bit complexity b of the predicates in the test: the total number of bits in the numerator after cancellation. This number is obtained as a byproduct of exact rational evaluation. It ranges from about a thousand to almost 2 million.

The degeneracy detection failure probability bound (Sec. 2) with $k = 2$ is at most 4×10^{-7} because b is at most 1810577 (for test 4b), and $n \approx 9.3 \times 10^7$ by the prime number theorem. For $k = 5$, the bound is 10^{-19}. We never see a zero residue for an unambiguous subexpression of an ambiguous predicate, and so the estimated failure rate is zero. We ran test 4b 250 times with k=1000. Each run had 44 million unambiguous subexpressions and 200,000 nonzero ambiguous predicates. The latter is derived as $p(a/100)(1-d/100) \approx 0.2$ million. The number of zero residues for each type was 6404 and 29. This implies a zero residue rate of 1 in 1.7 billion for both populations. Thus the rate for the former appears to be a good proxy for the latter. Furthermore, both are close to $1/q$ for a random 32-bit prime q, 1 in 3 billion, corresponding to a uniform distribution of residue values. In contrast, the provable bound for $k = 1$ and $b = 1810577$ is 1 in 1700. So high bit complexity has some effect, but not nearly as much as the bound indicates. Using the measured zero residue rate, the estimated degeneracy detection failure rate for $k = 2$ is 3×10^{-19}, similar to the provable bound for $k = 5$.

We can only derive the bound for $k = 5$ by evaluating in exact rational arithmetic to determine the bit-complexity b after cancellation. In contrast, the estimated rate requires negligible overhead to compute since the SDD algorithm already calculates the residues for the unambiguous subexpressions of ambiguous predicates. To estimate the probability that the geometric construction failed, the individual predicate probability $3 \cdot 10^{-19}$ should be multiplied times the number of nonzero ambiguous predicate, such as the 200,000 for test4b. The largest of these is 56 million for test 5b.

5.2 Algebraic predicates

The last three tests use both algebraic and rational predicates. 6) We repeat a test from our prior work on 4DOF motion planning [17]: sort 100000 angles at which four randomly generated pairs of robot and ob-

Table 1: Test inputs.

#	Algorithm	Shape 1	Facets	Shape 2	Facets	Shape 3	Facets
1a	packing	cube	12	glacier	32	sphere	760
1b	packing	glacier	32	glacier	32	glacier	32
1c	packing	glacier	32	sphere	760	sphere	760
1e	packing	sphere	760	sphere	760	sphere	760
2a	CDT	bull	12400				
2b	CDT	ear	32236				
2c	CDT	horse	39694				
3a	CDT of Minkowski sum	bull	12400	glacier	32		
3b	CDT of Minkowski sum	ear	32236	glacier	32		
3c	CDT of Minkowski sum	horse	39694	glacier	32		
4a	mesh	bull	12400				
4b	mesh	ear	32236				
4c	mesh	horse	39694				
5a	4DOF free space	frustum	12	tworooms	122		
5b	4DOF free space	plus	44	lattice-room	204		
6	4DOF rotations	not applicable					
7a	4DOF path	frustum	12	tworooms	122		
7b	4DOF path	plus	44	lattice-room	204		
8a	3DOF free space	r1	4	o1	736		
8b	3DOF free space	r1	4	o2	2640		
8c	3DOF free space	r1	4	o3	4628		
8d	3DOF free space	r2	14	o4	8068		

Table 2: Rational predicates: # test, p predicates in millions, a percent of p that are ambiguous, d percent of a that are degenerate, t predicate evaluation time in seconds, f, m, e percent of t for floating point, modular, and arbitrary precision arithmetic, c total CPU time in seconds, $\times t$ and $\times c$ multipliers of t and of c for exact evaluation, and b maximum bit complexity.

#	p	a	d	t	f	m	e	c	$\times t$	$\times c$	b
1a	38	9	97	11	80	13	7	30	10	4	5307
1b	35	9	93	11	72	15	13	23	13	7	13925
1c	49	9	76	19	60	19	21	49	4	2	3484
1d	902	8	79	274	66	18	16	666	6	3	9362
2a	3	1	98	0.6	75	25	0	1.8	1	1	1064
2b	6	0	98	1.2	96	4	0	4.0	1	1	942
2c	8	0	99	1.4	97	3	0	5.2	1	1	921
3a	29	2	77	11	63	9	28	37	10	4	4362
3b	54	1	51	25	53	9	38	74	5	2	4373
3c	109	1	69	33	70	7	23	113	6	3	4429
4a	45	3	92	12	83	6	11	35	4	2	600364
4b	194	2	95	51	90	4	6	167	38	12	1810577
4c	90	1	88	23	90	3	7	90	1	1	65680
5a	1113	5	91	252	79	10	11	420	216	131	292091
5b	1026	22	75	704	60	23	17	1114	100	63	115587

stacle features can have simultaneous contacts. 7) We generate a path in the test 5 approximate free space. 8) We mesh the free space of a 3DOF polyhedron that translates and rotates in a plane, which we compute with our prior algorithm [20].

In test 6, the robot and obstacle features are each generated from a pool of 12 vertices with random coordinates. Four robot/obstacle feature pairs define a rational angle polynomial. Angle parameters s and t are zeros of angle polynomials f and g and yield unit vectors

$$u = \left(\frac{1-s^2}{1+s^2}, \frac{2s}{1+s^2}\right) \qquad \text{and} \qquad v = \left(\frac{1-t^2}{1+t^2}, \frac{2t}{1+t^2}\right).$$

Table 3: Algebraic predicates: # test, p predicates in millions, a percent of p that are ambiguous, l and i percent of a that are algebraic and identities, t predicate evaluation time in seconds, f, m, e percent of t for floating point, modular, and arbitrary precision arithmetic, c total CPU time in seconds, and 10^{-r} error probability.

#	p	a	l	i	t	f	m	e	c	r
6a	35	0.2	96	96	51	5	2	93	57	*
6b	35	0.2	96	96	46	5	1	94	53	*
7a	0.2	13	45	1	1	8	13	79	1	28
7b	0.1	5	21	0	0	9	12	79	0	36
8a	1	5	48	37	1	33	14	53	2	20
8b	4	7	49	20	2	34	16	50	6	30
8c	11	6	44	22	6	35	14	51	15	16
8d	105	5	26	10	40	42	10	48	93	18

Vectors with $\text{sign}(u_y) = \text{sign}(v_y)$ (equivalent to $\text{sign}(s) = \text{sign}(t)$) are ordered by $\text{sign}(u \times v)$ (in 2D $u \times v = u_x v_y - u_y v_x$). Test 6a tests for identity by determining if $f(t) = 0$. It is also required to check that $s' - t \neq 0$ for every zero $s' \neq s$ of g. Test 6b checks if $u \times v = 0$ directly. In test 7, the path consists of rotations plus shortest paths on the polyhedron boundary of the approximate free space for a fixed angle. In test 8, we approximate the boundary patches with triangles that conform at patch boundaries, compute the arrangement of the triangles, and return the union of the cells with positive winding numbers.

There is no practical prior general degeneracy detection algorithm. The only option is separation bounds and these are impossibly small in every test. For test 6, our prior degeneracy detection algorithm is a table lookup of the factorizations of f and g, which is much faster than SDD but requires much specialized work to categorize all the angle polynomials for the domain. For test 7, our prior work prevents degeneracy by repeated geometric rounding [16], which is extremely slow (Sec. 1.2).

Table 3 shows the test results. We set $h = 265$ in the perturbation algorithm. We obtained this value by setting $h = 106$ (two times double precision) and increasing it by increments of 53 until the error estimate became tiny for tests 7 and 8. The error cannot be estimated in test 6 because every ambiguous predicate is an identity. The predicate values are the same for $h = 212$ and $h = 265$, which shows that the error estimate is conservative. Empirically, ambiguous at h bits is equivalent to identity, so algebraic predicates never require more than h bits and there are no restarts. As noted for the rational case, the error rate should be multiplied by the number of nonzero ambiguous expressions to obtain the probability that the computation failed. The error rate is very conservative, since this rate is zero.

Table 4 compares the perturbation-based (Sec. 4) and quotient-ring-based (Sec. 3) algorithms on predicates from tests 6 and 8. Test 7 is unsuitable for the residue method because the predicates required to construct a shortest path traversing m faces of a polyhedron are degree m in m square roots of rational expressions. The test 6 predicates are a) $f(t)$ or b) $(1-s^2)(2t)-(2s)(1-t^2)$ where s and t are zeros of f and g, respectively. The test 8 predicates are low-degree polynomials in three or four coordinates of points. The coordinates of a point are rational expressions in a zero of a polynomial of degree 2 or 4. The residue algorithm always returns the same result as the perturbation algorithm, which provides further evidence that the latter is correct. We compare the running times on predicates where floating point filtering fails, hence degeneracy detection is required. The residue algorithm with rational SDD ($k = 2$) is faster than the perturbation algorithm on predicates with $n = 1$ algebraic numbers but is 4 times slower with $n = 2$, 400 times slower with $n = 3$, and 1150 times slower with $n = 4$. The residue algorithm using exact rational evaluation for ambiguous predicates is 2 to 24 times slower than using SDD.

Table 4: Predicate evaluation: # test, n algebraic arguments, p predicates, t perturbation algorithm time in microseconds, $\times p$ and $\times e$ multipliers for residue algorithm with SDD and exact evaluation.

#	n	p	t	$\times p$	$\times e$
6a	1	84000	560	0.2	2.4
6b	2	84000	509	3.5	84
8	3	32500	25	400	1040
8	4	22000	64	1150	2600

6 Discussion

The tests confirm the claims in the introduction. 1) Identities are common in algorithms that construct objects. Tests 1, 5, 7, and 8 have many constructions and 5%–10% of the predicates are identities, whereas the other tests have few constructions and under 2% identities. 2) Identities are a computational bottleneck for prior degeneracy detection algorithms. For rational predicates, exact evaluation is median 8 and maximum 216 times slower than SDD on tests 1, 3, 4, and 5, which have high expression depth. For algebraic predicates, separation bounds are useless for all the tests. 3) SDD is reliable and fast. For parameter settings with a minuscule estimated error rate ($k = 2$ and $h = 256$), the predicate evaluation time is at most 60% of the CPU time. Hence, no alternate algorithm could reduce the overall running time by more than a factor of two. 4) The running time grows slowly with the parameters k and h, so there is no need for fine-tuning. For example, $h = 265$ is larger than necessary for most of the

tests, but the additional cost is insignificant. 5) For algebraic predicates, the perturbation-based algorithm far outperforms the quotient-ring-based algorithm.

The error estimate for the perturbation-based SDD algorithm depends on the assumption that the predicate is analytic on $[0, r]$. In practice, a value of h that yields a small estimate also ensures that ambiguity at h is equivalent to identity. This implies not only that a) a nondegenerate yet ambiguous $e(p)$ is unlikely but also that b) an identity $e(p)$ rarely becomes a non-identity $e(q)$. Since identity is more restrictive than non-identity, c) going from non-identity to identity is even less likely. The probability of an undetected analytic failure is the product of (a) and (b). Although we do not have an analysis of these probabilities as for an analytic false identity, we feel it is safe to disregard their product in comparison to the analytic false identity probability.

References

[1] C. Arluck, V. Milenkovic, and E. Sacks. Approximate free space construction and maximum clearance path planning for a four degree of freedom robot. In *Proceedings of the Canadian Conference on Computational Geometry*, 2018.

[2] F. Avnaim and J.-D. Boissonnat. Simultaneous containment of several polygons. In *Symposium on Computational Geometry*, pages 242–247, 1987.

[3] E. Berberich, P. Emeliyanenko, A. Kobel, and M. Sagraloff. Exact symbolic-numeric computation of planar algebraic curves. *Theoretical Computer Science*, 491(C):1–32, 2013.

[4] J. Blömer. A probabilistic zero-test for expressions involving roots of rational numbers. In G. Bilardi, G. F. Italiano, A. Pietracaprina, and G. Pucci, editors, *Algorithms — ESA' 98*, pages 151–162. Springer Berlin Heidelberg, 1998.

[5] H. Bronnimann, C. Burnikel, and S. Pion. Interval arithmetic yields efficient dynamic filters for computational geometry. *Discrete Applied Mathematics*, 109(1-2):25–47, 2001.

[6] H. Brönnimann, I. Z. Emiris, V. Y. Y. Pan, and S. Pion. Sign determination in residue number systems. *Theoretical Computer Science*, 210:173–197, 1999.

[7] C. Burnikel, S. Funke, K. Mehlhorn, S. Schirra, and S. Schmitt. A separation bound for real algebraic expressions. *Algorithmica*, 55(1):14–28, 2009.

[8] CGAL, Computational Geometry Algorithms Library. http://www.cgal.org.

[9] I. Emiris, B. Mourrain, and E. Tsigaridas. Separation bounds for polynomial systems. *Journal of Symbolic Computation*, page 128–151, 2020.

[10] I. Z. Emiris. A Complete Implementation for Computing General Dimensional Convex Hulls. Technical Report RR-2551, INRIA, May 1995.

[11] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A multiple precision binary floating point library with correct rounding. *ACM Transactions on Mathematical Software*, 33:13, 2007.

[12] D. Halperin. Controlled perturbation for certified geometric computing with fixed-precision arithmetic. In *ICMS*, pages 92–95, 2010.

[13] C. Li, S. Pion, and C. Yap. Recent progress in exact geometric computation. *Journal of Logic and Algebraic Programming*, 64:85–111, 2005.

[14] J. Masterjohn, V. Milenkovic, and E. Sacks. Finding intersections of algebraic curves in a convex region using encasement. In *Proceedings of the Canadian Conference on Computational Geometry*, pages 91–97, 2018.

[15] K. Melhorn and S. Näher. *The LEDA platform for combinatorial and geometric computing*. Cambridge University Press, 1999.

[16] V. Milenkovic and E. Sacks. Geometric rounding and feature separation in meshes. *Computer-Aided Design*, 108:12–18, 2019.

[17] V. Milenkovic, E. Sacks, and N. Butt. Fast detection of degenerate predicates in free space construction. *International Journal of Computational Geometry & Applications*, 29(03):219–237, 2019.

[18] V. Milenkovic, E. Sacks, and S. Trac. Robust free space computation for curved planar bodies. *IEEE Transactions on Automation Science and Engineering*, 10(4):875–883, 2013.

[19] R. E. Moore. *Methods and Applications of Interval Analysis*. SIAM Studies in Applied Mathematics. SIAM, Philadelphia, 1979.

[20] E. Sacks, N. Butt, and V. Milenkovic. Robust free space construction for a polyhedron with planar motion. *Computer-Aided Design*, 90C:18–26, 2017.

[21] E. Sacks and V. Milenkovic. Robust cascading of operations on polyhedra. *Computer-Aided Design*, 46:216–220, Jan. 2014.

[22] J. R. Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete and Computational Geometry*, 18:305–363, 1997.

[23] H. Si and K. Gartner. 3D boundary recovery by constrained delaunay tetrahedralization. *International Journal for Numerical Methods in Engineering*, 85:1341–1364, 2011.

[24] H. Si and J. R. Shewchuk. Incrementally constructing and updating constrained Delaunay tetrahedralizations with finite-precision coordinates. *Engineering with Computers*, 30:253–269, 2014.

[25] C. Yap. Robust geometric computation. In J. E. Goodman and J. O'Rourke, editors, *Handbook of discrete and computational geometry*, chapter 41, pages 927–952. CRC Press, Boca Raton, FL, second edition, 2004.

Unfolding Some Classes of One-Layer Polycubes

Josef Minařík*

Abstract

An *unfolding* of a polyhedron is a cutting along its surface such that the surface remains connected and it can be flattened to the plane without any overlap. An *edge-unfolding* is a restricted kind of unfolding, we are only allowed to cut along the edges of the faces of the polyhedron. A *polycube* is a special case of orthogonal polyhedron formed by glueing several unit cubes together face-to-face. In the case of polycubes, the edges of all cubes are available for cuts in edge-unfolding. We focus on one-layer polycubes and present several algorithms to unfold some classes of them. We show that it is possible to edge-unfold any one-layer polycube with cubic holes, thin horizontal holes and separable rectangular holes. The question of unfolding general one-layer polycubes remains open.

1 Introduction

An *orthogonal polyhedron* is a polyhedron whose edges are parallel to the Cartesian axes and whose faces meet at right angles. Each face of an orthogonal polyhedron is parallel to one of the Cartesian coordinate planes. A *polycube* is a special case of an orthogonal polyhedra. It is formed by glueing several unit cubes together by whole faces. Polycubes are three-dimensional generalisations of planar polyominoes. A one-layer polycube is a polycube of height 1. In other words, the centers of all unit cubes are in one plane. One-layer polycubes with non-zero genus have some *holes* in them. If a hole consists of only one missing unit cube, we call this hole *cubic*.

An *unfolding* of a polyhedron is a cutting along its surface such that the surface remains connected and it can be flattened to the plane without any overlap. We usually only care about interior overlap and there may be touching edges after unfolding to the plane. *Edge-unfolding* is a restricted kind of unfolding. In this case, we can only cut along the edges of the faces of the polyhedron. It is quite easy to show that there exist non-convex orthogonal polyhedra that cannot be edge-unfolded [7]. We are mostly interested in edge-unfolding of polycubes. In the case of polycubes, the edges of all cubes are available for cuts. This means that we can cut the faces of our polyhedron along the edges of the 1×1 grid. Different kinds of unfolding are discussed in greater detail in [6] and [8].

Unfoldings of many classes of orthogonal polyhedra have been studied. For example orthostacks, orthotubes [1] or Manhattan towers [4]. There are also known unfoldings of special cases of polycubes, such as well-separated orthotrees [3]. One-layer orthogonal polyhedra with arbitrary genus g can be edge-unfolded using only $2(g-1)$ additional cuts [2]. Kiou, Poon and Wei proved that it is possible to unfold one-layer polycubes with sparse cubic holes [5], which are one-layer polycubes such that each connected component in a column contains at most one hole. We generalize this result and present an algorithm for unfolding general one-layer polycubes with cubic holes.

Theorem 1 *It is possible to edge-unfold any one-layer polycube with cubic holes.*

In Sections 2.5 and 2.6 we further generalize this approach for other classes of one-layer polycubes.

Definition 1 *A hole is called thin horizontal if it is a rectangle of height 1.*

Theorem 2 *It is possible to edge-unfold any one-layer polycube with thin horizontal holes.*

Definition 2 *We call a set of rectangles separable if it satisfies the following property. If we extend any edge of any rectangle to a line, it does not cut any other rectangle.*

Theorem 3 *It is possible to edge-unfold any one-layer polycube with separable rectangular holes.*

2 Results

2.1 Definitions

Let us consider a one-layer polycube $\mathcal{P}$ placed in the xy plane such that the centers of all cubes have integer coordinates. The exact position of the polycube is not important, we only need to be able to index the cubes by coordinates. By a cube with coordinates x, y we mean a cube whose center has such coordinates. Let us denote the set of holes $\mathcal{H}$. A one-layer polycube $\mathcal{P}$ has a top base T and bottom base B. There also is

*Department of Applied Mathematics, Faculty of Mathematics and Physics, Charles University

an *external boundary* E and several *internal boundaries* $\mathcal{I} = \{I_h \mid h \in \mathcal{H}\}$, each corresponding to some hole h. The boundaries are formed by cyclic stripes of unit squares.

Since we are only interested in one-layer polycubes, we will display them as 2-dimensional objects. In all the figures, we are looking at the polycube from above, which means that we see the top base. With respect to that, we will be using terms such as "left", "right", "up" and "down" to describe directions. For example, the boundary of a hole consists of four not necessarily connected parts: left, right, upper and lower.

We require the surface of $\mathcal{P}$ to be *simple*, that is, every edge of $\mathcal{P}$ is incident to exactly two 1×1 squares on the surface of $\mathcal{P}$. The holes are not allowed to "touch" each other by corners nor to "touch" the external boundary, examples of such disallowed configurations are in Figure 1.

(a) Example of holes touching.

(b) Example of a hole touching the external boundary.

Figure 1: Examples of polycubes that are **not** allowed.

We will describe several algorithms for unfolding polycubes. Let n denote the number of unit cubes forming $\mathcal{P}$. All of the presented algorithms can be implemented in $\mathcal{O}(n)$ time if we are provided with a reasonable representation of the polycube as input (for example a sorted list of all unit cubes). We mainly focus on the existence of the unfolding and the existence of such an algorithm is more important for us than the exact implementation. However, an implementation of all the presented algorithms should be mostly straightforward.

2.2 No holes

Let us start with a simple example to get familiar with the techniques we will be using. Without holes, we only need to unfold B, T and E. The algorithm starts with the external boundary E. The external boundary can be unfolded to a single stripe of height 1. Let us place this stripe horizontally in the plane. We do not cut B and T. We simply connect them to the unfolded E, each being placed in a different half-plane. They are connected to E by the cube with the lowest y coordinate (if there are two or more, we can choose one arbitrarily). The resulting shape is connected and it is easy to see that there are no overlaps. See Figure 2 for an example of an unfolding of a polycube without holes.

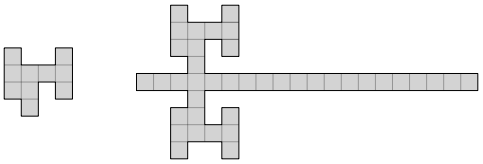
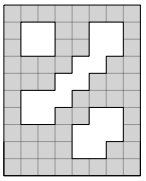


Figure 2: Unfolding of a one-layer polycube without holes.

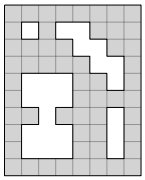
Note that this is an edge-unfolding in the standard sense, we only used cuts along the faces of the polyhedron. We did not use any additional cuts along the edges of the unit cubes.

2.3 Wide holes

Definition 3 *A hole h is called narrow if I_h contains two squares with distance 1 facing each other. A hole is wide if it is not narrow. In other words, we say that a hole is wide if it satisfies the following property. If there is a missing cube with a center on coordinates $[x, y]$, then there is at least one missing cube at coordinates $[x+1, y], [x-1, y]$ and at least one missing cube at coordinates $[x, y-1], [x, y+1]$.*



(a) Examples of wide holes.



(b) Examples of **non**-wide holes.

Figure 3: Examples of wide and non-wide holes.

We can unfold one-layer polycubes with wide holes using the following algorithm. We start by unfolding B, T and E in the same way as above in Section 2.2. Thanks to the wideness of the holes, there is a lot of space inside B and T. For every hole h, we will unfold I_h in two steps. In the first step, we unfold the upper and the lower faces of I_h. In the second step, we unfold the left and the right faces of I_h. In the first step, we unfold parts of I_h into the top base T, inside the holes. There will be no overlap because the holes are wide. The second step is almost the same, the only difference is that we use the bottom base B instead. Figure 4 shows an example of an unfolding produce by this algorithm.

We again used only cuts along the edges of $\mathcal{P}$.

2.4 Cubic holes

This algorithm is slightly more complicated, we will need to cut T and B. Note that cutting T or B is necessary to unfold even a single cubic hole. The idea

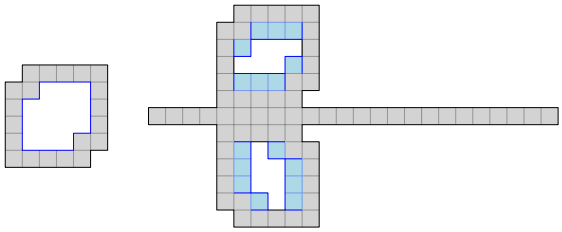


Figure 4: Unfolding of a one-layer polycube with a wide hole.

is similar to the algorithm in Section 2.3, we will unfold some parts (the upper and the lower faces) of the internal boundaries by connecting them to T and some of them (the left and the right faces) by connecting them to B.

The first steps are still the same: we unfold the external boundary E. Now let us color the squares of T using orange and red. The squares whose y-coordinate is 0 or 1 modulo 4 will be orange, the remaining ones will be red. In other words, we are coloring pairs of rows orange and red. Example of such coloring can be seen in Figure 6. Consider the connected components formed by orange or red squares, which would be formed by cutting edges separating squares of different colors. The leftmost and the rightmost square of every connected component must be incident to E. This is due to the holes being cubic; they are not big enough to separate components.

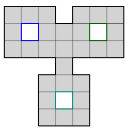


Figure 5: Example of a one-layer polycube with cubic holes.

(a) Coloring of the top base.

(b) Coloring of the bottom base.

Figure 6: Coloring of the one-layer polycube with cubic holes in Figure 5.

We will connect all the orange components to the external boundary on their left side by their leftmost square. Analogously every red component will be connected to the boundary by its rightmost square. The current stage of unfolding is illustrated in Figure 7. Quite simple casework shows that there is a distance of at least 2 between any pair of connected components

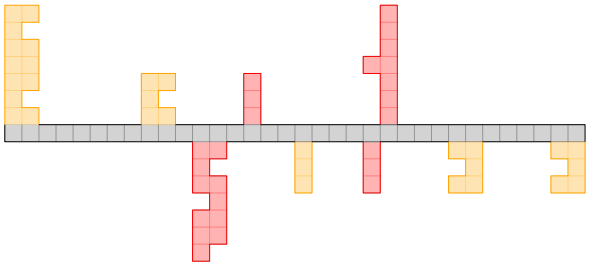


Figure 7: The first step of unfolding the polycube in Figure 5.

after placing them in the plane next to unfolded E. Suppose there are two stripes that have a distance of less than two. There are two cases:

1. Both of the stripes have the same color. We can suppose without loss of generality they are orange. Now consider where these stripes come from in the polycube. They either come from the same pair of rows or a different pair of rows. In the first case, the distance would have to be at least 3, in the second case, it would have to be at least 2, a contradiction. See Figure 8 for an illustration.

2. The stripes have different colors. Without loss of generality, we can assume that the left stripe is orange. Let us again consider where those stripes were before the unfolding. If they don't come from the neighboring pair of rows, the distance would obviously have to be at least 4. There are two remaining (symmetric) cases: the red rows could be either above or below the orange rows. In both of those cases, the distance is at least 2, contradiction again, see Figure 9.

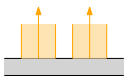
Now, we will take every left or right face of the internal boundaries and connect it to the only square of the already unfolded top base it is incident to. There are no overlaps because the connected components have a distance of at least 2 and there is enough space for two unit squares between them.

We repeat the process for the bottom base B. This time, we color pairs of columns instead of rows. This base and parts of holes are unfolded to the opposite half-plane so there will be no overlaps with previously placed parts.

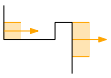
2.5 Thin horizontal holes

The approach in Section 2.4 can be quite easily generalized to holes of dimensions $1 \times k$, but only if all of them are oriented in the same way (either all horizontal or all vertical).

Definition 4 *A hole is called thin horizontal if it is a rectangle of height 1.*



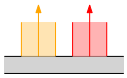
(a) Suppose the distance of two orange stripes is at most 1.



(b) In the first case, the distance after unfolding must be at least 3.

(c) In the second case, the distance after unfolding must be at least 2.

Figure 8: Two orange stripes cannot be too close to each other.



(a) Suppose the distance of an orange and red stripe is at most 1.

(b) In the first case, the distance after unfolding must be at least 2.

(c) In the second case, the distance after unfolding must be at least 2.

Figure 9: Two stripes of different colors cannot be too close to each other.

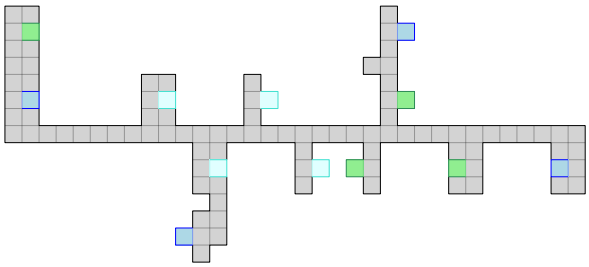
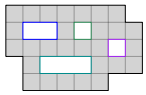
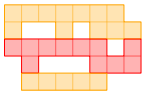


Figure 10: Unfolding of the one-layer polycube with cubic holes in Figure 5.



(a) Example of a polycube with thin horizontal holes.



(b) Coloring of the top base.

Figure 11: Coloring of a one-layer polycube with thin horizontal holes.

Let us start by unfolding E, T and longer (horizontal - upper and lower) faces of holes in the same way as in Section 2.4. The Figures 12 and 13 show the first two steps of the algorithm.

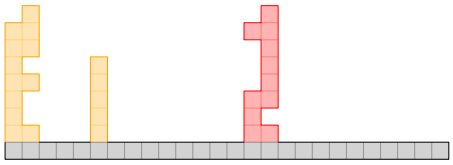


Figure 12: The first step of unfolding the polycube in Figure 11.

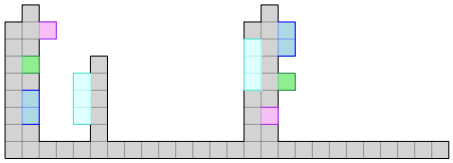


Figure 13: The second step of unfolding the polycube in Figure 11.

It remains to unfold the bottom base B and the short (left and right) faces of holes. We cannot do that in the same way as before, because if we cut B into stripes of width 2, they would not be necessarily incident to the external boundary. We can instead connect one face of each hole $h \in \mathcal{H}$ to one of the two already unfolded faces of I_h. In case of holes in the orange stripes, we unfold the right face, in case of holes in the red stripes, we unfold the left face. Let us look at the already unfolded horizontal faces of I_h. One of the faces is unfolded "inside" of a stripe but the other is "outside". For example, consider a hole in the lower row of a red stripe: the upper face of this hole is unfolded "inside" the red stripe while the lower face is unfolded "outside" of an orange stripe. The face unfolded outside has empty space around it and we can connect the one face of I_h here (this face is only one 1×1 square). There cannot be an overlap - we are outside a stripe, so there could only be a face of some hole or external boundary. External boundary cannot be there because it has distance at least 1 from all holes (and it also lies in the opposite direction than the one in which we place the face).

The same is true for holes, they are at a distance of at least 1 from each other, so the unfolded longer faces are not next to each other. Two faces unfolded in this step cannot overlap either because they are unfolded in the same direction.

The last part is the bottom base B and exactly one face of every hole. This is rather simple since all of the remaining faces are just 1×1 squares. We can unfold the rest in a similar fashion to unfolding wide holes in Section 2.3. See the Figure 14 for an example of the last steps.

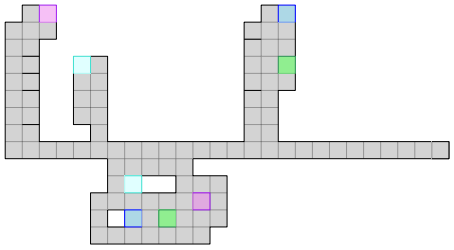


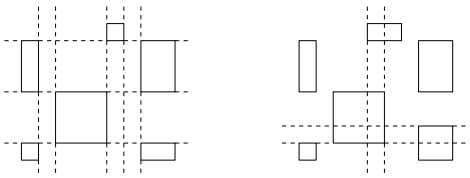
Figure 14: Unfolding of the polycube in Figure 11.

2.6 Separable rectangular holes

A slightly more general class of one-layer polycubes than the polycubes with cubic holes can also be unfolded using a similar algorithm.

Definition 5 *We call a set of rectangles separable if it satisfies the following property. If we extend any edge of any rectangle to a line, it does not cut into any other rectangle (it does not contain an interior point of any other rectangle).*

One-layer polycubes whose holes are separable rectangles can be unfolded by an algorithm very similar to the one in Section 2.4. Note that cubic holes are trivially separable, thus one-layer polycubes with cubic holes can also be unfolded using this algorithm.



(a) Example of separable rectangles.

(b) Example of **non**-separable rectangles.

Figure 15: Examples of separable and non-separable rectangles.

Let us extend the edges of all rectangles that are parallel to x-axis to lines. This creates several horizontal

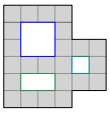


Figure 16: Example of a one-layer polycube with separable rectangular holes.

stripes. Analogously, we can create vertical stripes. Instead of coloring pairs of neighboring rows or columns of T and B as in Section 2.4, we color pairs of neighboring horizontal stripes. You can see an example of such coloring in Figure 17. The distance between any pair of stripes is again at least 2 for the same reasons as in the algorithm for cubic holes. We omit the case analysis this time.

(a) Coloring of the top base.

(b) Coloring of the bottom base.

Figure 17: Coloring of the polycube in Figure 16.

The rest of the algorithm is the same as in Section 2.4. We consider connected components of both colors. The leftmost and rightmost squares of connected components are incident to E and will be connected on the left or right side depending on their color. We then unfold the horizontal and vertical faces of internal boundaries separately. Since the distance between neighboring stripes is at least 2, there are no overlaps. Figures 18 and 19 show the steps of this algorithm.

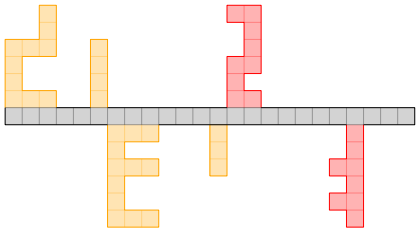


Figure 18: The first step of unfolding of the polycube in Figure 16.

3 Conclusion

We presented several linear-time algorithms for edge-unfolding of special cases of one-layer polycubes. The question of unfolding one-layer polycubes with arbitrary holes remains open. Interestingly, we are able to unfold one-layer polycubes with very small (cubic) holes and with very big (wide) holes. These are, in some sense, opposite types of one-layer polycubes. Generalising our

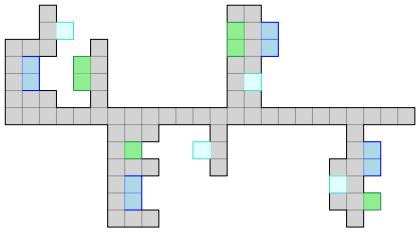


Figure 19: Unfolding of the polycube in Figure 16.

approach to unfold other classes of one-layer polycubes seems rather difficult since it relies on being able to cut the top and bottom bases into stripes such that all the connected components are incident to the external boundary.

References

[1] T. Biedl, E. Demaine, M. Demaine, A. Lubiw, M. Overmars, J. O'Rourke, S. Robbins, and S. Whitesides. Unfolding some classes of orthogonal polyhedra. *Proc. 10th Canad. Conf. Comput. Geom.*, 01 1998.

[2] Y.-J. Chang and H.-C. Yen. Improved algorithms for grid-unfolding orthogonal polyhedra. *Internat. J. Comput. Geom. Appl.*, 27(1-2):33–56, 2017.

[3] M. Damian, R. Flatland, H. Meijer, and J. O'Rourke. Unfolding well-separated orthotrees. pages 23–25, 2005.

[4] M. Damian, R. Flatland, and J. O'Rourke. Unfolding Manhattan towers. *Comput. Geom.*, 40(2):102–114, 2008.

[5] M.-H. Liou, S.-H. Poon, and Y.-J. Wei. On edge-unfolding one-layer lattice polyhedra with cubic holes. *Computing and combinatorics*, 8591:251–262, 2014.

[6] J. O'Rourke. Unfolding orthogonal polyhedra. *Surveys on discrete and computational geometry*, 453:307–317, 2008.

[7] J. O'Rourke. Unfolding polyhedra. 08 2008. `https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.150.8196`

[8] J. O'Rourke. Unfolding polyhedra. *Proc. 31st Canad. Conf. Comput. Geom.*, 2019.

Diverse Non Crossing Matchings*

Neeldhara Misra† Harshil Mittal‡ Saraswati Girish Nanoti§

Abstract

A perfect matching M on a set P of n points is a collection of line segments with endpoints from P such that every point belongs to exactly one segment. A matching is *non-crossing* if the line segments do not cross. Two matchings M and N are said to be *compatible* if there are no crossings among any pair of line segments in $M \cup N$. We introduce a notion of diverse non-crossing matchings: a pair of perfect matchings M and N are k-diverse if, for every $p \in P$, the distance between the matched partners of p in M and N is at least k. In this contribution, we describe a polynomial time algorithm to determine if a set of points in convex position admits two compatible and perfect NCMs that are k-diverse. For points in convex position, we also show that if a perfect matching M is given as input, then we can determine, in polynomial time, if another perfect matching N exists that is compatible with M and is such that M and N are k-diverse. Finally, we also establish that every point set in general position admits a pair of compatible and perfect NCMs. The first two results also hold for bichromatic points, and we also give a characterization for when a bichromatic point set in convex position admits a pair of perfect and disjoint NCMs.

1 Introduction

Matching problems involve partitioning a set of objects into pairs subject to some constraints. For example, in the context of graphs, we are given a binary relation over the set of objects and require the pairs to be related. In a geometric setting, the set typically consists of geometric objects (Aloupis et al., 2013), and such problems have received a lot of attention because of their practical relevance.

Our focus is on the setting of matching points using line segments. In particular, given a set P of n points in the plane $\mathbb{R}^2$, we are interested in matching them with straight line segments. We focus on perfect non-crossing matchings (NCMs), i.e, matchings where every point is matched and no two line segments cross. Unless otherwise mentioned, we assume that all matchings are perfect.

It turns out that any collection of points admits a NCM and that this can be found in $O(n \log n)$ time (Hershberger and Suri, 1990; Lo et al., 1994). Many studies on NCMs focus on optimizing some structural property of the matching, such as the maximum, minimum, or average edge length. Two NCMs M and N are said to be *disjoint* if every point has a different matched partner in both matchings, and *compatible* if the segments in $M \cup N$ do not cross.

For optimization problems, the decision or search version of the question seeks to find *some* optimal solution, while the counting version asks to enumerate *all* optimal solutions. In many application scenarios, the former is not sufficient, while the latter is too demanding in terms of computational expense. This motivates the notion of demanding not all but a select collection of solutions. In most applications, the requirement is not just for a multitude of solutions, but for an "interesting" collection of solutions: for example, informally speaking, solutions that are minor variations of one another and are very similar may not be very useful in most settings.

The existence of a *diverse* collection of solutions has been explored in several settings recently. Studies on diverse solutions have focused on a wide array of problems including, but are not limited to, vertex cover (Baste et al., 2022), matchings (Fomin et al., 2020), stable matchings (Ganesh et al., 2021), matroids (Fomin et al., 2021), satisfiability (Nadel, 2011), Kemeny rank aggregation (Arrighi et al., 2021), etc.

To propose that we find "diverse" solutions, we need a notion of distance between solutions. In the setting of matchings between points in $\mathbb{R}^2$, a natural notion of "distance between matchings M and N" would be an aggregation of the distance between the matched partners of all the points in the two matchings. The aggregation function that we work with picks out the smallest such distance. In particular, using $M(\cdot)$ to denote the matched partner of a point p in a matching M, we define the distance between two matchings M and N over a point set P as $\min_{p \in P} d(M(p), N(p))$. Note that

*The authors are grateful for support from the SERB-MATRICS grant MTR/2017/001033, the SERB-ECR grant ECR/2018/002967, and CSIR. We also thank the anonymous referees for their detailed comments.

†IIT Gandhinagar , neeldhara.m@iitgn.ac.in
‡IIT Gandhinagar , mittal_harshil@iitgn.ac.in
§IIT Gandhinagar , nanoti_saraswati@iitgn.ac.in

the we have used the term "distance" informally and this function does not satisfy the triangle inequality. We say that a collection of matchings $\mathcal{M}$ is k-diverse for some positive number k if the distance between every pair of matchings in $\mathcal{M}$ has a distance of at least k between them. Throughout our discussions, we focus on the problem of finding **two** matchings.

Our Contributions. We propose the following natural computational questions:

DIVERSE NCMS
(DIVERSE COMPATIBLE NCMS)

Input: A set P of $2n$ points and a positive rational number k.
Question: Does P admit two perfect matchings that are k-diverse and compatible, i.e., two DC-NCM's?

ANOTHER DIVERSE NCM
(ANOTHER DIVERSE COMPATIBLE NCM)

Input: A set P of $2n$ points, a perfect matching M over P, and a positive rational number k.
Question: Is there a perfect NCM N over P such that M and N are k-diverse (and compatible)?

We first show that any monochromatic point set P in general position with an even number of points such that $|P| \geqslant 4$ admits two compatible NCMs. Note that this is easy to see for points in convex position: a set of alternating edges on the convex hull and the remaining edges of the convex hull form a pair of compatible matchings. For points in general position, we generalize this idea by considering the layer decomposition and peeling off convex layers with an even number of points, and carefully matching across layers when we encounter layers with an odd number of points. We also characterize bichromatic point sets in convex position that admit two disjoint non-crossing matchings[1].

Theorem 1 (Disjoint Matchings) *Any point set P in general position admits two compatible perfect NCMs. A bichromatic point set P in convex position admits two disjoint and perfect NCMs if and only if the orbit of each point contains at least two points of the opposite colour.*

We next propose the computational problem of finding a matching that is diverse with respect to and, optionally, compatible with a given matching. We show that when points are in convex position, we can find such a matching in polynomial time. We use a dynamic programming approach here, considering subproblems corresponding to contiguous subintervals of the convex hull.

Theorem 2 (Another Diverse Matching) *For both monochromatic and bichromatic points in convex position, the problems* ANOTHER DIVERSE NCM *and* ANOTHER DIVERSE COMPATIBLE NCM *admit polynomial time algorithms.*

Finally, we consider the problem of finding a pair of diverse and compatible matchings. We demonstrate a polynomial time algorithm for points in convex position. For this algorithm, we note that any solution can be viewed equivalently as a collection of disjoint non-overlapping polygons. We prove a structural lemma which shows that there always exists an optimal solution consisting of polygons with a constant number of sides. We can then leverage this to come up with a dynamic programming algorithm that considers, as before, subproblems corresponding to contiguous subintervals of the convex hull, and makes progress by guessing all possible choices for the polygon that the first point on the subinterval belongs to.

Theorem 3 (Diverse Compatible Matching) *For both monochromatic and bichromatic points in convex position,* DIVERSE COMPATIBLE NCMS *admits a polynomial time algorithm.*

Related Work. The task of finding a matching that minimizes the length of the longest edge is called the bottleneck NCM problem and is known to be NP-complete in general and tractable for points in convex position and other special cases, and has been well-studied for monochromatic and bichromatic points (Abu-Affash et al., 2014; Carlsson et al., 2015; Savić and Stojaković, 2017, 2022; Biniaz et al., 2014).

Other variants of the problem such as those which involve minimizing the length of the shortest edge or maximizing the length of the longest edge are tractable (Mantas et al., 2021). Finally, to the best of our knowledge, the complexity of finding a matching that maximizes the length of the shortest edge is open.

In the context of the setting where we have a point set and a matching, it was conjectured by Aichholzer et al. (2009) that for every perfect matching M of a point set P such that $|P|$ is a multiple of four, there is another perfect matching, N of P such that M and N are compatible. This was subsequently proved by Ishaque et al. (2012) using a constructive argument that also leads to an efficient method for constructing the matching N. It is also known that the conjecture does not hold when $|P|$ is not a multiple of four Aichholzer et al. (2009).

[1] We refer the reader to Section 2 for the formal definitions of the terminology used here.

Organization of the paper. Due to lack of space, we defer the proofs of Theorem 1 and Theorem 2 to the full version of the paper. We provide most of the details towards showing Theorem 3 in Section 3, only deferring the argument of correctness and remarks about the bichromatic case to the full version.

2 Preliminaries

In the setting of monochromatic points, we use P typically to denote a set of $2n$ points in $\mathbb{R}^2$ with $n > 1$. When we work with points in general position, we will use $\mathcal{P}$ to denote the convex hull of P. In case of convex point sets, we label the points of P by $p_0, p_1, \ldots, p_{2n-1}$ in positive (counterclockwise) direction around the convex hull. To simplify the notation, we will generally use only indices when referring to points. We write $\{i, \ldots, j\}$ to represent the sequence $i, i+1, i+2, \ldots, j-1, j$. All operations are calculated modulo $2n$. Note that i is not necessarily less than j, and that $\{i, \ldots, j\}$ is not the same as $\{j, \ldots, i\}$.

A bichromatic set of points is a point set P equipped with a coloring function $\mathfrak{c} : P \to \{0, 1\}$ that classifies each point as either "red" (points for which $\mathfrak{c}(p) = 0$) or "blue" (points for which $\mathfrak{c}(p) = 1$). We usually denote these sets by R and B respectively, with $P = R \cup B$ and $|R| = |B| = n$, and again, we assume $n > 1$.

We say that two line segments s and t in the plane *cross* if there is a point on the plane which is not an endpoint of either s and t that belongs to both s and t. In particular, note that if $s = t$, then s and t cross each other.

The *convex hull* of a point set is the smallest convex polygon that contains all the points of it. The *convex layers* or the *onion decomposition* of a set of points are a sequence of nested convex polygons having the points as their vertices. The outermost one is the convex hull of the points and the rest are formed in the same way recursively. The innermost layer may be degenerate, consisting only of one or two points. The number of polygons in onion decomposition of a point set is called its *layer depth*.

A *perfect matching* on the set P is a set of n straight line segments whose endpoints are points in P such that each point is the endpoint of exactly one line segment. For bichromatic points sets, we further require that each line segment has one red and one blue endpoint. If the line segments do not cross, we refer to such a matching as a (bichromatic) non-crossing matching. All matchings are both perfect and non-crossing unless mentioned otherwise.

We usually use the notation M or N to refer to matchings. With a slight abuse of notation, given a matching M over P and a point $p \in P$, we use $M(p)$ to denote the matched partner of p in M, that is, the point q such that the segment connecting p and q belongs to M. Two matchings M and N are called *disjoint* if the matched partners of all points in p are different in both, i.e, for all $p \in P$, we have that $M(p) \neq N(p)$, and *compatible* if the segments in the multiset $M \cup N$ do not cross. Note that all compatible matchings are disjoint, while the converse may not be true.

We define the *distance* between two matchings M and N over a point set P, denoted $D_P(M, N)$, as $\min_{p \in P}(\text{dist}(M(p), N(p)))$, where $dist(\cdot, \cdot)$ denotes the Euclidean distance between two points. We also refer to this as the diversity of the set $\{M, N\}$ or the diversity between M and N. Further, we say that a pair of matchings M and N over P are k-diverse if $D_P(M, N) \geq k$. Note that if M and N are *not* disjoint, then they are 0-diverse. If the point set P is clear from the context, we may drop the subscript P from the notation for distances and diversity.

We now introduce some terminology that is relevant to bichromatic point sets.

Definition 1 (Balanced, Blue-heavy, Red-heavy) *A set of points is balanced if it contains the same number of red and blue points. If the set has more red (blue) points than blue (red), we say that it is red-heavy (blue-heavy).*

Lemma 1 (Savić and Stojaković (2022)) *Every balanced set of points can be matched.*

Definition 2 (Feasible pair.) *We say that (i, j) is a feasible pair if there exists a matching containing (i, j). We refer to i as a feasible neighbour of j and vice versa.*

Lemma 2 (Savić and Stojaković (2022)) *A pair (i, j) is feasible if and only if i and j have different colors and $\{i, \ldots, j\}$ is balanced.*

Definition 3 (Functions o^+ and o^-.) *[Savić and Stojaković (2022)] By $o^+(i)$ we denote the first point starting from i in the positive direction such that $(i, o^+(i))$ is feasible. By $o^-(i)$ we denote the first point starting from i in the negative direction such that $(o^-(i), i)$ is feasible.*

As we assume that the given point set is balanced, Lemma 2 guarantees that both o^+ and o^- are well-defined. It also turns out that o^- is the inverse function of o^+ as mentioned in Savić and Stojaković (2022). We denote the composition of o^+ function k times on a point p as $o^k(p)$ and also use the notation $o(p)$ to mean $o^+(p)$.

Definition 4 (Orbit) *[Savić and Stojaković (2022)]*

An orbit of i, denoted by $\mathcal{O}(i)$, is defined by $\mathcal{O}(i) := \left\{o^k(i) : k \in \mathbb{Z}\right\}$. By $\mathcal{O}(P)$ we denote the set of all orbits of a convex point set P, that is $\mathcal{O}(P) := \{\mathcal{O}(i) : i \in P\}$.

3 DC-NCM for points in Convex Position

Suppose that the points of P are in convex position. Let $\mathcal{F}$ be a collection of even-length simple convex polygons, each of length ≥ 4. We say that $\mathcal{F}$ is a *feasible collection of polygons on P* if the following hold true:

- Every $p \in P$ is a vertex of exactly one polygon in $\mathcal{F}$, and every polygon in $\mathcal{F}$ has all its vertices in P.
- No edge of a polygon in $\mathcal{F}$ crosses an edge of another polygon in $\mathcal{F}$.

For any even-length simple polygon T of length ≥ 4,

- let $partners(T)$ denote the set of all unordered pairs $\{u, v\}$ of vertices of T such that exactly one vertex of T appears between u and v, when one traverses from u to v in counter-clockwise direction along the boundary of T.
- let $quality(T)$ denote the minimum of dist(u, v) over all pairs $\{u, v\}$ in $partners(T)$.

For any feasible collection $\mathcal{F}$ of polygons on P, let $quality(\mathcal{F})$ denote the minimum of $quality(T)$ over all polygons T in $\mathcal{F}$.

Note that for any $k > 0$, the following are equivalent:

- There exists a pair of compatible perfect NCMs M and N on P such that $D_P(M, N) \geq k$.
- There exists a feasible collection $\mathcal{F}$ of polygons on P such that $quality(\mathcal{F}) \geq k$.

This claim follows from the fact that the union of the line segments in any pair of compatible NCMs over P is a collection of even-length simple convex polygons whose vertices partition P and do not cross, i.e, a feasible collection of polygons on P.

Thus, our goal is to find a collection $\mathcal{F}$ of feasible polygons on P for which $quality(\mathcal{F})$ is maximized.

Let A and B be non-empty sets of real numbers. We say that A *dominates* B if for every $x \in A$, we have $x \geq y$ for some $y \in B$. Note that

- If $A \subseteq B$, then A dominates B.
- A dominates B if and only if $\min(A) \geq \min(B)$.

Our dynamic programming algorithm relies on the following structural lemma, which says the following: if $\mathcal{F}$ is a feasible collection of polygons on P with quality s, then we can find a (potentially different) $\mathcal{F}'$ which is a feasible collection of polygons on P whose quality is no worse than s, and further, every polygon in $\mathcal{F}'$ has four or six vertices. This allows us to devise a polynomial time algorithm based on "guessing" the nature of the polygons that the points belong to in some final solution.

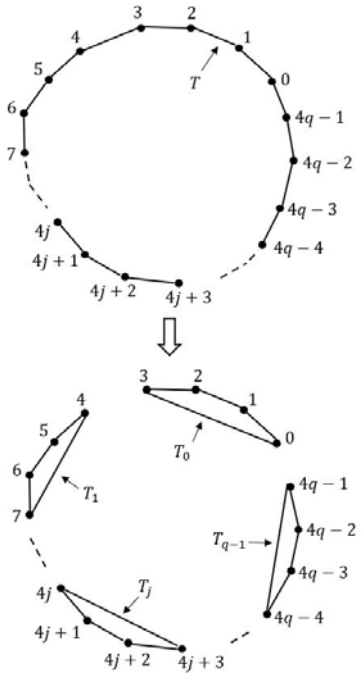


Figure 1: Breaking up a polygon on $4q$ points.

Our proof for the structural lemma considers two scenarios. First, when the number of vertices of a polygon T in $\mathcal{F}$ is a multiple of four, we simply "break" it into four-length polygons. In this situation, we introduce no new pairs into the set of matched partners (i.e, $partners(T') \subseteq partners(T)$ for any T' generated by the breaking procedure) , and so the quality of the solution is not affected. The other situation is that the number of vertices of a polygon T in $\mathcal{F}$ is of the form $4q + 2$. In this case, we find a six-length polygon T' and organize the remaining $4q - 4$ points into $(q - 1)$ polygons as in the previous case. The choice of T' is made carefully so as to ensure that the overall quality of the solution thus obtained is no worse than the original.

Lemma 3 *Let $\mathcal{F}$ be a feasible collection of polygons on P. Then, there exists a feasible collection $\mathcal{F}'$ of polygons on P such that*

- *Every polygon in $\mathcal{F}'$ has length either 4 or 6.*
- $quality(\mathcal{F}') \geq quality(\mathcal{F})$

Proof.
Let $\mathcal{F}'$ be a family of polygons on P obtained from $\mathcal{F}$ as follows: For each polygon T in $\mathcal{F}$ of length > 6,

Case 1: T has length $4q$, for some integer $q \geq 2$

Let $0, 1, 2 \ldots, 4q - 1$ denote the vertices of T, appearing in that order as one traverses in counter-clockwise direction along its boundary. We replace T with q simple convex polygons $T_0, T_1, \ldots, T_{q-1}$, each of length 4, where

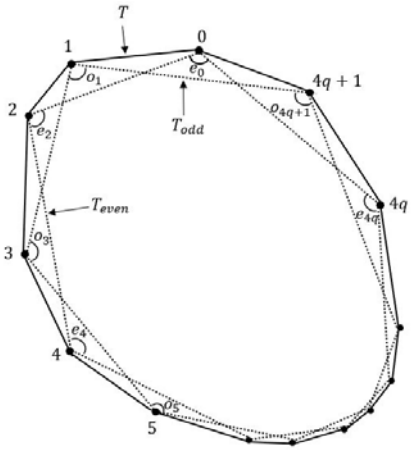


Figure 2: The even and odd polygons T_{even} and T_{odd}.

- T_0 has vertices $0, 1, 2, 3$.
- T_1 has vertices $4, 5, 6, 7$.

  ⋮

  ⋮
- T_{q-1} has vertices $4q-4, 4q-3, 4q-2, 4q-1$.

Let $0 \leq j \leq q-1$. Note that

$$\begin{aligned} partners(T_j) &= \Big\{\{4j, 4j+2\}, \{4j+1, 4j+3\}\Big\} \\ &\subseteq partners(T) \end{aligned}$$

So, we have

$$\min_{\{u,v\} \in partners(T_j)} \Big(\text{dist}(u,v)\Big) \geq \min_{\{u,v\} \in partners(T)} \Big(\text{dist}(u,v)\Big)$$

That is, $quality(T_j) \geq quality(T)$.

Thus,

$$\min_{0 \leq j \leq q-1} \Big(quality(T_j)\Big) \geq quality(T)$$

Case 2: T has length $4q+2$, for some integer $q \geq 2$

Let $0, 1, 2, \ldots, 4q+1$ denote the vertices of T, appearing in that order as one traverses in counter-clockwise direction along its boundary.

Let T_{even} and T_{odd} denote the simple convex polygons, each of length $2q+1$, on the vertices $0, 2, 4, \ldots, 4q$ and $1, 3, 5, \ldots, 4q+1$ respectively. Let $e_0, e_2, e_4, \ldots, e_{4q}$ denote the interior angles of T_{even}, at the vertices $0, 2, 4, \ldots, 4q$ respectively. Let $o_1, o_3, o_5, \ldots, o_{4q+1}$ denote the interior angles of T_{odd}, at the vertices $1, 3, 5, \ldots, 4q+1$ respectively.

For any simple convex polygon, since its exterior angles sum up to 2π, at most two of them are $> \frac{2\pi}{3}$. So, at most two of its interior angles are $< \frac{\pi}{3}$.

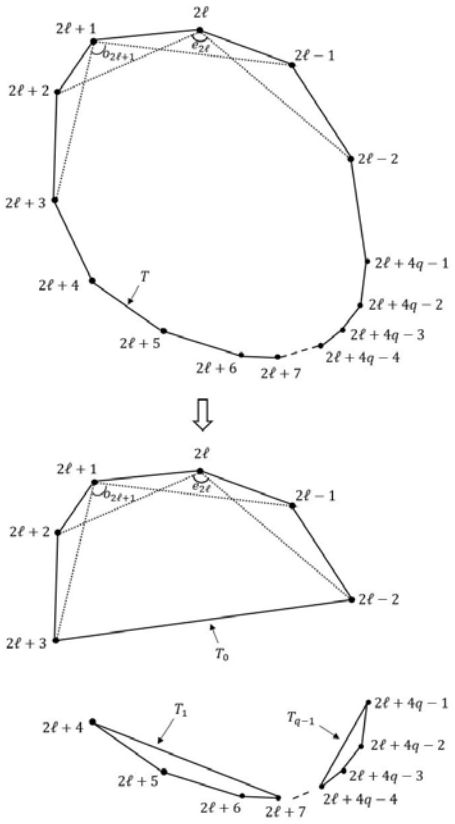


Figure 3: Breaking up a polygon on $4q+2$ points.

Thus, each of T_{even} and T_{odd} has at most two interior angles that are $< \frac{\pi}{3}$. That is, at most two of $e_0, e_2, \ldots, e_{4q}$ are $< \frac{\pi}{3}$, and at most two of $o_1, o_3, \ldots, o_{4q+1}$ are $< \frac{\pi}{3}$. So, among the ≥ 5 pairs of angles $(e_0, o_1), (e_2, o_3), (e_4, o_5), \ldots, (e_{4q}, o_{4q+1})$, there is at least one pair, say $(e_{2\ell}, o_{2\ell+1})$, such that each of $e_{2\ell}$ and $o_{2\ell+1}$ is $\geq \frac{\pi}{3}$.

We replace T with a simple convex polygon T_0 of length 6, and $q-1$ simple convex polygons $T_1, \ldots, T_{q-1}$, each of length 4 (c.f. Figure 3), where:

- T_0 has vertices
  $2\ell-2, 2\ell-1, 2\ell, 2\ell+1, 2\ell+2, 2\ell+3$.
- T_1 has vertices $2\ell+4, 2\ell+5, 2\ell+6, 2\ell+7$.
- T_2 has vertices $2\ell+8, 2\ell+9, 2\ell+10, 2\ell+11$.

  ⋮

  ⋮
- T_{q-1} has vertices
  $2\ell+4q-4, 2\ell+4q-3, 2\ell+4q-2, 2\ell+4q-1$.

Here, the additions are modulo $4q+2$.

Let $1 \leq j \leq q-1$. Note that

$$\begin{aligned} partners(T_j) = \Big\{ &\{2\ell+4j, 2\ell+4j+2\}, \\ &\{2\ell+4j+1, 2\ell+4j+3\} \Big\} \\ \subseteq\ & partners(T) \end{aligned}$$

So, we have

$$\min_{\{u,v\} \in partners(T_j)} \Big(\text{dist}(u,v) \Big) \geq \min_{\{u,v\} \in partners(T)} \Big(\text{dist}(u,v) \Big)$$

That is, $quality(T_j) \geq quality(T)$.

Next, we show that $quality(T_0) \geq quality(T)$.

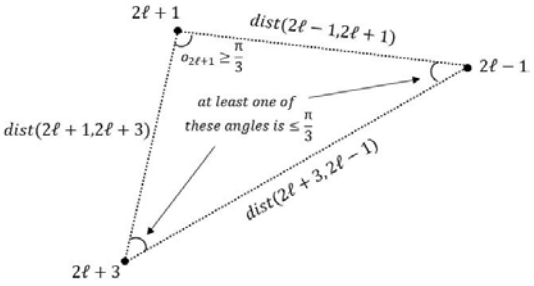
Let

$$A := \Big\{ \text{dist}(u,v) \mid \{u,v\} \in partners(T_0) \Big\}$$

$$B := \Big\{ \text{dist}(u,v) \mid \{u,v\} \in partners(T) \Big\}$$

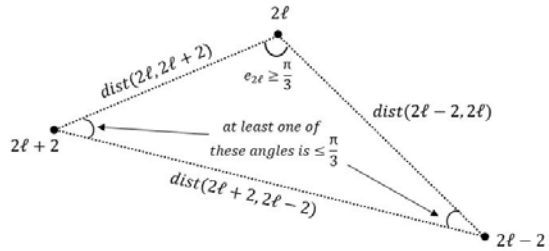
Note that

$$\begin{aligned} partners(T_0) \setminus partners(T) = \Big\{ &\{2\ell+3, 2\ell-1\}, \\ &\{2\ell+2, 2\ell-2\} \Big\} \end{aligned}$$



Consider the triangle formed by the points $2\ell-1$, $2\ell+1, 2\ell+3$. Here, as $o_{2\ell+1} \geq \frac{\pi}{3}$,

$$\text{dist}(2\ell+3, 2\ell-1) \geq \min \begin{pmatrix} \text{dist}(2\ell-1, 2\ell+1), \\ \text{dist}(2\ell+1, 2\ell+3) \end{pmatrix}.$$



Consider the triangle formed by the points $2\ell-2$, $2\ell, 2\ell+2$. Here, as $e_{2\ell} \geq \frac{\pi}{3}$,

$$\text{dist}(2\ell+2, 2\ell-2) \geq \min \begin{pmatrix} \text{dist}(2\ell-2, 2\ell), \\ \text{dist}(2\ell, 2\ell+2) \end{pmatrix}.$$

Also, note that $partner(T)$ contains the pairs $\{2\ell-1, 2\ell+1\}, \{2\ell+1, 2\ell+3\}, \{2\ell-2, 2\ell\}, \{2\ell, 2\ell+2\}$. Therefore, A dominates B and so, $\min(A) \geq \min(B)$.

That is,

$$\min_{\{u,v\} \in partners(T_0)} \Big(\text{dist}(u,v) \Big) \geq \min_{\{u,v\} \in partners(T)} \Big(\text{dist}(u,v) \Big)$$

Thus, $quality(T_0) \geq quality(T)$.

Hence, we have

$$\min_{0 \leq j \leq q-1} \Big(quality(T_j) \Big) \geq quality(T),$$

and this concludes the proof. □

Based on the lemma, we have the following dynamic programming approach: Let $0, 1, \ldots, 2n-1$ denote the points of P in counter-clockwise order. For every $0 \leq i, j \leq 2n-1$ such that $(j-i)$ is odd, let $Q_{i,j}$ denote the set of points $\{i, i+1, \ldots, j\}$ and let:

$$\mathbb{T}(i,j) = \begin{cases} \max D_{Q_{ij}}(M,N) & \text{if } j-i \geq 3, \\ -\infty & \text{if } j-i = 1, \\ +\infty & \text{if } j-i < 0; \end{cases}$$

where the max is taken over all pairs of disjoint compatible perfect NCMs M and N over the point set $Q_{i,j}$.

Note that $\mathbb{T}(0, 2n-1)$ is the value of the optimal solution. We compute and store $\mathbb{T}(i,j)$'s using the following recurrence:

$$\mathbb{T}(i,j) = \begin{cases} \max(\alpha(i,j), \beta(i,j)) & \text{if } j-i \geq 5, \\ \alpha(i,j) & \text{if } j-i = 3, \\ -\infty & \text{if } j-i = 1, \\ +\infty & \text{if } j-i < 0, \end{cases}$$

where $\alpha(i,j)$ is given by:

$$\begin{matrix} \max \\ i < p_1 < p_2 < p_3 \leqslant j : \\ p_1 - i \text{ is odd} \\ p_2 - p_1 \text{ is odd} \\ p_3 - p_2 \text{ is odd} \end{matrix} \quad \min \begin{pmatrix} \text{dist}\,(i, p_2)\,, \\ \text{dist}\,(p_1, p_3)\,, \\ \mathbb{T}\,(i+1, p_1-1)\,, \\ \mathbb{T}\,(p_1+1, p_2-1)\,, \\ \mathbb{T}\,(p_2+1, p_3-1)\,, \\ \mathbb{T}\,(p_3+1, j) \end{pmatrix}$$

and $\beta(i,j)$ is given by:

$$\max_{\substack{i<q_1<q_2<q_3<q_4<q_5\leqslant j:\\ q_1-i \text{ is odd}\\ q_2-q_1 \text{ is odd}\\ q_3-q_2 \text{ is odd}\\ q_4-q_3 \text{ is odd}\\ q_5-q_4 \text{ is odd}}} \min \begin{pmatrix} \operatorname{dist}(i,q_2), \\ \operatorname{dist}(q_1,q_3), \\ \operatorname{dist}(q_2,q_4), \\ \operatorname{dist}(q_3,q_5), \\ \operatorname{dist}(q_4,i), \\ \operatorname{dist}(q_5,q_1), \\ \mathbb{T}(i+1,q_1-1), \\ \mathbb{T}(q_1+1,q_2-1), \\ \mathbb{T}(q_2+1,q_3-1), \\ \mathbb{T}(q_3+1,q_4-1), \\ \mathbb{T}(q_4+1,q_5-1), \\ \mathbb{T}(q_5+1,j) \end{pmatrix}.$$

We remark that the recurrences are well-defined. The overall intuition for the recurrences above is the following: fix an arbitrary solution that has the property guaranteed by Lemma 3. We attempt to "guess" the type and vertices of the polygon that the first point belongs to in this solution. For each fixed guess, we have a natural partition of the remaining points into smaller sub-instances (see Figures 4 and 5). It is easy to identify invalid guesses, by which we mean a polygon which is such that there is no solution that contains it.

For any valid guess, the recurrence gives us the best possible extension, i.e, the best possible diversity achievable among solutions that contain the guessed polygon. All that remains is to pick the best choice among all choices of polygons that contain the first point. The overall running time is polynomially bounded because we only have to worry about polygons with a constant number of vertices. We make this argument more explicit in the Appendix. We also note that the running time of our algorithm is $\mathcal{O}(n^7)$ since the DP table has $\mathcal{O}(n^2)$ indices and the computation at each index is $\mathcal{O}(n^5)$.

We now sketch the correctness of the dynamic programming approach proposed in the context of Theorem 3. Consider the subproblem given by the points $i, i+1, \ldots, j$. Consider the space of all solutions $\mathcal{S}$ that have the property guaranteed by Lemma 3 and partition it into two parts: $\mathcal{S}_4 \subseteq \mathcal{S}$ consists of all solutions where the point i belongs to a polygon with four sides; and $\mathcal{S}_6 \subseteq \mathcal{S}$ consists of all solutions where the point i belongs to a polygon with six sides.

Let $A^\star$ and $B^\star$ denote arbitrary optimal solutions among all the solutions in $\mathcal{S}_4$ and $\mathcal{S}_6$, respectively. Further, let $a^\star$ and $b^\star$ denote the corresponding costs. Note that the cost of the optimal solution for this subproblem is $\max(a^\star, b^\star)$.

We now argue that $\alpha(i,j)$ correctly computes the value of $a^\star$. Once again, for every choice of points $i < p_1 < p_2 < p_3 \leq j$ in $\binom{Q_{i+1,j}}{3}$, let $\mathcal{S}_4[[p_1,p_2,p_3]]$ denote the set

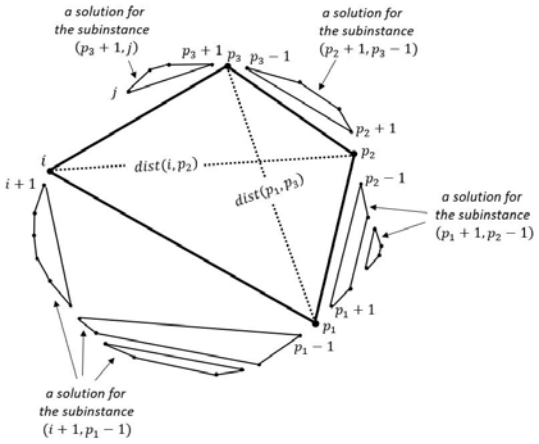


Figure 4: An example of how a base polygon divides the subproblem on $Q_{i,j}$ further into four smaller instances.

of all solutions in $\mathcal{S}_4$ where the polygon containing the point i also contains the points p_1, p_2, p_3. Note that if it is *not* the case that $p_1 - i$ is odd and $p_2 - p_1$ is odd and $p_3 - p_2$ is odd , then $\mathcal{S}_4[[p_1,p_2,p_3]] = \emptyset$, since for any such combination of points, there is no valid solution containing the polygon formed by the points $\{i, p_1, p_2, p_3\}$. For any valid combination, we know that the best solution in $\mathcal{S}_4[[p_1,p_2,p_3]]$ is captured by taking the union of the best solutions for the following subinstances: $(i+1, p_1-1)$, (p_1+1, p_2-1), (p_2+1, p_3-1), and (p_3+1, j) corresponding to the four "chunks" of points "carved out" by the polygon (see Figure 4); along with the polygon formed by the points $\{i, p_1, p_2, p_3\}$. Note that there are no points *inside* the polygon whose vertices are $\{i, p_1, p_2, p_3\}$ since the original point set is in convex position. Further, note that it is reasonable to consider these subinstances independently since no solution that contains the polygon formed by $\{i, p_1, p_2, p_3\}$ will contain a polygon with points from two distinct segments among the segments listed above. The proof can now be completed using a standard strong induction argument, and we defer the details to the full version.

4 Concluding Remarks

We introduced the notion of diverse non-crossing matchings. While we show that DIVERSE COMPATIBLE NCMS can be solved in polynomial time for points in convex position, the complexity of the closely related problem DIVERSE NCMS (where we drop the demand for compatibility from the solution matchings) remains open even for convex point sets. The complexity of all problems considered for more general inputs remains open. We also believe that exploring other notions of diversity, based on either different aggregation func-

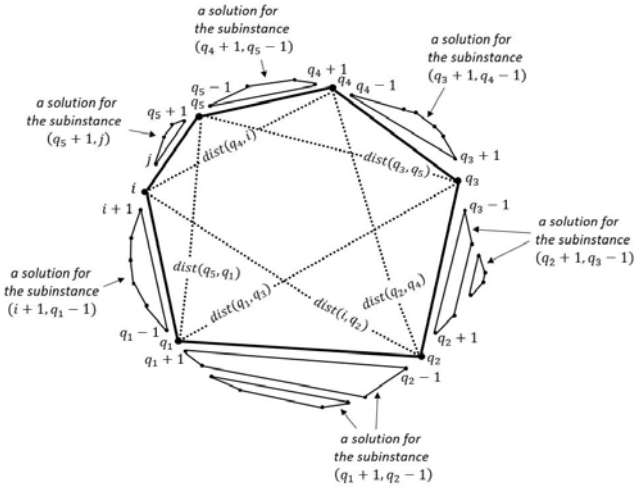


Figure 5: An example of how a base polygon divides the subproblem on $Q_{i,j}$ further into six smaller instances.

tions (e.g, sum instead of minimum), or other notions of distance (different from Euclidean), would also pose interesting directions for future research. We also propose to study the problems proposed here for more than two matchings.

References

Abu-Affash, A. K., Carmi, P., Katz, M. J., and Trabelsi, Y. (2014). Bottleneck non-crossing matching in the plane. *Computational Geometry*, 47(3):447–457. (↑ 2)

Aichholzer, O., Bereg, S., Dumitrescu, A., García, A., Huemer, C., Hurtado, F., Kano, M., Márquez, A., Rappaport, D., Smorodinsky, S., Souvaine, D., Urrutia, J., and Wood, D. R. (2009). Compatible geometric matchings. *Computational Geometry*, 42(6):617–626. (↑ 2)

Aloupis, G., Cardinal, J., Collette, S., Demaine, E. D., Demaine, M. L., Dulieu, M., Fabila-Monroy, R., Hart, V., Hurtado, F., Langerman, S., Saumell, M., Seara, C., and Taslakian, P. (2013). Non-crossing matchings of points with geometric objects. *Computational Geometry*, 46(1):78–92. (↑ 1)

Arrighi, E., Fernau, H., Lokshtanov, D., de Oliveira Oliveira, M., and Wolf, P. (2021). Diversity in Kemeny rank aggregation: A parameterized approach. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI*, pages 10–16. ijcai.org. (↑ 1)

Baste, J., Fellows, M. R., Jaffke, L., Masarík, T., de Oliveira Oliveira, M., Philip, G., and Rosamond, F. A. (2022). Diversity of solutions: An exploration through the lens of fixed-parameter tractability theory. *Journal of Artifical Intelligence*, 303:103644. (↑ 1)

Biniaz, A., Maheshwari, A., and Smid, M. H. M. (2014). Bottleneck bichromatic plane matching of points. In *Proceedings of the 26th Canadian Conference on Computational Geometry, CCCG 2014*. (↑ 2)

Carlsson, J. G., Armbruster, B., Rahul, S., and Bellam, H. (2015). A bottleneck matching problem with edge-crossing constraints. *International Journal of Computational Geometry and Applications*, 25(4):245–262. (↑ 2)

Fomin, F. V., Golovach, P. A., Jaffke, L., Philip, G., and Sagunov, D. (2020). Diverse pairs of matchings. In *Proceedings of the 31st International Symposium on Algorithms and Computation, ISAAC*, pages 26:1–26:12. (↑ 1)

Fomin, F. V., Golovach, P. A., Panolan, F., Philip, G., and Saurabh, S. (2021). Diverse collections in matroids and graphs. In *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science, (STACS) 2021*, pages 31:1–31:14. (↑ 1)

Ganesh, A., HV, V. P., Nimbhorkar, P., and Philip, G. (2021). Disjoint stable matchings in linear time. In Kowalik, L., Pilipczuk, M., and Rzazewski, P., editors, *Proceedings of the 47th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, pages 94–105. Springer. (↑ 1)

Hershberger, J. and Suri, S. (1990). Applications of a semi-dynamic convex hull algorithm. In *Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory (SWAT)*, pages 380–392. Springer. (↑ 1)

Ishaque, M., Souvaine, D. L., and Tóth, C. D. (2012). Disjoint compatible geometric matchings. *Discrete and Computational Geometry*, 49(1):89–131. (↑ 2)

Lo, C., Matousek, J., and Steiger, W. L. (1994). Algorithms for ham-sandwich cuts. *Discrete and Computational Geometry*, 11:433–452. (↑ 1)

Mantas, I., Savic, M., and Schrezenmaier, H. (2021). New variants of perfect non-crossing matchings. In *Proceedings of the 7th International Conference on Algorithms and Discrete Applied Mathematics, CALDAM*, pages 151–164. Springer. (↑ 2)

Nadel, A. (2011). Generating diverse solutions in SAT. In Sakallah, K. A. and Simon, L., editors, *Proceedings of the 14th International Conference on the Theory and Applications of Satisfiability Testing (SAT)*, pages 287–301. (↑ 1)

Savić, M. and Stojaković, M. (2017). Faster bottleneck non-crossing matchings of points in convex position. *Computational Geometry*, 65:27–34. (↑ 2)

Savić, M. and Stojaković, M. (2022). Structural properties of bichromatic non-crossing matchings. *Applied Mathematics and Computation*, 415:126695. (↑ 2, 3)

Maximum Weight Convex Polytope

Mohammad Ali Abam* Ali Mohammad Lavasani† Denis Pankratov‡

Abstract

We study the maximum weight convex polytope problem, in which the goal is to find a convex polytope maximizing the total weight of enclosed points. Prior to this work, the only known result for this problem was an $O(n^3)$ algorithm for the case of 2 dimensions due to Bautista et al. We show that the problem becomes $\mathcal{NP}$-hard to solve exactly in 3 dimensions, and $\mathcal{NP}$-hard to approximate within $n^{1/2-\epsilon}$ for any $\epsilon > 0$ in 4 or more dimensions. We also give a new algorithm for 2 dimensions, albeit with the same $O(n^3)$ running time complexity as that of the algorithm of Bautsita et al.

1 Introduction

Suppose you are given a set of n points S in $\mathbb{R}^d$ with weights $w : S \to \mathbb{R}$; note that weights can be positive or negative. The weight of a polytope P is defined as $w(P) = \sum_{x \in S \cap P} w(x)$. In the *maximum weight convex polytope* problem, or $MWCP$ for short, the goal is to find a convex polytope of maximum weight. This is a rather natural and fundamental computational geometry question.

$MWCP$ with a binary weight function, such as $w : S \to \{+1, -1\}$, belongs to a large class of computational geometry problems on bichromatic point sets with weights $\{+1, -1\}$ corresponding to two colors, typically "red" and "blue". For example, in the maximum box problem one is given a set of r red points and a set of b blue points in the plane and the goal is to find an axis-aligned rectangle which maximizes the number of blue points and does not contain any red points. Liu and Nediak [10] gave an exact $O(r \log r + r + b^2 \log b)$ algorithm, and Eckstein et al. [5] construct an efficient branch-and-bound algorithm motivated by a problem in data analysis. Liu and Nediak [10] also show how to solve efficiently a related bichromatic separability with two boxes problem, introduced by Cortés et al. [2].

$MWCP$ is also related to bichromatic discrepancy problems, where one is given two finite sets of points S^+ and S^- in $\mathbb{R}^d$, and the goal is to find an axis aligned parallelepiped (also called a box) B maximizing the difference between the number of the points of S^+ and S^- inside the box, i.e. $||B \cap S^+| - |B \cap S^-||$. Let $n = |S^+ \cup S^-|$ denote the total number of points. Dobkin et al. [4] solved this problem in $\mathbb{R}^2$ in $O(n^2 \log n)$ time. Liu and Nediak [10] presented a 2-factor approximation for this problem in $\mathbb{R}^2$ with $O(n \log^2 n)$ running time.

In another related problem, namely, numerical discrepancy problem, one is given a set of n points $S \subset [0, 1]^2$. The goal is to find a box B that maximizes the numerical discrepancy of B defined as $||B \cap S|/|S| - \mu(B)|$, where $\mu(B)$ denotes the area of B. Observe that the numerical discrepancy of B can be thought of as measuring the deviation of the empirical distribution from the uniform distribution. Dobkin et al. [4] solved this problem in $\mathbb{R}^2$ in $O(n^2 \log^2 n)$ time. Liu and Nediak [10] presented a 2-factor approximation for this problem in $\mathbb{R}^2$ with $O(n \log^3 n)$ running time.

The above problems introduce constraints on the shape of the solution, namely that the convex polygon must be an axis-aligned parallelepiped. In another variation studied by González-Aguilar et al. [7] the geometric shape of the solution is restricted to be a rectilinear convex hull of points (note that the rectilinear convex hull is not necessarily a convex subset of $\mathbb{R}^2$). González-Aguilar et al. [7] gave an $O(n^3)$ algorithm for this problem.

We note that the above problems are very similar to our problem at first glance. A deeper investigation shows that the nature of restriction on the solution set is crucial for the above problems and algorithms for them, and so new ideas and techniques are needed for $MWCP$ problem. There is one other problem that is directly relevant to $MWCP$, and that is the optimal islands problem studied by Bautista et al. [1]. In this problem, one is given a set S of n points colored with 2 colors in the plane. A subset $\mathcal{I} \in S$ is called an island of S, if $\mathcal{I}$ is an intersection of S and a convex set C. Bautista et al. [1] gave an $O(n^3)$-time algorithm to find a monochromatic island of maximum cardinality. Their algorithm can also be used to solve the $MWCP$ problem in 2 dimensions.

The class of problems to which $MWCP$ belongs have important practical applications in data analysis and machine learning. In particular, Bautista et al. [1] were motivated by clustering applications. Given a training dataset of points $S \subset \mathbb{R}^d$ that are labelled with two colors "red" and "blue", in a classification problem one is

*Department of Computer Engineering, Sharif University of Technology abam@sharif.edu

†Department of Computer Science and Software Engineering, Concordia University, ali.mohammadlavasani@concordia.ca

‡Department of Computer Science and Software Engineering, Concordia University, denis.pankratov@concordia.ca

interested in a simple description of a region of space corresponding to the class of "red" points, for example. One possibility is to use convex hulls for such a description (see, for example, Kudo et al. [9]). If dataset is 2-dimensional one arrives naturally at the optimal islands problem. However, datasets are often noisy, so one should not expect to see large monochromatic islands, so perhaps weighted version of the problem, such as $MWCP$, might be more suitable. A bigger issue is that in classification problems datasets are often high dimensional and one cannot always hope to obtain clusters by projecting to 2 dimensions first. Thus, for clustering applications it is important to be able to solve $MWCP$ efficiently in high dimensions. This is the question we tackle in this paper. Alas, we show that $MWCP$ is $\mathcal{NP}$-hard in 3 dimensions (Theorem 8), and that it is $\mathcal{NP}$-hard to approximate within $n^{1/2-\epsilon}$ for any $\epsilon > 0$ in 4 dimensions even with binary weights (Theorem 11). We also give a completely new algorithm for 2 dimensions with running time $O(n^3)$ matching Bautista et al.

2 Preliminaries

Whenever we write "polytope" in this paper we mean a convex polytope. S denotes the input set of n points in $\mathbb{R}^d$ for $d \geq 1$ and a weight function is denoted by $w : S \to \mathbb{R}$. The weight of a polytope P, denoted by $w(P)$, is defined as follows:

$$w(P) = \sum_{v \in S \cap P} w(x)$$

In $MWCP$ problem, the goal is to find a polytope with maximum weight. Note that points $v \in S$ with 0-weight do not affect weight of any polytope, and so they can be removed from the input in a preprocessing step. Henceforth, we assume that for all $v \in S$ we have $w(v) \neq 0$. We use S^- and S^+ for the subsets of points of S with negative and positive weights respectively. For a set of points $C \subset \mathbb{R}^d$ we let $conv(C)$ denote the convex hull of C. With a slight abuse of notation, we define $w(C) = w(conv(C))$. A subset $C \subseteq S^+$ is *maximal* if for every $v \in C$, $w(C) > w(C \setminus \{v\})$.

Recall that a polytope has two standard equivalent descriptions: $\mathcal{V}$-polytope is described as a convex hull of vertices, and $\mathcal{H}$-polytope is described as an intersection of half-spaces. We shall primarily work with $\mathcal{V}$-polytopes due to the nature of $MWCP$ problem. We let $vert(P)$ denote the set of vertices of a polytope P. Vertices of a polytope are also its 0-faces and edges of a polytope are its 1-faces. We state a few facts about polytopes here that will be used later in the paper; for a more thorough introduction to polytope theory, the reader is referred to the excellent lecture notes of Ziegler [11].

Fact 1 ($\mathcal{V}$-polytope definition) *Let $P \subseteq \mathbb{R}^d$ be a polytope and $v \in \mathbb{R}^d$ be a point. $v \in P$ if and only if there is a convex combination of $vert(P)$ equal to v.*

Fact 2 *Let $P \subseteq \mathbb{R}^d$ be a polytope and F be a face of P. The face F is a polytope, with $vert(F) = F \cap vert(P)$.*

Let $P \subseteq \mathbb{R}^d$ be a polytope and F be a face of P. For a hyperplane h such that $F \subseteq h$ we define h^- and h^+ to be the open half spaces bounded by h such that $h^- \cap P = \emptyset$.

A polytope $P \in \mathbb{R}^d$ is a *polytope embedding* of a graph $G(V, E)$ if there exist a one-to-one function $f : V \to vert(P)$ such that if $(u, v) \in E$ then $(f(v), f(u))$ is an edge of P. Note that P may have some extra edges compared to G. If P has exactly $|E|$ edges, then we call this embedding a *polytope realization* of G.

3 Results

In this section we present our results for the $MWCP$ problem beginning with an overview of upper bounds in Section 3.1 (where we present a new algorithm for 2 dimensions), followed by lower bounds for 3 and 4 dimensions in Section 3.2.

3.1 Upper bounds for 1 and 2 dimensions

We begin with a simple observation: we can assume without loss of generality that vertices of a maximum weight polytope are elements of S^+.

Lemma 3 *For every set S of points in $\mathbb{R}^d$, there exists a maximum weight polytope P with $vert(P) \subseteq S^+$.*

Proof. Let P be a maximum weight polytope and define $C = S^+ \cap P$. The convex hull $conv(C)$ is a subset of P that has all the positive points of P. Thus, $w(C) \geq w(P)$. Since $vert(conv(C)) \subseteq S^+$, we have that $conv(C)$ satisfies the conditions of the lemma. □

The above lemma implies that to solve $MWCP$ it is sufficient to find a set $C \subseteq S^+$ with maximum weight of its convex hall. In particular, when $d = 1$ the $MWCP$ problem reduces to the maximum subarray problem (consider the array of weights of points in S in increasing order of their x-coordinates). The following result is immediate from well known algorithms for the maximum subarray problems.

Theorem 4 *The $MWCP$ problem in 1 dimension ($d = 1$) is solvable in $O(n \log n)$ time. Moreover, if input points are sorted the problem is solvable in $O(n)$ time.*

Bautista et al. [1] gave a dynamic programming algorithm that solves the $MWCP$ problem in 2 dimensions in $O(n^3)$ time. Their algorithm is based on a triangulation of a convex polytope from a topmost *anchor* vertex.

Theorem 5 (Bautista et al. [1]) *The MWCP problem is solvable in $O(n^3)$ time in 2 dimensions ($d = 2$).*

In the rest of this section we present a new algorithm which solves $MWCP$ problem in 2 dimensions, albeit with the same $O(n^3)$ running time. Our algorithm is based on a different decomposition (see Figure 1), and is arguably simpler than the algorithm of Bautista et al.

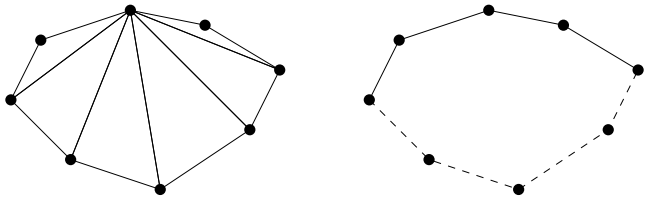


Figure 1: Two decompositions of a polytope which form a basis of two dynamic programming approaches. In the approach of Bautista et al. (shown on the left) a polytope is decomposed via a triangulation from an anchor (topmost) vertex. In our approach (shown on the right) a polytope is decomposed into two paths from a leftmost to a rightmost vertex: top concave path (shown solid) and bottom convex path (shown dashed).

Without loss of generality we can assume that no two points of S have the same x-coordinates. Otherwise in $O(n^2)$ we can find line ℓ such that is not parallel to any line passing through two point in S. Then we can rotate the axes so that the y-axis becomes parallel to ℓ.

Let $p_1, \ldots, p_n$ be the points in S sorted from left to right by their x-coordinates. Consider a directed edge from p_i to p_j for every $i < j$. Weight of the edge $p_i \to p_j$, denoted by $w(p_i, p_j)$, is the sum of all the weights of points p_k such that $i < k < j$ and p_k is below the line segment joining p_i and p_j. We can use brute-force algorithm to compute $w(p_i, p_j)$ for all $i < j$ in $O(n^3)$ time. Thus, we assume that all these weights have been precomputed and are available to us when we need them. A *path* is a sequence of connected edges. For a path $\mathcal{P}$ we define its weight, denoted by $w(\mathcal{P})$, to be the sum of the weights of its edges and its vertices. For a path $\mathcal{P}$ we define its sub-weight, denoted by $w^-(\mathcal{P})$, to be the sum of the weights of its edges only.

A polygon P can be represented as a concave path $\mathcal{C}$ and a convex path $\mathcal{V}$ between its leftmost and its rightmost vertices (see Figure 1). Thus the weight of P is equal $w(\mathcal{C}) - w^-(\mathcal{V})$. We shall present a dynamic programming algorithm to solve the optimization version of the problem, where we are interested in computing the weight of a maximum weight polygon only. The algorithm can be easily modified to find a maximum weight polygon itself by the standard technique of remembering which choices resulted in individual entries of the dynamic programming tables.

For every $i < j \leq k$, let $C[i, j, k]$ (respectively $V[i, j, k]$) be the maximum (respectively, minimum) weight (respectively, sub-weight) of a concave (respectively, convex) path from p_i to p_k such that the first edge is $p_i \to p_j$. We denote the maximum weight of a polygon with leftmost vertex p_i and rightmost vertex p_k by $M[i, k]$. If $i = k$ then $M[i, k] = w(p_k)$, and if $i < k$ then $M[i, k]$ can be computed as:

$$M[i,k] = \max_{j:i<j\leq k} C[i,j,k] - \min_{j:i<j\leq k} V[i,j,k].$$

The solution to the overall problem is then given by the $\max_{i \leq k} M[i, k]$.

In the remainder, we explain how the table $C[i, j, k]$ can be computed. The table $V[i, j, k]$ is computed analogously with some trivial modifications (such as excluding contribution of vertices of the path, replacing concavity with convexity, and replacing maximization objective with minimization objective).

In the algorithm, we have to check whether a line segment joining vertices p and q can be extended to a vertex r with $p.x < q.x < r.x$ while maintaining concavity. This can be tested by checking whether the vector $r - p$ is turned clockwise relative to the vector $q - p$ (see Figure 2). In turn, this can be achieved by checking the sign of 2-dimensional cross-product, denoted by $\times_2$, and defined as $v_1 \times_2 v_2 = v_1.x \cdot v_2.y - v_1.y \cdot v_2.x$. To summarize we have that the path $p \to q \to r$ is concave if and only if[1] $(r - p) \times_2 (q - p) > 0$.

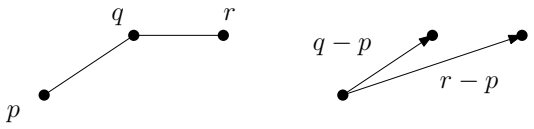


Figure 2: The path $p \to q \to r$ is concave if and only if vector $r - p$ is turned clockwise relative to vector $q - p$.

Base cases for the table $C[i, j, k]$ are the following:

$$C[i,k,k] = w(p_i,p_k) + w(p_i) + w(p_k) \qquad \text{if } i < k$$
$$C[i,j,k] = -\infty \qquad \text{if } i < j < k \text{ and } (p_k - p_i) \times_2 (p_j - p_i) < 0$$

It is clear that the other entries $C[i, j, k]$ with $i < j < k$ can be computed according to the following formula:

$$C[i,j,k] = \max_{j'}\{w(p_i,p_j) + w(p_i) + C[j,j',k] : \quad (1)$$
$$j < j' \leq k \text{ and } (p_{j'} - p_i) \times_2 (p_j - p_i) > 0\}.$$

[1] A bit of care is needed to handle inputs that are not in general position. If three points p, q, r with $p.x < q.x < r.x$ are collinear then $(r - p) \times_2 (q - p) = 0$, and the path p, q, r should be considered concave. However, this makes q not a vertex of the resulting polytope, as it appears in the middle of an edge. In our description, we tacitly assumed that points are in general position to simplify the presentation. It is easy to extend our algorithm to handle points not in general position.

A naive computation of the above table takes $O(n^4)$ time, since the table has $O(n^3)$ entries and each entry can be computed in $O(n)$ time. Next, we show a trick of how the time complexity can be reduced to $O(n^3)$. The idea is for a fixed j and k to fill in entries $C[i,j,k]$ for all i in $O(n)$ time.

We precompute in $O(n^2 \log n)$ total time for all j two lists: $L_j = (l_1, \ldots, l_{j-1})$ and $R_j = (r_1, \ldots, r_{n-j})$. L_j (R_j) consists of points $\{p_1, \ldots, p_{j-1}\}$ (respectively, $\{p_{j+1}, \ldots, p_n\}$) to the left (respectively, to the right) of p_j and sorted in clockwise order with respect to p_j as the origin.

Now, fix a pair of indices $j < k$. In $O(n)$ time it is easy to compute $D[j', k] = \max_{j''}\{C[j, j'', k] : j'' \leq k$ and $p_{j''}$ is either $p_{j'}$ or appears after $p_{j'}$ in $R_j\}$. Define the first compatible j' for the given i, j, denoted by $fc(i,j)$, as the first $p_{j'}$ appearing in R_j such that $p_i \to p_j \to p_{j'}$ is concave. Then it is clear that $C[i,j,k]$ can be equivalently restated as follows:

$$C[i,j,k] = w(p_i, p_j) + w(p_i) + D[fc(i,j), k].$$

This is because, every $p_{j''}$ that appears after $fc(i,j)$ in R_j also forms a concave path $p_i \to p_j \to p_{j''}$. Thus, the third term $D[fc(i,j),k]$ in the above equation is exactly the same as the third term in Equation (1).

Lastly, it is left to observe that as one considers points p_i in the order in which they appear in L_j, the corresponding sequence of $fc(i,j)$ also forms an increasing sequence in R_j. Thus, by maintaining a running pointer into R_j one can compute $fc(i,j)$ in $O(n)$ time for all $p_i \in L_j$. This finishes the description of the algorithm. One readily checks that all precomputing steps take $O(n^3)$, base cases of $C[i,j,k]$ can also be computed in $O(n^3)$ time, and all other entries can be computed in $O(n^3)$ as well, by iterating over all pairs $j < k$ and filling in $C[i,j,k]$ for all i in $O(n)$ time.

3.2 Lower bounds for 3 and 4 dimensions

Recall that a *strict reduction* from an optimization problem $\mathcal{A}$ to an optimization problem $\mathcal{B}$ is a pair of functions (f, g), where f maps instances x of $\mathcal{A}$ to instances $f(x)$ of $\mathcal{B}$ and g maps solutions y of $\mathcal{B}$ to solutions $g(y)$ of $\mathcal{A}$, such that the approximation ratio achieved by solution y on instance $f(x)$ of $\mathcal{B}$ is at least as good as the approximation ratio achieved by solution $g(y)$ on instance x of $\mathcal{A}$. All our lower bound results in this section are based on the following technical lemma.

Lemma 6 *Let $\mathcal{G}$ be a graph family. If for every $G \in \mathcal{G}$ a polytope embedding of G into $\mathbb{R}^d$ can be found in polynomial time and bit complexity polynomial in n, then there is a strict reduction from the maximum independent set on $\mathcal{G}$ to $MWCP$ in d dimensions with weights $\{+1, -1\}$.*

Proof. Given input instance $G = (V, E)$ to the maximum independent set on $\mathcal{G}$, we let P be the result of applying the polytope embedding to G. Let $S^+ := vert(P)$ and assign $+1$ weight to every vertex in S^+. Create set S^- by adding two points with weights of -1 at two arbitrary positions of every graph edge. Let $S = S^+ \cup S^-$. For a negative point $v \in S^-$, let $p_1(v), p_2(v) \in S^+$ be positive-weighted vertices such that v was placed on the edge joining $p_1(v)$ with $p_2(v)$ and $n(v)$ be the other negative point on that edge. See Figure 3 for an example.

We claim that for a subset $C \subseteq S^+$, there exist a negative point $v \in S^-$ in $conv(C)$ if and only if $p_1(v), p_2(v) \in C$. One direction is clear: if $p_1(v), p_2(v) \in C$ then by Fact 1 $n(v)$ and v are in $conv(C)$. For the other direction, assume that $v \in conv(C)$. Let e be the edge between $p_1(v)$ and $p_2(v)$. By the definition of P, there exist a hyperplane h_e such that $S^+ \cap h_e^- = \emptyset$. Therefore $C \cap h_e^- = \emptyset$ and $F := conv(C) \cap h_e$ is a face of $conv(C)$. $F \neq \emptyset$ since v is in h_e and $conv(C)$. Only vertices of S^+ in h_e are $\{p_1(v), p_2(v)\}$. By Fact 2 $vert(F) = F \cap vert(conv(C)) \subseteq h_e \cap S^+ = \{p_1(v), p_2(v)\}$. Without loss of generality suppose $vert(F) = \{p_1(v)\}$, this implies $v \notin F$ which is a contradiction. Thus $vert(F) = \{p_1(v), p_2(v)\}$ and $p_1(v), p_2(v) \in C$.

Let $C \subseteq S^+$ be a maximal subset. We claim that $conv(C)$ contains no negative points and all positive points in $conv(C)$ are precisely the vertices of $conv(C)$. First, suppose there exists a negative point $v \in conv(C)$ thus $p_1(v), p_2(v) \in C$ and $n(v) \in conv(C)$. $w(C \backslash \{p_1(v)\}) \geq w(C) + 2 - 1 > w(C)$ since $v, n(v), p_1(v) \notin conv(C \backslash \{v_i\})$. This is a contradiction to maximality of C. Second, suppose there exist a positive point $v \in S$ in $conv(C) \setminus vert(conv(C))$. Because v is a vertex of P there exist a hyperplane h_v such that $S^+ \cap h_v^- = \emptyset$. Therefore $C \cap h_e^- = \emptyset$ and v is a vertex of $conv(C)$ which is a contradiction.

Therefore, we can conclude that $w(C) = |C|$ if $C \subseteq S^+$ is maximal. Next, we prove there exists a maximal subset $C \subseteq S^+$ if and only if there exist an independent set $\mathcal{I} \subseteq V$ such that $w(C) = |\mathcal{I}|$.

If: Let $\mathcal{I} \subseteq V$ be an independent set and $C \subseteq S^+$ be the set of corresponding vertices of $\mathcal{I}$ in S^+. Because there is no edge between vertices in $\mathcal{I}$, there is no graph edge between vertices in C. Thus there are no negative points in $conv(C)$. Since all vertices inside $conv(C)$ are positive, C is a maximal subset and $w(C) = |C| = |\mathcal{I}|$.

Only if: Let $C \subseteq S^+$ be a maximal subset and let $\mathcal{I} \subseteq V$ be the set of corresponding vertices of C in G. Because C is a maximal subset, there is no negative point in $conv(C)$, and there is no graph edge between vertices of C. Thus the set of corresponding vertices of C in G is an independent set. $|\mathcal{I}| = w(C)$ since $w(C) = |C|$.

Without loss of generality we can suppose every approximation algorithm for $MWCP$ outputs a maxi-

mal subset of S^+. Thus there exist a strict reduction from the maximum independent set problem of graph $G(V, E)$ to $MWCP$ in $\mathbb{R}^d$. □

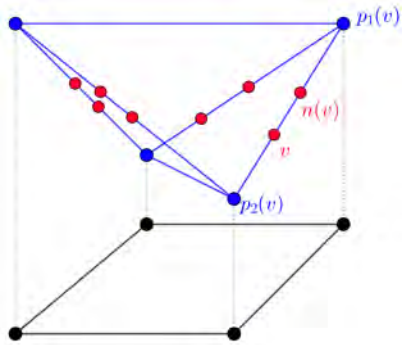


Figure 3: A graph and its embedding in $\mathbb{R}^3$. Black points and edges are the graph and blue points and edges are the embedding of the graph. Red points are added negative points. And an example of v, $n(v)$, $p_1(v)$, and $p_2(v)$ is shown.

We obtain the lower bound for 3 dimensions by applying Lemma 6 to the class $\mathcal{G}$ of planar graphs. We note that the maximum independent set problem is $\mathcal{NP}$-hard even for planar graphs [6]. Our lower bound relies on the polynomial embedding in 3 dimensions due to Das et al. [3]. A *maximal planar graph* is a planar graph such that an addition of any new edge results in a non-planar graph.

Lemma 7 (Das et al. [3]) *Given a maximal planar graph $G(V, E)$ with n vertices, a polytope realization of G in $\mathbb{R}^3$ can be found in $O(n)$ time and with bit complexity polynomial in n.*

Thus the following theorem can be easily deduced from Lemmas 7 and 6.

Theorem 8 *Let S be a set of n points in $\mathbb{R}^3$ with weight function w, finding $MWCP$ of S is $\mathcal{NP}$-hard even if $w : S \to \{-1, +1\}$.*

Proof. Let $\mathcal{G}$ be the family of all planar graphs. By adding edges to a planar graph G we can make it maximal. The polytope realization of the new maximal planar graph is also a polytope embedding of G. Thus with Lemma 7 we can conclude for every $G \in \mathcal{G}$ a polytope embedding of G in $\mathbb{R}^3$ can be found in polynomial time and with polynomial bit complexity. By Lemma 6, there is a strict reduction from maximum independent set on planar graphs to $MWCP$ with weights $\{+1, -1\}$, hence it is an $\mathcal{NP}$-hard problem. □

Let S be the set of points (i, i^2, i^3, i^4) for $1 \leq i \leq n$ in $\mathbb{R}^4$. The convex hull of S is known as the cyclic polytope on n vertices in $\mathbb{R}^4$ and it is a polytope realization of a complete graph with n vertices (for more details, see, for example, [11]).

Lemma 9 *Given a complete graph K_n with n vertices, a polytope realization of it in $\mathbb{R}^4$ can be found in $O(n)$ time with a bit complexity polynomial in n.*

We can use Lemma 9 to show that $MWCP$ in 4 dimensions is as hard as independent set on arbitrary graphs. Zuckerman [12], strengthening an earlier result of Håstad [8], showed that it is $\mathcal{NP}$-hard to approximate independent set on arbitrary graphs within $n^{1-\epsilon}$ factor for any $\epsilon > 0$.

Theorem 10 (Zuckerman [12]) *For any $\epsilon > 0$ it is $\mathcal{NP}$-hard to approximate maximum independent set to within $n^{1-\epsilon}$.*

Combining the above ingredients we establish the inapproximability of $MWCP$ in 4 dimensions and higher.

Theorem 11 *For any $\epsilon > 0$ it is $\mathcal{NP}$-hard to approximate $MWCP$ in 4 dimensions (or higher) with weights $\{+1, -1\}$ to within $n^{1/2-\epsilon}$.*

Proof. Let $\mathcal{G}$ be the family of all finite graphs. By Lemma 9 for every $G \in \mathcal{G}$ a polytope embedding of G in polynomial time and with polynomial bit complexity can be found (recall that the embedding is allowed to have extra edges compared to G). By Lemma 6, there is a strict reduction from maximum independent set on general graphs to $MWCP$ with weights $\{+1, -1\}$. Since Theorem 10 is expressed in terms of input size, it is left to observe that the reduction of Lemma 6 produces instances of $MWCP$ with the number of points that is at most quadratic in the number of vertices of the input graph. □

4 Conclusion and Discussion

In this work, we extended our understanding of the complexity of $MWCP$ as a function of the ambient dimension d. Based on our work and previous work of Bautista et al. [1], the following picture emerges:

1. For $d = 1$, $MWCP$ is solvable in $O(n \log n)$ time exactly (simple observation);
2. For $d = 2$, $MWCP$ is solvable in $O(n^3)$ time (Bautista et al. [1] with another algorithm presented in this work);
3. For $d = 3$, $MWCP$ is not solvable in polynomial time unless $\mathcal{P} = \mathcal{NP}$ (this work);
4. For $d \geq 4$, $MWCP$ is $\mathcal{NP}$-hard to approximate to within $n^{1/2-\epsilon}$ for any $\epsilon > 0$ (this work).

The above list immediately suggests several open problems, the following two of which are of particular interest:

Open Problem 1 *Find an algorithm with better time complexity than $O(n^3)$ for MWCP in 2 dimensions or prove a lower bound probably with some fine-grained hypothesis.*

Open Problem 2 *Determine if MWCP can be approximated within a constant factor in 3 dimensions.*

We conjecture that the answer to the first open problem is that there is no algorithm significantly faster than $O(n^3)$. In light of the second open problem, it is tempting to consider what approximation guarantees are provided by polytopes with constantly many vertices. As the following result demonstrates, constant approximation cannot be guaranteed by such solutions even in $2D$.

Theorem 12 *By restricting solutions to polytopes with constant number of vertices one can not achieve a constant factor approximation for MWCP even in $\mathbb{R}^2$ and even for $\{+1, -1\}$ weights.*

Proof. Let P be a regular n-gon and let the weight of each vertex be $+1$. Put a vertex with weight -1 outside of P on the perpendicular bisector of each edge of P at ϵ-distance away from the edge. Choose ϵ so that line segments joining every two consecutive negative points cross P. This defines the instance of $MWCP$ with P being an optimal solution of weight n.

Let $v_1, v_2, \ldots, v_n$ and $u_1, u_2, \ldots u_n$ be vertices of the clockwise order of S^+ and S^-, respectively, such that u_i has ϵ-distance with the edge between v_i and v_{i+1} ($v_{n+1} := v_1$).

Let C be a convex k-gon, we claim $w(C) \leq k$. Observe that what makes this claim non-trivial is that we cannot assume that $vert(C) \subseteq S^+$ as in Lemma 3, since we have an additional restriction of exactly k vertices.

$\overline{C \backslash P}$ (the closure of $C \backslash P$) is a set of vertices, edges and non-convex polygons. Let C' be one of these non-convex polygons. It suffices to show $w(C') \leq |vert(C) \cap vert(C')|$. Without loss of generality suppose $vert(C') \cap S^+ = \{v_1, v_2, ..., v_r\}$.

Let *outer negative points* be the set $\{u_{i_1}, u_{i_2}, \ldots, u_{i_\ell}\} \subseteq \{u_1, u_2, \ldots u_{r-1}\}$ such that for every $1 \leq j \leq \ell$, $u_{i_j} \notin C'$. For each $1 \leq j \leq \ell$ associate u_{i_j} to the edge e of C' that crosses the shortest line between u_{i_j} and P. By the choice of ϵ two vertices of e are in $vert(C') \cap vert(C)$ and no edge is associated to more than one outer negative point. Thus $|vert(C') \cap vert(C)| \geq \ell + 1$. On the other hand there is at most r positive and at least $r - 1 - l$ negative points in C' thus $w(C') \leq l + 1 \leq |vert(C') \cap vert(C)|$. □

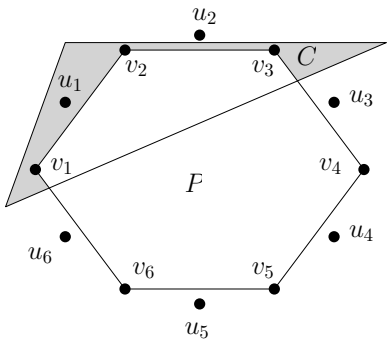


Figure 4: Illustration of the proof of Theorem 12. Here, $n = 6, k = 3$, we chose C to result only in a single C', which is shown as a shaded area. We have $l = 1$ with $u_{i_1} = u_2$ and vertex u_2 is associated with the topmost edge of C'. We have $w(C') = w(v_1) + w(v_2) + w(v_3) + w(u_1) = 3 - 1 = 2 = l + 1$.

References

[1] C. Bautista-Santiago, J. M. Díaz-Báñez, D. Lara, P. Pérez-Lantero, J. Urrutia, and I. Ventura. Computing optimal islands. *Operational Research Letters*, 39(4):246–251, 2011.

[2] C. Cortés, J. M. Díaz-Báñez, P. Pérez-Lantero, C. Seara, J. Urrutia, and I. Ventura. Bichromatic separability with two boxes: A general approach. *Journal of Algorithms*, 64(2-3):79–88, 2009.

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

[4] D. P. Dobkin, D. Gunopulos, and W. Maass. Computing the maximum bichromatic discrepancy, with applications to computer graphics and machine learning. *journal of computer and system sciences*, 52(3):453–470, 1996.

[5] J. Eckstein, P. L. Hammer, Y. Liu, M. Nediak, and B. Simeone. The maximum box problem and its application to data analysis. *Computational Optimization and Applications*, 23(3):285–298, 2002.

[6] M. R. Garey and D. S. Johnson. Computers and intractability. *A Guide to the*, 1979.

[7] H. González-Aguilar, D. Orden, P. Pérez-Lantero, D. Rappaport, C. Seara, J. Tejel, and J. Urrutia. Maximum rectilinear convex subsets. In *International Symposium on Fundamentals of Computation Theory*, pages 274–291. Springer, 2019.

[8] J. Håstad. Clique is hard to approximate within $n^{(1-\epsilon)}$. In *Acta Mathematica*, pages 627–636, 1996.

[9] M. Kudo, Y. Torii, Y. Mori, and M. Shimbo. Approximation of class regions by quasi convex hulls. *Pattern Recognition Letters*, 19(9):777–786, 1998.

[10] Y. Liu and M. Nediak. Planar case of the maximum box and related problems. In *Canadian Conference on Computational Geometry*, volume 3, pages 11–13, 2003.

[11] G. M. Ziegler. *Lectures on polytopes*, volume 152. Springer Science & Business Media, 2012.

[12] D. Zuckerman. Linear degree extractors and the inapproximability of max clique and chromatic number. *Theory of Computing*, 3(6):103–128, 2007.

High-Dimensional Axis-Aligned Bounding Box with Outliers

Ali Mostafavi* Ali Hamzeh†

Abstract

Given n points in d-dimensional space and a parameter z, we study the problem of finding the smallest axis-aligned bounding box that covers at least $n - z$ points and labels the remaining points as outliers. We consider two measures for the size of bounding box: length of the largest side and sum of lengths of the sides. We give two algorithms for the former case: one that uses at most $2z$ outliers but gives a bounding box which is at most as large as the optimal bounding box and another algorithm that uses at most z outliers but gives a box than can be twice as large as the optimal box. We also prove a matching lower bound for the approximation factor of these algorithms.

For the sum of the sides objective, we give a bi-criteria approximation algorithm that finds a box which is at most $O(\log d)$ larger than the optimal box by removing $O(z)$ points.

1 Introduction

We study the classic problem of finding the smallest axis-aligned box that contains a set of points in the setting where we are allowed to ignore z points by labelling them as outliers. The axis-aligned bounding box problem is extensively studied in literature because of its applications in pattern recognition [15], computer graphics [18, 17] and VLSI design [14]. Aggarwal *et al.* [1] gave an exact algorithm to minimize the perimeter for the case where points are in two dimensional plane which runs in $O((n-z)^2 n \log n)$ time. Eppstein and Erickson [8] showed that the time complexity can be improved to $O((n-z)^2 n)$ in the planar case and also gave an $O((n-z)n \log n + (n-z)^{d/2-1} n \log^2 (n-z))$ algorithm for the L_∞ objective (maximum side length) in higher dimensions.

Segal and Kedem [16] gave an $O(n + z(n-z)^2)$ algorithm for minimizing area and perimeter in the plane which is faster than previous ones when z is small. They also extend their algorithm to 3-dimensional space which runs in $O(n + z(n-z)^2 + (n-z)^5)$ time.

Ahn *et al.* [2] studied (p, z) *box covering* problem in the plane: find p disjoint axis-aligned rectangles that cover at least $n - z$ points minimizing the area of the largest box. They gave an $O(n + z^3)$ algorithm for the case where $p = 1$ (which is the same as our problem). They also show that this problem is NP-hard for general p. Atanassov *et al.* [3] studied many geometric problems with outliers and gave an $O(n + z^3)$ algorithm for minimum perimeter rectangle in the plane.

Kaplan *et al.* [10] developed algorithms for the minimum area and minimum perimeter rectangle with outliers which run in $O(n^{2.5} \log^2 n)$ and $O(n(n-z)^{1.5} \log (n-z) \log n)$ respectively.

de Berg *et al.* [7] considered the case where z is large ($n - z$ is small) and gave an $O(n(n-z)^2 \log n + n \log^2 n)$ time algorithm. They also studied the "dual" problem of covering the maximum number of points using a rectangle with area at most α and gave a randomized algorithm with running time $O(\frac{n}{\epsilon^4} \log^3 n \log \frac{1}{\epsilon})$ which covers at least $(1-\epsilon)\kappa^*$ points with high probability where κ^* is the maximum number of points coverable with such rectangle.

Guo and Li [9] presented an algorithm with running time $O(kz^3 + kzn + n^2 \log n)$ for the minimum area rectangle in the plane where k denotes the number of points on the first $z + 1$ convex layers. This algorithm can be faster than previous ones when z and k are small.

Chan and Har-Peled [5] improved the running time for both the area and perimeter to $O(n(n-z) \log \frac{n}{n-z} \log (n-z))$ when points are in the two dimensional plane, they also gave an algorithm which finds a rectangle with area at most $1 + \epsilon$ times the area of the optimal rectangle and runs in $O((1/\epsilon)^3 \log (1/\epsilon) n \log n)$.

Bae [4] studied the related *minimum width cuboidal shell* problem with outliers where cuboidal shell is defined as the area between a cube and its inward offset. They give an algorithm with running time $O(z^{2d} n)$.

In this paper, we present the first polynomial-time bi-criteria approximation algorithms for axis-aligned bounding box with outliers in high dimensional spaces. An (α, β)-approximation algorithm for axis aligned bounding box is one that achieves an objective value at most α times the optimal value by removing at most βz outliers. We consider two objectives: the maximum side length of the bounding box (L_∞) and the sum of the side lengths of the bounding box (L_1) which reduces to perimeter in 2-dimensional case.

- For the L_∞ objective, we give $(1, 2)$ and $(2, 1)$-approximation algorithms. Moreover, we prove ap-

*Department of Computer Engineering, Shiraz University, a.hr.mostafavi@gmail.com

†Department of Computer Engineering, Shiraz University, ali@shirazu.ac.ir

proximating the objective function with a factor better than 2 is NP-hard if we are not allowed to use more than z outliers.

- For the L_1 objective, we give a $(O(\log d), O(1))$-approximation algorithm.

2 Definitions and Terminology

Let $P \subset \mathbb{R}^d$ be a set of points in d-dimensional euclidean space and let $n = |P|$ be the number of points. Let $l_j(P)$ denote the extent of P in the j-th dimension, that is:

$$l_j(P) = \max_{p,q \in P} p_j - q_j$$

In this paper, we explore the following extent measures of the point set:

- Maximum side length of the axis-aligned bounding box:
$$L_\infty(P) = \max_{j=1}^{d} l_j(P)$$
- The sum of side lengths of the axis-aligned bounding box:
$$L_1(P) = \sum_{j=1}^{d} l_j(P)$$

Let $f(x)$ be any function on subsets of P, the problem of minimizing f with z outliers is to find z points in P such that removal of these points minimizes f, that is:

$$Z^* = \operatorname*{arg\,min}_{Z \subset P, |Z|=z} f(P \setminus Z)$$

we denote the optimal value of this function by $\text{OPT} = f(P \setminus Z^*)$ and the optimal set of points by $P^* = P \setminus Z^*$. For convenience we denote the optimal outlier points for the L_∞ objective by $Z^*_\infty(P,z) = \arg\min_{Z \subset P, |Z|=z} L_\infty(P \setminus Z)$ and the optimal non-outlier points are $P^*_\infty(P,z) = P \setminus Z^*_\infty(P,z)$ and the optimal objective value is $L^*_\infty(P,z) = L_\infty(P^*_\infty)$. $Z^*_1(P,z)$, $P^*_1(P,z)$ and $L^*_1(P,z)$ are defined analogously. For brevity, we will omit (P,z) when their value is obvious from the context (for example instead of $L^*_\infty(P,z)$ we just write L^*_∞).

3 Approximating L_∞ in High Dimensions

In this section we give the following results for the minimum-L_∞ axis-aligned bounding box with outliers:

1. In subsection 3.1 we give an approximation algorithm that labels $2z$ points as outliers but guarantees that the L_∞ value of the remaining points is less than $L^*_\infty(P,z)$
2. In subsection 3.2 we give an approximation algorithm that labels at most z points as outliers and guarantees that the L_∞ value of the remaining points is at most $2L^*_\infty(P,z)$
3. In subsection 3.3 we prove that under some reasonable assumptions both above approximation factors are optimal.

3.1 A $(1,2)$-approximation Algorithm

We repeatedly find the dimension with maximum side length and remove two extreme points along this dimension. We claim that Algorithm 1 achieves the optimal value of L_∞ and removes at most $2z$ points.

Algorithm 1 Approximation Algorithm for L_∞ in High Dimensions

```
1: procedure L∞-APPROXIMATION1(P)
2:     for i ← 1...z do
3:         d_i = arg max_{j=1}^{d} l_j(P)
4:         p_i^min, p_i^max = extreme points of P in d_i
5:         P ← P \ {p_i^min, p_i^max}
6:     return P
```

Lemma 1 *Let $Z_i = \{p_i^{\min}, p_i^{\max}\}$ be the two points removed in the i-th iteration of the for loop in Line 2, then one of the following holds:*

- $Z_i \cap Z^*_\infty \neq \emptyset$
- $L_\infty(P) \leq L^*_\infty$

Proof. Let P^*_∞ be the optimal set of points with their z outliers removed. Suppose $Z_i \cap Z^*_\infty = \emptyset$, therefore $Z_i \subseteq P^*_\infty$ and since L_∞ is a monotone function (that is, it can never increase by deleting points), we have:

$$L_\infty(P^*_\infty) \geq L_\infty(Z_i) = L_\infty(P)$$

□

Theorem 2 *The points returned by Algorithm 1 achieve at most the optimal value of L_∞.*

Proof. If at any point during Algorithm 1 we have $Z_i \cap Z^*_\infty = \emptyset$, then by Lemma 1 we have already achieved the optimum value (and we will never increase this value because L_∞ is a monotone function of points). Otherwise we have $\forall i : Z_i \cap Z^*_\infty \neq \emptyset$ and $\forall i, j : Z_i \cap Z_j = \emptyset$. Therefore for each i we have removed at least one new point from Z^*_∞, so after z iterations we have removed all z points from Z^*_∞.

□

The loop at line 2 is executed z times and each execution takes $O(nd)$ time so the overall runtime of the algorithm is $O(ndz)$. This runtime can be improved when dz is $o(n)$ by exploiting the following observation:

Observation 1 *The extreme point of P^*_∞ with the lowest (highest) coordinate in j-th dimension is among the $z+1$ extreme points of P with lowest (highest) j-th coordinate.*

Proof. This follows trivially from the fact that we have at most z outliers, therefore one of the $z+1$ most extreme points must be in the optimal solution. □

Therefore, instead of considering all n points, we can consider only the points which are among the $z+1$ most extreme points in some dimension. There are at most $2d(z+1)$ such points and they can be found in $O(nd)$ time using standard selection algorithms [6]. Running the algorithm only on these points reduces the overall runtime to $O(nd + d^2z^2)$.

3.2 A $(2,1)$-approximation Algorithm

Observation 1 implies that if a point is not among the $2dz$ most extreme points, it can not be an outlier and therefore it must be included in the optimal box. Additionally, any cube with side length l can be covered with a cube with side length $2l$ centered at any point inside the cube. Therefore if all except z points can be covered with a cube of maximum side length l, then all but z points can be covered with a cube centered at any non-outlier point with side length $2l$. This suggests Algorithm 2 for the case when $n > 2dz$. We know that the point c found in line 3 can not be an outlier, and therefore, we must be able to cover $n-z$ points with a cube centered on c and length at most $2L^*_\infty$.

Algorithm 2 $(2,1)$-approximation Algorithm for L_∞ in High Dimensions

1: **procedure** L_∞-Approximation2(P)
2: Let P' be the points of P with $2dz$ most extreme points in each dimension deleted
3: Let c be an arbitrary point in P'
4: Find the smallest l such that a cube with side length l centered at c can cover at least $n-z$ points in P
5: Label all the points not covered in the cube found in line 4 as outliers and return the cube as the solution

Line 2 can be performed in $O(nd)$ time using selection algorithms. Line 4 can be performed in $O(nd \log n)$ time by performing a binary search on the distance of c to the points in P. So the overall runtime of Algorithm 2 is $O(nd \log n)$.

3.3 Hardness of Approximation

We show that unless $P = NP$, we can not approximate the value of $L^*_\infty(P,z)$ with a factor better than 2 when no approximation on z is allowed. This proves that Algorithm 2 is optimal and justifies our approximation on z in Algorithm 1.

We convert an instance of Vertex-Cover [11] problem to an instance of minimum L_∞ axis-aligned bounding box with outliers such that the optimal value is 1 if the graph has a vertex cover of size k and is 2 otherwise (which means an approximation algorithm with a factor better than 2 can distinguish between these cases). Let (V,E,k) be an instance of Vertex-Cover problem where V is the set of nodes and $E \subseteq V \times V$ is the set of edges. Our goal is to determine if the graph has a vertex cover of size at most k. We arbitrarily assign directions to edges of $|E|$ and convert each vertex into a point in $\mathbb{R}^{|E|}$. Note that there is a dimension corresponding to each edge in the graph, let d_e denote the dimension corresponding to edge e. Let p_v denote the point corresponding to vertex $v \in V$. We use the following rule to determine p_v:

$$p_v(d_e) = \begin{cases} -1, & \text{if } e = (v,*). \\ 1, & \text{if } e = (*,v). \\ 0, & \text{otherwise.} \end{cases}$$

Theorem 3 *Let $P = \{p_v | v \in V\}$. Then:*

$$L^*_\infty(P,k) = \begin{cases} 1, & \textit{if } (V,E) \textit{ has a vertex cover of size } k. \\ 2, & \textit{otherwise.} \end{cases}$$

Proof. Suppose (V,E) has a vertex cover $C \subset V$ where $|C| = k$. We label the points corresponding to vertices in C as outliers, let $P' = P \setminus \{p_v | v \in C\}$ be the remaining points. There are exactly two points in P with non-zero value for each edge and since we know we have removed at least one of the endpoints of each edge in P', there is at most one point in P' with non-zero value in each dimension. Therefore the points of P' can be covered with a box with maximum side length of 1.

Conversely, we can convert any solution where $L^*_\infty(P,z) = 1$ to a vertex cover by selecting the vertices corresponding to outliers in P which means that if (V,E) doesn't have a vertex cover with size k then $L^*_\infty(P,k) > 1$, but the only other possible value for $L^*_\infty(P,k)$ is 2. □

Assuming that unique games conjecture [12] is true, it is impossible to approximate Vertex-Cover with a factor better than two [13] and we can strengthen Theorem 3 to claim that no (α,β)-approximation is possible where α and β are simultaneously less than 2. This proves that both Algorithms 1 and 2 are pareto-optimal.

4 Approximating L_1 in High Dimensions

In this section we develop bi-criteria approximation algorithms for the L_1 objective. First, we warm up by developing $(d, 2)$ and $(1, 2d)$-approximation algorithms in subsections 4.1 and 4.2 respectively. Then in subsection 4.3 we show how to combine the ideas of these algorithms to develop an algorithm that achieves a reasonable approximation factor both for the objective value and number of outliers.

4.1 A $(d, 2)$-approximation Algorithm

We claim that Algorithm 1 already gives us a $(d, 2)$ approximation factor.

Theorem 4 *Algorithm 1 is a $(d, 2)$-approximation algorithm for L_1.*

Proof. Let P^*_∞ and P^*_1 be the optimal set of points (with z outliers removed) for L_∞ and L_1 respectively and P_W be the result of applying Algorithm 1 on P. We have:

$$\begin{aligned} L_1(P_W) &= \sum_{j=1}^{d} l_j(P_W) \\ &\leq d \max_{j=1}^{d} l_j(P_W) \\ &\leq d \max_{j=1}^{d} l_j(P^*_\infty) \\ &\leq d \max_{j=1}^{d} l_j(P^*_1) \\ &\leq d \sum_{j=1}^{d} l_j(P^*_1) \\ &= dL_1(P^*_1) \end{aligned}$$

□

4.2 A $(1, 2d)$-approximation Algorithm

This algorithm is very similar to Algorithm 1, however, instead of deleting the extreme points just in the largest dimension, we delete all $2d$ extreme points in all dimensions.

Algorithm 3 Simple Approximation Algorithm for L_1 in High Dimensions

```
1: procedure L1-Approximation(P)
2:     for i ← 1...z do
3:         Z_i = at most 2d extreme points of P in each dimension
4:         P ← P \ Z_i
5:     return P
```

We omit the full proof of correctness and approximation factor because it is essentially the same as the proof of Theorem 2. The main idea is that at each step, either we have already achieved optimality or at least one of the deleted points is in Z^*_1.

4.3 A Better Approximation Algorithm

In this section, we assume that the optimal value $\text{OPT} = L^*_1(P, z)$ is known. We will show how to remove this assumption in subsection 4.3.1. The main idea is the same as Algorithm 3, however, instead of removing the two extreme points of all dimensions, we only consider the "large" dimensions and ignore "small" dimensions by proving that their contribution to the solution can not be too large (Theorem 9). And since we know the value of OPT, we know that there can not be too many "large" dimensions in the optimal solution, therefore we know that a good portion of the deleted points are actual outliers (Theorem 8).

Procedure Prune is called $\log_{\frac{2}{1+\epsilon}} d$ times. Each run of Prune takes $O(n^2 d)$ time so the overall runtime of Algorithm 4 is $O(n^2 d \log_{\frac{2}{1+\epsilon}} d)$.

Algorithm 4 Approximation Algorithm for L_1 in High Dimensions

```
1: procedure Prune(P, D)
2:     n ← |D|
3:     while |D|> (1+ε)n/2 do
4:         for d in D do
5:             if l_d(P) > 2OPT/n then
6:                 Remove the two extreme points of P in dimension d
7:             else
8:                 D ← D \ {d}
9:     return P, D
10: procedure L1-Approximation(P)
11:     D = {1, 2, ..., d}
12:     for i ← 1... ⌈log_{2/(1+ε)} d⌉ do
13:         P, D ← Prune(P, D)
14:     return P
```

Lemma 5 *Let $D, |D|> 0$ be a set of dimensions and let D' be the set of remaining dimensions after running* Prune(P, D) *then* $\frac{|D'|}{|D|} \leq \frac{1+\epsilon}{2}$.

Proof. Line 3 removes dimensions until this condition is satisfied (we know that it can be eventually satisfied because we can reduce the value to 0 by removing all points). □

Corollary 6 *Let D_i be D after the i-th iteration of the loop at line 12. Then $|D_i| \leq (\frac{1+\epsilon}{2})^i d$.*

Proof. $|D_0|$ is d and for each i we have $|D_{i+1}| \leq \frac{1+\epsilon}{2}|D_i|$. By multiplying all these inequalities we get:

$$|D_i| \leq \left(\frac{1+\epsilon}{2}\right)^i |D_0| = \left(\frac{1+\epsilon}{2}\right)^i d$$

□

Lemma 7 *Let $P_1^* = P \setminus Z_1^*$ be the optimal set of points and $C = \{j : l_j(P_1^*) > \frac{2\text{OPT}}{n}\}$. Then $|C| \leq \frac{n}{2}$.*

Proof.

$$\begin{aligned}\text{OPT} &= \sum_{j=1}^{d} l_j(P_1^*) \geq \sum_{j \in C} l_j(P_1^*) \\ &\geq \sum_{j \in C} \frac{2\text{OPT}}{n} = |C|\frac{2\text{OPT}}{n} \implies \\ \text{OPT} \geq |C|\frac{2\text{OPT}}{n} &\implies \frac{n}{2} \geq |C|\end{aligned}$$

□

Theorem 8 *Algorithm 4 removes at most $4(\epsilon^{-1}z + 2d)$ points.*

Proof. Let Z_i be the points removed the i-th time PRUNE was called, let D_i be the input dimensions to this iteration and let α_i be the number of times while loop in Line 3 was executed. We have:

$$|Z_i| \leq 2\alpha_i |D_i| = 2(\alpha_i - 1)|D_i| + 2|D_i|$$

which means:

$$\frac{|Z_i| - 2|D_i|}{2|D_i|} \leq \alpha_i - 1$$

Now, for the first $\alpha_i - 1$ iterations, we are sure that there are at least $\frac{(1+\epsilon)|D_i|}{2}$ dimensions remaining in D which we call active dimensions (this might not be true for the last iteration because we remove points inside the loop). On the other hand, Lemma 7 tells us that there are at most $\frac{|D_i|}{2}$ dimensions in the optimal solution whose extent is more than $\frac{2\text{OPT}}{|D_i|}$. This means that at least $\frac{(1+\epsilon)|D_i|}{2} - \frac{|D_i|}{2} = \frac{\epsilon|D_i|}{2}$ of these dimensions are not optimal (the extent of points in these dimensions is still larger than the extent of optimal points in this dimension) which means at each non-final iteration we remove at least $\frac{\epsilon|D_i|}{2}$ points from Z_1^*. Thus we have:

$$|Z_i \cap Z_1^*| \geq (\alpha_i - 1)\frac{\epsilon|D_i|}{2} \geq \frac{|Z_i| - 2|D_i|}{2|D_i|}\frac{\epsilon|D_i|}{2} = \frac{\epsilon(|Z_i| - 2|D_i|)}{4}$$

therefore $4\epsilon^{-1}|Z_i \cap Z_1^*| + 2|D_i| \geq |Z_i|$. Now we can bound the number of points removed in all iterations (we use I to denote the number of iterations which is $\left\lceil \log_{\frac{2}{1+\epsilon}} d \right\rceil$):

$$\begin{aligned}\sum_{i=1}^{I} |Z_i| &\leq \sum_{i=1}^{I} 4\epsilon^{-1}|Z_i \cap Z_1^*| + 2|D_i| \\ &= \sum_{i=1}^{I} 4\epsilon^{-1}|Z_i \cap Z_1^*| + 2\sum_{i=1}^{I} |D_i|\end{aligned}$$

we have

$$\sum_{i=1}^{I} |Z_i \cap Z_1^*| \leq |Z_1^*|$$

because the Z_is are disjoint so the first sum is less than $4\epsilon^{-1}z$. For the second sum we apply Corollary 6 and sum the geometric series to get:

$$\sum_{i=1}^{I} |D_i| \leq \sum_{i=1}^{I} \left(\frac{1+\epsilon}{2}\right)^i d < \frac{2d}{1-\epsilon} < 4d$$

we assumed $\epsilon < \frac{1}{2}$ for the last inequality. Putting this all together we have:

$$\sum_{i=1}^{I} |Z_i| \leq 4(\epsilon^{-1}z + 2d)$$

□

Theorem 9 *The points returned by Algorithm 4 achieve an L_1 value of at most* $\text{OPT}(2\left\lceil \log_{\frac{2}{1+\epsilon}} d \right\rceil)$.

Proof. Each time PRUNE is called the deactivated dimensions have extent at most $\frac{2\text{OPT}}{|D_i|}$ and there are at most $|D_i|$ of them. So the total contribution of these dimensions to the objective is at most $\frac{2\text{OPT}}{|D_i|}|D_i| = 2\text{OPT}$. And PRUNE is called at most $\left\lceil \log_{\frac{2}{1+\epsilon}} d \right\rceil$ times after which there remains no active dimensions. □

4.3.1 How to Find OPT ?

In Section 4.3 we assumed we knew the value of OPT. Here we will show how to remove this assumption. Let L and U be a lower bound and an upper bound on the value of OPT. For example U can be $L_1(P)$ and L can be $\frac{1}{d}$ times the result of running Algorithm 1 on P. Let $\phi = \frac{U}{L}$. Run Algorithm 4 in parallel $\log_{1+\delta} \phi$ times each time with a guess of $L(1+\delta)^i$ for the value of OPT. For one of these guesses we have $\text{OPT} \leq L(1+\delta)^i \leq (1+\delta)\text{OPT}$ and for that guess Algorithm 4 is guaranteed to succeed with a value of at most $2(1+\delta)\left\lceil \log_{\frac{2}{1+\epsilon}} d \right\rceil \text{OPT}$.

While this method probably works well in most practical datasets, one can design adversarial datasets where the value of ϕ is unbounded. Here we will describe a method to find bounds whose ratio is guaranteed to be a polynomial. Let P_W be the result of applying Algorithm 1 on P and $W = L_\infty(P_W)$. Let $L_\infty^*(P, z)$ and

$L_1^*(P, z)$ be the optimal value of L_∞ and L_1 objectives with z outliers on point set P. We have:

$$W \leq L_\infty^*(P, z) \leq L_1^*(P, z) \tag{1}$$

On the other hand, applying Algorithm 1 removes at most $2z$ points, so $L_1(P_W)$ is an upper bound on the value of $L_1^*(P, 2z)$.

$$\begin{aligned} L_1^*(P, 2z) \leq L_1(P_W) &= \sum_{j=1}^{d} l_j(P_W) \\ &\leq d \max_{j=1}^{d} l_j(P_W) = dW \end{aligned} \tag{2}$$

We set $L = W, U = dW$ and we use $z' = 2z$ as the number of outliers in Algorithm 4.

$$\phi = \frac{U}{L} = \frac{dW}{W} = d \implies \log_{1+\delta} \phi = O(\delta^{-1} \log d)$$

Equation 2 ensures that our algorithm succeeds for some estimate of OPT and Equation 1 ensures that at least some of our estimates are less than OPT. So either OPT is greater than dW or there is some estimate between OPT and $(1 + \delta)$OPT. Either way, our algorithm succeeds for these estimates with approximation factor given in Theorem 9. This algorithm blows up our approximation factor for the number of outliers at most by a factor of two (from $4(\epsilon^{-1}z + 2d)$ to $8(\epsilon^{-1}z + d)$).

5 Conclusion

We presented simple bi-criteria approximation algorithms for minimum axis-aligned bounding box in high-dimensional euclidean spaces for the maximum side length and sum of side lengths objectives. While this problem has been previously studied in low-dimensional settings, as far as we are aware this is the first work that studies this problem in high-dimensional euclidean spaces. We proved that our algorithms for the maximum side length objective are pareto-optimal under some reasonable assumptions.

We plan on (i) improving the approximation factors for the sum of side length algorithm, (ii) prove matching lower bounds for sum of side length objective, and (iii) studying other high-dimensional problems in the presence of outliers.

References

[1] Alok Aggarwal, Hiroshi Imai, Naoki Katoh, and Subhash Suri. Finding k points with minimum diameter and related problems. *Journal of Algorithms*, 12(1):38–56, 1991.

[2] Hee-Kap Ahn, Sang Won Bae, Erik D Demaine, Martin L Demaine, Sang-Sub Kim, Matias Korman, Iris Reinbacher, and Wanbin Son. Covering points by disjoint boxes with outliers. *Computational Geometry*, 44(3):178–190, 2011.

[3] Rossen Atanassov, Prosenjit Bose, Mathieu Couture, Anil Maheshwari, Pat Morin, Michel Paquette, Michiel Smid, and Stefanie Wuhrer. Algorithms for optimal outlier removal. *Journal of Discrete Algorithms*, 7(2):239–248, 2009.

[4] Sang Won Bae. Minimum-width cuboidal shells with outliers. *Journal of Computing Science and Engineering*, 14(1):1–8, 2020.

[5] Timothy M. Chan and Sariel Har-Peled. Smallest k-enclosing rectangle revisited. *Discrete and Computational Geometry*, 66(2):769–791, 2021.

[6] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2009.

[7] Mark De Berg, Sergio Cabello, Otfried Cheong, David Eppstein, and Christian Knauer. Covering many points with a small-area box. *Journal of Computational Geometry*, 10(1):207–222, 2019.

[8] David Eppstein and Jeff Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete & Computational Geometry*, 11(3):321–350, 1994.

[9] Zhengyang Guo and Yi Li. Minimum enclosing parallelogram with outliers. *arXiv preprint arXiv:2003.01900*, 2020.

[10] Haim Kaplan, Sasanka Roy, and Micha Sharir. Finding axis-parallel rectangles of fixed perimeter or area containing the largest number of points. *Computational Geometry: Theory and Applications*, 81(52):1–11, 2019.

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

[12] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 767–775, 2002.

[13] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate to within 2- ε. *Journal of Computer and System Sciences*, 74(3):335–349, 2008.

[14] Maharaj Mukherjee and Kanad Chakraborty. A polynomial-time optimization algorithm for a rectilinear partitioning problem with applications in VLSI design automation. *Information Processing Letters*, 83(1):41–48, 2002.

[15] Sung Hee Park and Jae-young Kim. Unsupervised Clustering with Axis-Aligned Rectangular Regions. `http://cs229.stanford.edu/proj2009/ParkKim.pdf`.

[16] Michael Segal and Klara Kedem. Enclosing k points in the smallest axis parallel rectangle. *Information Processing Letters*, 65(2):95–99, 1998.

[17] Daiki Takeshita. AABB pruning: Pruning of neighborhood search for uniform grid using axis-aligned bounding box. *The Journal of the Society for Art and Science*, 19(1):1–8, 2020.

[18] Bruce Walter, Kavita Bala, Milind Kulkarni, and Keshav Pingali. Fast agglomerative clustering for rendering. *RT'08 - IEEE/EG Symposium on Interactive Ray Tracing 2008, Proceedings*, pages 81–86, 2008.

Quasi-Twisting Convex Polyhedra

Thomas C. Hull* Anna Lubiw† Klara Mundilova‡ Chie Nara§
Joseph O'Rourke¶ Josef Tkadlec‖ Ryuhei Uehara**

Abstract

We introduce a notion we call ***quasi-twisting*** that cuts a convex polyhedron P into two halves and re-glues the halves to form a different convex polyhedron. The cut is along a simple closed quasigeodesic. We initiate the study of the range of polyhedra produced by quasi-twisting P, and in particular, whether P can "quasi-twist flat," i.e., produce a flat, doubly-covered polygon. We establish a sufficient condition and some necessary conditions, which allow us to show that of the five Platonic solids, the tetrahedron, cube, and octahedron can quasi-twist flat. We conjecture that the dodecahedron and icosahedron cannot quasi-twist flat, and prove that they cannot under certain restrictions. Many open problems remain.

1 Introduction

A geodesic γ on a convex polyhedron P is a path that has exactly π surface angle to either side at every point of γ. So geodesics cannot pass through vertices. A ***quasigeodesic*** has at most π angle to each side at every point, and so can pass through vertices. Whereas most convex polyhedra have no simple closed geodesic [10], every convex polyhedron has at least three simple closed quasigeodesics, a result of Pogorelov from 1949 [16].

In this paper we introduce an operation we call ***quasi-twisting***, which applies to any convex polyhedron P and any simple closed quasigeodesic (or geodesic) Q on P. We imagine cutting along Q to separate P into two "halves" A and B, above and below Q, each with boundary $\partial A, \partial B$. Let L be the length of Q. Keeping B fixed, "glue" ∂A to ∂B, but shifted by some fraction of L. (A and B are considered flexible but isometric surfaces during this gluing.) So A is "twisted" with respect to B. By Alexandrov's Gluing Theorem, the result is a convex polyhedron $\tilde{P}$: the lengths of the boundaries ∂A, ∂B are equal, so the gluing results in a closed shape homeomorphic to a sphere. And because of the $\leq \pi$ quasigeodesic condition, both ∂A and ∂B are convex and so the re-gluing maintains $\leq 2\pi$ at every point along the seam.

Example. Before proceeding further, we illustrate with an example. P is a unit cube, and quasigeodesic Q is the path through six vertices illustrated in Fig. 1(a). (We identify a vertex either by its index i, or as v_i, whichever is more convenient.) The angles to one side of Q alternate between $\pi/2$ and π.

Cutting Q leaves A and B composed of three faces each, with vertex v_7 interior to A and antipodal vertex v_1 interior to B. The boundaries ∂A and ∂B each include a copy of the six vertices of Q. Now we twist A one unit counterclockwise from above, matching vertices of A to B as follows.

$$\begin{array}{cccccc} 2 & 6 & 5 & 8 & 4 & 3 \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ 3 & 2 & 6 & 5 & 8 & 4 \end{array}$$

Three of the six vertices along Q cease to be vertices in $\tilde{P}$. For example, $v_6 \to v_2$ joins π to π surface angle. The result is a triangular bipyramid with base an equilateral triangle of side length 2, and altitudes to v_1 and v_7 of length $\sqrt{2/3}$.

1.1 Related Work

Reshaping. Previous work on reshaping convex polyhedra relies on Alexandrov's Gluing Theorem [1, p.100] (which we abbreviate AGT):

Theorem AGT *If polygons are glued together satisfying three conditions:*

1. *All their perimeters are glued, without gaps or overlap.*
2. *At most 2π surface angle is glued at any point.*
3. *The result is homeomorphic to a sphere.*

*Western New England University, Springfield, MA, USA. thul@wne.edu, supported by NSF grant DMS-1906202.
†University of Waterloo, Waterloo, ON, Canada. alubiw@uwaterloo.ca
‡Massachusetts Institute of Technology, MA, USA. kmundil@mit.edu
§Meiji University, Tokyo, Japan. cnara@jeans.ocn.ne.jp.
¶Smith College, Northampton, MA, USA. jorourke@smith.edu.
‖Harvard University, Cambridge, MA, USA. tkadlec@math.harvard.edu.
**Japan Advanced Institute of Science and Technology, Ishikawa, Japan. uehara@jaist.ac.jp

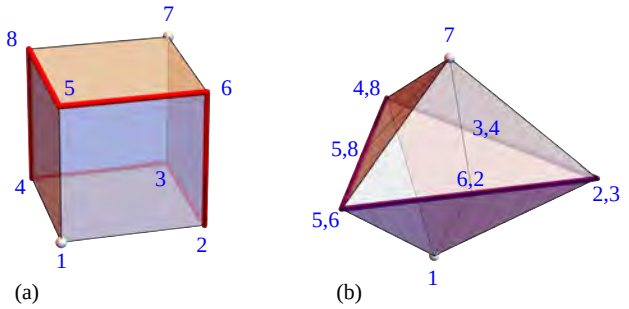


Figure 1: Vertices identified by their indices. (a) $Q = 265843$. (b) After quasi-twisting 1 unit: $5 \to 6, 6 \to 2, 2 \to 3$, etc.

Then the result is a convex polyhedron (possibly degenerated to a doubly-covered convex polygon), unique up to rigid motions.

A decade ago it was shown that every convex polyhedron could be unfolded to a single planar piece (possibly overlapping) and refolded to a different convex polyhedron [5]. A recent significant extension of this line of investigation showed (among other results) that any of the five Platonic solids can transform to any other by a sequence of at most six unfold-refold steps [4].

In a different direction, it was shown in [13] that, under mild conditions, a vertex can be excised from a convex polyhedron and transplanted elsewhere to create a new convex polyhedron. And [14] showed that any convex polyhedron can be converted to (a scaled copy) of any other via a sequence of vertex "tailorings"—excising a vertex along a digon and suturing the cut closed.

All of these reshaping results rely heavily on Alexandrov's Gluing Theorem, whose proof is non-constructive. There is no practical algorithm for actually constructing the three-dimensional shape of the polyhedron guaranteed by AGT; only an impracticable pseudo-polynomial-time algorithm is available [11]. However, ad hoc calculations suffice for polyhedra with a few vertices (say, 8), or significant symmetries. And it seems possible that testing whether the polyhedron guaranteed by AGT is a doubly-covered polygon is easier than the general case, although one attempt in this direction did not achieve polynomial-time [12].

Quasigeodesics. Pogorelov's proof that there are always at least three simple closed quasigeodesics on a convex polyhedron is also non-constructive, and it has long been an open problem to design an algorithm to find a simple closed quasigeodesic (Open Problem 24.2 [7]). Recently there has been significant progress. First, a pseudopolynomial algorithm for finding at least one closed quasigeodesic (not necessarily simple) was detailed in [6]. Second, an exponential algorithm for finding all the simple closed quasigeodesics was described in [3]. Despite this progress, there remains no practical algorithm for finding simple closed quasigeodesics.

Questions. The quasi-twist operation suggests many questions, most fundamentally: From a given P, what quasi-twisted $\tilde{P}$ can result? A few remarks:

- Every P can be twisted to some $\tilde{P} \neq P$, because every P has simple closed quasigeodesics.
- $\tilde{P}$ could have as many as twice the number of vertices as P, and as few as half the number. For example, if P is a doubly-covered regular n-gon and Q its boundary, then quasi-twisting by angle π/n leads to $\tilde{P}$ with double the number of vertices. Reversing the roles of P and $\tilde{P}$ halves the number of vertices.
- Quasi-twisting P can lead to an uncountably infinite number of incongruent $\tilde{P}$. For example, quasi-twisting a doubly-covered n-gon by different angles in $(0, \pi/n)$ leads to incongruent $\tilde{P}$.

The regular n-gon example connects to *D-forms*, gluing two congruent convex shapes along their perimeters [7].

Since it is impractical both to find quasigeodesics and to apply AGT, algorithmic questions are currently unapproachable. Here we start the investigation of the natural question: Which P can be ***quasi-twisted flat***, i.e., is it possible to quasi-twist P to a doubly-covered polygon? We further narrow the question to the five Platonic solids. We show that the regular tetrahedron, the cube, and the regular octahedron can all quasi-twist flat. We conjecture that neither the dodecahedron nor the icosahedron can quasi-twist flat. Along the way, we establish some sufficient conditions for flattening by quasi-twists, and some necessary conditions, leaving complete characterization unresolved.

For brevity, henceforth we shorten "simple closed quasigeodesic" to ***quasigeodesic***, and "simple closed geodesic" to ***geodesic***. In contrast, a ***geodesic segment*** is a geodesic path between distinct vertices on P. A quasigeodesic is composed of geodesic segments.

2 Flattening with Perimeter Q

In the simplest examples of quasi-twisting to a doubly-covered polygon, the quasigeodesic Q becomes the perimeter of the doubly-covered polygon. We begin by giving necessary and sufficient conditions for this. Later we show that the regular tetrahedron and the cube satisfy these conditions, and in fact, we can even find a suitable Q in the 1-skeleton of P.

Lemma 1 *Quasi-twisting polyhedron P along a quasi-geodesic Q with twist τ produces a doubly-covered polygon whose perimeter is Q if and only if:*

1. *Q passes through every vertex of P;*
2. *At every point aligned by τ, the angles on the two sides of Q are equal.*

Proof. Suppose the conditions hold. Because Q includes every vertex of P, the interiors of the two sides A and B are empty of vertices, i.e., they are flat polygons. Because τ aligns points with equal angles (not only at the vertices of the polygon, but also along the sides), the two flat polygons are the same, so the result is a doubly-covered polygon.

For the other direction, if Q is the perimeter of a doubly-covered polygon, then Q must have passed through every vertex of P, and the twist τ has aligned equal angles. □

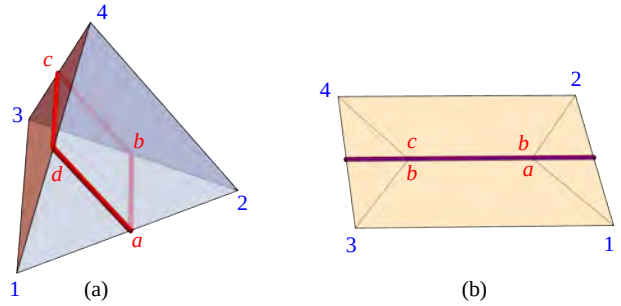


Figure 2: (a) $Q = abcd$. (b) Quasi-twisting results in a doubly-covered $1 \times \sqrt{3}/2$ rectangle.

This lemma is not, however, the only way to flatten P by quasi-twisting. We have several examples of P and Q that twist to flat polygons but which do not satisfy the conditions of the lemma. Perhaps the simplest is Q determined by the midplane of a regular tetrahedron, shown in Fig. 2(a). Here Q is a closed geodesic through no vertices, with two vertices to each side. If the edges are unit length, a twist by $\frac{1}{2}$, matching $abcd$ to $bcda$, results in a $1 \times \sqrt{3}/2$ doubly-covered rectangle, shown in (b). The four vertices become the corners of the rectangle and Q is not the boundary of the rectangle. We will show another example in Fig. 10.

3 Tetrahedron

The only quasigeodesic Q (up to relabeling) that includes all four vertices of a regular tetrahedron is shown in Fig. 3(a). Cutting Q partitions P into A and B, each alternating angles $\frac{\pi}{3}$ and $\frac{2\pi}{3}$. Quasi-twisting A one unit lines up the angles to match, as required by Lemma 1, resulting in a doubly-covered parallelogram, again alternating $\frac{\pi}{3}$ and $\frac{2\pi}{3}$ angles.

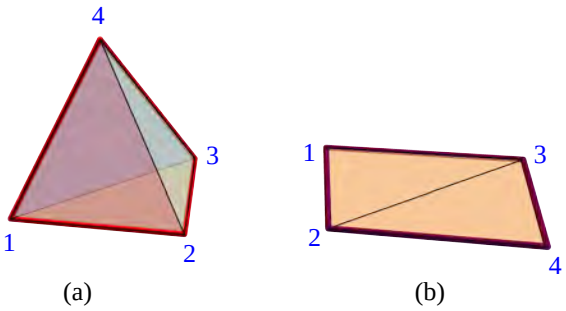


Figure 3: (a) Regular tetrahedron twists to (b) doubly-covered parallelogram.

4 Cube

We again follow Lemma 1. On a cube there is again one (up to relabelings) 8-vertex quasigeodesic, as shown in Fig. 4(a), alternating angles $\pi/2$ and π along Q. (The 3D shape of Q is sometimes known as a "napkin holder.") Quasi-twisting 2 units aligns the equal angles and results in the 3×1 doubly-covered rectangle shown in (b) of the figure: v_5, v_6, v_7, v_8 become flat with incident angle $\pi + \pi$, and the other four vertices have doubled angle $\pi/2$.

Using the same Q but quasi-twisting 1 unit results in a doubly-covered hexagon, where Q is not the boundary of the hexagon.

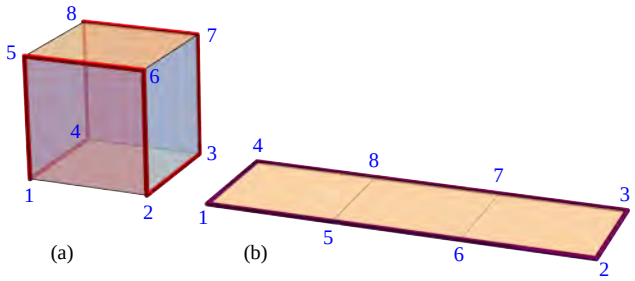


Figure 4: (a) $Q = 15623784$. (b) 3×1 doubly-covered rectangle.

5 Octahedron

Here we still use a quasigeodesic Q passing through every vertex of P, but we deviate from Lemma 1 in that we no longer align equal angles. The 6-vertex quasigeodesic Q shown in Fig. 5(a) has angles π times $\frac{1}{3}, \frac{2}{3}, 1, \frac{1}{3}, \frac{2}{3}, 1$. Fig. 5(b) shows that A and B each consists of four faces. Fig. 6(a) shows those faces unfolded, and (b) the result of shifting A by one unit. Gluing ∂A to ∂B after this shift results in creasing the layout as shown in (c), which folds to a doubly-covered $1 \times \sqrt{3}$ rectangle.

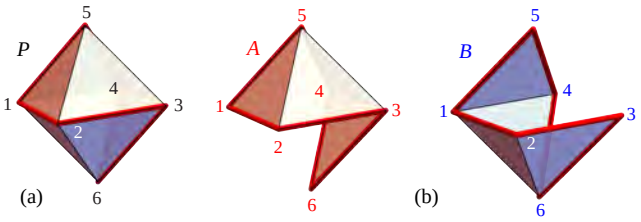


Figure 5: (a) $Q = 123645$. (b) A and B each consist of four faces.

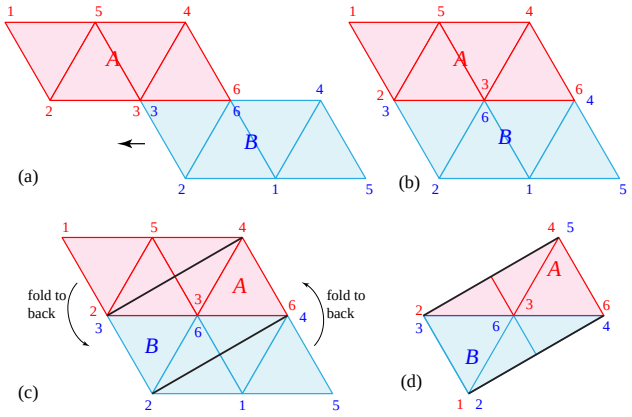


Figure 6: (a) A and B unfolded. (b) Shifting B one unit leftward. (c) Crease lines black. (d) Final doubly-covered $1 \times \sqrt{3}$ rectangle.

Quasi-twisting the different quasigeodesic shown in Fig. 7 by 1 unit leads to a doubly-covered equilateral triangle.

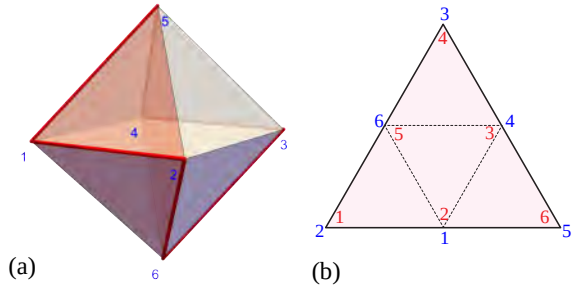


Figure 7: (a) Quasigeodesic $Q = 126345$. (b) Doubly-covered equilateral triangle, side length 2.

6 Dodecahedron and Icosahedron

We conjecture that neither the dodecahedron nor the icosahedron can quasi-twist to a doubly-covered polygon. We provide support for this conjecture by showing that the approach followed above—namely to use a quasigeodesic through all vertices—is not possible for the icosahedron or dodecahedron because no such quasigeodesics exist. We then discuss the challenges remaining to prove the conjecture, challenges that indicate what may be needed for a broader understanding of quasi-twists.

Lemma 2 *Any quasigeodesic Q on the icosahedron passes through at most 10 of the 12 vertices.*

Proof. Suppose Q passes through m vertices. Partition Q at the edges of the icosahedron into ***segments*** so that each segment is either: an edge of the icosahedron; a segment that crosses a face and is incident to one vertex of that face (we call these ***rays***); or a segment that crosses a face and is not incident to a vertex of that face. Suppose there are k edge segments, and r ray segments. Our counting argument need not include segments of the third type. First observe that Q consists of m vertex-to-vertex paths, each of which is an edge, or includes exactly two rays (one at either end of the path). Thus $m = k + \frac{1}{2}r$, so $r = 2m - 2k$.

Next, we claim (the argument is below) that each of the 20 triangle faces can contain at most one edge or ray segment. Since an edge is contained in two triangles, it counts twice. Thus $2k + r \leq 20$, and substituting $r = 2m - 2k$ gives $2m \leq 20$, so $m \leq 10$.

To prove the claim, suppose a face contains two edge or ray segments. If they are incident to the same vertex, then they must be consecutive on Q, and the angle between them is $\leq \frac{\pi}{3}$, leaving $\geq \frac{4\pi}{3}$ to the other side, violating the quasigeodesic condition. Otherwise (since a triangle does not have two vertex-disjoint edges), one segment must be a ray segment, say from vertex v to the opposite edge, and the other edge/ray segment must intersect it, see Figure 8(a). □

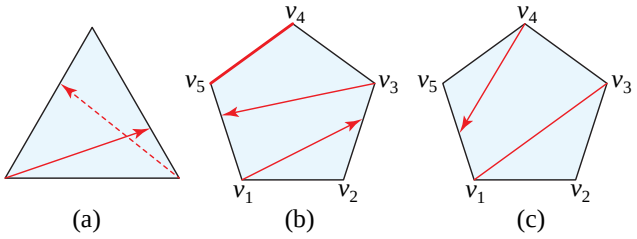


Figure 8: (a) Triangle of icosahedron. (b) Two rays and one edge in a dodecahedron face. (c) A chord v_1v_3 and a ray.

We now turn to the dodecahedron. The argument parallels that of the icosahedron.

Lemma 3 *Any quasigeodesic on the dodecahedron passes through at most* 18 *of the* 20 *vertices.*

Proof. Suppose Q passes through m vertices. We follow the same plan as in Lemma 2. Partition Q at the

edges of the dodecahedron into *segments*. Here we have four types of segments: an edge of the dodecahedron; a segment that crosses a face from vertex to vertex (we call these ***chords***); a segment that crosses a face and is incident to one vertex (again ***rays***); or a segment that crosses a face and is not incident to any vertex of that face. Suppose there are k edge segments, c chord segments, and r ray segments. Again the counting argument need not include crossing segments, the last type. Henceforth we use ***segments*** to refer to edges, chords, and rays. Since Q consists of m vertex-to-vertex paths, each of which is an edge, a chord, or includes exactly two rays, we have $m = k + c + \frac{1}{2}r$ and thus $r = 2m - 2k - 2c$.

We make two claims about segments within a face to complete our counting argument.

- A face can contain three segments, but cannot contain four segments.
- If a face contains a chord then it has at most one other segment, and that other segment cannot be a chord.

See Fig. 8(b,c). These claims are proved below. The claims imply that each of the c chords is in a unique face with at most one other edge/ray segment, and the remaining $12-c$ faces each contain at most three edge/ray segments. Again, each edge segment counts twice since it lies in two faces. Thus $2k + r \le c + 3(12 - c)$. Substituting for r we obtain $2m - 2c \le c + 3(12 - c)$, which gives $2m \le 36$, so $m \le 18$.

To prove the claims, first suppose a face contains two segments incident to the same vertex. Then they must be consecutive on Q, and the angle between them is $\le 108°$ leaving $\ge 216°$ to the other side, violating the quasigeodesic condition.

Next, consider the (disjoint) segments in the face. See Fig. 8(b). Each one cuts the face into two "sides," and we say that an "empty side" is a piece that contains no other segment. There are at least two empty sides and each (closed) empty side contains at least two vertices, leaving only one remaining vertex for a third (and last) segment. Thus there are at most three segments. There cannot be two (disjoint) chords, and if there is a chord, then one of its sides must be empty, and contains three vertices. Furthermore, a second empty side contains two vertices, so there cannot be a third segment. □

We do not believe either Lemma 2 or Lemma 3 is tight, in that it seems neither 10- nor 18-vertex quasigeodesics are achievable on the icosahedron and dodecahedron respectively.

6.1 Conjecture Revisited

Having eliminated the possibility of using Lemma 1, the dodecahedron and the icosahedron could only twist to a flat polygon if Q does not include all the vertices. Then the vertices not on Q must flatten to the boundary of the doubly-covered convex polygon.

Again we use $\tilde{P}$ to represent P after quasi-twisting flat, and $\tilde{Q}$ to represent the image of Q on $\tilde{P}$. Thus $\tilde{Q}$ is a quasigeodesic on $\tilde{P}$. Note that $\partial\tilde{P}$ is itself a quasigeodesic of $\tilde{P}$ and indeed a *straightest* such quasigeodesic in the sense that it bisects the angle at each vertex through which it passes.[1]

If Q does not pass through all the vertices of P, then, by Lemma 1, $\tilde{Q}$ is different from $\partial\tilde{P}$, though they may share edges (see for example Figs. 2 and 6). It is tempting to imagine that there will be only two intersections between $\tilde{Q}$ and $\partial\tilde{P}$ as in those examples, in which case the two sections of $\partial\tilde{P}$ would lift to straightest quasigeodesic paths on A and B. However, the following example shows that there may be more than two intersections.

Let P be the doubly-covered rectangle shown in Fig. 9(a), and Q the horizontal bisector on the front and back, with v_1, v_3 above Q in A and v_2, v_4 below in B. P is already flattened, but we can still quasi-twist. Quasi-twisting by $\sqrt{2}/2$ leads to the doubly-covered unit square shown in (b). Directing $\tilde{Q}$ as $wxyz$, we have v_1, v_3 left of Q and v_2, v_4 right. This is not surprising, as the interiors of A and B are unaffected by quasi-twisting and gluing along their boundaries. What is perhaps surprising is that $\partial\tilde{P}$ is partitioned into four sections by $\tilde{Q}$, not the two sections one might expect.

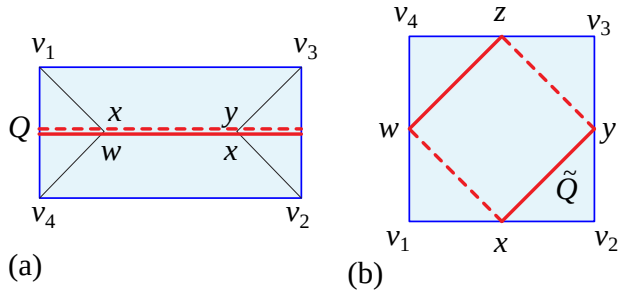


Figure 9: (a) $\sqrt{2} \times \sqrt{2}/2$ doubly-covered rectangle. (b) 1×1 doubly-covered square. Dashed portion of Q lie on the back side.

At the other extreme, one might ask if P can quasi-twist flat using a geodesic, i.e., one that does not pass through any vertices of P. This is possible for the tetrahedron (Fig. 2), and also for the octahedron (Fig. 10). There are three different geodesics on the cube [9, 8], but our experimental results show that none of them permit flat quasi-twisting. We conjecture that the icosahedron and dodecahedron cannot be twisted flat at true geodesics either. Fuchs and Fuchs [9] categorize the three possible geodesics on the icosahedron, but the case of the dodecahedron is apparently unresolved.

[1] The term "straightest geodesic" is from [17].

To summarize, we do not know if the icosahedron or dodecadedron can quasi-twist flat. A main difficulty is that we lack an understanding of when a quasigeodesic allows a flat quasi-twisting. Another impediment is that there is no complete inventory of the (simple) quasigeodesics on the dodecahedron or icosahedron. Just recently a 1-vertex quasigeodesic on the dodecahedron was found [2]. Even tetrahedra can have as many as 34 incongruent quasigeodesics [15].

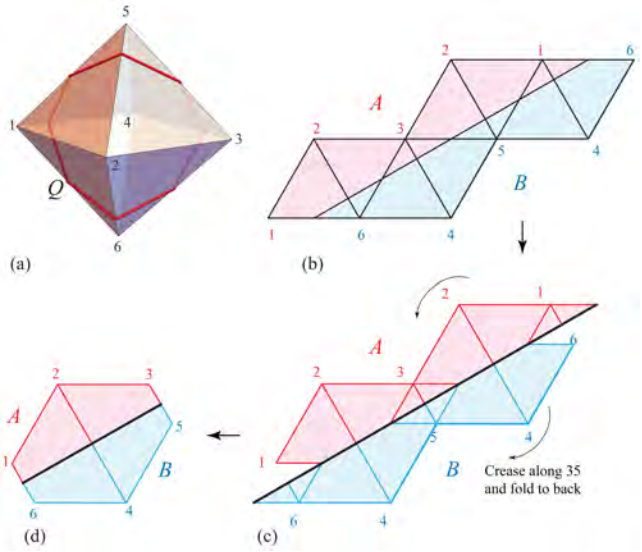


Figure 10: (a) Geodesic on octahedron. (b) A and B unfolded. (c) After quasi-twisting by $\sqrt{3}/2$. (d) Doubly-covered hexagon.

7 Open Problems

Because the quasi-twisting concept is new, almost every question one could pose is open. It would be interesting to know which polyhedra can be obtained from P by repeated quasi-twisting. Finding more substantive necessary conditions for quasi-twisting flat could resolve flattening the Platonic solids.

We emphasized quasi-twisting from P to a flat polyhedron $\tilde{P}$. The reverse viewpoint is equally interesting. We mentioned in Section 1 that a doubly-covered regular n-gon could be viewed as a discrete version of a D-form. It is natural to explore what shapes can be quasi-twisted from doubly-covered convex polygons. Even restricting to doubly-covered rectangles is interesting. For example, Fig. 11 illustrates quasi-twisting a doubly-covered square using the perimeter as the quasigeodesic.

Acknowledgements. We thank all the participants at the 3rd Virtual Workshop on Computational Geometry (2022), directed by Erik Demaine, where this research was initiated. We also thank the referees for helpful suggestions.

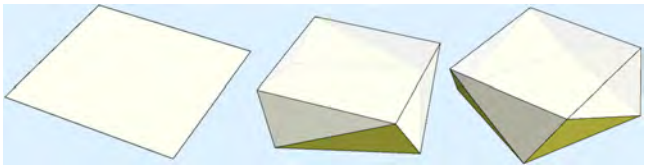


Figure 11: Snapshots of quasi-quasi-twisting a doubly-covered square. See `https://klaramundilova.com/square-twist/`.

References

[1] A. D. Alexandrov. *Convex Polyhedra*. Springer-Verlag, Berlin, 2005. Monographs in Mathematics. Translation of the 1950 Russian edition by N. S. Dairbekov, S. S. Kutateladze, and A. B. Sossinsky.

[2] J. S. Athreya and D. Aulicino. A trajectory from a vertex to itself on the dodecahedron. *Amer. Math. Monthly*, 126(2):161–162, 2019.

[3] J. Chartier and A. de Mesmay. Finding weakly simple closed quasigeodesics on polyhedral spheres. In *38th Symp. Comput. Geom. (SoCG)*, Leibniz Internat. Proc. Informatics (LIPIcs), pages 27:1–27:16, June 2022.

[4] E. D. Demaine, M. L. Demaine, Y. Diomidov, T. Kamata, R. Uehara, and H. A. Zhang. Any regular polyhedron can transform to another by $o(1)$ refoldings. In *Proc. 33rd Canad. Conf. Comput. Geom.*, pages 332–342, 2021.

[5] E. D. Demaine, M. L. Demaine, J. i Itoh, A. Lubiw, C. Nara, and J. O'Rourke. Refold rigidity of convex polyhedra. In *28th European Workshop Comput. Geom. (EuroCG)*, pages 273–276, Mar. 2012.

[6] E. D. Demaine, A. C. Hesterberg, and J. S. Ku. Finding closed quasigeodesics on convex polyhedra. *Discrete Comput. Geom.*, 2021. arXiv:2008.00589. SoCG2020. To appear *Discrete Comput. Geom.*

[7] E. D. Demaine and J. O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, 2007. `http://www.gfalop.org`.

[8] D. Fuchs. Periodic billiard trajectories in regular polygons and closed geodesics on regular polyhedra. *Geometriae Dedicata*, 170(1):319–333, 2014.

[9] D. B. Fuchs and E. Fuchs. Closed geodesics on regular polyhedra. *Moscow Mathematical Journal*, 7(2):265–279, 2007.

[10] P. Gruber. A typical convex surface contains no closed geodesic. *J. Reine Angew. Math.*, 416:195–205, 1991.

[11] D. Kane, G. N. Price, and E. D. Demaine. A pseudopolynomial algorithm for Alexandrov's theorem. In *Workshop Algorithms Data Struct. (WADS)*, pages 435–446. Springer, 2009.

[12] J. O'Rourke. On flat polyhedra deriving from Alexandrov's theorem. arXiv:1007.2016v2 [cs.CG]: `arxiv.org/abs/1007.2016v2`, July 2010.

[13] J. O'Rourke. Vertex-transplants on a convex polyhedron. In *Proc. 32nd Canad. Conf. Comput. Geom.*, pages 138–143, Aug. 2020.

[14] J. O'Rourke and C. Vîlcu. Reshaping Convex Polyhedra. arXiv 2107.03153v2 [math.MG]: `arxiv.org/abs/2107.03153`, July 2021.

[15] J. O'Rourke and C. Vîlcu. Simple closed quasigeodesics on tetrahedra. *Information*, 13:238–258, May 2022.

[16] A. V. Pogorelov. Quasi-geodesic lines on a convex surface. *Mat. Sb.*, 25(62):275–306, 1949. English transl., *Amer. Math. Soc. Transl.* 74, 1952.

[17] K. Polthier and M. Schmies. Straightest geodesics on polyhedral surfaces. In *ACM SIGGRAPH 2006 Courses*, pages 30–38. 2006.

ZHED is NP-complete

Sagnik Saha* Erik D. Demaine†

Abstract

We prove that the 2017 puzzle game ZHED is NP-complete, even with just tiles numbered 1. Such a puzzle is defined by a set of unit-square tiles in a square grid, and a target square of the grid. A move consists of selecting a previously unselected tile and then filling the next unfilled square in a chosen direction from that tile (similar to Tipover and Cross Purposes). We prove the NP-completeness of deciding whether the target square can be filled, by a reduction from rectilinear planar monotone 3SAT.

1 Introduction

ZHED [Gro17] is a puzzle game available for Android, iOS, Switch, and Steam, first released in 2017. An instance of this puzzle is played on a $n \times n$ square grid; refer to Figure 1 (left). One of the squares is designated as the ***target***. Several of the other squares are ***filled*** by ***tiles***. Each initial tile has an integer ***number*** between 1 and $n-1$ written on it.

In each move, the player selects one of the remaining numbered tiles and a direction (up, down, left, or right); refer to Figure 1. If the tile was numbered k, the move replaces the tile with a blank tile (keeping the square filled, but removing the number) and fills the k closest unfilled squares in the specified direction with blank tiles. The objective of the game is to fill the target square.

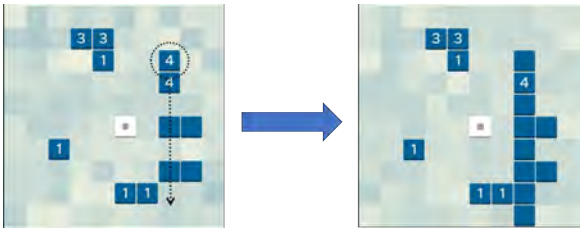


Figure 1: A typical move in ZHED. Tiles are drawn in dark blue. The target square has a small square in the middle.

The number of moves in such a puzzle is bounded by the number of numbered tiles, so the puzzle is in NP. In this paper, we prove that the puzzle is in fact NP-complete, even when all tiles are numbered 1.

The mechanic of ZHED moves is similar to the idea of a tower of specified height that falls over in a specified direction. This "falling tower" mechanic is present in the puzzle board game Tipover, which is NP-complete [Hea06], and the 2-player board game Cross Purposes, which is PSPACE-complete [Hea05]. The key distinction is that both Tipover and Cross Purposes prevent a tower from falling on top of occupied/filled squares.

2 Rectilinear Planar Monotone 3SAT

Our NP-hardness reduction is from the known NP-complete problem Rectilinear Planar Monotone 3SAT (henceforth called ***RPM-3SAT***) [BK12]; refer to Figure 2. In this problem, we are given a Boolean formula $\mathcal{F}$ over n Boolean variables $x_1, x_2, \ldots, x_n$. Formula $\mathcal{F}$ is in conjunctive normal form, so the formula $\mathcal{F}$ is the logical AND ($\wedge$) of m ***clauses***. Each clause in $\mathcal{F}$ is a logical OR ($\vee$) of at most three ***literals***, and the literals in each clause are either all positive, consisting of non-negated variables of the form x_i, or all negative, consisting of negated variables of the form $\overline{x_i}$. (The last is the "monotonicity" property.) We are also given a planar embedding of the variable–clause incidence graph connecting each variable to the clauses containing them, which satisfies the following five conditions:

1. All variables and clauses are grid-aligned rectangles of height 1.
2. All of the variables lie along a single horizontal line.
3. All edges lie along vertical lines.
4. All positive clauses lie above the line of the variables.
5. All negative clauses lie below the line of the variables.

The goal in the RPM-3SAT problem is to determine whether there exists a value assignment to the n Boolean variables that satisfies $\mathcal{F}$. It is known to be NP-complete [BK12].

3 Reduction: Puzzle Construction

We now show how to build a ZHED puzzle that represents an arbitrary instance of RPM-3SAT, in such a

*Work done while at MIT.
†MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA, edemaine@mit.edu

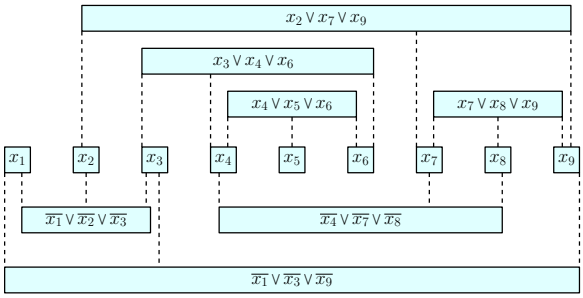


Figure 2: A sample instance of Rectilinear Planar Monotone 3SAT (RPM-3SAT). This planar embedding represents the conjunctive-normal-form formula $(x_2 \vee x_7 \vee x_9) \wedge (x_3 \vee x_4 \vee x_6) \wedge (x_4 \vee x_5 \vee x_6) \wedge (x_7 \vee x_8 \vee x_9) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}) \wedge (\overline{x_4} \vee \overline{x_7} \vee \overline{x_8}) \wedge (\overline{x_1} \vee \overline{x_3} \vee \overline{x_9})$.

way that the puzzle we construct will be solvable if and only if the original problem is satisfiable (as proved in Section 4). We start with a description of the various gadgets in our construction, and then describe how they are combined together. In our construction, we only need to use tiles with the number 1, so in the figures we omit the number on tiles.

3.1 Threshold Gadget: Wires, AND, OR

Our main workhorse is the ***threshold gadget***, which consists of a line of tiles on alternate squares, as shown in Figure 3. The empty squares between the tiles each function as a ***source***. The threshold gadget is parameterized by a nonnegative integer k, and we call the square at a distance of $k + 1$ from the last tile in the gadget (on the same line in the direction the gadget is meant to be expanded) the ***target***.

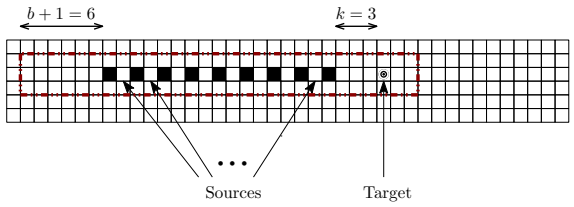


Figure 3: Threshold gadget. We assume that at most 5 of the source squares can be filled by outside tiles, and this gadget activates as long as at least 3 of those squares are filled before the tiles in this gadget are expanded rightwards.

Threshold property. This gadget has the property that the target square can be filled if and only if at least k of the source squares were already filled by tiles from other gadgets. The intended activation is to expand each tile in the gadget toward the target, in decreasing order of distance. Without any sources already filled, this will reach only one square from the last tile in the gadget. If, however, j sources were already filled, then it will reach distance $j + 1$. Thus, the target will get filled if and only if $j \geq k$.

Wire, turn, AND, OR. This gadget is versatile, and forms the basis of several other gadgets. If we connect only one source to other gadgets and set $k = 1$, then we get simple ***wire gadget*** which propagate a signal (represented by a square being filled) from that source to the target. We can ***turn*** the wire by connecting the target of one wire at a source of another orthogonal wire. On the other hand, if $m > 1$ sources are attached to other gadgets (overlapping each source with the target of the other gadget, representing m input signals), then we obtain an ***OR gadget*** by setting $k = 1$, and we obtain an ***AND gadget*** by setting $k = m$.

Bounding region. Because of the spacing between the tiles, a threshold gadget can only possibly fill squares on that line within a distance of $b + 1$ from one end of the gadget, where b is the maximum number of sources that can be filled by other gadgets. Because each tile has the number 1, the gadget can also affect at most the two adjacent lines of squares. Therefore, the dotted rectangle in Figure 3 serves as a ***bounding region*** of the gadget. No sequence of moves in the overall puzzle can lead to a tile in the threshold gadget directly filling a square outside its bounding region.

Chaining threshold gadgets. As alluded to above, we can chain threshold gadgets by making the target of one a source of the next, as shown in Figure 4. This allows us to aggregate signals and turn wires, as chaining necessarily introduces a right-angle turn. During activation, we simply expand the first gadget's tiles first, which (maybe) fills its target square, and then expand the tiles of the second gadget.

This interaction is one of only two ways different gadgets are allowed to interact with each other in our construction. The first gadget's bounding region grows by 1 unit in the direction it propagates, because the second gadget may be expanded before it. The composite bounding region is simply the union of the two gadgets' individual bounding regions.

3.2 Shift Gadget

To resolve parity issues in our construction, we sometimes need to shift threshold gadgets by 1 square. To do so, we simply add an extra tile at the beginning of the gadget, adjacent to the old starting tile, as in Figure 5. To activate this gadget, we expand all tiles toward the

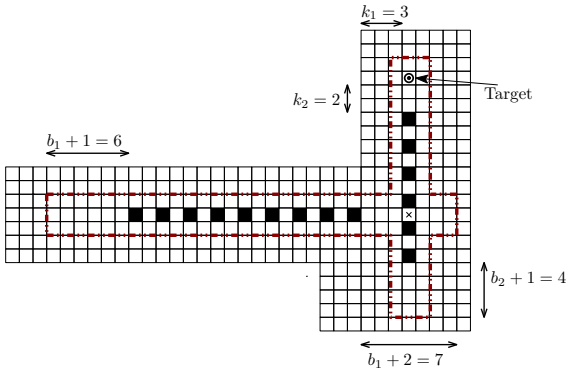


Figure 4: Chaining together two threshold gadgets (with parameters k_1 and k_2 respectively). The target of the horizontal threshold gadget (marked by $\times$) acts as a source for the vertical threshold gadget.

target, in decreasing order by distance (as in the threshold gadget).

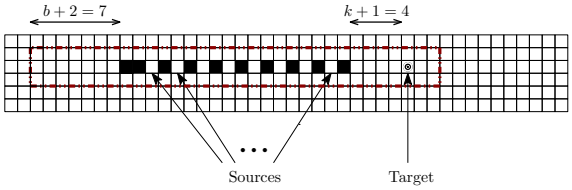


Figure 5: Shift gadget. The threshold gadget from Figure 3 is shifted by one square to the right.

Bounding region. As a result of the extra tile, the bounding region of the original threshold gadget expands by 1 square in the forward direction, and 2 squares in the backward direction (counting the new tile itself, as well as the extra square it can potentially fill). We move the target square one square forward, and everything else remains unchanged.

3.3 Variable Gadget

A ***variable gadget*** consists of a large even number L of tiles consecutive on a horizontal row; refer to Figure 6. These tiles are all meant to be expanded in the same direction, either left or right. In our construction, expanding all tiles left corresponds to setting the variable to ***false***, and expanding all tiles to the right corresponds to setting the variable to ***true***.

Activating a variable gadget as intended fills L squares in one direction. We attach vertical wires that propagate the filling of certain squares to the left or right of the tiles (to clause gadgets), thus "reading" the value of the variable. All such wires are at a distance

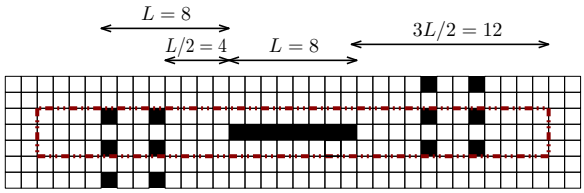


Figure 6: Variable gadget with $L = 8$ tiles, along with four threshold gadgets "reading" the value of this variable. The two threshold gadgets right of the variable can get one of their source squares filled if the variable tiles are all expanded right, setting it to true. The two gadgets on the left can be activated when the variable is set to false.

between $L/2 + 1$ and L from the variable gadget. This choice ensures that an unintended activation of the gadget, which expands some tiles left and other tiles right, cannot activate wires on both sides of the variable. For example, if x of tiles are expanded left and y are expanded right, then at least one of x and y will be less than $L/2 + 1$, so no wire in that direction will be activated.

We choose the value of L based on the number of vertical wires needed. In the rest of the construction described below, we specify the number of wires needed for each clause gadget, and the gaps needed between consecutive wire. Any value of L large enough so that all of those wires fit within $L/2 - 1$ columns suffices.

Bounding region. The bounding region of the variable gadget is again only 3 rows high because all the tiles have number 1. There are less than $L/2$ vertical wires on either side within the reach of the tiles in this gadget, so we obtain a upper bound on the bounding region of the variable gadget by supposing that all of those wires were already expanded before any tile in this gadget is expanded. Thus the bounding region of a variable gadget spans at most $L + L/2 = 3L/2$ columns outside the tiles in each direction, as shown in Figure 6.

3.4 Clause Gadget

The threshold gadget enables us to build a ***clause gadget***; refer to Figure 7. Because our 3SAT instance is monotone, each clause uses only positive or only negative literals. For positive clauses, we create a horizontal threshold gadget going rightwards above the variables; and for negative clauses, we create the threshold gadget below the variables. Then we connect the clause to corresponding variables (on the left side of the variable for positive clauses, and on the right side of the variable for negative clauses) via vertical "thick wires". A ***thick wire*** is a group of g parallel wires connecting the same

gadgets, each separated 4 units apart from each other; thus, a single input actually advances the reach of the gadget by g. For the threshold gadget to function as an OR of the literals, we set its k to g (instead of 1). We refer to the combination of the thick wires and the horizontal OR gadget as the clause gadget. Thus the target square for the clause gadget is $g+1$ distance to the right of the rightmost tile in the gadget. The thickness g is a parameter that may vary across different clauses and will be decided later.

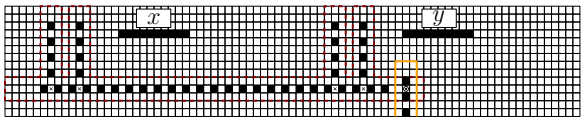


Figure 7: Clause gadget for $(\overline{x} \vee \overline{y})$. We use thick wires, each made of 2 individual wires, to connect the clause with the variables. The vertical wire chained from the OR gadget (with the orange bounding region) is a clause propagator that uses the target square of this clause gadget as a source, and propagates the signal elsewhere.

Bounding regions. The clause gadget consists mainly of a threshold gadget which can have $3g$ filled sources, so it already has a defined bounding region. By our choice of separation, the bounding regions of the individual wires comprising each thick wire do not overlap.

3.5 Crossover Gadget

We use a ***crossover gadget*** when we need a vertical threshold gadget to intersect the horizontal threshold gadget belonging to a clause. As shown in Figure 8, we simply position the two threshold gadgets so that the intersection square is blank (with four tiles surrounding that intersection square like a plus sign). Next, we increase the k of the vertical threshold gadget by 1. Essentially, we assume that the horizontal gadget will be activated before the vertical one, and we compensate for the extra filled source square for the vertical gadget by moving its target square one unit farther. If the player violates this assumption, the horizontal gadget will effectively have its k off by 1, an error we will tolerate using thick wires (see below).

Bounding region. The bounding region of the crossover gadget is easy to calculate because there are only four tiles involved. At the point of intersection, a 3×3 square centered at the intersection point suffices. The bounding regions of both gadgets involved get longer in their long direction by 2 units (1 unit on each end) because of the potential extra source square that may be filled.

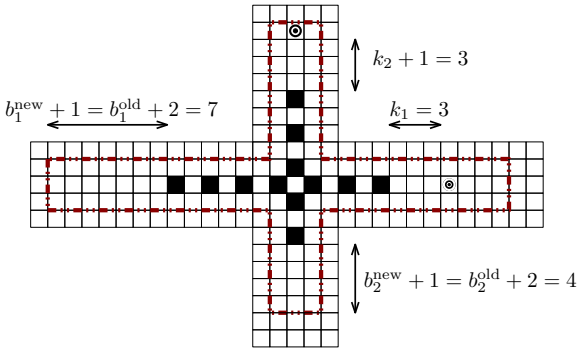


Figure 8: Crossover gadget. The horizontal threshold gadget is meant to be expanded first, so its target remains unchanged. The vertical threshold gadget is to be activated second, when the intersection square is already filled in. This pushes the target by one square.

3.6 Putting It All Together

To construct the overall puzzle, we transform the planar embedding of the provided RPM-3SAT instance; refer to Figure 9. All the variable gadgets go on the central horizontal row, spaced far enough apart that none of their bounding regions overlap. We then draw the clause gadgets according to the provided clause rectangles (above and below the central row for positive and negative clauses respectively). We space out the clauses more than the original drawing in order to leave room for the thick-wire connections between variables and clauses. Because we started with a planar monotone drawing, we obtain a noncrossing drawing.

Next we add vertical wires (henceforth referred to as ***clause propagators***) to transport the output signals from all of the target squares of the positive clauses up. We place an AND gadget horizontally on a row high enough so that its bounding region doesn't collide with the bounding region of any clause gadget, and ensure that the target of all positive clause propagators coincide with sources of the AND gadget. We perform a symmetric construction below the negative clauses, resulting in an upper and a lower AND gadget.

Each clause propagator may intersect with the horizontal threshold gadgets from some of the other clauses (nested above this one). We use a crossover gadget for each such point of intersection. For each clause, we count the number x of crossovers in its OR gadget, and set the thickness g of the thick wires attached to the clause gadget to be strictly larger: $g > x$. This choice ensures that all the propagators intersecting a clause combined have less influence on the clause threshold gadget than a single thick wire, so no clause target can be satisfied by the propagator intersections alone.

We design the upper and lower AND gadgets to both go rightwards and have their targets on the same col-

umn, which is farther right than the bounding region of every other gadget. We combine these two signals using a vertical AND gadget, with a target square near the bottom right of the board. This square is the actual ***target*** for the entire puzzle in our construction.

The parity issue in the above construction is that the threshold gadget has a repeating unit of size 2, and hence we need to account for parity in lining everything up. This issue may cause problems when we want multiple threshold gadget targets to be on the same line for a chained gadget. Parity issues may also become significant for crossover gadgets, which require the intersection square to be empty. We fix these issues using shift gadgets. This approach allows us to adjust the rows of the clause gadgets and the columns of the clause propagators without parity restrictions. If we put all the clause gadgets on rows of the same parity, we can ensure that all the crossover gadgets line up. We use the same trick to ensure proper chaining of the clause propagators into the upper and lower AND gadgets, and the chaining of those two AND gadgets into the final vertical AND gadget. See Figure 9 for an example.

Puzzle size. During this construction, we space the gadgets out sufficiently so that no two bounding regions collide except as part of an intended interaction (through crossover gadgets or chaining threshold gadgets). The thickness g of each clause gadget's thick wires are determined at this stage, and the size L of each variable gadget then depends on the total number of individual wires using it. We'll show that because all of our gadgets have bounding-box dimensions of polynomial size, the overall board remains polynomial sized in the input parameters.

Note that there are a total of m clause propagators corresponding to the m clause gadgets. Therefore, any given clause gadget can have a maximum of $m-1$ crossovers in its OR gadget, and we don't need any thick wire with thickness g exceeding m. Each variable can be connected to at most m clauses, and may therefore need m thick wires *reading* it. So we need to accommodate a maximum of m^2 vertical wires, each spaced 4 units apart within $L/2-1$ columns. This implies that the variables can all have sizes $L \leq 8m^2+2$.

We can draw the clause gadgets' horizontal wires just long enough to intersect all the thick wires comprising them, and therefore the bounding region of a clause gadget should end no more than $3g+2 \leq 3m^2+2$ units to the right of the rightmost thick wire. To account for at most m crossover gadgets intersecting the clause, it can extend a further m units.

The horizontal dimension of the puzzle only needs to be big enough to fit the bounding regions of all the variable and clause gadgets, followed by the AND gadgets. The total length of the bounding region of a variable gadget

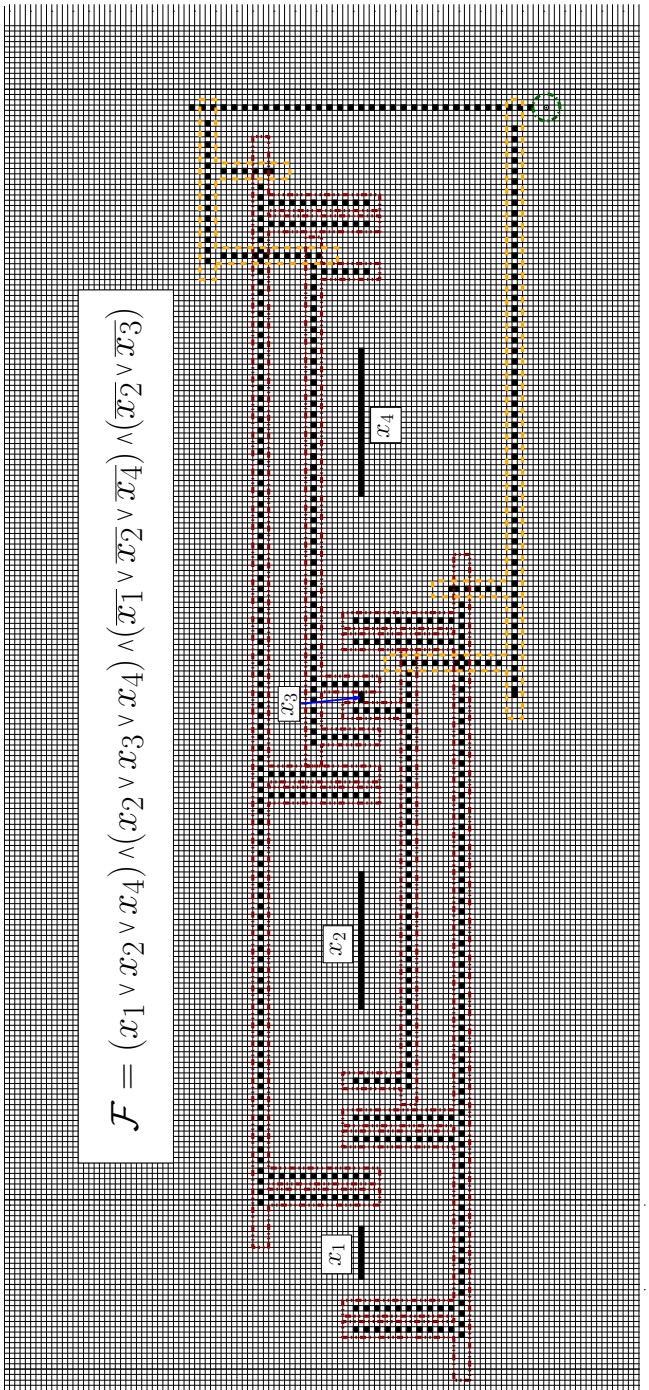


Figure 9: The entire puzzle board for a sample formula. The bounding regions for the clause gadgets are marked in red. The bounding regions of the clause propagators together with the upper and lower AND gadgets are colored orange. The final target square is at the lower-right of the board, marked by a green circle. Note that different variables have different sizes L depending on how many wires connect them to clause gadgets: v_3 uses only 2 tiles while v_4 uses 28. Similarly, the first and third clauses use thick wires while the other two use single wires to connect with variables. We use shift gadgets for the lower AND gadget and for a couple of clause propagators.

is $7L$, and the n variables account for $7n(8m^2+2)$. The clause gadgets can add another $3m^2+m+2$. The upper and lower AND gadgets have up to m clause propagators, and the vertical AND gadget therefore needs to be another m squares to the right of the rightmost clause gadget's bounding region. This shows that the horizontal dimension of our generated puzzle is $\mathcal{O}(nm^2)$.
The vertical size of the puzzle is much easier to upper bound. The bounding region of a basic horizontal threshold gadget is only 3 squares tall. Our generated puzzle grows vertically because we need to draw multiple clause gadgets on top of each other without their bounding regions colliding. All the vertical wires can only be as long as necessary to intersect with the variable gadgets and the horizontal OR gadgets of the clauses. Therefore, the total height of our puzzle is $\mathcal{O}(m)$.

4 Reduction Correctness

Finally we show that the construction of Section 3 is a valid reduction, i.e., an instance of RPM-3SAT is satisfiable if and only the obtained ZHED puzzle is solvable.

4.1 Satisfiable Formula ⇒ Puzzle Solution

Suppose there exists an assignment of the Boolean variables v_is that makes the formula evaluate to true. Then we solve the constructed puzzle as follows.

First we expand each of the variable tiles. If v_i = true in the solution, we expand the corresponding L tiles rightward; otherwise, we expand the tiles leftward. Then we expand all the vertical wires connecting variables to clauses, toward the clauses. Next we expand the threshold gadgets in the clause wires rightward. Because each clause is satisfied in the formula, each of the clause gadgets will be activated by at least one thick wire, and hence each of the target squares for the clause gadgets will be filled after this stage.

Now we expand all of the clause propagator wires. These propagate the filled source corresponding to their clause's target square to the extreme upper and lower horizontal AND gadgets, filling all of their sources. Next, we expand the upper and lower AND gadgets rightward, which fills the two sources of the extreme-right vertical AND gadget. Finally, we expand the last vertical AND gadget downwards, which fills the puzzle target as desired.

4.2 Puzzle Solution ⇒ Satisfiable Formula

Suppose we have a solution to the ZHED puzzle. We argue about the different gadgets in the reverse order from the intended activation sequence of Section 4.1, and use the fact that a square can only be filled by gadgets whose bounding regions contain it, to obtain a satisfying Boolean assignment. We call a threshold gadget ***successfully activated*** if it fills its target square.

- The puzzle's target square only belongs to the bounding region of the extreme-right vertical AND gadget. Therefore that AND gadget was successfully activated, with at least $k = 2$ of its sources filled by other gadgets.
- There are only 2 sources of the extreme-right vertical AND gadget that might be filled by other gadgets, so both of them must have been. This implies that both the extreme upper and lower horizontal AND gadgets were successfully activated.
- For the extreme horizontal AND gadgets to be successfully activated, all clause propagators had to have successfully activated their target squares.
- This implies that all clause gadgets successfully activated their target squares.
- If none of the thick wires activated for any particular clause gadget, then there will not be enough sources in that clause's OR gadget that belong to other gadget's bounding regions (specifically, crossovers) to reach its target square. Therefore, for every clause gadget, there is at least one thick wire with one or more component wires successfully activated.
- For any individual wire connecting a variable to a clause to be successfully activated, over half of the tiles in the corresponding variable gadget must be expanded in the direction of that clause (right if the clause is positive and left otherwise). We set each variable to false or true based on which direction (among left or right) the majority of the tiles of its corresponding variable gadget expanded; only clauses in this direction will be satisfied by the variable gadget. We set the value arbitrarily if there is a tie, in which case the variable did not satisfy any clauses.
- This assignment must be a satisfying assignment, because all clauses must have at least one variable satisfying them.

4.3 Main Theorem

We just established that determining whether a ZHED puzzle is solvable is NP-hard. It is also easy to prove membership in NP: a solution can be described by a sequence of moves, and if there are n tiles, each move takes $O(\log n)$ bits to describe. The number of moves cannot exceed the number n of tiles, so the total size of a proposed solution is polynomial in n. We can verify the solution in polynomial time by simply maintaining

the state of the board after each move and simulating the moves. Together, these assertions prove our main result:

Theorem 1 *It is NP-complete to decide if a ZHED puzzle is solvable, even when all tile numbers are* 1*.*

References

[Hea05] Robert A. Hearn. "Amazons, Konane, and Cross Purposes are PSPACE-complete". In: *Games of No Chance III: Proceedings of the BIRS Workshop on Combinatorial Games*. 2005, pp. 287–306. DOI: 10.1017/CBO9780511807251.015.

[Hea06] Robert A. Hearn. "Tipover is NP-complete". In: *The Mathematical Intelligencer* 28 (June 2006), pp. 10–14. DOI: 10.1007/BF02986878.

[BK12] Mark de Berg and Amirali Khosravi. "Optimal Binary Space Partitions for Segments in the Plane". In: *International Journal of Computational Geometry & Applications* 22.3 (2012), pp. 187–206. DOI: 10.1142/S0218195912500045.

[Gro17] Ground Control Games. *ZHED*. https://playzhed.com/. 2017.

Efficient Graph Reconstruction and Representation Using Augmented Persistence Diagrams

Brittany Terese Fasy* Samuel Micka† David L. Millman‡ Anna Schenfisch§ Lucy Williams¶

Abstract

Persistent homology is a tool that can be employed to summarize the *shape* of data by quantifying homological features. When the data is an object in $\mathbb{R}^d$, the (augmented) persistent homology transform ((A)PHT) is a family of persistence diagrams, parameterized by directions in the ambient space. A recent advance in understanding the PHT used the framework of reconstruction in order to find finite a set of directions to faithfully represent the shape, a result that is of both theoretical and practical interest. In this paper, we improve upon this result and present an improved algorithm for graph—and, more generally one-skeleton—reconstruction. The improvement comes in reconstructing the edges, where we use a radial binary (multi-)search. The binary search employed takes advantage of the fact that the edges can be ordered radially with respect to a reference plane, a feature unique to graphs.

1 Introduction

At the heart of inverse problems in the field of topological data analysis is the following question: *how many persistence diagrams are needed to faithfully represent a shape?* Since the introduction of persistence diagrams, it has been known that many "shapes" can share the same persistence diagram. With enough persistence diagrams, we arrive at a set of parameterized diagrams (that is, diagrams labeled by direction) that uniquely, or *faithfully* represents the underlying shape. Moreover, the set of parameterized diagrams is *faithful* if and only if it can be used to *reconstruct* the underlying shape.

The foundation for asking the question of how many diagrams are needed for a faithful representation was first introduced in [16], where Turner et al. defined the Persistent Homology Transform (PHT) and the Euler characteristic curve transform (ECCT). These transforms map a simplicial complex embedded in $\mathbb{R}^d$ (a shape) to sets of persistence diagrams (respectively, Euler characteristic curves) parmeterized by $\mathbb{S}^{d-1}$, the set of all directions in $\mathbb{R}^d$. They showed that, up to mild general position assumptions, no two simplicial complexes can correspond to the same PHT or ECCT, i.e., they showed that the uncountably infinite sets making up the PHT and ECCT faithfully represent the shape.

However, the PHT and ECCT are uncountably infinite sets of diagrams (and Euler characteristic curves). Thus, to bridge the gap between the theory and what can be used in practice, a discretization of the PHT was needed; several papers stepped up to the challenge and proved that there exists a finite discretizations of topological transforms that are faithful for simplicial and cubical complexes [1,2,4,5,7,8,12]. Beyond proving the existence of these finite faithful sets, Belton et al. [2] explicitly give an algorithm for using an oracle to reconstruct graphs embedded in $\mathbb{R}^d$ with n vertices. Their reconstruction uses $n^2-n+d+1$ augmented persistence diagrams in $O(dn^{d+1} + n^4 + (d+n^2)\Pi)$ time [2, Theorem 17], where $\Theta(\Pi)$ is the time complexity it takes for an oracle to produce answer a persistence diagram querry. Since the direction-labeled diagram set can be used for reconstructing the underlying graph, it is a faithful discretization of the augmented PHT (APHT).

In the current work, we give a faithful discretization of the APHT using $\Theta(d+m\log n)$ diagrams and $\Theta(dn^{d+1}+d\Pi + n^2 + m\log n(\log n + d + \Pi))$ time. This result is an improvement over [2], both in the size of the set and in the speed of reconstruction. The crux is in the improvmeent to edge reconstruction. While the method of [2] uses a linear scan of all possible edges for each vertex, resulting in a quadratic number of diagrams needed, here we show that we can detect edges with $\Theta(m\log n)$ diagrams using a radial binary multi-search.

2 Background and Tools

In this section, we provide definitions of the tools used in the remainder of the paper. We make use of standard notation such as using e_i for the ith standard basis vector in $\mathbb{R}^d$, where $1 \le i \le d$. We use the notation (V, E) for a graph and its vertex and edge sets, and use $n = |V|$ and $m = |E|$. We assume the reader is familar with

*School of Computing and Department of Mathematical Sciences, Montana State University, brittany.fasy@montana.edu

†Mathematics & Computer Science Department, Western Colorado University, smicka@western.edu

‡School of Computing, Montana State University, david.millman@montana.edu

§Department of Mathematical Sciences, Montana State University, annaschenfisch@montana.edu

¶School of Computing, Montana State University, luciawilliams@montana.edu

standard concepts in topology (such as simplicial complexes, simplicial homology, Betti numbers), and note that further information can be found in [6] for a general introduction and in [7, Section 2.1] for a detailed definition of the augmented persistence diagram.

First, we define our general postition assumption.

Assumption 1 *Let $V \subset \mathbb{R}^d$ be a finite set with $d \geq 2$. We say V is in* general position *if the following properties are satisfied:*

(i) Every set of $d+1$ points is affinely independent.

(ii) No three points are colinear after orthogonal projection into the space $\pi(\mathbb{R}^d)$, where $\pi : \mathbb{R}^d \to \mathbb{R}^2$ is the orthogonal projection onto the plane spanned by the first two basis elements, e_1 and e_2.

(iii) Every point has a unique height with respect to the direction e_2.

We call a graph GP-immersed[1] *iff its vertex set is in general position in $\mathbb{R}^d$.*

We note that (iii) is not strictly necessary, however, it is convenient for simplicity of exposition. How to handle this degeneracy is discussed in Appendix B.

Given a graph GP-immersed in $\mathbb{R}^d$, we can filter the graph based on the height in any direction s in the sphere of directions $\mathbb{S}^{d-1}$. To do so, we assign each simplex a height. A vertex $v \in V$ is assigned the height $s \cdot v$, and an edge $[v_0, v_1] \in E$ is assigned the height $\max\{s \cdot v_0, s \cdot v_1\}$. This function, mapping vertices and edges to heights, is known as a *filter* function, which we use to compute persistent homology.

2.1 Persistence and the Oracle Framework

Given a filtered simplicial complex (that is, a simplicial complex with each simplex assigned a "height"), the corresponding *augmented persistence diagram (APD)* is a record of of all homological events throughout the filtration. A *birth event* is the introduction or appearance of a new homological feature, and a *death event* is the merging of two features. A death is paired with the most recent of the birth-labeled features that it merges together, creating a birth-death pairing, leaving the remaining feature labeled by the elder birth height. In an APD, every simplex corresponds to exactly one event (resulting in some pairings where the birth and death heights are equal). As a result, by construction, APDs contain at least one event at the height of each vertex.

For a simplicial complex (e.g., a graph) GP-immersed in $\mathbb{R}^d$ and a direction in S^{d-1}, we use the *lower-star filtration*: the nested sequence of graphs that arise by looking at all simplices at or below a given height and allowing that height to grow from $-\infty$ to ∞. Throughout this paper, we denote the i-dimensional APD by $\widehat{\mathcal{D}}_i(s)$ and write $\widehat{\mathcal{D}}(s) = \sqcup_i \widehat{\mathcal{D}}_i(s)$, omitting the graph itself from the notation (as it is always clear from context).[2]

The *(augmented) persistence homology transform ((A)PHT)* is the set of (augmented) diagrams of lower-star filtrations in all possible directions, parameterized by the direction. That is, the set $X = \{(\widehat{\mathcal{D}}(s), s)\}_{s \in \mathbb{S}^{d-1}}$. A *faithful discretization* is a finite subset of X from which all other elements of X can be deduced (and, by [16], corresponds to a unique simplicial complex). The introduction of the (augmented) persistent homology transform has sparked related research in applications of shape comparison [3,9–11,14,15,17,18]. As such, finding a minimal faithful discretization is important for the applicability of the (A)PHT. In what follows, we will only consider APDs, and we may shorten notation and refer to an APD by the word *diagram*.

In this work, we assume an oracle framework. That is, we assume that we have no knowledge of the shape itself, but we have access to an oracle from which we can query directional diagrams.

Definition 2 (Oracle) *For a graph (V, E) GP-immersed in $\mathbb{R}^d$ and a direction $s \in \mathbb{S}^{d-1}$, the operation* `Oracle`(s) *returns the diagram $\widehat{\mathcal{D}}(s)$. We define $\Theta(\Pi)$ to be the time complexity of this oracle query and note that the space complexity of $\widehat{\mathcal{D}}(s)$ is $\Theta(n+m)$. We assume that the data structure returned by the oracle allows queries for specific birth or death values in $\Theta(\log n)$ time (for example, the we could have two arrays of persistences points, one sorted by birth values and one sorted by death values).*

2.2 Constructions and Data Structures

In this subsection, we introduce the edge arc object and other definitions useful for computing properties of immersed graphs. Throughout this paper, we project points in $\mathbb{R}^d$ to the (e_1, e_2)-plane. As a result, we use "above (below)" without stating with respect to which direction as shorthand for "above (below) with respect to the direction e_2." This direction is intentionally chosen (and used in our GP assumption), as it corresponds to our intuition of above (below) in the figures. When we measure an angle of a vector x, denoted $\measuredangle x$, we mean the angle that $\pi(x)$ makes with the positive e_1 axis.

Given a direction s and a vertex in a graph immersed in $\mathbb{R}^d$, we classify each edge (v, v') as either an "incoming" edge, when v' is below v with respect to s, or an "outgoing" edge, when v' is above v with respect to s. Note that all incoming edges have the same height as the vertex with respect to the e_2 direction.

[1] Here, we use *immersed* rather than *embedded* in order to allow intersections of edges. Note, however, that this can only happen when $d = 2$.

[2] When calculating diagrams, we count $\widehat{\mathcal{D}}(s)$ as one diagram, not multiple.

Definition 3 (Indegree) *Let (V, E) be a graph GP-immersed in $\mathbb{R}^d$. Let $v \in V$ and $s \in \mathbb{S}^{d-1}$. The* indegree of v in direction s, *denoted* $\texttt{Indeg}(v, s)$, *is the number of edges incident to v with height $s \cdot v$.*

The following lemma relates the number of edges at a given height to points in the APD.

Lemma 4 (Edge Count) *Let (V, E) be a graph and let $c \in \mathbb{R}$. Let $f\colon V \sqcup E \to \mathbb{R}$ be a filter function. Then, the edges in E with a function value of c are in one-to-one correspondence with the following multiset of points in $\widehat{\mathcal{D}}(f)$, the diagram corresponding to f:*

$$\begin{aligned}\big\{(b,d) \in \widehat{\mathcal{D}}_1(f) \text{ s.t. } b = c\big\} \\ \cup \big\{(b,d) \in \widehat{\mathcal{D}}_0(f) \text{ s.t. } d = c\big\}.\end{aligned} \tag{1}$$

In other words, each edge corresponds to either a birth of a one-dimensional homological feature or a death of a zero-dimensional feature in $\widehat{\mathcal{D}}(f)$. For more details and a generalized proof, see [7, Appendix A].

If f is a lower-star filtration in direction $s \in \mathbb{S}^{d-1}$, we note that whenever a vertex v has a unique height with respect to a direction s, the cardinality of the multiset above is exactly $\texttt{Indeg}(v, s)$.

Lemma 5 (Indegree Computation) *Let (V, E) be a graph GP-immersed in $\mathbb{R}^d$. Let $v \in V$ and let $s \in \mathbb{S}^{d-1}$ such that $s \cdot v \neq s \cdot v'$ for any $v' \neq v \in V$. Then, $\texttt{Indeg}(v, s)$ can be computed via the oracle using one diagram and $\Theta(\log n + \Pi)$ time.*

Proof. Let $\widehat{\mathcal{D}} = \texttt{Oracle}(s)$. By the assumption on s, the height of v with respect to the direction s is unique. Hence, we know that any edge at height $c = s \cdot v$ must be incident to v. Thus, by the definition of indegree, an edge has the height c if and only if it contributes to the indegree of v in direction s. Using Lemma 4, we count these edges by counting one-dimensional births and zero-dimensional deaths at height c. Since $\widehat{\mathcal{D}}_0$ and $\widehat{\mathcal{D}}_1$ are sorted by both birth and death values (see Definition 2) and since $\widehat{\mathcal{D}}$ has $\Theta(n+m)$ points, searching for these events takes $\Theta(\log n + \log m)$. Adding $\Theta(\Pi)$ for the oracle query and recalling that $m = O(n)$, the total runtime is $\Theta(\log n + \Pi)$. □

We conclude this section by introducing a data structure, the *edge arc object*; see Table 1 for a summary of the attributes of an edge arc and Figure 1 for an example. An edge arc represents the region in the (e_1, e_2)-plane centered at v that is swept out between the two angles α_1 and α_2 (the word 'arc' is referring to the arc of angles between α_1 and α_2, where the angle is measured with respect to the postive e_1 axis). We only consider edge arcs in the upper half-space, with respect to the e_2 direction, so the maximal edge arc is the upper half-plane and the start and stop angles always satisfy $0 \leq \alpha_1 \leq \alpha_2 \leq \pi$. An edge arc stores an array

Table 1: Attributes of the edge arc object.

EA	**Edge Arc**
v	Vertex around which the edge arc is centered
(α_1, α_2)	Start and stop angles of the arc, with respect to the e_1 direction
verts	Array of vertices in arc radially ordered clockwise in (e_1, e_2)-plane
count	Number of edges incident to v within the arc

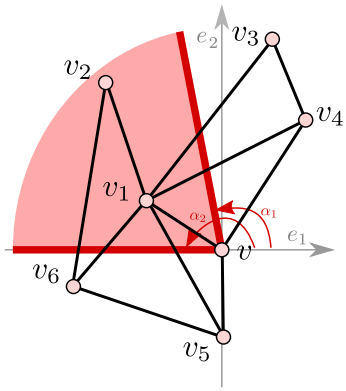


Figure 1: An edge arc EA centered at vertex $EA.v = v$. Other attributes of the edge arc include its start and stop angles, $EA.\alpha_1 = 1.75$ radians and $EA.\alpha_2 = \pi \approx 3.14$ radians, the array of vertices $EA.verts = \{v_1, v_2\}$, and the count of edges $EA.count = 1$. Here, we also see that $\texttt{Indeg}(v, e_1) = 1$ and $\texttt{Indeg}(v, e_2) = 2$.

of vertices sorted radially clockwise about $\pi(v)$ in the (e_1, e_2)-plane in decreasing angle with the e_1-direction. By construction, the first vertex in the array must be closest to α_2 and the last closest to α_1. The edge arc also stores the count of edges of E that have vertices from *verts* as endpoints. In implementation, the angles α_1 and α_2 do not need to be stored directly, but we include them in psuedocode and discussions for clarity.

Given some arc EA centered at vertex $v \in V$, we need to be able to compute $EA.count$, the number of edges contained EA that are adjacent to v. The following lemma provides such a computation. We omit a proof because it is a straightforward adaptation of [7, Theorem 16] and [2, Lemma 13].

Lemma 6 (Arc Count) *Let (V, E) be a graph GP-immersed in $\mathbb{R}^d$. Let EA be an edge arc object, and let $v = EA.v$. Let $s \in \mathbb{S}^{d-1}$ be the direction perpendicular to α_2 so that the arc is entirely below $s \cdot v$. Let E_* denote the edges with height $s \cdot v$ that are not contained in EA. If no vertex in V is at the same height as v in direction s, then*

$$EA.count = \texttt{Indeg}(v, s) - |E_*|.$$

As an illustration, again, consider Figure 1. Consider the direction $s = e^{i(\alpha_2 - \pi/2)}$, which is perpendicular to $e^{i\alpha}$ by construction. In addition, the edge arc is below $s \cdot v$ (specifically, all vertices in $EA.verts$ are below $s \cdot v$). Then, $\texttt{Indeg}(v, s) = 2$ and $E_* = \{[v, v_5]\}$. By Lemma 6, $EA.count = \texttt{Indeg}(v, s) - |E_*| = 2 - 1 = 1$. When we say that a list of vertices or edges is sorted clockwise around v, we mean that the list is sorted clockwise (cw) around $\pi(v)$ once projected into the (e_1, e_2)-plane with the largest angle first.

3 Fast Reconstruction

In this section, we provide an algorithm to reconstruct a graph (and, more generally, a one-skeleton of a simplicial complex) using the oracle. We start with an algorithm to find the edges, provided the vertex locations are known. We end with describing the complete graph reconstruction method.

3.1 Fast Edge Reconstruction

In this subsection, we assume we have a graph (V, E), where the vertex set V is known, but E is unknown. Using the oracle and the known vertex locations, we provide a reconstruction algorithm to find all edges in E (Algorithm 3). This algorithm is a sweepline algorithm in direction e_2 that, for each vertex processed in the sweep, performs a radial binary multi-search (Algorithm 2). This search is enabled by an algorithm that splits an edge arc object into two edge arcs, each containing half of the vertices (Algorithm 1). We provide the algorithms and relevant theorem statements here, but defer the proofs to Appendix A.

Algorithm 1 $\texttt{SplitArc}(EA, bigedges, \theta)$

Input: EA, an edge arc; $bigedges$, an array of all edges $(EA.v, v') \in E$ such that $\measuredangle\pi(v' - EA.v) < EA.\alpha_1$; θ, the minimum angle defined by any three vertices in $\pi(V)$

Output: EA_ℓ and EA_r, edge arcs satisfying the properties in Theorem 7

1: $n_v \leftarrow |EA.verts|$
2: $mid \leftarrow \lceil \frac{n_v}{2} \rceil$
3: $\alpha \leftarrow \measuredangle\pi(EA.verts[mid] - EA.v) - \theta/2$
4: $s \leftarrow e^{i(\alpha - \frac{\pi}{2})}$
5: $m_\ell \leftarrow \texttt{Indeg}(EA.v, s) - |\{b \in bigedges \mid \measuredangle b < \pi + \alpha\}|$
6: $m_r \leftarrow EA.count - m_\ell$
7: $EA_\ell \leftarrow$ edge arc where $EA_\ell.v = EA.v$, $EA_\ell.\alpha_1 = EA.\alpha_1$, $EA_\ell.\alpha_2 = \alpha$, $EA_\ell.verts = EA.verts[: mid]$, and $EA_\ell.count = m_\ell$
8: $EA_r \leftarrow$ edge arc where $EA_r.v = EA.v$, $EA_r.\alpha_1 = \alpha$, $EA_r.\alpha_2 = EA.\alpha_2$, $EA_r.verts = EA.verts[mid + 1 :]$, and $EA_r.count = m_r$
9: **return** (EA_ℓ, EA_r)

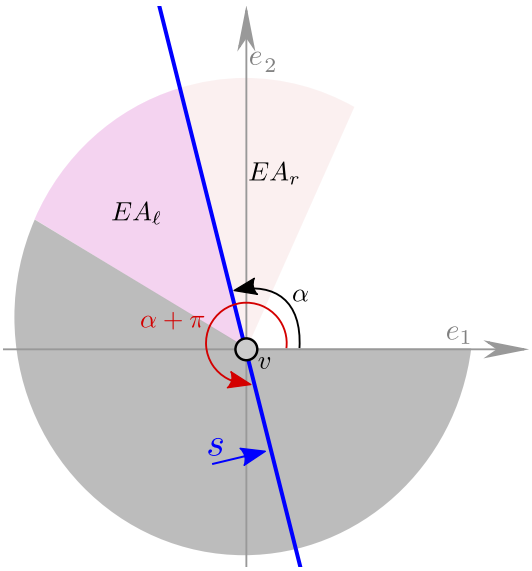


Figure 2: The splitting of edge arc EA into EA_ℓ and EA_r, as in Algorithm 1. The large gray region is the region containing all edges of $bigedges$. That is, all edges whose angle with the positive e_1-axis is at least $EA.\alpha_1$. On Line 5 of the algorithm, we compute the number of edges in EA_ℓ by first computing the indegree of $EA.v$ in direction s from the diagram in direction s, then we subtract the number of edges in $bigedges$ that are below the height of $EA.v$ in direction s (i.e., below the blue line). By the pigeonhole principal, we find $EA_r.count = EA.count - EA_\ell.count$.

In Algorithm 1, we find a direction s in the (e_1, e_2)-plane such that half of the vertices in $EA.verts$ are above v and half are below v with respect to the direction s. This allows us to create a new edge arcs corresponding to each half; see Figure 2. The properties of Algorithm 1 are described in the following theorem.

Theorem 7 (Arc Splitting) *Algorithm 1 uses one diagram and $\Theta(\log n + d + \Pi)$ time to split EA into two new edge arcs EA_ℓ and EA_r with the properties:*

(i) The sets $EA_r.verts$ and $EA_\ell.verts$ partition the vertex set $EA.verts$ such that the vertices in $EA_\ell.verts$ come before those in $EA_r.verts$, with respect to the clockwise ordering around $EA.v$.

(ii) $|EA_\ell.verts| = \lceil \frac{1}{2}|EA.verts| \rceil$.

(iii) $|EA_r.verts| = \lfloor \frac{1}{2}|EA.verts| \rfloor$.

In Algorithm 2, we use Algorithm 1 to find all outgoing edges from a given vertex. In particular, the algorithm maintains a stack of edge arc objects. When processing an edge arc (the while loop in Lines 4–16), we are determining which of the vertices in $verts$ form edges with v. If an edge arc has $count = 0$, it contains

Algorithm 2 `UpEdges`$(v, V_v, in_v, \theta, \widehat{\mathcal{D}})$

Input: $v \in V$; V_v, array of all vertices in V above v, ordered clockwise; in_v, array of all incoming edges of v, sorted radially clockwise; θ, the minimum angle formed by any three vertices in $\pi(V)$; and $\widehat{\mathcal{D}}$, the APD in direction e_2
Output: array of all outgoing edges of v

1: *indeg* ← indegree of v in direction $-e_2$.
2: *eastack* ← a stack of edge arc objects, initialized with a single edge arc A, where $A.v = v$, $A.\alpha_1 = 0$, $A.\alpha_2 = \pi$, $A.count = indeg$, and $A.verts = V_v$
3: $E_v \leftarrow \emptyset$
4: **while** *eastack* is not empty **do**
5: $EA \leftarrow eastack.pop()$
6: **if** $EA.count = 0$ **then**
7: **Continue** to top of while loop
8: **end if**
9: **if** $|EA.count| = |EA.verts|$ **then**
10: Append $v \times EA.verts$ to E_v, in order
11: **Continue** to top of while loop
12: **end if**
13: $(EA_\ell, EA_r) \leftarrow$ `SplitArc`$(EA, in_v \cup E_v, \theta)$
14: Push EA_r onto *eastack*
15: Push EA_ℓ onto *eastack*
16: **end while**
17: **return** E_v

no edges, and it can be ignored (Lines 6–8). If it has *count* exactly equal to the number of vertices in *verts*, each vertex in *verts* must form an edge with v (Lines 9–12). Otherwise, as demonstrated in Figure 3, the edge arc is split in half using Algorithm 1 and each half is put on the stack to be processed.

Theorem 8 (Finding Edges Above a Vertex) *Algorithm 2 finds the sorted array of edges above v using $\Theta(\deg(v)\log n)$ augmented persistence diagrams in $\Theta((\deg(v)\log n)(\log n + d + \Pi))$ time.*

Finally, our main algorithm (Algorithm 3) is a sweepline algorithm, where we consider the vertices in increasing order of their e_2-coordinates and find the outgoing edges of the vertex being considered.

Theorem 9 (Edge Reconstruction) *Let (V, E) be a graph GP-immersed in $\mathbb{R}^d$. Given V, Algorithm 3 reconstructs E using $\Theta(m \log n)$ augmented persistence diagrams in $\Theta(n^2 + m\log n(\log n + d + \Pi))$ time.*

3.2 Putting it Together: Full Reconstruction

The results of Section 3.1 are related to just part of the full process of reconstruction, since reconstruction begins with no knowledge of the underlying simplicial complex. Identifying the location of all vertices is the first step, and is one that has been previously examined in detail. In particular, Belton et al. provide an algorithm to reconstruct V in $\Theta(dn^{d+1} + d\Pi)$ time and $d+1$ oracle queries; see [2, Algorithm 1 & Theorem 9]. Together with Theorem 9, we obtain the following runtime and diagram count for a full reconstruction process.

Algorithm 3 `FindEdges`(V)

Input: V, array of all vertices in the unknown graph
Output: E, array of all edges in the unknown graph

1: $\widehat{\mathcal{D}} \leftarrow$ `Oracle`$(-e_2)$
2: $E \leftarrow \{\}$
3: *vertsabove* ← for each $v \in V$, an array clockwise ordering all vertices in V that are above v
4: $\theta \leftarrow$ min angle defined by any three vertices of $\pi(V)$
5: **for** v in V, in increasing height in direction e_2 **do**
6: *inedges* ← clockwise sorted array of edges in E incident to v
7: $E+=$ `UpEdges`$(v, vertsabove[v], inedges, \theta, \widehat{\mathcal{D}})$
8: **end for**
9: **return** E

Theorem 10 (Graph Reconstruction) *Using an oracle, we can reconstruct an unknown graph immersed in $\mathbb{R}^d$ using $\Theta(d + m\log n)$ diagrams in $\Theta(dn^{d+1} + d\Pi + n^2 + m\log n(\log n + d + \Pi))$ time.*

We omit a proof of Theorem 10, as it simply combines the results of [2, Theorem 9] and Theorem 9 of the current paper. Observing that the methods presented here are immediately applicable in the reconstruction of one-skeletons of general simplicial complexes, we have the following corollary:

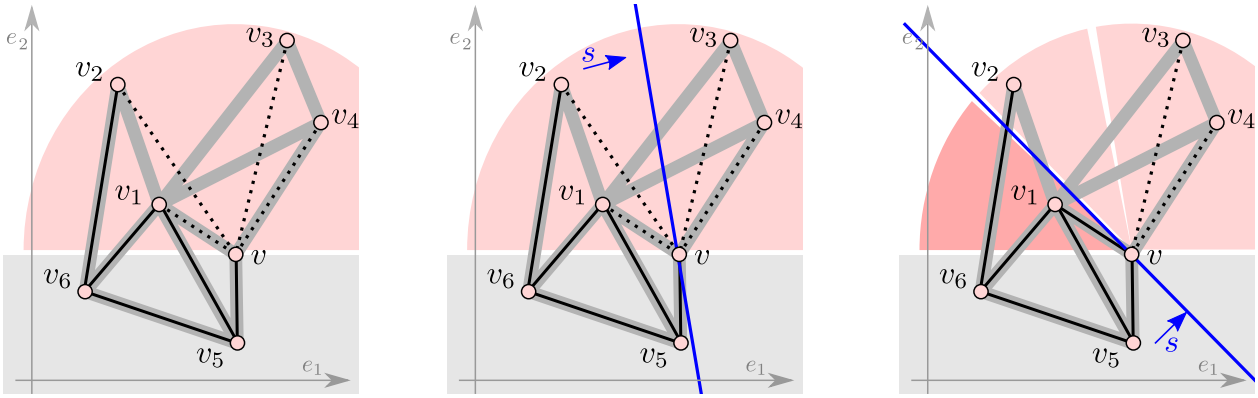
Corollary 11 (One-Skeleton Reconstruction) *Let K be an unknown simplicial complex GP-immersed in $\mathbb{R}^d$. Algorithm 1 of [2] and Algorithm 3 of the current paper reconstruct the one-skeleton of K using $\Theta(d + m\log n)$ augmented persistence diagrams in $\Theta(dn^{d+1} + d\Pi + n^2 + m\log n(\log n + d + \Pi))$ time.*

Finally, we note that embedding a graph (or simplicial complex) in $\mathbb{R}^2$ is a special case, as $m = O(n)$ and d is constant. In addition, by [2, Theorem 6], vertex reconstruction of a graph embedded in $\mathbb{R}^2$ can be done with three diagrams and $\Theta(n\log n + \Pi)$ time. Hence, we obtain a result for plane graph reconstruction:

Corollary 12 (Reconstruction in $\mathbb{R}^2$) *We can use an oracle to reconstruct the one-skeleton of an unknown simplicial complex embedded in $\mathbb{R}^2$ using $O(n\log n)$ diagrams and $O(n^2 + n\Pi\log n)$ time.*

4 Discussion

One way of proving that a discretization of the APHT is faithful is through the method of *reconstructing* the



(a) At vertex v. (b) Split edge arc. (c) Split again.

Figure 3: We demonstrate one step of Algorithm 2. (a) By assumption, we initially know $[v_5, v] \in E$. From Line 1 of Algorithm 2 we also know that two of the four vertices above v are adjacent to v. Thus, we create an edge arc object EA with $EA.count = 2$, and $EA.verts = (v_1, v_2, v_3, v_4)$. (b) In Algorithm 1, we choose a direction s such that half of the vertices in EA are below v. We use this split to create two edge arcs, EA_r and EA_ℓ, corresponding to the pink shaded regions on the right and left of the blue line defined by s. We push EA_r onto a stack to be processed later and focus on the arc EA_ℓ. Since two edges contribute to v's indegree in direction s and one is the known edge $[v_5, v]$, we have $EA.count = 2-1 = 1$. (c) Next, we find a new direction s that splits $EA_\ell.verts$ into two sets of size one. We push the set above s onto our stack. The edge arc containing only v_1 also has $EA.count = 2 - 1 = 1$, so $[v_1, v] \in E$. After all steps of Algorithm 2 are applied to find the edges above a particular vertex, Algorithm 2 is then applied to the next highest vertex, eventually processing every vertex in V in a sweep (Algorithm 3).

underlying simplicial complex. That is, by showing that the underlying simplicial complex can be recovered with the data of the discretization alone. In this paper, we take that approach and provide an algorithm for reconstructing a graph immersed in $\mathbb{R}^d$. We use fewer persistence diagrams than presented in alternate approaches. For example, the algorithm that we present for edge reconstruction (when the vertex locations are known) uses $\Theta(m \log n)$ diagrams. In contrast, [2, Theorem 16] uses $n^2 - n$ diagrams. Note that, for a very dense edge set, that is, when $m = \Theta(n^2)$, the method in [2, Theorem 16] uses fewer diagrams. However, if $m = O(n)$, as is common in many complexes, the represntation computed in this paper has fewer diagrams. Moreover, we emphasize that the number of diagrams is not exponential in the ambient dimension.

One might hope to use binary search strategies to reconstruct a simplicial complexe, but the methods presented here are unique to one-skeletons. Radially ordering higher dimensional simplices is not well-defined, and this issue prevents the methods presented here from being immediately transferrable. On the other hand, with the representation in this paper being output-sensitive (as opposed to testing if every pair of vertices is a simplex), we have hope for the discretization of the (A)PHT of a simplicial complex immersed in $\mathbb{R}^d$ being proportional to the size of the complex itself.

We also observe that not all diagrams used in our reconstruction algorithms were strictly necessary (i.e., the set of diagrams used were not a minimal faithful set). One straightforward way to reduce the number of diagrams used without altering the method much would be to split the region above a vertex in the sweep into arcs that contain exactly the same number of edges as vertices, or no edges. This property can then be validated by a simple difference of indegrees. In ongoing work, we hope to make these claims precise. We also hope to extend our methods to use topological descriptors that are *not* dimension-returning (such as augmented Euler Characteristic curves).

References

[1] R. L. Belton, B. T. Fasy, R. Mertz, S. Micka, D. L. Millman, D. Salinas, A. Schenfisch, J. Schupbach, and L. Williams. Learning simplicial complexes from persistence diagrams. In *Canadian Conference on Computational Geometry*, August 2018. Also available at arXiv:1805.10716.

[2] R. L. Belton, B. T. Fasy, R. Mertz, S. Micka, D. L. Millman, D. Salinas, A. Schenfisch, J. Schupbach, and L. Williams. Reconstructing embedded graphs from persistence diagrams. *Computational Geometry: Theory and Applications*, 2020.

[3] P. Bendich, J. S. Marron, E. Miller, A. Pieloch, and S. Skwerer. Persistent homology analysis of brain artery trees. *The Annals of Applied Statistics*, 10(1):198, 2016.

[4] L. M. Betthauser. *Topological Reconstruction of Grayscale Images*. PhD thesis, University of Florida, 2018.

[5] J. Curry, S. Mukherjee, and K. Turner. How many directions determine a shape and other sufficiency results for two topological transforms. arXiv:1805.09782, 2018.

[6] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. American Mathematical Society, 2010.

[7] B. T. Fasy, S. Micka, D. L. Millman, A. Schenfisch, and L. Williams. A faithful discretization of the augmented persistent homology transform. 2022. arXiv:1912.12759.

[8] R. Ghrist, R. Levanger, and H. Mai. Persistent homology and Euler integral transforms. *Journal of Applied and Computational Topology*, 2(1-2):55–60, 2018.

[9] C. Giusti, E. Pastalkova, C. Curto, and V. Itskov. Clique topology reveals intrinsic geometric structure in neural correlations. *Proceedings of the National Academy of Sciences*, 112(44):13455–13460, 2015.

[10] P. Lawson, A. B. Sholl, J. Q. Brown, B. T. Fasy, and C. Wenk. Persistent homology for the quantitative evaluation of architectural features in prostate cancer histology. *Scientific Reports*, 9, 2019.

[11] Y. Lee, S. D. Barthel, P. Dłotko, S. M. Moosavi, K. Hess, and B. Smit. Quantifying similarity of pore-geometry in nanoporous materials. *Nature Communications*, 8:15396, 2017.

[12] S. A. Micka. *Searching and Reconstruction: Algorithms with Topological Descriptors*. PhD thesis, Montana State University, 2020.

[13] D. L. Millman and V. Verma. A slow algorithm for computing the Gabriel graph with double precision. *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry*, 2011.

[14] A. H. Rizvi, P. G. Camara, E. K. Kandror, T. J. Roberts, I. Schieren, T. Maniatis, and R. Rabadan. Single-cell topological RNA-seq analysis reveals insights into cellular differentiation and development. *Nature Biotechnology*, 35(6):551, 2017.

[15] G. Singh, F. Mémoli, and G. E. Carlsson. Topological methods for the analysis of high dimensional data sets and 3d object recognition. *SPBG*, 91:100, 2007.

[16] K. Turner, S. Mukherjee, and D. M. Boyer. Persistent homology transform for modeling shapes and surfaces. *Information and Inference: A Journal of the IMA*, 3(4):310–344, 2014.

[17] S. Tymochko, E. Munch, J. Dunion, K. Corbosiero, and R. Torn. Using persistent homology to quantify a diurnal cycle in hurricanes. *Pattern Recognition Letters*, 2020.

[18] Y. Wang, H. Ombao, and M. K. Chung. Statistical persistent homology of brain signals. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1125–1129. IEEE, 2019.

A Algorithmic Proofs

In this appendix, we provide the proofs omitted from Section 3. These proofs provide justification for the runtimes, diagram compelxity, and correctness of the algorithms presented in this paper.

A.1 Proof of Theorem 7

Theorem 7 (Arc Splitting) *Algorithm 1 uses one diagram and $\Theta(\log n + d + \Pi)$ time to split EA into two new edge arcs EA_ℓ and EA_r with the properties:*

(i) The sets $EA_r.verts$ and $EA_\ell.verts$ partition the vertex set $EA.verts$ such that the vertices in $EA_\ell.verts$ come before those in $EA_r.verts$, with respect to the clockwise ordering around $EA.v$.

(ii) $|EA_\ell.verts| = \lceil \frac{1}{2}|EA.verts| \rceil$.

(iii) $|EA_r.verts| = \lfloor \frac{1}{2}|EA.verts| \rfloor$.

Proof. For the runtime, we walk through the algorithm and analyze the time and diagram complexity of each line. In Lines 1–3, we find the angle α that splits $EA.verts$ into two equal sets, then in Line 4 compute a direction s orthogonal to α. See Figure 2. Lines 1–4 use no diagrams and can be done in constant time when restricting our attention to the (e_1, e_2)-plane. However, we need s to be a direction in $\mathbb{R}^d$ (as opposed to only in the (e_1, e_2)-plane), so the computation takes $\Theta(d)$ time.[3] Specifically, s is the vector

$$s = e^{\frac{1}{2}i(2\alpha - \pi - \theta)} \tag{2}$$
$$= \left(\cos\left(\alpha - \frac{1}{2}\pi - \frac{1}{2}\theta\right), \sin\left(\alpha - \frac{1}{2}\pi - \frac{1}{2}\theta\right), 0, 0, \ldots, 0\right).$$

To compute m_ℓ in Line 5, we compute $\texttt{Indeg}(v, s)$ then subtract the cardinality of the set $S := \{b \in \textit{bigedges} \mid \measuredangle b < \pi + \alpha\}$, where $\measuredangle b$ is taken to mean the angle b makes with the e_1-axis, when viewed as a vector with $EA.v$ as the origin. By Lemma 5, we compute $\texttt{Indeg}(v, s)$ via the oracle using one diagram and $\Theta(\log n + \Pi)$ time. Since *bigedges* is sorted and since s lies in the (e_1, e_2)-plane, we can find the set S in $\Theta(\log(|\textit{bigedges}|))$ time. The subtraction in Line 5 takes constant time, as does Line 6.

In Lines 7 and 8, we create two edge arc objects. The time complexity of creating them is proportional to the size of the obejcts themselves. All attributes of edge arc objects, except the array of vertices (*verts*), are constant size. By construction, $EA_\ell.verts$ and $EA_r.verts$ split $EA.verts$ into two sets, which can be done naïvely in $\Theta(d|EA.verts|)$ time by walking through $EA.verts$ and storing each one explicitly. However, we improve this to $\Theta(\log |EA.verts|)$ time if we have a globally accessible array of vertices (sorted cw around v) and just computes the pointers to the beginning and end of the subarrays corresponding to the *verts* attributes of the new edge arc objects. In total, Algorithm 1 and takes $\Theta(d + \log n +$

[3] With some clever data structures, this $\Theta(d)$ can be reduced to constant time. For example, we could require vectors in $\mathbb{R}^2$ are automatically padded with 0's to become vectors in $\mathbb{R}^d$ when needed. However, this is out of the scope of the real RAM model of computation.

$\Pi + \log(|bigedges|) + 1 + \log(|EA.verts|)) = \Theta(\log n + d + \Pi)$ time and uses uses one diagram.

Now that we have walked through the algorithm and established the runtime and diagram complexity, we prove correctness. To do so, we first show that EA_ℓ and EA_r are edge arc objects. In particular, this means showing that they have the correct values for *count* and *verts*. We prove this for EA_ℓ; the proof for EA_r follows a similar argument.

EA_ℓ.count: We must show that $EA_\ell.count$ is the number of edges in EA_ℓ incident to $EA_\ell.v$. By Lemma 4, the value returned from $\texttt{Indeg}(EA.v, s)$ counts all edges incident to $EA.v$ and below $s \cdot EA.v$ in direction s. By Lemma 6, this is exactly the total number of edges in EA_ℓ plus edges $(EA.v, v') \in E_v$ for which $s \cdot v' < s \cdot EA.v$. Thus, by subtracting $|\{b \in bigedges \mid \measuredangle b < \pi + \alpha\}|$ from $\texttt{Indeg}(EA.v, s)$ on Line 5, we are left with m_ℓ, the number of edges incident to $EA.v$ contained in EA_ℓ. Setting $EA_\ell.count = m_\ell$ on Line 7, we see that $EA_\ell.count$ is correct.

EA_ℓ.verts: We must show that $EA_\ell.verts$ contains an array of all verices contained in EA_ℓ radially ordered clockwise. This follows from the fact that $EA.verts$ is all vertices contained in EA ordered clockwise, so when we restrict $EA.verts$ to $EA.verts[: mid]$ on Line 7, we are eliminating vertices not contained in EA_ℓ, so $EA_\ell.verts$ is correct.

Next, we prove Statement (i). Recall that $EA.verts$ orders the vertices in decreasing angle with e_1. In Line 3, $\measuredangle\pi(EA.verts[mid] - EA.v)$ is the angle made by $EA.v$ with the middle vertex. We tilt this angle by $\theta/2$ on Line 3 to obtain angle α. By construction of α,

$$\measuredangle\pi(EA.verts[mid] - EA.v) > \alpha.$$

By definition of θ, the angle α satisfies:

$$\alpha > \measuredangle\pi(EA.verts[mid + 1] - EA.v).$$

Since the array $EA.verts$ is sorted, all vectors in the set $\pi(EA.verts[: mid] - EA.v)$ have an angle of at least α with e_1 and all vectors in $\pi(EA.verts[: mid] - EA.v)$ have an angle of at most α.

By Lines 1–2 and Line 5, we know that $EA.verts$ contains the first $m = \lceil \frac{1}{2}|EA.verts| \rceil$ vertices in $EA.verts$. Hence, Statement (ii) holds. Statement (iii) follows from Statements (i) and (ii). □

A.2 Proof of Theorem 8

Theorem 8 (Finding Edges Above a Vertex) *Algorithm 2 finds the sorted array of edges above v using $\Theta(\deg(v) \log n)$ augmented persistence diagrams in $\Theta((\deg(v) \log n)(\log n + d + \Pi))$ time.*

Proof. First, we analyze the time complexity of the algorithm and the number of diagrams it requires. By Lemma 5, Line 1 can be computed in $\theta(\log n)$ time (since we are given the diagram and do not need an additional oracle query). Storing $A.verts$ by storing a pointer to V_v, we initialize *eastack* and E_v in Lines 2 and 3 in constant time.

To analyze the complexity of the loop in Lines 4–16, we first note that this is a radial binary multi-search. When processing an edge arc, we decide whether all edges have been found or if we need to split the edge arc. If there is only one edge in the arc (i.e., $EA.count = 1$), then this loop is a binary search for an edge, using the angle with e_1 in the (e_1, e_2)-plane as the search key. When $EA.count > 1$, we search for all edges, finding them in decreasing angle order (since arcs with larger angles are added after arcs of smaller angles). The if statement in Lines 9–12 is where the edges are added to E_v. Note that this shortcuts additional edge arc splitting by stopping the process once we find that the number of edges in the arc is equal to the number of potential vertices that can form the edges. As a result, each edge above v contributes to $O(\log n)$ edge arcs being added to *eastack* and, in the case that every other vertex is incident to an edge with v, we have $\Theta(\log n)$ edge arcs added to the stack. All operations in the while loop are constant time, except splitting the edge arc object in Line 13, which uses one diagram and takes $\Theta(\log n + d + \Pi)$ time.

The complexity of Algorithm 2 is dominated by the complexity of the while loop: the algorithm uses $\Theta(\deg(v) \log n)$ augmented persistence diagrams and takes takes $\Theta((\deg(v) \log n)(\log n + d + \Pi))$ time.

To prove correctness of this algorithm, we state the loop invariant for the while loop:

(i) For $(v, v') \in E$:

- If $\measuredangle(v' - v) > E_v.\alpha_1$, then (v, v') is either in E_v or in_v.
- If $\measuredangle(v' - v) > E_v.\alpha_1$, then v' is in *verts* for some edge arc in *eastack*
- $\measuredangle(v' - v) \neq E_v.\alpha_1$

(ii) The edge arc stack is clockwise-ordered.

This loop invariant ensures that the call to Algorithm 1 in Line 13 has valid input and that all outgoing edges are found when the algorithm terminates. □

A.3 Proof of Theorem 9

Theorem 9 (Edge Reconstruction) *Let (V, E) be a graph GP-immersed in $\mathbb{R}^d$. Given V, Algorithm 3 reconstructs E using $\Theta(m \log n)$ augmented persistence diagrams in $\Theta(n^2 + m \log n(\log n + d + \Pi))$ time.*

Proof. We first analyze the runtime and diagram count for Algorithm 3 by walking through the algorithm line-by-line. In Line 1, we ask the oracle for the diagram in direction $-e_2$, which takes $\Theta(\Pi)$ time. In [2, Theorem 14 (Edge Reconstruction)], simultaneously find the cyclic ordering around all vertices in $\Theta(n^2)$ time by Lemmas 1 and 2 of [13]. In Line 3, we do that as well; however, we do not store vertices that are above v in the array $vertsabove[v]$, and thus this line takes $\Theta(n^2)$ time. We note that such a cyclic ordering exists around each vertex by Assumption 1projectedindep. Once we have *vertsabove*, to find the minimum angle defined by any three vertices of V, we check all angles between vectors $vertsabove[v][i] - v$ and $vertsabove[v][i+1] - v$ in Line 4 in $\Theta(n + m)$ time.

The for loop in Lines 5–8 is repeated n times, once for each vertex in V. To determine the order of processing the vertices in V, we follow the births in $\widehat{\mathcal{D}}_0$, in decreasing order (since $\widehat{\mathcal{D}}_0$ is the lower-star filtration in direction $-e_2$).

Thus, finding the order takes $\Theta(n)$ time. In each iteration, we compute the incoming edges (those whose other vertex is below v) in Line 6 followed by all outgoing edges (those whose other vertex is above v) in Line 7. By Assumption 1(iii), every edge is either incoming or outgoing with respect to direction e_2. Thus, all edges incident to v are in E once E is updated in Line 6. By Theorem 8, when processing vertex v, the call to Algorithm 2 on Line 7 takes $\Theta((\deg(v)\log n)(\log n + d + \Pi))$ time and uses $\Theta(\deg(v)\log n)$ diagrams. Summing over all vertices, we see that the loop in Lines 5–8 takes

$$\sum_{v\in V} \Theta((\deg(v)\log n)(\log n + d + \Pi)) = \Theta(m \log n(\log n + d + \Pi))$$

time and uses $\sum_{v\in V} \Theta(\deg(v)\log n) = \Theta(m\log n)$ augmented persistence diagrams.

In total, Algorithm 3 takes $\Theta(\Pi) + \Theta(n^2) + \Theta(n+m) + \Theta(n) + \Theta(m\log n(\log n + d + \Pi)) = \Theta(n^2 + m\log n(\log n + d + \Pi))$ time and uses $\Theta(1) + \Theta(m\log n) = \Theta(m\log n)$ diagrams.

Next, we prove the correctness of Algorithm 3 (i.e., that all edges are found). In order to process vertices in order of their heights in the e_2 direction, we first sort them in Line 5. For $1 \le j \le n$, let v_j be the j^{th} vertex in this ordering. To show that Algorithm 3 finds all edges in E, we consider the loop invariant (LI): when we process v_j, all edges with maximum vertex height equal or less than the height of v_j are known. The LI is trivially true for v_1. We now assume that it is true for iteration j, and show that it must be true for iteration $j+1$. By assumption, all edges (v_i, v_j) with $1 \le i < j$ are known, and so by Theorem 8, Algorithm 2 finds all edges (v_k, v_j), where $k > j$, and we add them to the edge set E. Note that, by assumption, all edges (v_x, v_i) for $1 \le i \le j$ are also already known, and so the invariant is maintained. Thus, after the loop terminates, all edges are found. □

B Basis

In Assumption 1(iii), we assume all vertices of the underlying graph are unique with respect to the first basis direction e_2. In this appendix, we provide details of how to *find* a basis where all vertices have a unique height with respect to the second basis direction. I.e., this appendix allows us to remove one general position assumption by showing it can be satisfied deterministically, at an added cost of $\Theta(|P|\log|P| + d + \Pi)$ time.

Lemma 13 (Creation of Orthonormal Basis) *Given a point set $P \subset \mathbb{R}^d$ satisfying Assumption 1(i) and Assumption 1(ii), we can use two diagrams and $\Theta(|P|\log|P| + d + \Pi)$ time to create the orthonormal basis $\{b_1, b_2, e_3, e_4, \ldots, e_d\}$ so that all points of P have a unique height in direction b_2.*

Proof. Algorithm 6 (`Tilt`) of [7] takes diagrams from two linearly independent directions $s, s' \in \mathbb{S}^{d-1}$, the point set P, and returns a direction s_* in $\Theta(|P|\log|P| + d + \Pi)$ time[4] so that the following properties holds for all $p_1, p_2 \in P$:

(i) If p_1 is strictly above (below) p_2 with respect to direction s, then p_1 is strictly above (below, respectively) p_2 with respect to direction s_*.

(ii) If p_1 and p_2 are at the same height with respect to direction s and p_1 is strictly above (below) p_2 with respect to direction s', then p_1 is strictly above (respectively, below) p_2 with respect to direction s_*.

(iii) If p_1 is is at the same height as p_2 with respect to both directions s and s', then p_1 and p_2 are at the same height with respect to direction s_*.

A proof of correctness is given in [7, Lemma 32 (Tilt)].

We start with the standard basis for $\mathbb{R}^d$, $\{e_1, e_2, \ldots, e_d\}$, and we replace the first two basis elements as follows. Let b_2 be the direction obtain by using `Tilt` with $s = e_1$, $s' = e_2$, and $P = P$.

By Assumption 1(ii), no three points of P are colinear when projected onto the first two coordinates. In particular, this means no two points share the same heights in both the e_1 and e_2 directions. Then, by Statements (i)-(ii) above, the direction b_2 must order all vertices of P uniquely. Using only the first two coordinates of b_2 and e_1, we then perform Gram Schmidt orthanormalization to compute the first two coordinates of b_1. More precisely, letting $b_i^{(j)}$ denote the jth coordinate of b_i, we compute

$$\begin{pmatrix} b_1^{(1)} \\ b_1^{(2)} \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} - \frac{\left\langle \begin{pmatrix} b_2^{(1)} & b_2^{(2)} \end{pmatrix}^T, \begin{pmatrix} 1 & 0 \end{pmatrix}^T \right\rangle}{\left|\left| \begin{pmatrix} b_2^{(1)} & b_2^{(2)} \end{pmatrix}^T \right|\right|^2} \begin{pmatrix} b_2^{(1)} \\ b_2^{(2)} \end{pmatrix} \quad (3)$$

We then set $b_1^{(j)} = 0$ for $2 < j \le d$, so that $b_1 \in \text{span}\{e_1, e_2\}$, $b_2 \perp b_1$, and $||b_1|| = 1$. Only considering the first two coordinates of b_2 and e_1 means this process takes constant time. The remaining e_i for $2 \le i \le d$ can be used to fill the basis.

Finally, we have a basis satisfying all assumptions of Assumption 1, namely, $\{b_1, b_2, e_3, e_4, \ldots, e_d\}$. □

[4] While [7] does not account for diagram computation time, there are two diagrams used in this process, hence our addition of $\Theta(\Pi)$ to the total runtime.

On the Biplanar and k-Planar Crossing Numbers

Alireza Shavali* Hamid Zarrabi-Zadeh†

Abstract

The biplanar crossing number of a graph G is the minimum number of crossings over all possible drawings of the edges of G in two disjoint planes. We present new bounds on the biplanar crossing number of complete graphs and complete bipartite graphs. In particular, we prove that the biplanar crossing number of complete bipartite graphs can be approximated to within a factor better than 3, improving over the best previously known approximation factor of 4.03. For complete graphs, we prove an approximation factor of 3.17, improving the best previously known factor of 4.34. We provide similar improved bounds for the k-planar crossing number of complete graphs and complete bipartite graphs, for any positive integer k.

1 Introduction

An embedding (or drawing) of a graph G in the Euclidean plane is a mapping of the vertices of G to distinct points in the plane and a mapping of edges to smooth curves between their corresponding vertices. A planar embedding of a graph is a drawing of the graph in the plane such that edges intersect only at their endpoints. A graph admitting such a drawing is called planar. A *biplanar embedding* of a graph $G = (V, E)$ is a decomposition of the graph into two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ such that $E = E_1 \cup E_2$ and $E_1 \cap E_2 = \emptyset$, together with planar embeddings of G_1 and G_2. In this case, we call G biplanar. Biplanar embeddings are central to the computation of thickness of graphs [13], with applications to VLSI design [14]. It is well-known that planarity can be recognized in linear time, while biplanarity testing is NP-complete [12].

Let $cr(G)$ be the minimum number of edge crossings over all drawings of G in the plane, and let $cr_k(G)$ be the minimum of $cr(G_1) + \cdots + cr(G_k)$ over all possible decompositions of G into k subgraphs $G_1, \ldots, G_k$. We call $cr(G)$ the *crossing number* of G, and $cr_k(G)$ the *k-planar crossing number* of G. Throughout this paper, we only consider *simple drawings* for each subgraph G_i, in which no two edges intersect more than once, and no three edges intersect at a point (such drawings are sometimes called nice drawings). Moreover, we denote by n the number of vertices, and by m the number of edges of a graph.

Determining the crossing number of complete graphs and complete bipartite graphs has been the subject of extensive research over the past decades. In 1955, Zarankiewicz [20] conjectured that the crossing number $cr(K_{p,q})$ of the complete bipartite graph $K_{p,q}$ is equal to

$$Z(p,q) := \left\lfloor \frac{p}{2} \right\rfloor \left\lfloor \frac{p-1}{2} \right\rfloor \left\lfloor \frac{q}{2} \right\rfloor \left\lfloor \frac{q-1}{2} \right\rfloor .$$

He also established a drawing with that many crossings. In 1960, Guy [8] conjectured that the crossing number $cr(K_n)$ of the complete graph K_n is equal to

$$Z(n) := \frac{1}{4} \left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{n-1}{2} \right\rfloor \left\lfloor \frac{n-2}{2} \right\rfloor \left\lfloor \frac{n-3}{2} \right\rfloor .$$

Both conjectures have remained open after more than six decades. For the biplanar case, even formulating such conjectures seems to be hard. As noted in [4], techniques like embedding method and the bisection width method which are useful for bounding ordinary crossing numbers do not seem applicable to the biplanar case.

In 1971, Owens [14] described a biplanar embedding of K_n with almost $\frac{7}{24}Z(n)$ crossings. The construction was later improved by Durocher *et al.* [7], but the upper bound remained asymptotically the same. In 2006, Czabarka *et al.* [4] presented a biplanar embedding for $K_{p,q}$ with about $\frac{2}{9}Z(p,q)$ crossings. They also proved that $cr_2(K_n) \geq n^4/952$ and $cr_2(K_{p,q}) \geq p(p-1)q(q-1)/290$. Shahrokhi *et al.* [17] generalized these lower bounds to the k-planar case. Pach *et al.* [15] proved that for every graph G and any positive integer k, $cr_k(G) \leq \left(\frac{2}{k^2} - \frac{1}{k^3}\right) cr(G)$. This includes as a special case the inequality $cr_2(G) \leq \frac{3}{8}cr(G)$, originally proved by Czabarka *et al.* [5].

Our results. In this paper, we present several new bounds for approximating the biplanar and k-planar crossing number of complete graphs and complete bipartite graphs. Given a positive integer k and a real constant $\alpha \geq 1$, we say that $cr_k(K_n)$ is *approximated* to within a factor of α, if there is an upper bound $f(n)$ and a lower bound $g(n)$ on the value of $cr_k(K_n)$ such that $\lim_{n\to\infty} \frac{f(n)}{g(n)} \leq \alpha$. Here, α is called an *asymptotic approximation factor* for $cr_k(K_n)$. Similarly, we

*Department of Computer Engineering, Sharif University of Technology. Email: ashavali@ce.sharif.edu.

†Department of Computer Engineering, Sharif University of Technology. Email: zarrabi@sharif.edu.

say that $cr_k(K_{p,q})$ is approximated to within a factor of α, if there is an upper bound $f(p,q)$ and a lower bound $g(p,q)$ on the value of $cr_k(K_{p,q})$ such that $\lim_{p,q\to\infty} \frac{f(p,q)}{g(p,q)}$ exists and is no more than α. The results presented in this paper are summarized below.

- We prove that for all $p, q \geq 30$, $cr_2(K_{p,q}) \geq p(p-1)q(q-1)/213$. This significantly improves the best current lower bound of $cr_2(K_{p,q}) \geq p(p-1)q(q-1)/290$, due to Czabarka *et al.* [4]. Combined with the upper bound of $cr_2(K_{p,q}) \leq \frac{2}{9}Z(p,q)+o(p^2q^2)$[1] [4], our result implies an asymptotic approximation factor of 2.96 for $cr_2(K_{p,q})$, improving over the best previously known asymptotic factor of 4.03.

- For complete graphs, we show that $cr_2(K_n) \geq \frac{n^4}{694}$, improving the best current lower bound of $cr_2(K_n) \geq \frac{n^4}{952}$ [4]. Combined with the upper bound of $cr_2(K_n) \leq \frac{7}{24}Z(n) + o(n^4)$ due to Owens [14], we achieve an asymptotic approximation factor of 3.17 for $cr_2(K_n)$, improving the best previously known approximation factor of 4.34.

- We extend our lower bounds for the biplanar crossing number to the k-planar case, for any positive integer k. In particular, we show that for sufficiently large n, $cr_k(K_n) \geq n^4/(232k^2)$, improving the best current lower bound of $cr_k(K_n) \geq n^4/(432k^2)$, due to Shahrokhi *et al.* [17]. Considering the upper bound of $cr_k(K_n) \leq \frac{2}{k^2}Z(n)$ due to Pach *et al.* [15], we obtain an asymptotic approximation factor of 7.25 for $cr_k(K_n)$, improving the best current approximation factor of 13.5 available for $cr_k(K_n)$.

- Finally, we prove that for any positive integer k, $cr_k(K_{p,q}) \geq p(p-1)q(q-1)/(73.2k^2)$, improving the current lower bound of $cr_k(K_{p,q}) \geq p(p-1)q(q-1)/(108k^2)$ due to Shahrokhi *et al.* [17]. Combined with the upper bound of $cr_k(K_n) \leq \frac{2}{k^2}Z(p,q)$ [15], we obtain an asymptotic approximation factor of 9.15 for $cr_k(K_{p,q})$, improving the best current factor of 13.5.

A summary of the asymptotic approximation factors for the biplanar and k-planar crossing number of K_n and $K_{p,q}$ is presented in Table 1.

Table 1: Summary of asymptotic approximation factors for the biplanar and k-planar crossing numbers.

Crossing Number	Asymptotic Approx. Factor	Ref.
$cr_2(K_{p,q})$	4.03	[4]
	2.96	[This work]
$cr_2(K_n)$	4.34	[4, 14]
	3.17	[This work]
$cr_k(K_{p,q})$	13.5	[15, 17]
	9.15	[This work]
$cr_k(K_n)$	13.5	[15, 17]
	7.25	[This work]

2 Two Combinatorial Lemmas

We first present two combinatorial lemmas which are the main ingredients of our proofs. Our first lemma shows how we can derive a lower bound on the k-planar crossing number of a graph G based on a lower bound on the (ordinary) crossing number of that graph, if G belongs to a family of graphs closed under edge removal, such as simple graphs and bipartite graphs.

Lemma 1 *Let $\mathcal{G}$ be a hereditary class of graphs which is closed under removing edges. Let $f(x) = \alpha x$, for some positive constant α, and let $g(x)$ be an arbitrary function of x. If for every graph G in $\mathcal{G}$, $cr(G) \geq f(m) - g(n)$, then $cr_k(G) \geq f(m) - k \cdot g(n)$ for all $G \in \mathcal{G}$ and for all positive integers k.*

Proof. Fix a graph $G \in \mathcal{G}$. Let $G = \bigcup_{i=1}^{k} G_k$ be a decomposition of G into k subgraphs $G_i = (V, E_i)$ such that $\sum_{i=1}^{k} cr(G_i)$ is minimum. By the hereditary property of $\mathcal{G}$, each G_i is a member of $\mathcal{G}$, and hence $cr(G_i) \geq f(m_i) - g(n)$, where $m_i = |E_i|$. Therefore, $cr_k(G) = \sum_{i=1}^{k} cr(G_i) \geq \sum_{i=1}^{k}(f(m_i) - g(n)) = \alpha \sum_{i=1}^{k} m_i - \sum_{i=1}^{k} g(n) = f(m) - k \cdot g(n)$. □

Another combinatorial tool typically used for deriving lower bounds on the crossing number of graphs is the counting method (see, e.g., [9, 16]). We use the following generalization of the counting method in this paper.

Lemma 2 (Counting method) *Let G be a simple graph that contains α copies of a subgraph H. If in every k-planar drawing of G, each crossing of the edges belongs to at most β copies of H, then*

$$cr_k(G) \geq \left\lceil \frac{\alpha}{\beta} cr_k(H) \right\rceil.$$

Proof. Let D be a k-planar drawing of G, realizing $cr_k(G)$. For each of the α copies of H, D contains a k-planar drawing with at least $cr_k(H)$ crossings. Since each crossing is counted at most β times by our assumption, the lemma statement follows. Note that a ceiling is put in the right-hand side of the inequality, because $cr_k(G)$ is always an integer. □

[1] By definition, $f(x,y) = o(g(x,y))$ if $\lim_{x,y\to\infty} \frac{f(x,y)}{g(x,y)} = 0$.

3 Lower Bounds for Complete Bipartite Graphs

In this section, we provide new lower bounds on the biplanar crossing number of complete bipartite graphs. In particular, we improve the following bound due to Czabarka *et al.* [4] which states that for all $p, q \geq 10$,

$$cr_2(K_{p,q}) \geq \frac{p(p-1)q(q-1)}{290}.$$

From Euler's formula, we have $cr(G) \geq m-3(n-2)$ for simple graphs, and $cr(G) \geq m-2(n-2)$ for bipartite graphs. Using Lemma 1, we immediately get a lower bound of $cr_2(G) \geq m-6(n-2)$ for simple graphs, and a lower bound of $cr_2(G) \geq m-4(n-2)$ for bipartite graphs.

To establish stronger lower bounds, we need to incorporate more powerful ingredients. A graph is called k-planar, if it can be drawn in the plane in such a way that each edge has at most k crossings. It is known that every 1-planar drawing of a 1-planar graph has at most $n-2$ crossings [6]. (Note the difference between k-planar graphs, and k-planar crossing numbers.) Removing one edge per crossing yields a planar graph. Therefore, every 1-planar bipartite graph has at most $3n-6$ edges. Karpov [10] proved that for every 1-planar bipartite graph with at least 4 vertices, the inequality $m \leq 3n-8$ holds. In a recent work, Angelini *et al.* [2] proved that for every 2-planar bipartite graph we have $m \leq 3.5n-7$. We use these results to obtain the following stronger lower bound.

Lemma 3 *For every bipartite graph G with $n \geq 4$,*

$$cr_k(G) \geq 3m-(8.5n-19)k.$$

Proof. Let G be a bipartite graph with n vertices and m edges. Fix a drawing of G with a minimum number of crossings. If $m > 3.5n-7$, then by [2], there must be an edge in the drawing with at least three crossings. We repeatedly remove such an edge until we reach a drawing with $\lfloor 3.5n-7 \rfloor$ edges. Now, by Karpov's result, there must be an edge in the drawing with at least two crossings. We repeatedly remove such an edge until we reach a drawing with $3n-8$ edges. Let G' be the bipartite graph corresponding to the remaining drawing. We know by Euler's formula that $cr(G') \geq (3n-8)-2(n-2)$. Therefore,

$$\begin{aligned} cr(G) &\geq 3(m-\lfloor 3.5n-7 \rfloor)+2(\lfloor 3.5n-7 \rfloor-(3n-8)) \\ &\quad +(3n-8)-2(n-2) \\ &\geq 3m-\lfloor 3.5n-7 \rfloor-(3n-8)-2(n-2) \\ &\geq 3m-8.5n+19. \end{aligned}$$

Applying Lemma 1 yields $cr_k(G) \geq 3m-(8.5n-19)k$. □

For complete bipartite graphs, Lemma 3 implies that $cr_2(K_{p,q}) \geq 3pq-17(p+q)+38$, for all $p, q \geq 2$. We use Lemma 3 along with a counting argument to obtain the following improved bound on $cr_2(K_{p,q})$.

Theorem 4 *For all $p, q \geq 30$,*

$$cr_2(K_{p,q}) \geq \frac{p(p-1)q(q-1)}{213}.$$

Proof. Using the counting method (Lemma 2) for $K_{p,p}$ and $K_{p+1,p}$ we have

$$cr_2(K_{p+1,p}) \geq \left\lceil \frac{p+1}{p-1} cr_2(K_{p,p}) \right\rceil.$$

This is because $K_{p+1,p}$ contains $p+1$ copies of $K_{p,p}$, and each crossing realized by two edges, belongs to at most $\binom{p-1}{p-2} = p-1$ of these copies. Using a similar argument for $K_{p+1,p}$ and $K_{p+1,p+1}$, we get

$$cr_2(K_{p+1,p+1}) \geq \left\lceil \frac{p+1}{p-1} \left\lceil \frac{p+1}{p-1} cr_2(K_{p,p}) \right\rceil \right\rceil. \quad (1)$$

By Lemma 3, $cr_2(K_{15,15}) \geq 203$. Plugging into (1), yields $cr_2(K_{16,16}) \geq 266$, Now, we use the recurrence relation (1) iteratively from $p = 16$ to 30 to get

$$cr_2(K_{30,30}) \geq 3554. \quad (2)$$

We can now apply the counting method on $K_{30,30}$ and $K_{p,q}$ to obtain

$$\begin{aligned} cr_2(K_{p,q}) &\geq \frac{\binom{p}{30}\binom{q}{30}}{\binom{p-2}{28}\binom{q-2}{28}} cr_2(K_{30,30}) \\ &= \frac{p(p-1)q(q-1)}{30 \times 29 \times 30 \times 29} cr_2(K_{30,30}). \end{aligned}$$

Plugging (2) in the above inequality yields the theorem statement. □

Remark. The exact value of the denominator obtained in the above proof is around 212.97. One may continue applying the recurrence relation (1) to obtain better bounds for $K_{p,p}$, when $p > 30$. This leads to a slightly improved constant in the denominator, but it does not seem to reduce the constant below 212. Indeed, the denominator seems to converge to a value around 212.4, for large values of p.

4 Biplanar Crossing Number of Complete Graphs

We now consider the biplanar crossing number of complete graphs. Czabarka *et al.* [4] used a probabilistic method to prove that for large values of n,

$$cr_2(K_n) \geq \frac{n^4}{952}.$$

We improve this lower bound using the counting method.

Theorem 5 *For all $n \geq 24$,*

$$cr_2(K_n) \geq \frac{n(n-1)(n-2)(n-3)}{698}.$$

Proof. We know from [1] that for every G with $n \geq 3$, $cr(G) \geq 5m - \frac{139}{6}(n-2)$. Applying Lemma 1, we get

$$cr_2(G) \geq 5m - \frac{139}{3}(n-2).$$

This in particular implies $cr_2(K_{25}) \geq 435$. Now, we use the counting method (Lemma 2) on K_{25} and K_n to get

$$cr_2(K_n) \geq \frac{\binom{n}{25} cr_2(K_{25})}{\binom{n-4}{21}} \geq \frac{n(n-1)(n-2)(n-3)}{\frac{25\times24\times23\times22}{435}},$$

which implies the theorem statement. □

We can slightly improve this result, using an iterative counting method similar to what we used in the previous section.

Theorem 6 *For large values of n,*

$$cr_2(K_n) \geq \frac{n^4}{694}.$$

Proof. Using the counting method (Lemma 2) for K_n and K_{n+1} we have

$$cr_2(K_{n+1}) \geq \left\lceil \frac{(n+1)cr_2(K_n)}{n-3} \right\rceil. \tag{3}$$

Starting from $cr_2(K_{25}) \geq 435$, we use the recurrence relation (3) iteratively from $n = 25$ to 50 to obtain $cr_2(K_{50}) \geq 7965$. Now, we use the counting method on K_{50} and K_n to get

$$\begin{aligned} cr_2(K_n) &\geq \frac{\binom{n}{50} cr_2(K_{50})}{\binom{n-4}{46}} \\ &\geq \frac{n(n-1)(n-2)(n-3)}{\frac{50\times49\times48\times47}{7965}} \\ &\geq \frac{n(n-1)(n-2)(n-3)}{693.94}, \end{aligned}$$

which implies $cr_2(K_n) \geq \frac{n^4}{694}$ for sufficiently large n. □

5 k-Planar Crossing Number of K_n and $K_{p,q}$

In this section, we provide improved lower bounds on the k-planar crossing number of complete bipartite and complete graphs. Shahrokhi *et al.* [17] proved that for any positive integer k, and sufficiently large integers p, q, and n:

$$cr_k(K_{p,q}) \geq \frac{p(p-1)q(q-1)}{108k^2},$$

and

$$cr_k(K_n) \geq \frac{n(n-1)(n-2)(n-3)}{432k^2}.$$

We improve these results using the ideas developed in Sections 3 and 4.

Theorem 7 *For all $p, q \geq 8k+2$,*

$$cr_k(K_{p,q}) \geq \frac{p(p-1)q(q-1)}{73.2k^2}.$$

Proof. We apply the counting method (Lemma 2) on $K_{8k+2,8k+2}$ and $K_{p,q}$. By Lemma 3, for every bipartite graph G, $cr_k(G) \geq 3m - (8.5n - 19)k$. This yields

$$cr_k(K_{8k+2,8k+2}) \geq 56k^2 + 43k + 12.$$

Hence,

$$\begin{aligned} cr_k(K_{p,q}) &\geq \frac{\binom{p}{8k+2}\binom{q}{8k+2} cr_k(K_{8k+2,8k+2})}{\binom{p-2}{8k}\binom{q-2}{8k}} \\ &= \frac{p(p-1)q(q-1)cr_k(K_{8k+2,8k+2})}{(8k+2)(8k+1)(8k+2)(8k+1)} \\ &\geq \frac{p(p-1)q(q-1)}{\frac{(8k+2)^2(8k+1)^2}{56k^2+43k+12}} \\ &\geq \frac{p(p-1)q(q-1)}{\frac{512}{7}k^2}, \end{aligned}$$

which completes the proof. □

Theorem 8 *For all $n \geq 14k-3$,*

$$cr_k(K_n) \geq \frac{n(n-1)(n-2)(n-3)}{232k^2}.$$

Proof. We use the counting method (Lemma 2) for K_{14k-3} and K_n. Recall that for every G with $n \geq 3$, $cr(G) \geq 5m - \frac{139}{6}(n-2)$ [1]. Therefore, $cr_k(G) \geq 5m - \frac{139}{6}(n-2)k$ by Lemma 1. Thus,

$$cr_k(K_{14k-3}) \geq \frac{497}{3}k^2 - \frac{775}{6}k + 30.$$

Therefore,

$$\begin{aligned} cr_k(K_n) &\geq \frac{\binom{n}{14k-3} cr_k(K_{14k-3})}{\binom{n-4}{14k-7}} \\ &= \frac{n(n-1)(n-2)(n-3)cr_k(K_{14k-3})}{(14k-3)(14k-4)(14k-5)(14k-6)}, \end{aligned}$$

which implies the theorem. □

6 Conclusion

In this paper, we presented several improved bounds on the biplanar and k-planar crossing number of complete graphs and complete bipartite graphs. An obvious open problem is whether the asymptotic approximation factors presented in this paper can be further improved. Obtaining similar bounds on the k-planar crossing number of other graph classes is an intriguing open problem.

References

[1] E. Ackerman. On topological graphs with at most four crossings per edge. *Computational Geometry*, 85:1–37, 2019.

[2] P. Angelini, M. A. Bekos, M. Kaufmann, M. Pfister, and T. Ueckerdt. Beyond-planarity: Turán-type results for non-planar bipartite graphs. In *Proceedings of the 29th International Symposium on Algorithms and Computation*, 2018.

[3] J. Battle, F. Harary, and Y. Kodama. Every planar graph with nine points has a nonplanar complement. *Bulletin of the American Mathematical Society*, 68(6):569–571, 1962.

[4] É. Czabarka, O. Sýkora, L. A. Székely, and I. Vrťo. Biplanar crossing numbers I: A survey of results and problems. In *More sets, graphs and numbers*, pages 57–77. 2006.

[5] É. Czabarka, O. Sýkora, L. A. Székely, and I. Vrťo. Biplanar crossing numbers II: Comparing crossing numbers and biplanar crossing numbers using the probabilistic method. *Random Structures & Algorithms*, 33(4):480–496, 2008.

[6] J. Czap and D. Hudák. On drawings and decompositions of 1-planar graphs. *The electronic journal of combinatorics*, 20(2):54, 2013.

[7] S. Durocher, E. Gethner, and D. Mondal. On the biplanar crossing number of K_n. In *Proceedings of the 28th Canadian Conference on Computational Geometry*, pages 93–100, 2016.

[8] R. K. Guy. A combinatorial problem. *Nabla (Bulletin of the Malayan Mathematical Society)*, 7:68–72, 1960.

[9] R. K. Guy, T. Jenkyns, and J. Schaer. The toroidal crossing number of the complete graph. *Journal of Combinatorial Theory*, 4(4):376–390, 1968.

[10] D. Karpov. An upper bound on the number of edges in an almost planar bipartite graph. *Journal of Mathematical Sciences*, 196(6):737–746, 2014.

[11] A. Liebers. *Methods for Planarizing Graphs: A Survey and Annotated Bibliography*. PhD thesis, 1996.

[12] A. Mansfield. Determining the thickness of graphs is NP-hard. *Mathematical Proceedings of the Cambridge Philosophical Society*, 93(1):9–23, 1983.

[13] P. Mutzel, T. Odenthal, and M. Scharbrodt. The thickness of graphs: a survey. *Graphs and Combinatorics*, 14(1):59–73, 1998.

[14] A. Owens. On the biplanar crossing number. *IEEE Transactions on Circuit Theory*, 18(2):277–280, 1971.

[15] J. Pach, L. A. Székely, C. D. Tóth, and G. Tóth. Note on k-planar crossing numbers. *Computational Geometry*, 68:2–6, 2018.

[16] F. Shahrokhi, O. Sỳkora, L. A. Székely, and I. Vrto. Crossing numbers: bounds and applications. *Intuitive geometry*, 6:179–206, 1995.

[17] F. Shahrokhi, O. Sỳkora, L. A. Székely, and I. Vrto. On k-planar crossing numbers. *Discrete Applied Mathematics*, 155(9):1106–1115, 2007.

[18] W. T. Tutte. The non-biplanar character of the complete 9-graph. *Canadian Mathematical Bulletin*, 6(3):319–330, 1963.

[19] A. T. White and L. W. Beineke. Topological graph theory. *Selected Topics in Graph Theory*, 1:15–49, 1978.

[20] C. Zarankiewicz. On a problem of P. Turán concerning graphs. *Fundamenta Mathematicae*, 41(1):137–145, 1955.

A Constant Time Algorithm for Solving Simple Rolling Cube Mazes

Randal Tuggle* Davis Murphy† Nicholas Lorch‡

Abstract

In a rolling cube maze, a cube is placed on a board and the task is to roll it to a desired final space. Many variations of this puzzle exist. In this paper, we establish formal notation regarding rolling cube mazes and solve a simple variant: find a shortest path that puts a desired label on top at the final space. Utilizing several symmetries and reductions, we then produce a description of the solution path in constant time. This provides a framework for future researchers to develop algorithms to efficiently solve more complex mazes.

1 Introduction

Rolling cube puzzles were first popularized by Martin Gardner [3]. They consist of a labelled cube on a board with some task in mind. Mathematician Robert Abbott built on this to create a "Rolling Cube Maze", which considers an initial space and a final space, and asks to find a path to the final space. Rolling Cube Mazes have many variations, two of which are shown in Figure 1. In the left image, every space is labelled and a condition is applied such that when the cube lands on that space, the space's label must be face up before flipping onto that space (the spaces with asterisk mean any label is allowed). In the right image, there are no labelled spaces, but instead an initial and final space for the cube to start and end on respectively.

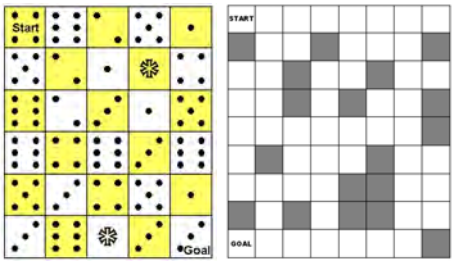


Figure 1: Example Rolling Cube Mazes (Images from Buchin et al. [2])

*Department of Computer Science, University of North Carolina at Chapel Hill, rtuggle99@gmail.com
†Department of Mathematics, Berry College, dkmurphy@outlook.com
‡Department of Statistics, University of Georgia, lorchnd@gmail.com

The rolling cube mazes we consider in this paper have a labelled final space and no blocked spaces. The aim is to find a sequence of moves that takes a cube from an initial position (x_i, y_i) to a final position (x_f, y_f) in the fewest moves such that the cube visits (x_f, y_f) only on the final move and the desired label ℓ ends on top. An important distinction to note is that in our problem, the final label must be face up after flipping onto the final space, not before.

In Section 2, we define notation. In Section 3, we present four techniques that allow us to greatly simplify the problem. In Section 4, we describe solutions for (x_i, y_i) and (x_f, y_f) that are sufficiently far apart. In Section 5, we describe solutions for all other (x_i, y_i) and (x_f, y_f). Finally, in Section 6, we prove that the complexity of our algorithm is O(1). We can formally define the problem as follows:

> **Problem:** Simplified Rolling Cube (SRC).
> **Instance:** board height m, board width n, initial space (x_i, y_i), final space (x_f, y_f), and desired final label ℓ, with the assumption that the cube starts in the standard orientation described in Section 2.1.
> **Solution:** A string description of moves that takes the cube from initial space (x_i, y_i) to final space (x_f, y_f) with desired label ℓ on top in the fewest moves without crossing over (x_f, y_f), or False if there is no solution.

2 Notation

2.1 Describing Faces and Assigning Labels

First, we name the faces of our cube according to the net provided. We define the North face to be the face that points North, the East face to be the face that points East, and so on. Then we define the Top face and Bottom face to be the face pointing away from and touching the board respectively. We assign labels to the starting faces of our cube according to the labeling of a standard right-handed die and, without loss of generality, create a standard starting orientation, pictured in Figure 2:

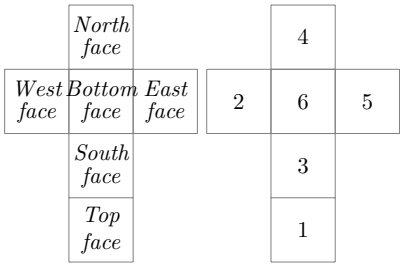


Figure 2: Face descriptions and starting labels

2.2 Describing Moves and Paths

We define North (N), South (S), East (E), and West (W) moves as flipping the cube onto the space immediately north/south/east/west of the cube's current position respectively. We also define an identity (I) move which leaves the cube in its current orientation and position. We represent a sequence of moves from one space to another as a Generalized Path String, or GPS. We can use the following grammar rules to define a GPS with Z being the start symbol

$$\begin{aligned} Z &\to (M)\{EXP\}Z \mid MZ \mid \epsilon \\ M &\to MM \mid I \mid N \mid E \mid S \mid W \\ EXP &\to \Delta + D \mid \Delta - D \mid D \\ \Delta &\to \Delta_x \mid \Delta_y \\ D &\to DD \mid 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \end{aligned}$$

We define $\Delta_x = |x_f - x_i|$ and $\Delta_y = |y_f - y_i|$. When we return the string, however, the string contains the literal characters "Δ_x" and "Δ_y" rather than the numbers they represent. We do this because Δ_x and Δ_y can be arbitrarily large, and we want the length of the string to be bounded by a constant. Δ is either Δ_x or Δ_y. EXP is an expression of the form Δ plus or minus some $D \in \mathbb{N}$ (our algorithm never uses $D > 4$). EXP evaluates to some number $d \in \mathbb{N}$. M is simply any sequence of the five basic moves described above. When we have $(M)\{EXP\}$, we take this to mean that we perform the moves M in parentheses consecutively d times.

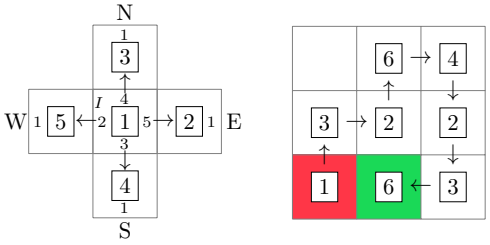


Figure 3: The moves N, E, S, W, I, and the path 'NENESSW' visualized

3 Simplifying the Problem

In this section, we introduce several techniques and a partitioning that together simplify our problem significantly. The first technique is "face-saving" which allows us to keep track of our desired final top label over arbitrarily long distances. The second technique is "Quadrant Mapping" which allows us to assume that $x_f \geq x_i$ and $y_f \geq y_i$. The third technique is a series of reductions which allows us to focus only on solving for $\ell = 1$ or $\ell = 6$. Following these techniques, we partition the displacements into two sets, large and small, which we handle differently.

3.1 Face Saving

In later proofs, we utilize the technique of "saving" the desired final top label onto one of the two faces that are unchanged by moving only along a single axis. This allows us to move the cube an arbitrary number of moves in either direction along that axis and still know exactly the face on which the desired final label is saved.

Definition 1 *A label ℓ is said to be **saved with respect to an axis** A if and only if moving along A keeps label ℓ on the same face.*

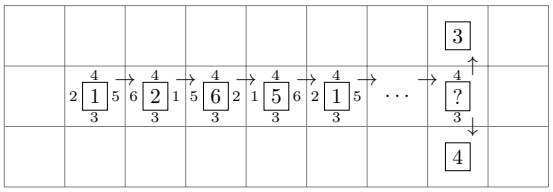


Figure 4: Saving 4 on the North face with respect to the E-W axis

3.2 Quadrant Mapping

We say (x_f, y_f) is in quadrant 1 if $x_f \geq x_i$ and $y_f \geq y_i$, in quadrant 2 if $x_f < x_i$ and $y_f \geq y_i$, in quadrant 3 if $x_f < x_i$ and $y_f < y_i$, and in quadrant 4 if $x_f \geq x_i$ and $y_f < y_i$.

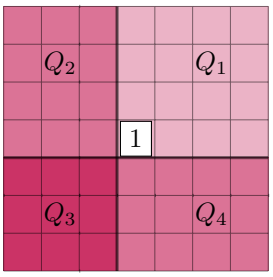


Figure 5: Quadrants

We can perform reflections to create a (x'_f, y'_f) and ℓ' that fall in quadrant 1 via the following steps:

- Step 1: start at (x_i, y_i) in the standard orientation and show (x_f, y_f) with desired label ℓ where (x_f, y_f) is
- Step 2: reflect the board across N-S axis if (x_f, y_f) began in $Q3$ or $Q4$
- Step 3: reflect the board across E-W axis if (x_f, y_f) began in $Q3$ or $Q2$
- Step 4: relabel cube to be in standard orientation and find ℓ' corresponding to the new labeling.

As seen in the pseudo code for *mapToQuad*1 in the appendix, we can define a tuple $q = (x_i < x_f, y_i < y_f)$ in which the first or second element of q is true if we are reflecting over the N-S or E-W axis respectively. After we generate our GPS, we can switch the E's with W's and N's with S's as needed to find an analogous path for the original quadrant. This conversion can be seen in the pseudo code for *convGPS* in the appendix.

3.3 Reductions

We can reduce the number of cases by noting certain symmetries. First, any path to (x_f, y_f) ending on $\ell = 2$ on an $m \times n$ board is analogous to a path to (y_f, x_f) ending on $\ell = 3$ on an $n \times m$ via swapping $N's$ with $E's$ and $S's$ with $W's$. The $\ell = 4$ and $\ell = 5$ cases share the same symmetry. The pseudo code for handling labels 2 and 5 is shown in the appendix. In the remainder of this section, we reduce the $\ell = 3, 4$ cases to either the $\ell = 1$ or $\ell = 6$ case.

Lemma 1 *Let $\ell = 3, 4$. For any GPS G that places ℓ on top in k moves, there exists a GPS that begins with $(E)\{i\}N$ or $(E)\{i\}S$ or $(W)\{i\}N$ or $(W)\{i\}S$ for some $i \in \{0, 1, 2, 3\}$ that also places ℓ on top at (x_f, y_f) in k moves.*

The idea for the reduction is that if there is some GPS G that places ℓ on top at (x_f, y_f), then there exists some GPS that begins with $i \in \{0, 1, 2, 3\}$ E or W moves followed by a N or S move that places ℓ on the top or bottom at some (x_r, y_r) and places ℓ on top at (x_f, y_f) in the same amount of moves as G.

Theorem 2 *The $\ell = 3$ and $\ell = 4$ cases can reduce to the $\ell = 1, 6$ cases in constant time*

3.4 Displacement Types

For ease of analyzing displacements, we denote the displacements before the reduction to $\ell = 1$ or $\ell = 6$ as Δ_x and Δ_y and the displacements after the reduction as Δ'_x and Δ'_y. That is, $\Delta_x = \Delta'_x + \delta_x$ and $\Delta_y = \Delta'_y + \delta_y$ for some natural numbers δ_x, δ_y. Furthermore, we define (x_r, y_r) to be the initial space after the reduction.

We denote the bottom left square of the $m \times n$ boards as $(1, 1)$. Note that if we ignore ending labels, going from (x_r, y_r) to (x_f, y_f) takes at least $|x_r - x_f| + |y_r - y_f|$, or $(\Delta'_x + \Delta'_y)$, moves. Unfortunately, finding a path to (x_f, y_f) with label ℓ on top in $(\Delta'_x + \Delta'_y)$ moves is not always possible.

For $\ell = 1, 6$, we define threshold values $\Delta_{row}, \Delta_{col}$ for Δ'_y, Δ'_x in Table 1 to separate our problem into cases requiring different GPS templates. Note that there are two sets of threshold values for $\ell = 6$. This is because we define two possible paths for $\ell = 6$ in Section 4, one starting with a N move and one starting with an E.

ℓ	# of rows apart (Δ_{row})	# of cols apart (Δ_{col})
1	2	2
6_N	4	2
6_E	2	4

Table 1: Threshold values for displacement categories

We can now define the large displacements (Section 4) to be the cases where $\Delta'_y \geq \Delta_{row}$ and $\Delta'_x \geq \Delta_{col}$ and small displacements to be all remaining cases.

4 Large displacements

To begin, we list the shortest string of moves required to get $\ell = 1, 6$ saved on either the North or East face:

ℓ	Prefixes	Face ℓ is on
1	N or E	North or East
6	EEN or NNE	North or East

Table 2: Prefixes to use for $\ell = 1, 6$

Once ℓ is saved on either the North face or the East face on some space, we can follow one of the paths depicted in the figure below:

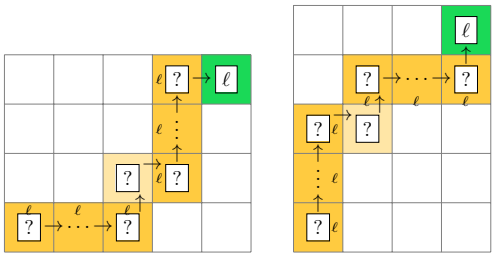


Figure 6: Large-displacement method once ℓ is saved on the North face (left) or East face (right)

Then, to generate a GPS, we can apply the aforementioned prefix to the path found from this face saving process. Doing this, we can describe the solutions for large displacements:

ℓ	Generalized Path Strings	
1	N	$(E)\{\Delta'_x - 2\}NE(N)\{\Delta'_y - 2\}E$
6_N	NNE	$(N)\{\Delta'_y - 4\}EN(E)\{\Delta'_x - 2\}N$
6_E	EEN	$(E)\{\Delta'_x - 4\}NE(N)\{\Delta'_y - 2\}E$

Table 3: Large Displacement GPS's

An important observation is that the number of moves in these GPS's is always $\Delta'_x + \Delta'_y$. Thus, these GPS's use the fewest moves to get from (x_r, y_r) to (x_f, y_f) with ℓ on top.

5 Small Displacements

When dealing with large displacements we did not need to worry about potentially rolling off the board or using a GPS that contains a value such as $\Delta'_x - 2$ which may now be negative. Thus, we must come up with a different way to find solutions when $\Delta'_x < \Delta_{col}$ or $\Delta'_y < \Delta_{row}$.

5.1 Breadth First Search Approach

One possible approach is to check all possible paths to (x_f, y_f) using a brute force algorithm and pick the shortest one. Using this approach, it would be impossible to find solutions in $O(1)$ time since since the complexity depends on m and n. So we do not use this approach in our actual algorithm. However, we use this approach to prove that some of our results are in fact solutions of fewest moves. In order to brute force all paths, we create a graph with one vertex for each possible position and orientation combination, or *state* [2], of a cube on a given board. Edges are between two vertices if their corresponding states are one N, E, S, or W move apart.

5.2 Systematic Approach

We begin our systematic approach by proving the following Theorem.

Theorem 3 *Assume there exists a GPS that begins at (x_i, y_i) on an $m \times n$ board and places ℓ on top at $(x_i + \Delta_x, y_i + \Delta_y)$ in $\Delta_x + \Delta_y + 2k$ moves for some $k \in \mathbb{N}$. Then for all (x_c, y_c) such that $x_c \geq k$ and $y_c \geq k$ on any board large enough to allow (x_c, y_c) and $(x_c + \Delta_x, y_c + \Delta_y)$ to exist, all GPS's from (x_c, y_c) to $(x_c+\Delta_x, y_c+\Delta_y)$ that place ℓ on top in the fewest moves are contained within an $(\Delta_y + 2k) \times (\Delta_x + 2k)$ rectangle such that the bottom left corner of this rectangle is $(x_c - k, y_c - k)$ and the upper right corner is $(x_c + \Delta_x + k, y_c + \Delta_y + k)$.*

The idea with Theorem 3 is that once we have a GPS for a given Δ'_x and Δ'_y, we can get an upper bound for the size of boards we need to check in order to find a GPS of fewest moves for the given Δ'_x and Δ'_y regardless of board size.

5.2.1 Symmetry

Because $\ell = 1, 6$ begins on the top and bottom faces respectively, any path to (x_f, y_f) with $\Delta'_y > \Delta'_x$ is analogous to a path to (y_f, x_f) with $\Delta'_x > \Delta'_y$, just by swapping Ns with Es and Ss with Ws. Therefore, for the following cases, we assume $\Delta'_x \geq \Delta'_y$.

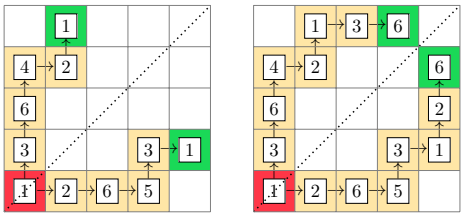


Figure 7: $\ell = 1$ (left) and $\ell = 6$ (right) GPS symmetry about diagonal

We now handle the $\ell = 1, 6$ small displacement cases. We begin by defining the function χ such that $\chi(\ell)$ is the number of E moves required to place ℓ on top from the starting orientation. For $\ell = 1, 6$ we find that $\chi(1) = 0$ and $\chi(6) = 2$. These are the main cases we use for small displacements:

1. $\Delta'_x = 0$ and $\Delta'_y = 0$
2. $\Delta'_x \equiv_4 \chi(\ell)$
3. $\Delta'_y = 0$
4. $\Delta'_y = 1$
5. $\Delta'_y = 2$
6. $\Delta'_y = 3$

To handle the small displacement cases we go through the above enumeration in order, handling a case only if the previous case has not been met.

5.2.2 $\Delta'_x = 0$ and $\Delta'_y = 0$

When $\ell = 1$, return I. Our formal statement of SRC allows the cube to be on the final space only when the correct label is on the top face, so return False when $\ell = 6$.

5.2.3 $\Delta'_x \equiv_4 \chi(\ell)$

By the definition of χ, using $4k + \chi(\ell)$ for all $k \in \mathbb{Z}$ consecutive E moves places ℓ on top when our cube is in starting orientation. Thus, when $\Delta'_x \equiv_4 \chi(\ell)$, a solution can be obtained by rolling to $(x_f - 1, 1)$ to save ℓ on the West face, then performing Δ'_y N moves, and finally placing ℓ on top via an E move as shown in Figure 8.

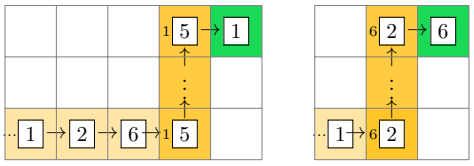


Figure 8: $\ell = 1$ (left) and $\ell = 6$ (right) when $\Delta'_x \equiv_4 \chi(\ell)$

5.2.4 $\Delta'_y = 0$

Theorem 4 *All paths from (x_i, y_i) to (x_f, y_f) require $\Delta_x + \Delta_y + 2k$ moves for some $k \in \mathbb{N}$.*

Theorem 5 *When $m > 1$, $\Delta'_y = 0$, $\Delta'_x > \chi(\ell)$, and $\Delta'_x \not\equiv_4 \chi(\ell)$, the shortest path from (x_r, y_r) to (x_f, y_f) with final label $\ell = 1, 6$ is exactly $\Delta'_x + \Delta'_y + 2$ moves.*

It follows that when $\ell = 1$, going either N or S then E as far as needed and then finally S or N, we get a GPS of fewest moves. We can also do something similar for $\ell = 6$ as shown in Figure 9:

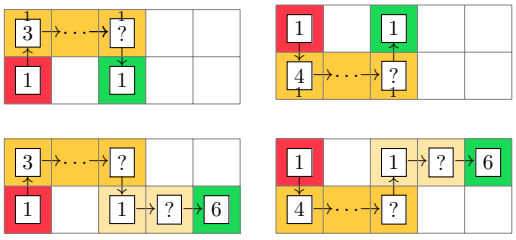


Figure 9: $\ell = 1$ (top) and $\ell = 6$ (bottom) when $m > 1$, $\Delta'_x \not\equiv_4 \chi(l)$ and $\Delta'_y = 0$

When $\ell = 6$ there is not room to do this when $\Delta'_x = 1$. Using the brute force algorithm, we checked on boards up to size 7×8 and found that the GPS's listed in *small1case4* and *small6case4* in the appendix were the shortest paths that place ℓ on top at (x_f, y_f). We know by Theorem 3 that these are the shortest paths.

5.2.5 $\Delta'_y = 1$

Theorem 6 *For the $\ell = 1, 6$ small displacement case, when $\Delta'_y = 1$, $\Delta'_x > 1$ and $\Delta'_x \not\equiv_4 \chi(\ell)$, there is no path that places ℓ on top at (x_f, y_f) in $\Delta'_x + \Delta'_y$ moves.*

Notice that the paths in Figure 10 place ℓ on top in $\Delta'_x + \Delta'_y + 2$ moves. Therefore, by Theorem 6, these GPS's are the shortest paths that place ℓ on top at (x_f, y_f) for $\ell = 6$ when $\Delta'_x > 1$ and for $\ell = 1$ when $\Delta'_x > 3$.

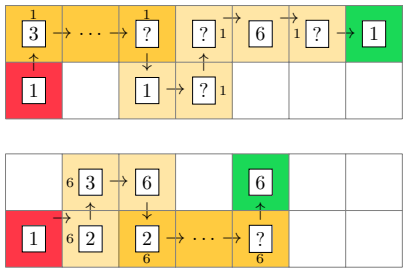


Figure 10: $\ell = 1$ (left) and $\ell = 6$ (right) when $m > 1$, $\Delta'_x \not\equiv_4 \chi(\ell)$ and $\Delta'_y = 1$

What remains of the $\Delta'_y = 1$ case is when $\ell = 1$ and $\Delta'_x \leq 3$ or when $\ell = 6$ and $\Delta'_x = 1$. Applying our brute force algorithm, we found the paths required $\Delta'_x + \Delta'_y + 6$ moves when $\ell = 1, 6, \Delta'_x = 1$. Thus by Theorem 3, we need to check only boards up to size 8×8 to find solutions of fewest moves.

5.2.6 $\Delta'_y = 2$

Note that if $\ell = 1$, we are not in the small displacement case. Therefore, $\ell = 6$. We can use the GPS depicted in Figure 11:

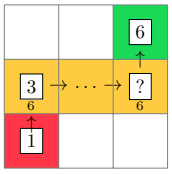


Figure 11: $\ell = 6$ when $\Delta'_y = 2$

5.2.7 $\Delta'_y = 3$

The only time we are in this case is when $\ell = 6$ and $\Delta'_x = 3$. The GPS depicted in Figure 12 is of fewest moves:

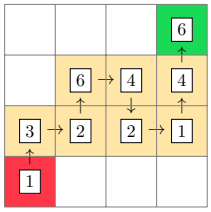


Figure 12: $\ell = 6$ when $\Delta'_y = \Delta'_x = 3$

5.2.8 Putting the cases together

The pseudo code *small*1 and *small*6 put the small displacement cases together via the helper functions seen in the appendix.

6 Finalized Approach

The main approach to solving a rolling cube maze is as follows. Reduce inputs to a quadrant 1 problem, reduce the inputs to a $\ell = 1$ or $\ell = 6$ problem, determine the displacement type, create a GPS, and invert the quadrant mapping to return a final GPS. Pseudo code can be seen in the appendix.

Theorem 7 *The length of the GPS returned by SRC is bounded by a constant and is generated in constant time*

7 Future Work

We believe that future work can now easily be done on other rolling cube mazes by utilizing the fact that a path from any space to any other space with a desired label can be generated in constant time. We implemented SRC and created the following visualization available on github to aid any potential researchers interested in exploring mazes with blocked spaces.

8 Acknowledgements

We would like to thank Professor Jack Snoeyink for reading our drafts and for giving helpful feedback.

References

[1] Robert Abbott SuperMazes: Mind Twisters for Puzzle Buffs, Game Nuts, and Other Smart People. *Prima Publishing*, 1997.

[2] Kevin Buchin and Maike Buchin and Erik D. Demaine and Martin L. Demaine and Dania El-Khechen and Sandor Fekete and Christian Knauer and André Schulz and Perouz Taslakian On rolling cube puzzles. *In Proc. 19th Canad. Conf. Comput. Geom.*, pages 141–148, 2007.

[3] Martin Gardner Mathematical games column. *Scientific American*, 209(6):144, 1963.

[4] Jiawei Yao Research on the Minimum Moves of Rolling Cube Puzzles *JAIST*, 2021.

Appendix

Proofs

Proof. (Lemma 1) Assume $\ell = 3, 4$ and assume some GPS G gets ℓ on top in k moves. Since ℓ is saved with respect to the $E - W$ axis, there must be a N or S in G in order to get ℓ on top. Without loss of generality, assume a N move exists in G. The N move is performed after some amount of E and/or W moves. As stated previously, an E move can be thought of as a negative W move. Thus, the sequence of E and W moves prior to the first N move can be written entirely as E's or entirely as W's. Without loss of generality, assume the moves can be represented entirely by E's. That is, the GPS G can be written as $(E)\{c\}NG_s$ for some c where G_s is the substring of G after the first N move.

We can write $c \equiv_4 i$ for some $i \in \{0, 1, 2, 3\}$. Note $(E)\{c\}$ will keep ℓ face saved. Then N will put $\ell = 3, 4$ on either the top or bottom respectively. Note that if we instead did $(E)\{i\}N$ first, the same would be true. Then $(E)\{c - i\}$ would keep $\ell = 3$ on top and $\ell = 4$ on bottom because every four E moves puts the labels back on the faces they started on. Thus the GPS $G = (E)\{c\}NG_s$ can be written as $(E)\{i\}N(E)\{c - i\}G_s$.

An analogous argument could have been made if G used W moves rather than E moves at the beginning. Furthermore, an analogous argument could be made if the move before G_s were a S move rather than a N move. ∎

Proof. (Theorem 2) As noted in the proof of Lemma 1, when $\ell = 3$, a N or S move must be performed at some point in order to get label ℓ on top. Note that when a first N move is performed, $\ell = 3$ is on top at some $(x_r, y_i + 1)$ and this can be treated as an instance of the $\ell = 1$ case, which is $SRC(m, n, x_r, y_i + 1, x_f, y_f, 1)$. Similarly, note that when a first S move is performed, $\ell = 3$ is on bottom and this can be treated as an instance of the $\ell = 6$ case. By Lemma 1, it follows that we only need to consider any paths that begin with 0,1,2, or 3 consecutive E's/W's followed by a N or S. Since we perform moves before reducing to $\ell = 1$ or $\ell = 6$, we are changing what Δ_x and Δ_y are. To take care of this, we can define a helper function *SRCReduction* which takes the same inputs as *SRC* along with two inputs δ_x and δ_y so that the number of moves in the returned GPS is correct. Also, since the E moves or W moves may change the quadrant (x_f, y_f) is in, we need to do quad mapping again before *SRCReduction* handles the $\ell = 1, 6$ case. This is not a problem because when $\ell = 1, 6$, quad mapping does not change the desired label. We will have that δ_x is the difference of $|x_f - x_i|$ and $|x_f - x_r|$ and δ_y is the difference of $|y_f - y_i|$ and $|y_f - y_r|$ where (x_r, y_r) is the position of the cube when the reduction is called. The body of *SRCReduction* can be seen in the appendix. Following these results, we get the pseudo code *handle*3 in the appendix, which handles the $\ell = 3$ case. Since $\ell = 3$ reduces to the $\ell = 1$ or $\ell = 6$ case and the $\ell = 2$ case is analogous to the $\ell = 3$ case, it follows that the $\ell = 2$ case reduces to the $\ell = 1$ or $\ell = 6$ case. By an analogous argument, we can handle the $\ell = 4, 5$ cases, as seen by the pseudo code for *handle*4 in the appendix. ∎

Proof. (Theorem 3) Assume there exists a GPS starting at (x_i, y_i) and placing ℓ on top at (x_f, y_f) in $\Delta_x + \Delta_y + 2k$ moves on an $m \times n$ board. Now consider a GPS G of fewest moves from (x_c, y_c) to $(x_c + \Delta_x, y_c + \Delta_y)$ ending with the same ℓ on top, for some $x_c > k$ and $y_c > k$. Assume for the sake of contradiction that G requires moving west or south of $(x_c - k, y_c - k)$ or north or east of $(x_c + \Delta_x + k, y_c + \Delta_y + k)$. Without loss of generality, assume G requires moving west of $(x_c - k, y_c - k)$. That is, there exists a position $(x_c - k - 1, y_d)$ for some y_d that is visited by G. Note that at a minimum,

this requires $|(x_c)-(x_c-k-1)| = k+1$ moves. Then, at the very minimum, going from this space (x_c-k-1, y_d) to the ending space $(x_c+\Delta_x, y_c+\Delta_y)$ requires $(x_c+\Delta_x)-(x_c-k-1)+(y_c+\Delta_y)-(y_d) = \Delta_x+\Delta_y+k+1+y_c-y_d$ moves. Thus in total G requires $(k+1)+(\Delta_x+\Delta_y+k+1+y_c-y_d) = \Delta_x+\Delta_y+2(k+1)+y_c-y_d \geq \Delta_x+\Delta_y+2(k+1)+0$ moves. Thus G requires more than $\Delta_x+\Delta_y+2k$ moves. This is a contradiction because we assumed G required the fewest moves and we already know a path of $\Delta_x+\Delta_y+2k$ exists.

Proof. (Theorem 4) This statement is equivalent to Theorem 1 in "Research on the Minimum Moves of Rolling Cube Puzzles" [4].

Proof. (Theorem 5) Consider a path from (x_r, y_r) to (x_f, y_f) with final label $\ell = 1, 6$ on a board with $m > 1$, such that $\Delta_y = 0$, $\Delta'_x > \chi(\ell)$ and $\Delta'_x \not\equiv_4 \chi(\ell)$. Since $\Delta'_y = 0$ and $\Delta'_x \not\equiv_4 \chi(\ell)$, in order to place ℓ on top at (x_f, y_f), a N and S move must be performed eventually. Thus, a solution must have more than $\Delta'_x + \Delta'_y$ moves. By Theorem 4, then, we know any solution must be at least $\Delta_x + \Delta_y + 2$ moves. Because $m > 1$, there is either a row above (x_r, y_r) or below (x_r, y_r). If there exists a row above (x_i, y_i), the GPS $N(E)\{\Delta'_x - \chi(\ell)\}S(E)\{\chi(\ell)\}$ places ℓ on top in $\Delta_x + \Delta'_y + 2$ moves. Similarly, if there exists a row below (x_r, y_r), the GPS $S(E)\{\Delta'_x - \chi(\ell)\}N(E)\{\chi(\ell)\}$ places ℓ on top in $\Delta'_x + \Delta'_y + 2$ moves. Thus, the shortest path under these conditions is exactly $\Delta'_x + \Delta'_y + 2$ moves.

Proof. (Theorem 6) Consider the $\ell = 1, 6$ small displacement case and assume $\Delta'_y = 1$, $\Delta'_x > 1$ and $\Delta'_x \not\equiv_4 \chi(\ell)$. Note that a path to (x_f, y_f) using $\Delta'_x + \Delta'_y$ moves must have exactly 1 N move, which will come after c E moves and be followed by $\Delta'_x - c$ E moves. If $c \equiv_4 0, 2$, we find that $\ell = 1$ will be saved on the North or South faces, respectively, and $\ell = 6$ will be saved on the South or North faces, respectively. This means that the remaining E moves cannot get $\ell = 1, 6$ on the top face. If $c \equiv_4 1, 3$, we see that $\ell = 1$ will be on the East and West faces, respectively, and remain there after the N move. Similarly, $\ell = 6$ will be on the West and East faces, respectively, and will also remain there after the N move. For both $c \equiv_4 1, 3$, the label ℓ will only be on top when the total number Δ'_x of E moves is of the form $4k + \chi(\ell)$. Since we know by initial assumption that this is not true of Δ'_x, we see that using only $\Delta'_x + \Delta'_y$ moves leaves us unable to place $\ell = 1$ or $\ell = 6$ on top at (x_f, y_f). Thus, by Theorem 4, we need at least $\Delta_x + \Delta'_y + 2$ moves to place $\ell = 1, 6$ on top at (x_f, y_f).

Proof. (Theorem 7) To avoid the returning a string whose length depends on the number of digits in $|x_f - x_i|$ and $|y_f - y_i|$, we return a GPS that uses the literal characters Δ_x and Δ_y. Note further that δ_x and δ_y are at most 3, so that is why we can use the numbers they represent rather than also using the literal characters. Therefore, the length of the GPS returned is bounded by a constant.
We have shown that we perform quadrant mapping and reduce any instance of the problem to the $\ell = 1, 6$ cases in constant time. Clearly, the large displacement case can be handled in constant time. As for the small case, as stated in Section 5.1, the complexity of the brute force algorithm used in some of our proofs depends on m and n. However, this algorithm is only necessary for a finite number of scenarios. We have provided the output for each of these scenarios and thus our proposed algorithm finds these solutions by performing a simple lookup. Finally, we can invert the quadrant mapping in constant time. Therefore, the final GPS is generated in constant time.

Algorithms

```
def mapToQuad1 (m,n,xi,yi,xf,yf,ℓ):
    set newL = ℓ and set q = [0,0]
    if xf < xi:
        q[0] = 1
        if ℓ = 2:
            newL = 5
        elif ℓ = 5
            newL = 2
    if yf < yi:
        q[1] = 1
        if ℓ = 3:
            newL = 4
        elif ℓ = 4
            newL = 3
    newXi = (n - xi) + 1 if q[0] == 1 else  xi
    newYi = (m - yi) + 1 if q[1] == 1 else  yi
    newXf = (n - xf) + 1 if q[0] == 1 else  xf
    newYf = (m - yf) + 1 if q[1] == 1 else  yf
    return (newXi ,newYi),(newXf,newYf),newL,q

def convGPS (q,GPS):
    if q[0] == 1
        switch all Es and Ws in GPS
    if q[1] == 1
        switch all Ns and Ss in GPS
    return GPS

def handle2(m,n,xi,yi,xf,yf):
    return handle3(n,m,yi,xi,yf,xf) swapping Ns with Es, Ss with Ws,
    and Δx with Δy

def handle5(m,n,xi,yi,xf,yf):
    return handle4(n,m,yi,xi,yf,xf) swapping Ns with Es and Ss with Ws,
    and Δx with Δy

def SRCReduction(m,n,xr,yr,xf,yf,ℓ,δx,δy):
    (m,n,xr,yr,xf,yf,ℓ,q) = mapToQuad1(m,n,xr,yr,xf,yf,ℓ)
    if q[0] == 1:
        δx = −δx
    if q[1] == 1:
        δy = −δy
    if ℓ == 1:
        GPS =  handle1(m,n,xr,yr,xf,yf,δx,δy)
    elif ℓ == 6:
        GPS =  handle6(m,n,xr,yr,xf,yf,δx,δy)
    return ConvGPS(q,GPS)

def handle3(m,n,xi,yi,xf,yf)
    initialize a list of paths
    for i in [0,1,2,3]:
        path1 = (E){i}N +
        SRCReduction(m,n,xi + i,yi+1,xf,yf,1,
        ||xf − xi| − |xf − (xi + i)||, ||yf − yi| − |yf − (yi + i)||)
        path2 = (E){i}S +
        SRCReduction(m,n,xi + i,yi−1,xf,yf, 6,
        ||xf − xi| − |xf − (xi + i)||, −||yf − yi| − |yf − (yi − i)||)
        path3 = (W){i}N +
        SRCReduction(m,n,xi - i,yi+1,xf,yf,1,
        −||xf − xi| − |xf − (xi − i)||, |yf − yi| − |yf − (yi + i)|)
        path4 = (W){i}S +
        SRCReduction(m,n,xi - i,yi−1,xf,yf,6,
        −||xf − xi| − |xf − (xi − i)||, −||yf − yi| − |yf − (yi − i)||)
        add each path to the list of paths
return the computed path of fewest moves or false if none exist

def handle4(m,n,xi,yi,xf,yf):
    initialize a list of paths
    for i in [0,1,2,3]:
        path1 = (E){i}N +
        SRCReduction(m,n,xi + i,yi+1,xf,yf,6,
```

```
        ||x_f − x_i| − |x_f − (x_i + i)||, ||y_f − y_i| − |y_f − (y_i + i)||)
        path2 = (E){i}S +
        SRCReduction(m,n,xi + i,yi-1,xf,yf,1,
        ||x_f − x_i| − |x_f − (x_i + i)||, −||y_f − y_i| − |y_f − (y_i − i)||)
        path3 = (W){i}N +
        SRCReduction(m,n,xi - i,yi+1,xf,yf,6,
        −||x_f − x_i| − |x_f − (x_i − i)||, ||y_f − y_i| − |y_f − (y_i + i)||)
        path4 = (W){i}S +
        SRCReduction(m,n,xi - i,yi-1,xf,yf,1,
        −||x_f − x_i| − |x_f − (x_i − i)||, −||y_f − y_i| − |y_f − (y_i − i)||)
        add each path to the list of paths
return the computed path of fewest moves or false if none exist

def handle1(m,n,xr,yr,xf,yf,δ_x,δ_y):
        dxPrime = xf - xr, dyPrime = yf - yr
        if dxPrime > 1 and dyPrime > 1:
            return the large displacement template
        else:
            return the small displacement template

def handle6(m,n,xr,yr,xf,yf,δ_x,δ_y):
        dxPrime = xf - xr,  dyPrime = yf - yr
        if dxPrime > 3 and dyPrime > 1 or dxPrime > 1 and dyPrime > 3:
            return the large displacement template
        else:
            return the small displacement template

def large1(δ_x,δ_y):
    return N(E){Δ_x - δ_x- 2}NE(N){Δ_y - δ_y - 2}E

def large6(dxPrime,dyPrime,δ_x,δ_y):
    if  dxPrime > 1 and dyPrime > 3:
        return NNE(N){Δ_y - δ_y - 4}EN(E){Δ_x - δ_x - 2}N
    elif dxPrime > 3 and dyPrime > 1:
        return  EEN(E){Δ_x - δ_x - 4}NE(N){Δ_y - δ_y - 2}E

def small1symmetry(m,n,xr,yr,xf,yf,dxPrime,dyPrime)
    answer = small1(n,m,yr,xr,yf,xf,dyPrime,dxPrime,δ_y ,δ_x )
    return answer but switch Ns with Es and Ss with Ws and Δ_x with Δ_y

def small6symmetry(m,n,xr,yr,xf,yf,dxPrime,dyPrime)
    answer = small6(n,m,yr,xr,yf,xf,dyPrime,dxPrime,δ_y ,δ_x )
    return answer but switch Ns with Es and Ss with Ws and Δ_x with Δ_y

def small1case1(m,n,xr,yr,xf,yf,dxPrime,dyPrime): return I

def small6case1(m,n,xr,yr,xf,yf,dxPrime,dyPrime): return False

def small1case2(m,n,xr,yr,xf,yf,dx,dy,δ_x,δ_y)
    return (E){Δ_x - δ_x - 1}(N){Δ_y - δ_y}E

def small6case2(m,n,xr,yr,xf,yf,dx,dy,δ_x,δ_y)
    return (E){Δ_x - δ_x - 1}(N){Δ_y - δ_y}E

def small1case3(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
    if yf < m:
        return N(E){Δ_x - δ_x}S
    elif yr > 1:
        return S(E){Δ_x - δ_x}N
    else:
        return False

def small6case3(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
    if m == 1:
        return False
    elif dxPrime > 1 and yf < m:
        return N(E){Δ_x - δ_x - 2}SEE
    elif dxPrime >  1 and yr > 1:
        return S(E){Δ_x - δ_x - 2}NEE
    elif dxPrime == 1:
        if xr > 1:
            return NWSEE
        elif yf < m - 1:
            return NENWSSE
        elif yr > 2
            return SESWNNE
        elif xr < n - 1 :
            if yf < m:
                return NEEESWW
            elif yr > 1:
                return SEEENWW
            elif (xr > 1 and yr > 1):
                return SWNESEENW
            else:
                return False

def small1case4(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
    if dxPrime > 3:
        return N(E){Δ_x - δ_x - 4}SENEEE
    elif dxPrime == 2 or dxPrime == 3:
        if yf < m:
            return N(E){Δ_x - δ_x - 2}NESE
        elif yr > 1:
            return ESEN(E){Δ_x - δ_x - 2}N
        else:
            return ENWS(E){Δ_x - δ_x}N
    elif dxPrime == 1:
        if yr > 1:
            return SENWNE
        elif xr > 1:
            return WNESEN
        elif yf < m - 1:
            return NNENWSES
        elif xf < n - 1:
            return EENESWNW
        else:
            return False

def small6case4(m,n,xr,yr,xf,yf,dxPrime,dyPrime, δ_x, δ_y)
    if dxPrime > 1:
        return ENES(E){Δ_x - δ_x - 2}N
    elif dxPrime == 1:
        if yr > 1:
            return ESWNEN
        elif xr > 1:
            return NWSENE
        elif yf < m-2:
            return NNNESWSE
        elif xf < n-2:
            return EEENWSWN
        else:
            return False

def small6case5(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
    return N(E){Δ_x - δ_x}N

def small6case6(m,n,xi,yi,xf,yf,dx,dy)
    return NENESENN

def small1(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        if dyPrime > dxPrime:
                return small1symmetry(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif (dxPrime,dyPrime) == (0,0):
                return small1case1(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif dxPrime == 0 mod 4:
                return small1case2(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif dyPrime == 0:
                return small1case3(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif dyPrime == 1:
                return small1case4(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)

def small6(m,n,xr,yr,xf,yf,dx,dy,δ_x,δ_y):
        if dyPrime > dxPrime:
            return small6symmetry(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif (dxPrime,dyPrime) == (0,0):
                return small1case1(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif dxPrime == 2 mod 4:
            return small6case2(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif dyPrime == 0:
            return small6case3(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif dyPrime == 1:
            return small6case4(m,n,xr,yr,xf,yf,dxPrime,dyPrime.δ_x,δ_y)
        elif dyPrime == 2:
            return small6case4(m,n,xr,yr,xf,yf,dxPrime,dyPrime,δ_x,δ_y)
        elif dyPrime == 3:
            return small6case6(m,n,xr,yr,xf,yf,dxPrime,dyPrime, δ_x,δ_y)

def SRC(m,n,xi,yi,xf,yf,ℓ):
        xi,yi,xf,yf,ℓ,q = mapToQuad1(m,n,xi,yi,xf,yf,ℓ)
        if ℓ == 1:
                GPS = handle1(m,n,xi,yi,xf,yf,0,0)
        elif ℓ == 2:
                GPS = handle2(m,n,xi,yi,xf,yf)
        elif ℓ == 3:
                GPS = handle3(m,n,xi,yi,xf,yf)
        elif ℓ == 4:
                GPS = handle4(m,n,xi,yi,xf,yf)
        elif ℓ == 5:
                GPS = handle5(m,n,xi,yi,xf,yf)
        elif ℓ == 6:
                GPS = handle6(m,n,xi,yi,xf,yf,0,0)
        return ConvGPS(q,GPS)
```

A Theory of Spherical Diagrams

Giovanni Viglietta*

Abstract

We introduce the axiomatic theory of Spherical Occlusion Diagrams as a tool to study certain combinatorial properties of polyhedra in $\mathbb{R}^3$, which are of central interest in the context of Art Gallery problems for polyhedra and other visibility-related problems in discrete and computational geometry.

1 Introduction

Geometric intuition. Consider a set $\mathcal{P}$ of internally disjoint opaque polygons in $\mathbb{R}^3$ and a viewpoint $v \in \mathbb{R}^3$ such that no vertex of any polygon in $\mathcal{P}$ is visible to v. An example is given by the set of six rectangles in Figure 1 (left) with respect to the point v located at the center of the arrangement.

Let S be a sphere centered at v that does not intersect any of the polygons in $\mathcal{P}$, and let $S_\mathcal{P}$ be the set of projections onto S of the portions of edges of polygons in $\mathcal{P}$ that are visible to v (i.e., where polygons occlude projection rays). We call $S_\mathcal{P}$ a *Spherical Occlusion Diagram*. Figure 1 (right) shows an example of such a projection.

In this paper we set out to study the combinatorial structure of Spherical Occlusion Diagrams.

Applications. Spherical Occlusion Diagrams naturally arise in visibility-related problems for arrangements of polygons in $\mathbb{R}^3$, and especially for polyhedra.

An example is found in [3], where an upper bound is given on the number of edge guards that solve the Art Gallery problem in a general polyhedron. That is, given a polyhedron $\mathcal{P}$, the problem is to find a (small) set of edges that collectively see the whole interior of $\mathcal{P}$ (refer to [2, 9] for more results on this problem). An edge e sees a point x if and only if there is a point $y \in e$ such that the line segment xy does not properly cross the boundary of $\mathcal{P}$.

The idea of [3] is to preliminarily select a (small) set E of edges that cover all vertices of $\mathcal{P}$. Note that E may be insufficient to guard the interior of $\mathcal{P}$, as some of its points may be invisible to all vertices; Figure 1 (center) shows an example. Thus, an additional (small) set of edges E' is selected, which collectively see all internal points of $\mathcal{P}$ that do not see any vertices. Clearly, $E \cup E'$ is a set of edges that see all internal points of $\mathcal{P}$.

The selection of the edges E' is carried out in [3] by means of an ad-hoc analysis of some properties of points that do not see any vertices of $\mathcal{P}$. Spherical Occlusion Diagrams offer a systematic and general tool to reason about points in a polyhedron that do not see any vertices.

Spherical Occlusion Diagrams have also provided a framework for proving the main result of [8]: Any point that sees no vertex of a polyhedron must see at least 8 of its edges, and that the bound is tight.

2 Axiomatic Theory

Toward an axiomatization. The construction outlined in Section 1 produces an arrangement $S_\mathcal{P}$ of arcs on the surface of a sphere S. For each arc $a \in S_\mathcal{P}$, let e_a be the edge of a polygon in $\mathcal{P}$ whose orthographic projection on S (partly occluded by other polygons) contains a. Since e_a is a line segment, a must be an arc of a great circle. The fact that each vertex of a polygon in $\mathcal{P}$ is occluded by some other polygon translates into the property that each endpoint of each arc in $S_\mathcal{P}$ must lie in the interior of another arc of $S_\mathcal{P}$. Also, since e_a is an edge of a polygon $P \in \mathcal{P}$, all arcs of $S_\mathcal{P}$ that end on the interior of a must reach it from the same side (as these correspond to edges of polygons partially hidden by P).

Axioms. In the following, S will denote the unit sphere immersed in $\mathbb{R}^3$, and we will abstract from a specific set of polygons $\mathcal{P}$ and a viewpoint v. Some terms will be useful.

Definition 1 *Let a and b be two non-collinear arcs of great circles on a sphere. If an endpoint p of a lies in the relative interior of b, we say that a* hits *b at p (or* feeds into *b at p) and b* blocks *a at p.*

We are now ready to formulate an abstract theory of Spherical Occlusion Diagrams.

Definition 2 *A* Spherical Occlusion Diagram*, or simply* Diagram*, is a finite non-empty collection $\mathcal{D}$ of arcs of great circles on the unit sphere in $\mathbb{R}^3$ satisfying the following axioms.*

A1. If two arcs $a, b \in \mathcal{D}$ have a non-empty intersection, then a hits b or b hits a.

*School of Information Science, Japan Advanced Institute of Science and Technology (JAIST), viglietta@gmail.com

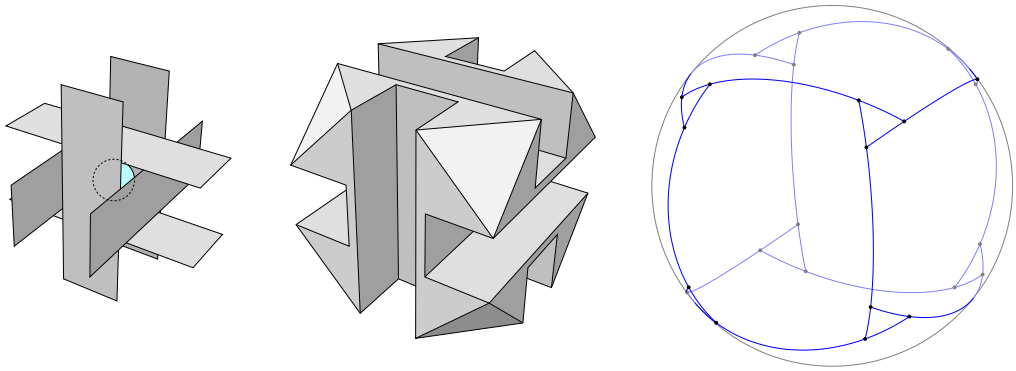


Figure 1: Construction of a Spherical Occlusion Diagram (right) from an arrangement of six rectangles (left) or a polyhedron whose central point does not see any vertices (center)

A2. Each arc in $\mathcal{D}$ is blocked by arcs of $\mathcal{D}$ at each endpoint.

A3. All arcs in $\mathcal{D}$ that hit the same arc of $\mathcal{D}$ reach it from the same side.

Figure 2 shows a Diagram with 18 arcs.

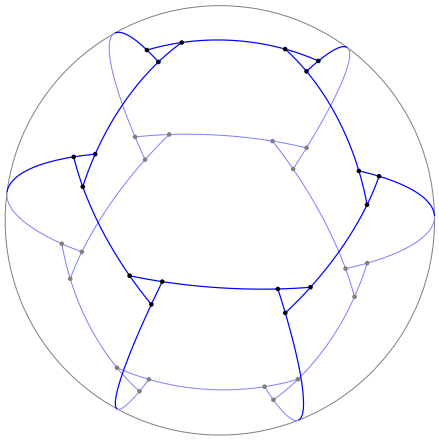


Figure 2: Example of a Diagram with 18 arcs

Realizability. It is immediate to recognize that the Diagrams $S_{\mathcal{P}}$ constructed in Section 1 indeed provide a model for our theory, as they satisfy all its axioms. The proof of the following statement is essentially contained in the first paragraph of Section 2.

Proposition 1 *Any set $S_{\mathcal{P}}$, as constructed in Section 1 for an arrangement of polygons $\mathcal{P}$ and a viewpoint v that sees no vertices of such polygons (re-scaled in such a way that S is the unit sphere), satisfies the axioms of Spherical Occlusion Diagrams, provided that v is in* general position *with respect to $\mathcal{P}$, i.e., no ray emanating from v intersects the interiors of more than two distinct edges of polygons of $\mathcal{P}$.* □

Observe that the general-position requirement in Proposition 1 is irrelevant in the context of the Art Gallery problem and was introduced only for the sake of a more aesthetically pleasing axiomatization of Diagrams. Indeed, the set of points in general position with respect to $\mathcal{P}$ is dense in $\mathbb{R}^3$, whereas the set of points that are visible to any finite set E of edges is topologically closed. Thus, for instance, if the edges in E collectively see all points in general position, then they also see all points that are not in general position.

Although there is compelling evidence that the converse of Proposition 1 is not true, we do not yet have a definitive answer to this fundamental problem, which we leave open. We actually formulate a stronger conjecture. We say that a Diagram $\mathcal{D}$ is *irreducible* if no proper subset of $\mathcal{D}$ is a Diagram.

Conjecture 1 *There is an irreducible Spherical Occlusion Diagram $\mathcal{D}$ (satisfying the conditions in Definition 2) such that $\mathcal{D} \neq S_{\mathcal{P}}$ for any set of internally disjoint polygons $\mathcal{P}$.*

It can be shown that Conjecture 1 is equivalent to its restricted version where $\mathcal{P}$ is a polyhedron of genus zero. Indeed, a set of polygons $\mathcal{P}$ that gives rise to a Diagram $\mathcal{D}$ with respect to a viewpoint v can easily be extended by adding a mesh of polygons whose edges are either shared with $\mathcal{P}$ or concealed from v by polygons in $\mathcal{P}$. The resulting polyhedron gives rise to the same Diagram $\mathcal{D}$.

3 Elementary Properties

We will prove some basic properties of Diagrams.

Proposition 2 *Every arc in a Diagram is strictly shorter than a great semicircle.*

Proof. Referring to Figure 3, assume that an arc a (in red) is at least as long as a great semicircle. Then, taking

an endpoint p of a as the North pole and a itself as the prime meridian, consider an arc b_0 that blocks a at p (which exists by Axiom 2). The arc b_0 has exactly one endpoint in the Eastern hemisphere; let b_1 be an arc that blocks b_0 at this endpoint. We can construct a sequence $(b_0, b_1, b_2, \dots)$ of arcs, each of which hits the next at a point of smaller (or equal) latitude, until one of them hits a from the East (this must eventually happen, because a is at least as long as a great semicircle). Note that no b_i other than b_0 can pass through p without contradicting Axiom 1. Symmetrically, we can construct a similar sequence of arcs starting from the endpoint of b_0 that lies in the Western hemisphere. The last arc of this sequence hits a from the West, contradicting Axiom 3. □

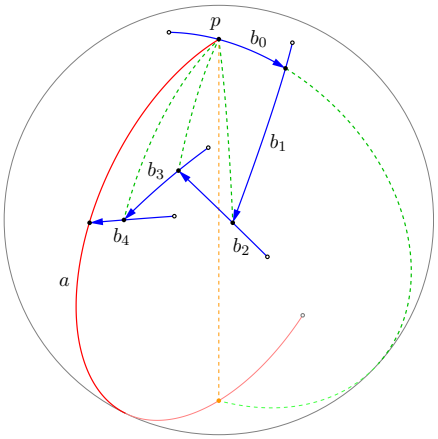


Figure 3: Proof of Proposition 2

We can now prove a stronger form of Axiom 2.

Proposition 3 *Every arc in a Diagram hits exactly two distinct arcs, one at each endpoint.*

Proof. Assume for a contradiction that an endpoint p of an arc a lies in the interior of two arcs b and c. Then b and c intersect at p. By Axiom 1, without loss of generality, b hits c at p, and therefore b and a share an endpoint, which contradicts Axiom 1. Thus, a hits at most one arc at each endpoint; by Axiom 2, it hits exactly one. Moreover, a cannot hit the same arc b at both endpoints p and p', or else p and p' would be antipodal points, and b would be longer than a great semicircle, contradicting Proposition 2. Thus, a hits exactly two distinct arcs. □

Proposition 4 *No two arcs in a Diagram feed into each other.*

Proof. Two arcs feeding into each other must be longer than a great semicircle, as Figure 4 shows. This contradicts Proposition 2. □

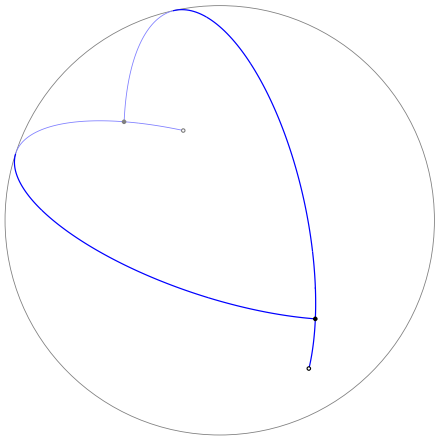


Figure 4: Proof of Proposition 4

Proposition 5 *A Diagram partitions the unit sphere into spherically convex regions.*

Proof. Let $\mathcal{D}$ be a Diagram, and let p and q be two points in the same connected component of $S \setminus \bigcup \mathcal{D}$. There is a chain C of arcs of great circles (drawn in green in Figure 5) that connects p and q without intersecting the Diagram. The arc of a great circle joining p with the third vertex of C (drawn in orange) does not intersect the Diagram either, or else we could reason as in Proposition 2 to construct a sequence of arcs of $\mathcal{D}$ which eventually intersect one of the first two arcs of C. Hence, we can simplify the chain by joining p with its third vertex. Proceeding by induction, we conclude that the arc of a great circle connecting p and q does not intersect $\mathcal{D}$, implying that each connected component of $S \setminus \bigcup \mathcal{D}$ is spherically convex. □

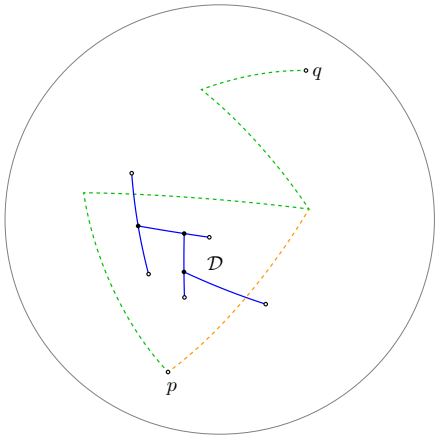


Figure 5: Proof of Proposition 5

Definition 3 *Each of the convex regions into which the unit sphere is partitioned by a Diagram is called a* tile.

It is easy to derive the following from Proposition 5.

Corollary 1 *In a Diagram, the topological closure of no tile contains two antipodal points. Moreover, the relative interior of any great semicircle on the unit sphere intersects some arc of the Diagram.* □

Proposition 6 *The union of all the arcs in a Diagram is a connected set of points.*

Proof. Let the union of the arcs in a Diagram $\mathcal{D}$ have two connected components, given by $\mathcal{D}_1$ and $\mathcal{D}_2$. Note that $\mathcal{D}_1$ and $\mathcal{D}_2$ individually satisfy all axioms, and therefore both are Diagrams. Hence, $\mathcal{D}_2$ is contained in a tile F determined by $\mathcal{D}_1$, as shown in Figure 6. Take two points $p, q \in F$ close to the boundary of F such that the arc of great circle connecting p and q (in orange) intersects $\mathcal{D}_2$. Observe that there exists a chain of arcs of great circle (in green) that connects p and q without intersecting $\mathcal{D}_1$ nor $\mathcal{D}_2$. Hence p and q are in the same tile determined by $\mathcal{D}$. However, since the arc pq intersects $\mathcal{D}$, the tile cannot be spherically convex, contradicting Proposition 5. □

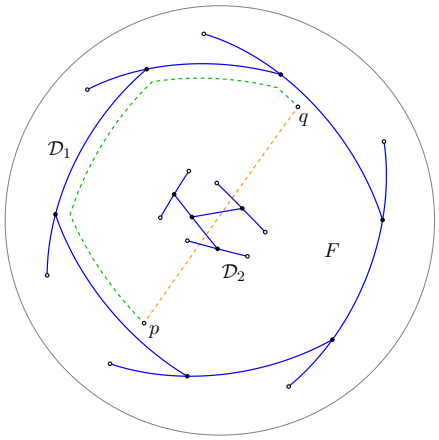


Figure 6: Proof of Proposition 6

Proposition 7 *A Diagram with n arcs partitions the unit sphere into $n + 2$ tiles.*

Proof. Every endpoint of an arc of a Diagram divides the arc it hits into two sub-arcs. The set of these sub-arcs induces a spherical drawing of a planar graph with $2n$ vertices and $3n$ edges. Each face of this drawing coincides with a tile of the Diagram. By Euler's formula, the number of faces is $3n - 2n + 2 = n + 2$. □

4 Swirls

There is a curious similarity between Diagrams and continuous vector fields on a sphere. According to the hairy ball theorem, "it is impossible to comb a hairy ball without creating cowlicks". Similarly, it is impossible to construct a Diagram without creating "swirls", as we shall see in this section.

Definition 4 *A* swirl *in a Diagram is a cycle of arcs, each of which feeds into the next going clockwise or counterclockwise. The spherically convex region enclosed by a swirl is called the* eye *of that swirl.*

Figure 2 shows a Diagram with six clockwise swirls and six counterclockwise swirls. Observe that, in an irreducible Diagram, the eye of each swirl coincides with a single tile; in general, the eye of a swirl is a union of tiles, as it may have internal arcs.

Definition 5 *The* swirl graph *of a Diagram $\mathcal{D}$ is the undirected multigraph on the set of swirls of $\mathcal{D}$ having an edge between two swirls for every arc in $\mathcal{D}$ shared by the two swirls.*

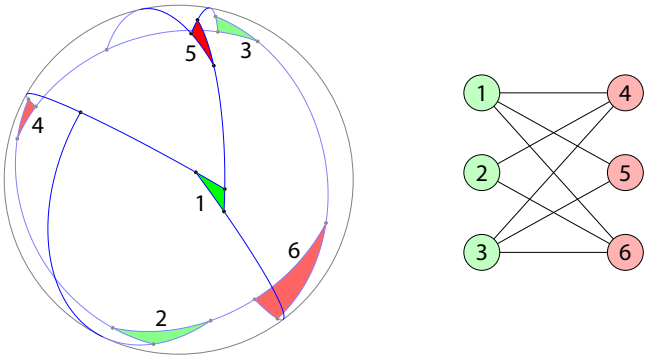


Figure 7: A Diagram and its swirl graph

In Figure 7, the eyes of clockwise swirls are colored green, and the eyes of counterclockwise swirls are colored red. Note that the swirl graph is simple and bipartite; this is true in general.

Theorem 1 *The swirl graph of any Diagram is a simple planar bipartite graph with non-empty partite sets.*

Proof. The swirl graph is spherical, hence planar. It is bipartite, where the partite sets correspond to clockwise and counterclockwise swirls, respectively. Indeed, if the same arc is shared by two concordant swirls (say, clockwise), then it is hit by arcs from both sides, violating Axiom 3.

Figure 8 shows how to find a clockwise and a counterclockwise swirl in any Diagram. For a clockwise swirl, start from any arc and follow it in any direction until it hits another arc. Then turn clockwise and follow this arc until it hits another arc, and so on. The sequence of arcs encountered is eventually periodic, and the period identifies a clockwise swirl. A counterclockwise swirl is found in a similar way.

To prove that the swirl graph is simple, assume for a contradiction that the swirl $\mathcal{S}_1$ shares two arcs a and b with another swirl $\mathcal{S}_2$. Then, the eye of $\mathcal{S}_2$ must be entirely contained in the spherical lune determined by

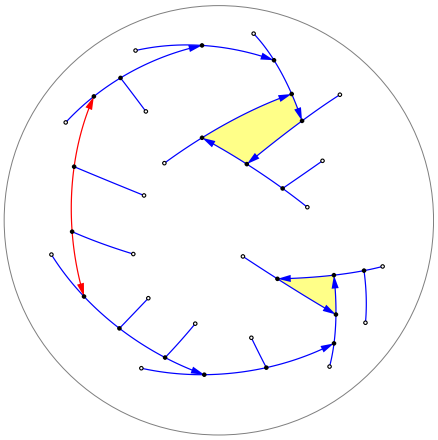


Figure 8: Finding swirls in a Diagram

a and b, as shown in Figure 9. Since the eye of $\mathcal{S}_2$ is bounded by a, it must lie in the region A. However, the eye of $\mathcal{S}_2$ is also bounded by b, and thus it must lie in the region B. This is a contradiction, since A and B are disjoint. □

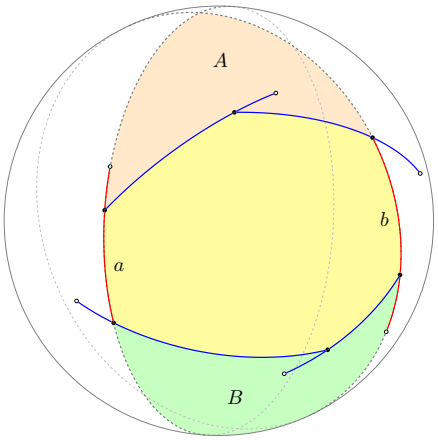


Figure 9: Two swirls cannot share more than one arc

More is actually known about swirl graphs.

Theorem 2 *Every Diagram has at least four swirls.* □

This result has been announced in [8]. From Theorem 2, it easily follows that every Diagram has at least eight arcs. On the other hand, Figure 10 shows an example of a Diagram with exactly eight arcs and exactly four swirls, which is therefore minimal.

It is not yet clear if there are Diagrams with only one clockwise swirl, but we believe this is not the case.

Conjecture 2 *Every Diagram has at least two clockwise and two counterclockwise swirls.*

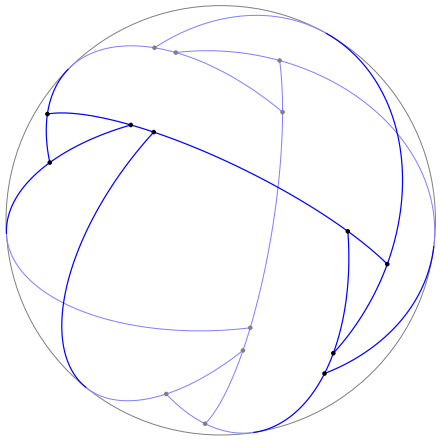


Figure 10: A Diagram with eight arcs and four swirls

5 Swirling Diagrams

This section is devoted to a special type of Diagrams whose arcs always meet forming swirls. Patterns arising in these Diagrams are found in modular origami, globe knots, rattan balls, etc.

Definition 6 *A Diagram is* swirling *if every arc is part of two swirls.*

An example of a swirling Diagram is found in Figure 2; further examples are in Figure 14. All of these Diagrams were obtained from convex polyhedra or, equivalently, from convex tilings of the sphere, by a process that we call *swirlification*.

Definition 7 *A subdivision of the unit sphere into strictly convex spherical polygons is* swirlable *if each polygon of the subdivision has an even number of edges.*

Proposition 8 *A subdivision of the unit sphere into strictly convex spherical polygons is swirlable if and only if its 1-skeleton is bipartite.*

Proof. The 1-skeleton is bipartite if and only if its has no odd cycles, which is true if and only if each face has an even number of edges. □

Hence, we can always deform the 1-skeleton of a swirlable tiling, turning each of its vertices into a swirl, going clockwise or counterclockwise according to the bipartition of the 1-skeleton. Conversely, by "shrinking" the eye of each swirl of a swirling Diagram to a point, one obtains a swirlable subdivision of the sphere.

In other words, the swirlification operation establishes a natural correspondence between swirling Diagrams and swirlable subdivisions of the sphere.

Theorem 3 *Every swirling Diagram is the swirlification of a swirlable subdivision of the sphere.* □

Note that we can also obtain a swirlable subdivision of the sphere by taking the dual of a subdivision whose vertices have even degree, or by truncating it. More generally, we have the following.

Proposition 9 *A subdivision of the sphere is swirlable if and only if its truncated dual is swirlable.* □

6 Uniform Diagrams

We now turn to a class of Diagrams that generalizes the swirling ones.

Definition 8 *A Diagram is* uniform *if every arc blocks exactly two arcs.*

Proposition 10 *A Diagram is uniform if and only if every arc blocks at most (respectively, at least) two arcs.*

Proof. By Proposition 3, each arc hits exactly two distinct arcs. Hence, each arc blocks two arcs on average. Thus, if every arc blocks at most two arcs (or at least two arcs), it must block exactly two arcs. □

Proposition 11 *Every uniform Diagram is irreducible.*

Proof. Let $\mathcal{D}$ be a uniform Diagram, and assume that there is a proper subset of arcs $\mathcal{D}' \subset \mathcal{D}$ that is itself a Diagram. By Proposition 6, $\mathcal{D}$ is connected; thus, removing arcs from $\mathcal{D}$ causes some arcs to block fewer than two arcs. Since $\mathcal{D}$ is uniform, it follows that the arcs of $\mathcal{D}'$ block fewer than two arcs on average, contradicting Proposition 3. □

Corollary 2 *In a uniform Diagram, the eye of each swirl coincides with a single tile.*

Proof. If the interior of the eye of a swirl contains some arcs, then such arcs can be removed without violating the Diagram axioms. Hence, such a Diagram is not irreducible, and by Proposition 11 it cannot be uniform. □

Theorem 4 *Every swirling Diagram is uniform.*

Proof. In a swirling Diagram, each arc a is part of two distinct swirls. By Theorem 1, these two swirls share no arcs other than a, and hence a must block one arc from each of them. Therefore, every arc in a swirling Diagram blocks at least two arcs, and by Proposition 10 the Diagram is uniform. □

The converse of Theorem 4 is not true in general, as Figure 11 shows.

Definition 9 *An endpoint of an arc of a Diagram is called a* non-swirling vertex *if it is not incident to the eye of any swirl. A walk on a Diagram is* non-swirling *if it only touches non-swirling vertices and, whenever it touches an arc, it follows it until it reaches one of its endpoints, without touching any other arc along the way. A cyclic non-swirling walk is called a* non-swirling cycle.

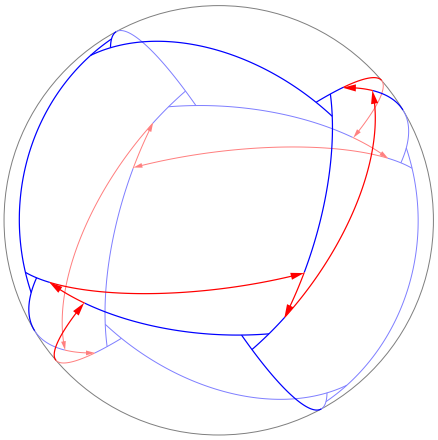


Figure 11: A uniform Diagram that is not swirling

Observe that there is a non-swirling cycle that covers all the non-swirling vertices of the Diagram in Figure 11 (drawn in red). This is not a coincidence.

Theorem 5 *In any uniform Diagram, all non-swirling vertices are covered by disjoint non-swirling cycles.*

Proof. Consider a non-swirling walk W on a uniform Diagram terminating at a non-swirling vertex p_i, endpoint of an arc a_i, as Figure 12 illustrates. We will prove that W can be extended to a longer non-swirling walk in a unique way.

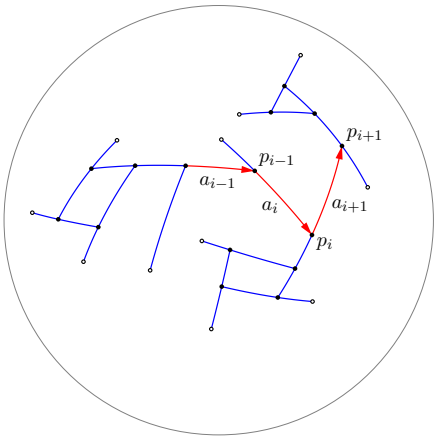


Figure 12: Proof of Theorem 5

Let a_{i+1} be the arc that blocks a_i at p_i. Since exactly two arcs feed into a_{i+1}, there is exactly one endpoint of a_{i+1}, say p_{i+1}, that can be reached from p_i without touching any arc other than a_{i+1}.

By definition of non-swirling walk, p_{i+1} can be used to extend W if and only if it is a non-swirling vertex. However, if p_{i+1} were incident to a swirl's eye E, then an arc of that swirl would either hit a_{i+1} between p_i and p_{i+1}, contradicting the fact that a_{i+1} blocks exactly two arcs, or it would hit a_{i+1} on the other side of p_i,

implying that E contains the arc a_i in its interior, which contradicts Corollary 2.

Hence, W can be extended uniquely to a non-swirling walk. By a similar reasoning, we argue that W can also be uniquely extended backwards to a non-swirling walk. Thus, W is part of a unique non-swirling cycle. Now we conclude the proof by inductively repeating the same argument with any remaining non-swirling vertices. □

We can construct uniform Diagrams with any number of arbitrarily long non-swirling cycles. An example with two non-swirling cycles is shown in Figure 13.

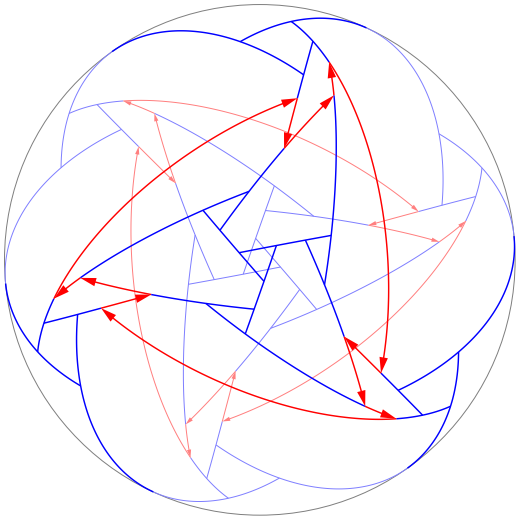


Figure 13: A uniform Diagram with two non-swirling cycles

7 Conclusions

We introduced the theory of Spherical Occlusion Diagrams and studied their basic properties, while also discussing some applications to visibility-related problems in discrete and computational geometry.

Although we strongly believe Conjecture 1 to be true, a related and more subtle question can be asked, inspired by previous work on weaving patterns [1, 7]. Namely, whether for every Diagram $\mathcal{D}$ there is a *combinatorially equivalent* Diagram $\mathcal{D}'$ and a set $\mathcal{P}$ of internally disjoint polygons such that $\mathcal{D}' = S_{\mathcal{P}}$. In other words, does every class of combinatorially equivalent Diagrams contain a realizable instance?

We have introduced three remarkable families of Diagrams: irreducible, uniform, and swirling. We proved that all swirling Diagrams are uniform, and all uniform Diagrams are irreducible; moreover, Theorem 5 reveals a deeper structural connection between swirling and uniform Diagrams. A complementary observation is that it seems to be possible to systematically transform any uniform Diagram into a swirling Diagram by "sliding" arcs' endpoints along other arcs and "merging" coincident arcs. Making this observation rigorous is left as a direction for future work.

More generally, we may wonder which Diagrams can be transformed into swirling ones by sequences of elementary operations on arcs (defining suitable "elementary operations" is in itself an open problem). The Diagram in Figure 10 shows that the question is not trivial. Indeed, this is the unique configuration of any Diagram with eight or fewer arcs; since the Diagram itself is not swirling, it cannot be transformed into a swirling one by means of operations that only rearrange or merge arcs.

Acknowledgments. The author is grateful to C. D. Tóth, J. Urrutia, and M. Yamashita for interesting discussions. The author also thanks the anonymous reviewers for improving the readability of this paper.

References

[1] S. Basu, R. Dhandapani, and R. Pollack. On the Realizable Weaving Patterns of Polynomial Curves in $\mathbb{R}^3$. In *Proceedings of the 12th International Symposium on Graph Drawing (GD 2004)*, pp. 36–42, 2004.

[2] N. M. Benbernou, E. D. Demaine, M. L. Demaine, A. Kurdia, J. O'Rourke, G. T. Toussaint, J. Urrutia, and G. Viglietta. Edge-Guarding Orthogonal Polyhedra. In *Proceedings of the 23rd Canadian Conference on Computational Geometry (CCCG)*, pp. 461–466, 2011.

[3] J. Cano, C. D. Tóth, J. Urrutia, and G. Viglietta. Edge Guards for Polyhedra in Three-Space. *Computational Geometry: Theory and Applications*, vol. 104, art. 101859, 2022.

[4] D. Eppstein, E. Mumford, B. Speckmann, and K. Verbeek. Area-Universal and Constrained Rectangular Layouts. *SIAM Journal on Computing*, vol. 41, no. 3, pp. 537–564, 2012.

[5] A. I. Merino, T. Mütze. Efficient Generation of Rectangulations via Permutation Languages. In *37th International Symposium on Computational Geometry (SoCG 2021)*, pp. 54:1–54:18, 2021.

[6] J. O'Rourke. Visibility. In J. E. Goodman, J. O'Rourke, and C. D. Tóth, editors, *Handbook of Discrete and Computational Geometry*, pp. 875–896, CRC Press, 2017.

[7] J. Pach, R. Pollack, and E. Welzl. Weaving Patterns of Lines and Line Segments in Space. *Algorithmica*, vol. 9, no. 6, pp. 561–571, 1993.

[8] C. D. Tóth, J. Urrutia, and G. Viglietta. Minimizing Visible Edges in Polyhedra. In *Proceedings of the 23rd Thailand-Japan Conference on Discrete and Computational Geometry, Graphs, and Games (TJCDCGGG 2020+1)*, pp. 70–71, 2021.

[9] G. Viglietta. Optimally Guarding 2-Reflex Orthogonal Polyhedra by Reflex Edge Guards. *Computational Geometry: Theory and Applications*, vol. 86, art. 101589, 2020.

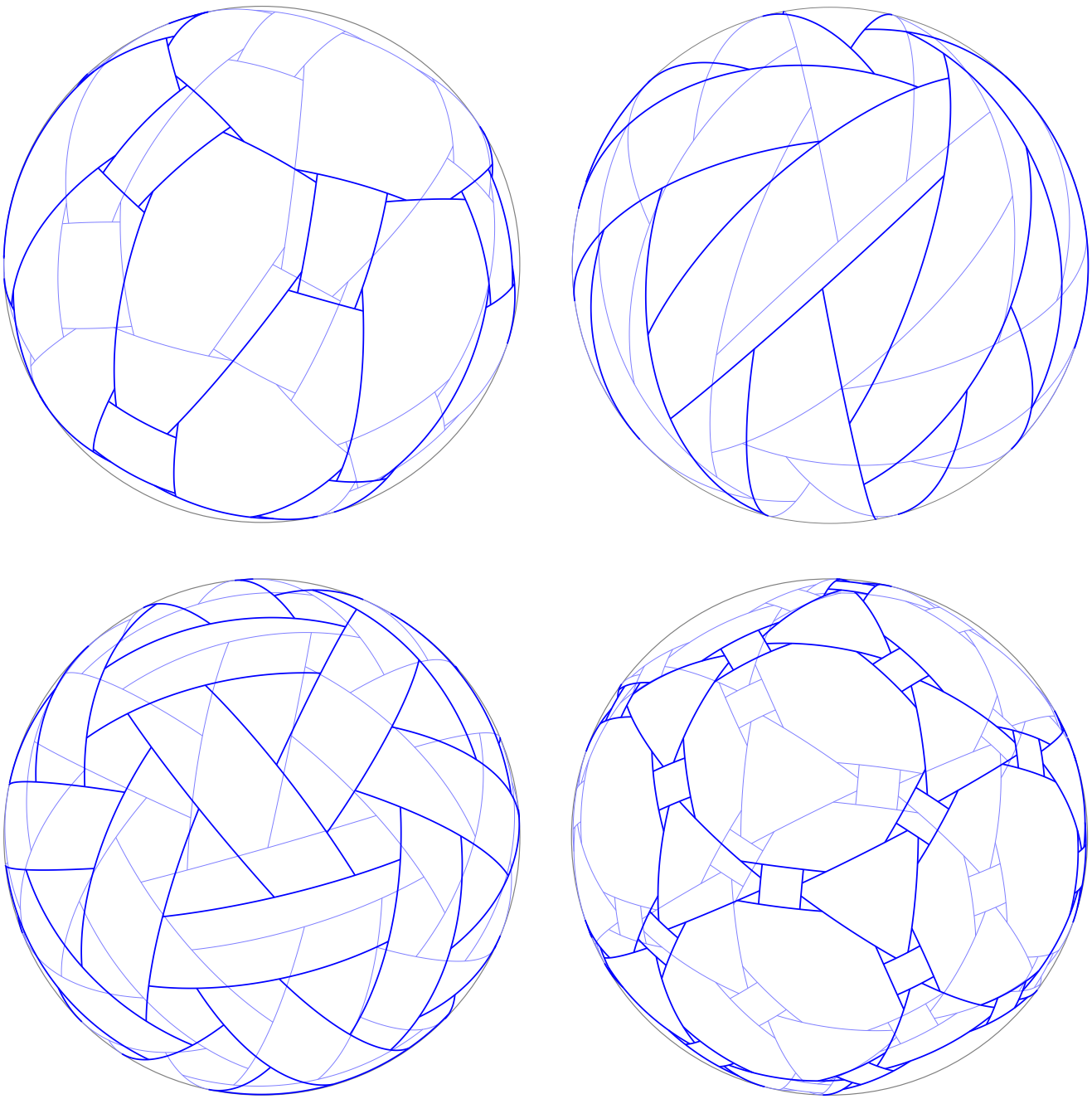


Figure 14: Examples of the swirlification method developed in Section 5 to produce swirling Diagrams from convex polyhedra with a bipartite 1-skeleton (or, equivalently, from swirlable subdivisions of the unit sphere). The pictures show swirling Diagrams resulting from a truncated antiprism, a trapezohedron, a rhombic triacontahedron, and a truncated icosidodecahedron, respectively.

Using Existential Theory of the Reals to Bound VC Dimension

Austin Watkins* Jeff M. Phillips†

Abstract

We provide new bounds on the VC dimension of range spaces beyond logical compositions of polynomials and other discrete geometric shapes. Our results address the VC dimension of a seemingly simple class of range spaces we call inflated polynomials, which are defined as the Minkowski sum of a polynomial and a ball; in $\mathbb{R}^2$ with degree p the VC dimension is $\Theta(p)$, and in $\mathbb{R}^d$ the bound is $O(dp^{O(d)})$. This addresses natural questions on learnability in the adversarially-robust setting for polynomial classifiers and of polynomially-defined trajectories. We use a connection between algebraic geometry and classic circuit-based approaches of bounding the VC dimension to derive our results. We believe this connection and our general results may find other applications in learning theory, range searching, and other aspects of computational geometry where the VC dimension plays a key role.

1 Introduction

This paper studies the VC dimension and learnability of regions defined by offsets from polynomial curves and surfaces, which we call *inflated polynomials*. These offsets are no longer polynomial and so little to nothing is known about the learnability of a large family of classes that arise this way. We provide new VC dimension bounds for this family of objects by a connection to the existential theory of the reals. Application of these inflated polynomials are broad and we highlight implications in sweeping out the region around a polynomial curve and in adversarially-robust learning.

The Vapnik-Chervonenkis-dimension (VC dimension) [37] is the central combinatorial complexity score for a range space or a function class. It intricately ties into many aspects of learning theory [1] where it bounds how many data samples are needed to learn over a function class, model theory [2, 4] where it ties into the rich structure of algebraic geometry, big data [15] where it governs the size and runtime for creating coresets, computational geometry [19, 5] where it describes the size of a hitting set, and data structures [8] where it characterizes a class of ranges that admit a near-linear size data structure which allows for sub-linear time range queries. In this paper, we significantly generalize the approaches to analyze function classes defined through non-polynomial and existential formulations.

Inflated polynomials. In particular, in this paper we focus on ranges defined by the Minkowski sum of a Euclidean ball and a polynomial; we call these *inflated polynomials*. A simple example of an inflated parabola in $\mathbb{R}^2$ is shown in Figure 1. In particular, observe that the boundary of this shape is *not* a polynomial, as clearly evidenced by the cusp point, directly above the minimum. Thus, due to this non-polynomial nature, among other complexities, the VC dimension of such shapes have no known bound [9]. Let us highlight two other grounded scenarios where such questions arise.

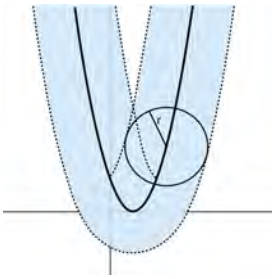


Figure 1: The inflated polynomial, shown in blue, of $(x-1)^2$ with radius r.

First, consider learning a polynomial classifier robustly, in the sense that it should protect against adversarial examples [35]. Typically, the goal is to learn a classifier so no, or few, correctly labeled examples can cross the decision boundary with small perturbations. While this is perhaps most problematic in complex classifiers [35, 18], learnability of robust classifiers has mostly been studied formally [34, 14, 29, 25, 16, 10] for linear (or near-linear) classifiers and/or when data classes have specific and known (uniform, Gaussian, accurate under Gaussian noise) distributions. While polynomial (and other kernel classifiers) can be "linearized" so the inner-product acts as a linear dot-product, this distance no longer measures the amount of perturbation required in the input space needed for a data point to cross the decision boundary. In particular, one goal is to learn a perfect polynomial classifier so that no data points are within a distance r of the decision boundary (measured using Euclidean distance in the input space). As we elaborate in Section 4, the number of samples needed to ensure that such a perfect and r-robust polynomial classifier on the sample will ensure

*School of Computer Science, Johns Hopkins University, awatki29@jhu.edu; This research was supported, in part, by DARPA GARD award HR00112020004 and NSF CAREER award IIS-1943251.

†University of Utah, School of Computing, jeffp@cs.utah.edu; Thanks to NSF IIS-1816149, CCF-2115677, and Visa Research.

at most ε-fraction of all data (with probability $1-\delta$) will be at least a distance r from the polynomial decision boundary is $O(\frac{\nu}{\varepsilon}\log\frac{\nu}{\varepsilon\delta})$, where ν is the VC dimension of the inflated polynomial around the decision boundary.

Second, consider a drone which moves through a neighborhood and transmits malicious computer code to Wi-Fi routers. We do not know the exact drone trajectory, but can model it as a polynomial curve of degree p, and know it is most effective within 30 meters. Thus its affected range is an inflated polynomial curve. How many Wi-Fi routers in the neighborhood do we need to randomly inspect to accurately estimate the drone path (i.e., we can predict the probability of malicious code within ε, with probability $1-\delta$)? The number of samples is $O(\frac{1}{\varepsilon^2}(\nu+\log\frac{1}{\delta}))$ where ν is the VC dimension of this inflated polynomial.

Results and techniques. In this paper we develop a family of techniques to bound the VC dimension of complex range spaces and apply them to the inflated polynomials and existentially defined sets. We build on traditional techniques for bounding the VC dimension [1, 17], which prior-to-this-work were restricted to polynomially defined sets, a few other specific options like sigmoid, and their compositions. This approach provides a set of simple operations and bounds the VC dimension by the number of such operations needed to determine inclusion in the set in question. For our work, as in [17], we combine this approach with a distinct set of tools from decision algorithms in logic, algebraic geometry, and the existential theory of the reals. While ultimately the proofs are simple; they rely on an observation that the computation model associated with most algebraic geometry is compatible with the simple operations of [1]. This allows us to bound the VC dimension of inflated polynomials and existentially-defined sets. Our main result is as follows:

Theorem 1 *The VC dimension of inflated polynomials in $\mathbb{R}^d$ of degree p is $O(dp^{O(d)})$, and for univariate polynomials, the bound is $\Theta(p)$.*

This provides the specific bound needed to address the two applications (polynomial path learning for detecting Wi-Fi manipulation and adversarially robust polynomial separators) highlighted above. In particular, for adversarially robust learning, we view this as an essential step in how to link the geometry of the decision boundary to the input space.

2 Background, Definitions, and Prior Work

As this paper unites several technical areas, we start with a fair number of definitions.

Polynomials. Central to our study are *real polynomials*, that is polynomials with real coefficients. When there are d variables $x_1,\ldots,x_d$, we denote these as $\mathbb{R}[x_1,\ldots,x_d]$. The *degree* of the polynomial is the maximum sum of exponents of the variables in any monomial. Such polynomials define functions f from $\mathbb{R}^d\to\mathbb{R}$. Hence they can also be viewed as d-dimensional objects in $\mathbb{R}^{d+1}$ which divide $\mathbb{R}^{d+1}$ into 3 sets. For $(x_1,\ldots,x_d,y)\in\mathbb{R}^{d+1}$ with $x=(x_1,\ldots,x_d)$, then it can be "below" if $y<f(x)$, "above" if $y>f(x)$, or "on" if $y=f(x)$.

The *Minkowski sum* between two sets $A,B\subset\mathbb{R}^{d+1}$ is the set of all pairwise additions between A and B, $\{a+b\mid a\in A,b\in B\}$, and is denoted $A\oplus B$. Let $\mathcal{M}_p^d$ be the set of all these *inflated polynomials* constructed as the Minkowski sum of the points "on" a polynomial (the set A) and a ball (the set B). Let $\mathcal{B}_r^d$ be the set of d-dimensional balls with radius r. That is, $\mathcal{M}_p^d=\{P\oplus B\mid B\in\mathcal{B}_r^{d+1},r\in\mathbb{R},P\in\mathbb{R}[x_1,\ldots,x_d]$ of degree at most $p\}$.

Range spaces and VC dimension. A *range space* is a tuple $(X,\mathcal{R})$, where X is called the *ground set* and $\mathcal{R}$ is called the *range set*, where all sets in the range set are a subset of the ground set. $\mathcal{R}$ is often defined in terms of geometric objects. $\mathcal{R}$ could be the set of disks for $X=\mathbb{R}^2$, intervals on $X=\mathbb{R}$, linear halfspaces on $X=\mathbb{R}^d$, or as points below (or on) polynomials in $X=\mathbb{R}^d$. When $X\subset\mathbb{R}^d$ is set of points, then these example ranges $\mathcal{R}$ are the induced subset of X contained in some such shape.

Similar to a restriction over a family of functions to a subset of the domain, we will define the projection of range space $\mathcal{R}$ onto $Y\subset X$ as $\mathcal{R}_{|Y}:=\{R\cap Y\mid R\in\mathcal{R}\}$. For a range space $(X,\mathcal{R})$, if the projection $\mathcal{R}_{|Y}$ contains all subsets of Y, then $\mathcal{R}$ *shatters* Y. The *VC dimension* of $(X,\mathcal{R})$ is the maximum cardinality of any shattered subset of X.

In this paper we mostly consider real ground sets $X=\mathbb{R}^d$ or $X\subset\mathbb{R}^d$, in which case a range space is defined by its range sets, and thus for simplicity we refer to the VC dimension of range sets, where the real ground set and corresponding range space are implicit. Simple examples of VC dimension ν include: for disks in $\mathbb{R}^2$ then $\nu=3$, for intervals in $\mathbb{R}^2$ then $\nu=2$, for linear halfspaces in $\mathbb{R}^d$ then $\nu=d+1$, and for polynomials of degree p in $\mathbb{R}^d$ then $\nu=O(d^p)$. For polynomials of *any* degree in $\mathbb{R}^d$ then ν is unbounded – it is infinite.

2.1 Sample Complexity

For a domain X consider a classifier function $h:X\to\{0,1\}$, it maps any element $x\in X$ to either 0 or 1. Then given a probability distribution μ on $Z=X\times\{0,1\}$, the *error of* h with respect to μ, written $\mathrm{er}_\mu(h)$, is the probability that $(x,y)\sim\mu$ such that $h(x)\neq y$. The goal of classification, for some family of classifiers $\mathcal{H}$ and

some μ, is to find an $h \in \mathcal{H}$ with $\text{er}_\mu(h)$ as small as possible.

The other side of this seeks to minimize the number of samples required to achieve a certain error on a learned classifier $h \in \mathcal{H}$ for some family $\mathcal{H}$. For a set of m samples $P = (x_1, y_1), \ldots, (x_m, y_m)$ drawn i.i.d. from μ, let μ_P be the sample distribution induced by this set. Let $\varepsilon, \delta \in (0, 1)$. The *sample complexity* is defined as the smallest m such that $\text{er}_{\mu_P}(h) \leq \text{er}_\mu(h) + \varepsilon$ holds with probability at least $1 - \delta$ for all $h \in \mathcal{H}$. Then for parameters $\varepsilon, \delta \in (0, 1)$ we seek to minimize m so that for all $h \in \mathcal{H}$ we have $\text{er}_{\mu_P}(h) \leq \text{er}_\mu(h) + \varepsilon$, with probability at least $1 - \delta$. Since this condition holds for $h^* := \inf_{h \in \mathcal{H}} \text{er}_{\mu_P}(h)$, we can then "learn" h^* on P and know it will ε-approximately hold (with probability at least $1 - \delta$) on μ.

The family of classifiers $\mathcal{H}$ defines a range set, and when $X \subset \mathbb{R}^d$, the VC dimension ν of this range space $(X, \mathcal{H})$ determines the sample complexity m. Vapnik and Chervonenkis [37] and refined by [23, 1] show that $m = O(\frac{1}{\varepsilon^2}(\nu + \log \frac{1}{\delta}))$ samples are sufficient.

When a perfect classifier h exists, one where $\text{er}_\mu(h) = 0$, the sample complexity is lower; using only $m = O(\frac{\nu}{\varepsilon} \log \frac{\nu}{\varepsilon\delta})$ samples is sufficient [20].

2.2 Methods of Bounding VC Dimension

There are two powerful methods for bounding complex range spaces. The first is via composition arguments, where we break (via unions and intersections) a complex range space into simple ranges for which bounds are known, and then bound the complex range by aggregating the effect of the simple ranges. The second is via circuit arguments, where computing set inclusion within a computational framework is used to derive an upper bound for the range space.

Composition argument. Let $(X, \mathcal{R}_1), \ldots, (X, \mathcal{R}_s)$ be a set of range spaces with VC dimension $\nu_1, \ldots, \nu_s$, respectively. Let $f(r_1, \ldots, r_s)$ be a function defined element-wise over the domain X (i.e., unions and intersections), that maps any s-tuple of sets $r_1 \in \mathcal{R}^1, \ldots, r_s \in \mathcal{R}^s$ into a subset of X. That is, f corresponds with a fixed logical formula (i.e., composed of $\vee$s and $\wedge$s) over s binary values determined by if $x \in X$ is in each range r_i. A element $x \in X$ is in the composite range $f(r_1, \ldots, r_s)$ if the logical function returns 1. This process defines a composition range set $\mathcal{R}^\oplus = \{f(r_1, \ldots, r_s) \mid r_1 \in \mathcal{R}_1, \ldots, r_s \in \mathcal{R}_s\}$. Har-Peled [19][Theorem 5.22] shows for the VC dimension of the associated range space $(X, \mathcal{R}^\oplus)$ is bounded by $O(s\nu(1 + \log s))$ where $\nu = \max\{\nu_i\}_{i=1}^s$.

Circuit argument. Goldberg and Jerrum [17], and slightly generalized to this form [1][Theorem 8.4], uses a circuit of *simple operations*, defined to consist of

- the arithmetic operations $+, -, \times$, and $/$ on real numbers,
- jumps conditioned on $>, \geq, <, \leq, =$ and $\neq$ as comparisons of real numbers, and
- output 0 or 1.

Then suppose h_a is a function from $\mathbb{R}^d$ to $\{0, 1\}$ parameterized by $a \in \mathbb{R}^k$. Let h_a define a range $H_a = \{x \in X \subset \mathbb{R}^d \mid h_a(x) = 1\}$ from the associated family of ranges $\mathcal{H} = \{R_a \mid a \in \mathbb{R}^k\}$. Suppose that h_a can be computed by an algorithm that takes as input the pair $(x, a) \in \mathbb{R}^d \times \mathbb{R}^k$ and returns $h_a(x)$ after no more than t simple operations. Then the VC dimension of $(\mathbb{R}^d, \mathcal{H})$ is at most $4d(t + 2)$.

While this approach (perhaps in combination with composition arguments) seems like it can be applied to handle most geometrically defined range spaces (say including inflated polynomials), there is an important omission from the simple operations: the square root operation. A square root is needed, for instance, to encode distance in a radius r ball. More importantly, simple composition arguments cannot be made, since an inflated polynomial is a union of an infinite number of these balls. Towards addressing some such goals (with respect to range spaces defined by polynomial curves), [13][Lemma 12] provides a special case where one can handle a square root inside of the circuit argument: Consider values $a, b, c, d \in \mathbb{R}$ with $b, d \geq 0$, then one can compute the truth values of $a + \sqrt{b} \leq c + \sqrt{d}$ and $a + \sqrt{b} \geq c + \sqrt{d}$ using $O(1)$ simple operations. However, restricting to this use of the square root, to apply this to general range sets in a metric space where the square root is needed, such as inflated polynomials, one would need to perform this comparison at an infinite number of points, or a composition of an infinite number of sets. So we will still require more powerful machinery from the existential theory of the reals in real algebraic geometry.

2.3 Algorithms in Real Algebraic Geometry

We next focus on the interpretation of algebraic geometry through the perspective of solutions to polynomial systems. We will mostly follow notation from [3].

P-atoms for P-formulas. For our purposes (specifying the field to be $\mathbb{R}$), a $\mathcal{P}$-*atom* is a polynomial equality or inequality; if $P \in \mathbb{R}[x_1, \ldots, x_d]$ then the options are $P = 0$, $P \neq 0$, $P > 0$, or $P < 0$. Similarly, a $\mathcal{P}$-*formula* is a combination of $\wedge, \vee, \neg, \forall, \exists$ with $\mathcal{P}$-atoms to form a logical statement. For example a $\mathcal{P}$-formula could be $\forall x \exists y (x^2 y + 2 > 0 \wedge y \leq 0)$. A *semialgebraic set* is a finite union of polynomial equalities and polynomial inequalities. For instance, $x^2 - y \leq 0 \cup x - y > 0$ is a semialgebraic set in the plane ($\mathbb{R}^2$). In [3] they detail a large number of algorithms on real polynomials. We will

use two key results from this work: Tarski queries and existential decidability over a subset of $\mathcal{P}$-formulas.

Arithmetic operations for algebraic geometry. As detailed in [4], the complexity of algorithms within algebraic geometry is given in terms of specified allowable arithmetic operations between elements of a chosen set. These operations and a chosen set define the structure of an algorithm. The structures which concern us are a ring, an integral domain, and an ordered integral domain. A *ring* structure defines the allowable operations to be $+, -, \times,$ and $= 0$, where $= 0$ is the unary operation of deciding if an element in the ring is zero. An *integral domain* structure defines, in addition to the ring structure, exact division $/$, between two elements given that division will be in the integral domain. An *ordered integral domain* structure defines, in addition to the integral domain structure, comparison between elements with $>, =,$ and $<$ operations.

Importantly, $\mathbb{R}$ is an ordered integral domain, and the "simple operations" in the circuit argument [17, 1] include all allowable arithmetic operations for a ordered integral domain. Hence a bound on arithmetic operations provides a bound on simple operations.

Univariate Tarski queries. The first real algebraic geometry result we use is Pollack's [3] Algorithm 9.5 for counting roots of a univariate polynomial. The cited form includes an extra parameter $Q \in \mathbb{R}[x]$ that represents a more general query called a Tarski query. By taking $Q = 1$ then a Tarski query is equivalent to computing the number of roots as given in Sturm's theorem,[1] which is specifically for univariate polynomials. Ultimately, a *univariate Tarski query* can take in a univariate polynomial $P \in \mathbb{R}[x] \setminus \{0\}$ (that is, not including the trivial 0 polynomial), it outputs the number of elements in $\{x \in \mathbb{R} \mid P(x) = 0\}$ using $O(p + 1)$ simple operations.

Decidability. Next we will use a result regarding decidability, specifically over the language that is the theory of real closed fields. The Tarski–Seidenberg Theorem implies that the theory of the real closed fields is decidable. Yet it was only with Collins's [11] use of cylindrical algebraic decomposition that a doubly exponential bound was found. There is a simpler problem which only allows for existential quantifiers. This problem is known as the existential theory of the reals, with the first singly exponential complexity provided by Renegar [31].

Consider first-order logical statements in the following form: $\exists x_1, \ldots \exists x_d\ F(x_1, \ldots, x_d)$ where $F(x_1, \ldots, x_d)$ is a quantifier free $\mathcal{P}$-formula. Determining if that statement is true or false is called the *decision problem for the existential theory of the reals*. When $\mathcal{P} \subset \mathbb{R}[x_1, \ldots, x_d]$ is a finite set of s polynomials each of degree at most p, then there is an algorithm to decide the truth of $\exists x_1, \ldots, \exists x_d\ F(x_1, \ldots, x_d)$ using $s^{d+1}p^{O(d)}$ simple operations.

[1] see Theorem 2.50 and Theorem 2.61 in [3]

3 New VC Dimension Bounds

We begin with a two-dimensional bound for univariate inflated polynomials, based on Tarski queries. Then we generalize to d-dimensional inflated polynomials using Renegar's algorithm. We provide a lower bound of the same order, which matches when $d = 1$.

3.1 Upper Bound of VC Dimension for Inflated Polynomials

We first translate inflated polynomials into the language of existential algebraic geometry. Consider range space $(\mathbb{R}^{d+1}, \mathcal{M}_p^d)$, and a query point $w \in \mathbb{R}^{d+1}$, and inflated polynomial $P_r \in \mathcal{M}_p^d$. Then w is in P_r if and only if $\exists x_0 \in P(\mathbb{R}^d)(\|w - (x_0, P(x_0))\|_2 \leq r)$ where P is the polynomial of the inflated polynomial P_r, and $(x_0, P(x_0))$ is a point on that polynomial in $\mathbb{R}^{d+1}$. A univariate degree-p polynomial curve in $\mathbb{R}^2$ is an element of $(\mathbb{R}^2, \mathcal{M}_p^1)$, which is the domain of our first upper bound.

Theorem 2 *The range space $(\mathbb{R}^2, \mathcal{M}_p^1)$, where $\mathcal{M}_p^1$ is composed of only univariate inflated polynomials, has VC dimension $O(p)$.*

Proof. We must find a point on the polynomial close enough to $w = (w_1, w_2)$. And $\|w - (x, P(x))\|_2 \leq r$ implies $(w_1 - x)^2 + (w_2 - P(x))^2 - r^2 \leq 0$. Notice that this is a polynomial inequality. As P is defined for all $\mathbb{R}$, the distance is unbounded from above and, due to the squared terms, that the final polynomial has even degree. Therefore, to determine if there exists an x that satisfies the inequality (≤ 0) above it is sufficient to count roots of the polynomial. That is, since the number of roots is the number of times a set satisfies $= 0$, and it is $+\infty$ as $x \to \{-\infty, +\infty\}$, then if the number of roots is non-zero, there must exist a point x where the ≤ 0 condition is satisfied. Using univariate Tarski queries, we can count the real roots of univariate polynomials in $O(p + 1)$ simple operations, with p the degree of P.

Then we can use the circuit argument with a bound on the number of free variables $d = 1$ and depth in simple operations of the circuit as $t = O(p + 1)$. Hence the VC dimension is $4d(t + 2) = O(p)$. □

Now we will generalize to multivariate polynomials by using a decision algorithm.

Theorem 3 *The range space $(\mathbb{R}^{d+1}, \mathcal{M}_p^d)$ of inflated polynomials in $\mathbb{R}^{d+1}$ has VC dimension $O(dp^{O(d)})$.*

Proof. Consider an inflated polynomial P_r of degree p and fix $w \in \mathbb{R}^{d+1}$. We must find a point $x \in \mathbb{R}^d$

on the polynomial close enough to w, satisfying $\|w - (x, P(x))\|_2 \leq r$, and equivalently $(w_1 - x_1)^2 + \ldots + (w_d - x_d)^2 + (w_{d+1} - P(x))^2 - r^2 \leq 0$. As before this is a polynomial inequality only with more free variables. Now we will invoke the existential theory of the reals decidability result of Renegar [31]. To do this we need to write the inequality into the logical structure desired by the algorithm.

$$
\begin{aligned}
&(\exists x_1) \ldots (\exists x_d) \\
&\Big((w_1 - x_1)^2 + \ldots + (w_d - x_d)^2 + (w_{d+1} - P(x))^2 - r^2 = 0 \\
&\vee (w_1 - x_1)^2 + \ldots + (w_d - x_d)^2 + (w_{d+1} - P(x))^2 - r^2 < 0 \Big) \\
&\equiv (\exists x_1) \ldots (\exists x_d) \\
&\Big((w_1 - x_1)^2 + \ldots + (w_d - x_d)^2 + (w_{d+1} - P(x))^2 - r^2 = 0 \Big) \\
&\vee (\exists x_1) \ldots (\exists x_d) \\
&\Big((w_1 - x_1)^2 + \ldots + (w_d - x_d)^2 + (w_{d+1} - P(x))^2 - r^2 < 0 \Big)
\end{aligned}
$$

Thus we have two d-variate polynomials we must evaluate. The existential theory of the reals algorithm takes $O(p^{O(d)})$ simple operations to evaluate for each $\mathcal{P}$-atom. Now we will use a circuit argument with d free variables/dimensions and $t = p^{O(d)}$ simple operations. Thus the VC dimension for one $\mathcal{P}$-atom is $O(dp^{O(d)})$. Then using a composition argument, we can combine these together increasing the bound only a constant factor. □

3.2 Lower Bound of VC Dimension for Inflated Polynomials via Interpolation

We show a lower bound of $\binom{d+p}{p}$ where p is the degree of the polynomial and d is the number of variables in the polynomial. The proof uses that a polynomial $P \in \mathbb{R}[x_1, \ldots, x_d]$ can uniquely interpolate $\binom{d+p}{p}$ points in $\mathbb{R}^{d+1}$. With some perturbation, we can always shatter sets of this size.

Theorem 4 *The lower bound of the VC dimension of* $(\mathbb{R}^{d+1}, \mathcal{M}_p^d)$ *is* $\binom{d+p}{p}$.

Proof. Given $(\mathbb{R}^{d+1}, \mathcal{M}_p^d)$ consider X, a set of points in $\mathbb{R}^{d+1}$ where $|X| = \binom{d+p}{p}$ points such that the sample matrix's determinant, as in [33], is nonzero. Let Z be a non empty element of the power set of X. To intersect all points in Z and none in $X \setminus Z$ we interpolate over Z and $\binom{d+p}{p} - |Z|$ *perturbed* points in $X \setminus Z$. We will perturb these points by adding ε to the final coordinate of the points in $X \setminus Z$. Let $P_r \in \mathcal{M}_p^d$ and P be the polynomial at the center of P_r. Recall that polynomial P is a function from $\mathbb{R}^d \to \mathbb{R}$. If we then interpolate using Lagrange interpolation detailed in [33] over the Z and the perturbed points of $X \setminus Z$ the function will not interpolate the original points of $X \setminus Z$. We know that perturbing these points does not affect the existence of the interpolant since changing the final coordinate of our set does not change the determinant of the sample matrix. We can then take r sufficiently small so that P_r does not contain any element of $X \setminus Z$. Therefore as we can interpolate any subset of $\binom{d+p}{p}$ points in this way the VC dimension of the range space must be at least $\binom{d+p}{p}$. □

If we are dealing with univariate polynomials then the curve lives in $\mathbb{R}^2$ and can shatter $\binom{1+p}{p} = p+1$ points by the above theorem. Note that this is a lower bound due to the fact that we are not using the expressiveness of the radius of the inflated polynomial to our advantage. Yet as the modification of the radius affects the inflated polynomial globally, not just locally, its expressiveness is limited.

Comment on tightness. We have an upper bound and a lower bound on the VC dimension of the inflated polynomial range space $(\mathbb{R}^{d+1}, \mathcal{M}_p^d)$. When p is constant then $\binom{d+p}{p} = \Theta(d^p)$ and when d is constant, then $\binom{d+p}{p} = \binom{d+p}{d} = \Theta(p^d)$. So for constant d, we have upper bound of $O(p^{O(d)})$ and lower bound of $\Omega(p^d)$. For $d = 1$, we have established $\Theta(p)$ VC dimension.

4 Application in Robust Adversarial Learning

We highlight an application in robust adversarial learning. Others implications can be found in Appendix A and by connecting to results in coresets [15], hitting sets [5], and range searching [8].

Adversarial attacks on classifiers refers to when someone makes small perturbations to input data so it fools a classifier. This phenomenon has been demonstrated in images, question answering, voice recognition, among other areas [35, 7, 34, 18]. Current defenses against adversarial robustness [24, 6, 12, 26, 27, 29] may have undesirable consequences, such as decreasing test accuracy, leading some to investigate a potential trade-off between accuracy and robustness [39, 36]. Yet, further investigation on robustness prevention methods and the separability of image datasets show that accuracy and robustness are obtainable for real-life data [38, 30]. Also, random smoothing of a classifier, a defense in which you randomly sample around points within the data to build robustness [10, 32, 22] has been effective in low dimensions yet may be untenable in high dimension [21].

To formalize this problem, we need to consider a classifier $h : \mathbb{R}^d \to \{-1, 1\}$. Let $B_\gamma(x) = \{x' \in \mathbb{R}^d \mid \|x' - x\|_2 \leq \gamma\}$ be the l_2 ball of radius γ around x, which describes the allowable perturbations around a data point $x \in \mathbb{R}^d$. We say a point $(x, y) \in \mathbb{R}^d \times \{-1, 1\}$ is γ-*safe* from h if all $x' \in B_\gamma(x)$ has that $h(x') = y$; this implies it is sufficiently far from the decision boundary.

The γ-*error* can be measured on a distribution μ as the probability a sample $(x, y) \sim \mu$ is not γ-safe.

Prior work has defined a few notions of adversarial robustness. [14] considers the expected minimum Euclidean distance γ of a point x to decision boundary of h, formally: $\mathbb{E}_{(x,y)\sim\mu}[\min_{x'\in\mathbb{R}^d} \|x' - x\|_2$ such that $h(x') \neq y$.] This line of work uses specific function classes (some linear and quadratic classifiers) $\mathcal{H}$, which can use the value $h(x)$ to upper bound the expected perturbation radius γ for specific distributions (e.g., μ is Gaussian or uniform for each class). [34] defines *robust classification error* as the probability of drawing a γ-safe point from μ, mostly focusing on l_∞ perturbations. They show for linear models on Gaussian mixture distributions μ that more samples are needed to generalize wrt robust classifiers error than just classification error.

Our work, extends this to more general distributions, more complex (polynomial) classifiers, and to Euclidean perturbations. An important point to make is that while polynomials can be linearized to a higher-dimensional space so whether a point is classified correctly by the polynomial is preserved, this does *not* preserve the *distance* to the decision boundary, and so such techniques cannot be directly applied to understand the learnability of these polynomial classification problems.

Let $\mathcal{H}_p = \{\text{sgn} \circ h \mid h \in \mathcal{P}_p\}$ where $\mathcal{P}_p = \{f \in \mathbb{R}[X_1, \ldots, X_d] : \deg(f) \leq p\}$. The key insight is to describe a range space $(\mathbb{R}^d \times \{-1, 1\}, \mathcal{R}_p)$ derived from $\mathcal{P}_p$ and a robustness parameter $\gamma > 0$. Each function $h \in \mathcal{H}_p$ maps to a function $g : \mathbb{R}^d \times \{-1, 1\} \to \{-1, 1\}$, where $g(x, y) = 1$ if and only if (x, y) is γ-safe with respect to h. This takes on two cases, if $y = +1$, then $h(x)$ must be positive and x not in the γ-inflated polynomial around the decision boundary. Similarly, if $y = -1$, then $h(x)$ must be negative and x not in the γ-inflated polynomial.

Lemma 5 *The VC dimension of* $(\mathbb{R}^d \times \{-1, 1\}, \mathcal{R}_p)$ *is* $O(p)$ *for* $d = 1$ *and* $O(dp^{O(d)})$ *for* $d > 1$.

Proof. We can apply the composition argument detailed in Section 2.2 to the two d-dimensional ranges considered: at $y = +1$ the complement of an inflated polynomial and a polynomial, and at $y = -1$ the complement of an inflated polynomial and a polynomial, all of degree p; see example in Figure 2. All of these ranges are derived from the same polynomial $f \in \mathbb{R}[X_1, \ldots, X_d]$, but this only restricts the range space and does not increase the VC dimension. For the composition of a constant number of range spaces, the VC dimension is asymptotically the max of them. The stated bounds follow from the inflated polynomial bounds in Theorem 2 and Theorem 3. □

Next we analyze the learnability of polynomial classifiers which are γ-robust; those deemed successful on data which is γ-safe. The previous lemma demonstrated that such classifiers can be characterized with range spaces

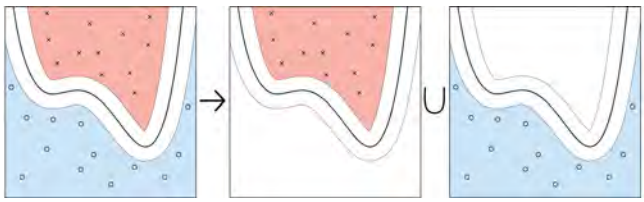


Figure 2: Decomposition of robust polynomial classification into ranges.

with bounded VC dimension, and directly linked to that for inflated polynomials.

We first focus on non-agnostic learning, where 0 error can be achieved on a sample from family $\mathcal{H}$. The *non-agnostic robust sample complexity* of a family $\mathcal{H}$, a parameter $\gamma > 0$, and a distribution μ is the size of the smallest iid sample $P = \{(x_i, y_i)\} \subset \mu$ so that for any $h \in \mathcal{H}$ with γ-error of zero on μ_P, then with probability at least $1 - \delta$, it has at most γ-error of ε on μ.

Theorem 6 *For any* $\gamma > 0$*, the non-agnostic robust sample complexity is* $O(\frac{p}{\varepsilon} \log \frac{p}{\varepsilon\delta})$ *for univariate polynomials of degree at most* p *and* $O(\frac{pd^{O(d)}}{\varepsilon} \log \frac{pd^{O(d)}}{\varepsilon\delta})$ *for* d*-variate polynomials of degree at most* p.

Proof. Let any function $g \in \mathcal{R}_p$ have $g(x, y) = 1$ iff the point $(x, y) \in \mathbb{R}^d \times \{-1, 1\}$ is γ-safe. By assumption of the theorem there is a function $g \in \mathcal{R}_p$ with $\text{er}_{\mu_P}(g) = 0$ on a sample P. Then by bounding the VC dimension in Lemma 5 and applying the non-agnostic bound of [20], we obtain the claimed result. □

We can also apply this to agnostic settings, where we cannot guarantee a perfect classifier. The *agnostic robust sample complexity* of a family $\mathcal{H}$, a parameter $\gamma > 0$, and a distribution μ is the size of the smallest iid sample $P\{(x_i, y_i)\} \subset \mu$ so that for any $h \in \mathcal{H}$ with γ-error of η on μ_P, then with probability at least $1 - \delta$, it has at most γ-error of $\eta + \varepsilon$ on μ. By the same argument as in Theorem 6 but applying the more general bound of [23], we obtain the following result which has no assumptions on the distribution μ.

Theorem 7 *For any* $\gamma > 0$*, the agnostic robust sample complexity is* $O(\frac{1}{\varepsilon^2}(p + \log \frac{1}{\delta}))$ *for univariate polynomials of degree at most* p *and* $O(\frac{1}{\varepsilon^2}(pd^{O(d)} + \log \frac{1}{\delta}))$ *for* d*-variate polynomials of degree at most* p.

5 Conclusion & Discussion

This paper uses a combination of traditional techniques of bounding VC dimension and algorithms in algebraic geometry to bound the VC dimension of complex range spaces. These techniques are useful for ranges defined

with a combination of polynomials and existential quantifiers, such as geometric ranges of all points within a fixed Euclidean distance from an object. These apply as long as the geometric object can be described as a polynomial, or by n polynomial pieces. A key example is the class of inflated polynomials; for one such range, a point x_0 is inside if *there exists* a ball, centered on the defining polynomial, which contains x_0. These results have implications in range searching, hitting sets, and learning on swept out polynomial curves, as well as in adversarial learning.

ℓ_∞ perturbations. The applications to adversarially-robust sample complexity we develop focus on how inflated polynomials correspond with robust classifiers, which allow any ℓ_2 perturbation of data and still have the correct classification. Other work in this subarea has considered ℓ_∞ perturbations. We remark here that the VC-dimension of a polynomial of degree p under ℓ_∞ perturbation may not require analysis with existential theory of the reals. We claim that the Minkowski sum of an ℓ_∞ ball with a polynomial of degree p in $\mathbb{R}^2$ can be described as the composition of 4 polynomial classifiers of degree p, and $O(p)$ linear segments. Thus, since the VC dimension of any one of the polynomial parts is $O(p)$, the composition of the $O(p)$ linear parts is $O(\log p)$, and the composition of these two aspects is $O(p)$.

References

[1] Martin Anthony and Peter L. Bartlett. *Neural Network Learning: Theoretical Foundations*. Cambridge University Press, Cambridge, 1999.

[2] Matthias Aschenbrenner et al. "Vapnik-Chervonenkis density in some theories without the independence property, I". In: *Trans. American Mathematical Society* 368 (Sept. 2011).

[3] S. Basu, R. Pollack, and Roy M.F. *Algorithms in Real Algebraic Geometry*. 2nd ed. Vol. 10. Springer-Verlag, Berlin Heidelberg, 2006.

[4] Saugata Basu. "Combinatorial complexity in O-minimal geometry". In: *Proceedings of The London Mathematical Society* 100 (Dec. 2006).

[5] Hervé Brönnimann and Michael T. Goodrich. "Almost Optimal Set Covers in Finite VC-Dimension". In: *Disc. & Comp. Geom.* (1995).

[6] J. Buckman et al. "Thermometer Encoding: One Hot Way To Resist Adversarial Examples". In: *ICLR*. 2018.

[7] Krzysztof Chalupka, P. Perona, and F. Eberhardt. "Visual Causal Feature Learning". In: *UAI*. 2015.

[8] Bernard Chazelle and Emo Welzl. "Quasi-optimal range searching in spaces of finite VC-dimension". In: *Disc. & Comp. Geom.* 4.5 (1989), pp. 467–489.

[9] Orfried Cheong, Anne Driemel, and Jeff Erickson. "Computational Geometry (Dagstuhl Seminar 17171)". In: *Dagstuhl Reports* 7.4 (2017). Ed. by Otfried Cheong, Anne Driemel, and Jeff Erickson, pp. 107–127. ISSN: 2192-5283.

[10] Jeremy M. Cohen, Elan Rosenfeld, and J. Z. Kolter. "Certified Adversarial Robustness via Randomized Smoothing". In: *ICML*. 2019.

[11] George E. Collins. "Quantifier elimination for real closed fields by cylindrical algebraic decompostion". In: *Automata Theory and Formal Languages*. Ed. by H. Brakhage. Springer, Berlin Heidelberg, 1975, pp. 134–183.

[12] Guneet S. Dhillon et al. "Stochastic Activation Pruning for Robust Adversarial Defense". In: *ArXiv* abs/1803.01442 (2018).

[13] Anne Driemel et al. "The VC dimension of metric balls under Fréchet and Hausdorff distances". In: *Disc. & Comp. Geom.* 66.4 (2021), pp. 1351–1381.

[14] Alhussein Fawzi, Omar Fawzi, and P. Frossard. "Analysis of classifiers' robustness to adversarial perturbations". In: *Machine Learning* 107 (2017), pp. 481–508.

[15] Dan Feldman and Michael Langberg. "A unified framework for approximating and clustering data". In: *STOC*. 2011, pp. 569–578.

[16] Jean-Yves Franceschi, Alhussein Fawzi, and Omar Fawzi. "Robustness of classifiers to uniform ℓ_p and Gaussian noise". In: *AISTATS*. 2018.

[17] Paul W. Goldberg and Mark R. Jerrum. "Bounding the Vapnik-Chervonenkis dimension of concept classes parameterized by real numbers". In: *Machine Learning* 18 (1995), pp. 131–148.

[18] I. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *CoRR* abs/1412.6572 (2015).

[19] Sariel Har-peled. *Geometric Approximation Algorithms*. USA: American Mathematical Society, Providence, Rhode Island, 2011. ISBN: 0821849115.

[20] David Haussler and Emo Welzl. "epsilon-Nets and Simplex Range Queries." In: *Disc. & Comp. Geom.* 2 (1987), pp. 127–151.

[21] Aounon Kumar et al. "Curse of Dimensionality on Randomized Smoothing for Certifiable Robustness". In: *ArXiv* abs/2002.03239 (2020).

[22] Guang-He Lee et al. "Tight Certificates of Adversarial Robustness for Randomly Smoothed Classifiers". In: *NeurIPS*. 2019.

[23] Yi Li, Philip M. Long, and Aravind Srinivasan. "Improved Bounds on the Samples Complexity of Learning". In: *J. Comp. and Sys. Sci.* 62 (2001), pp. 516–527.

[24] A. Madry et al. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *ArXiv* abs/1706.06083 (2018).

[25] Seyed-Mohsen Moosavi-Dezfooli et al. "Robustness of Classifiers to Universal Perturbations: A Geometric Perspective". In: *International Conference on Learning Representations*. 2018.

[26] Seyed-Mohsen Moosavi-Dezfooli et al. "Robustness via Curvature Regularization, and Vice Versa". In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2019).

[27] Nicolas Papernot et al. "Distillation as a Defense to Adversarial Perturbations Against Deep Neural Networks". In: *2016 IEEE Symposium on Security and Privacy (SP)* (2016), pp. 582–597.

[28] Jeff M. Phillips and Yan Zheng. "Subsampling in Smoothed Range Spaces". In: *Algorithmic Learning Theory*. 2015.

[29] Chongli Qin et al. "Adversarial Robustness through Local Linearization". In: *NeurIPS*. 2019.

[30] Aditi Raghunathan et al. "Understanding and Mitigating the Tradeoff Between Robustness and Accuracy". In: *ArXiv* abs/2002.10716 (2020).

[31] James Renegar. "On the Computational Complexity and Geometry of the First-Order Theory of the Reals, Part I: Introduction. Preliminaries. The Geometry of Semi-Algebraic Sets. The Decision Problem for the Existential Theory of the Reals". In: *J. Symb. Comput.* 13 (1992), pp. 255–300.

[32] Hadi Salman et al. "Provably Robust Deep Learning via Adversarially Trained Smoothed Classifiers". In: *NeurIPS*. 2019.

[33] Kamron Saniee. "A Simple Expression for Multivariate Lagrange Interpolation". In: *SIAM Undergraduate Research Online* (Jan. 2007).

[34] L. Schmidt et al. "Adversarially Robust Generalization Requires More Data". In: *NeurIPS*. 2018.

[35] Christian Szegedy et al. "Intriguing properties of neural networks". In: *CoRR* abs/1312.6199 (2014).

[36] D. Tsipras et al. "Robustness May Be at Odds with Accuracy". In: *arXiv: Machine Learning* (2019).

[37] Vladimir Vapnik and Alexey Chervonenkis. "On the Uniform Convergence of Relative Frequencies of Events to their Probabilities". In: *Theory of Probability and its Applications* 16 (1971).

[38] Y. Yang et al. "Adversarial Robustness Through Local Lipschitzness". In: *ArXiv* abs/2003.02460 (2020).

[39] Hongyang Zhang et al. "Theoretically Principled Trade-off between Robustness and Accuracy". In: *ICML*. 2019.

A Additional Implications

Smoothed range spaces. Another application related to robust adversarial learning is the idea of a "smoothed range space" [28], where the misclassification error is replaced around a binary decision boundary with a continuous function, where significantly misclassified points are given a penalty of 1, but points close to the boundary (under Euclidean distance) are given a penalty between 0 and 1 according to a continuous rate based on how close. Zheng and Phillips [28] showed that the VC dimension of the decision boundary expanded by a Euclidean distance of r in all directions (i.e., inflated the decision boundary) governs the sample complexity of this task. However, this bound was unknown for polynomial decision boundaries [9] until this paper. The relevant VC dimension is that of an inflated polynomial.

Inflated univariate spline classification. An inflated spline is a polynomial spline that has been inflated with radius r. A spline is a piecewise polynomial that preserves stronger continuity between pieces. Suppose we are unaware of an object's (perhaps a person or vehicle) location over time and that we make a modeling assumption that the object is traveling along a piecewise polynomial path. A piecewise polynomial curve, perhaps a natural cubic spline, could be a more natural assumption than a piecewise polygonal curve. Suppose there is a low-flying unmanned aerial vehicle (UAV) with a radio jamming device which is disrupting cellular and GPS signals within r meters. We would like to approximate the UAV's trajectory over time. How many devices with radio sensors (cell towers, GPS, etc.) do we need to test (build a binary classifier) with up to $1-\varepsilon$ accuracy, to induce the path the object took, with probability $1-\delta$. It was previously unknown how many radio sensors are required to be tested, yet in $\mathbb{R}^2$ with n polynomial pieces each with bounded degree p we know now the bound is $m = O(\frac{1}{\varepsilon^2}(np\log n + \log\frac{1}{\delta}))$. The specific application described in the Introduction with a polynomial curve, has $n = 1$, so the specific bound in that case is $m = O(\frac{1}{\varepsilon^2}(p + \log\frac{1}{\delta}))$.

Theorem 8 *If points $x \in \mathbb{R}^2$ within r distance of a univariate polynomial spline are classified as* 1 *and points outside r are classified as -1, then to induce a trajectory with ε error, $m = O(\frac{1}{\varepsilon^2}(np\log n + \log\frac{1}{\delta}))$ points randomly chosen are sufficient, with probability $1-\delta$.*

Proof. In $\mathbb{R}^2$, by Theorem 2, the VC dimension associated with each piece is $O(p)$, if its degree is bounded by p. Now we must only apply a composition argument over each piece to get the VC dimension. Therefore we find the following bound $O(np\log n)$ where n is the number of polynomial pieces used. Hence, the sample complexity for learning on inflated polynomial splines is $m = O(\frac{1}{\varepsilon^2}(np\log n + \log\frac{1}{\delta}))$. □

Budgeted Steiner Networks: Three Terminals with Equal Path Weights

Mario Szegedy Jingjin Yu *

Abstract

Given a set of terminals in 2D/3D, the network with the shortest total length that connects all terminals is a Steiner tree. At the other extreme, with enough total length budget, every terminal can be connected to every other terminal via a straight line, yielding a complete graph over all terminals that connects every pair of terminals with a shortest path. In this work, we study a generalization of Steiner trees, asking what happens between these two extremes. For a given total length budget, we seek a network structure that minimizes the sum of the weighted distances between pairs of terminals. Focusing on three terminals with equal pairwise path weights, we characterize the full evolutionary pathway between the Steiner tree and the complete graph, which contains interesting intermediate structures.

1 Introduction

Consider a scenario in which three or more terminals (e.g., the black nodes A, B, and C in Fig. 1) are to be connected using a (graph) network, the total length of which is limited.

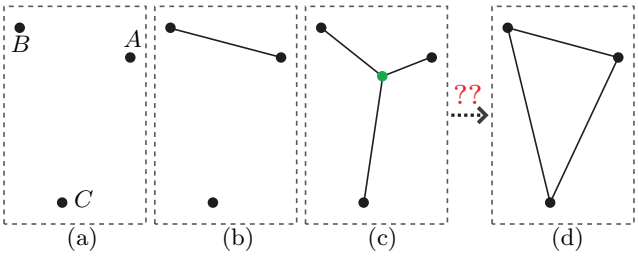


Figure 1: Evolution of a *budgeted Steiner network* over three (black) terminals as the budget increases. (a) Three terminals, A, B, and C, to be connected. (b) The minimal non-trivial network that connects two terminals. (c) The minimal network connecting all terminals, which is a Steiner tree. (d) With sufficient budget, the network is a complete graph. The question is, what happens between (c) and (d)?

At one extreme, the minimum length budget required to connect all terminals corresponds to the total length of the edges of a Steiner tree over the terminals (Fig. 1(c)). The well-known *Steiner tree problem* (`STP`) seeks optimal network structures for connecting a set of terminals while minimizing the total edge lengths [9,16]. `STP` generally asks for a minimally connected network, resulting in a topology that is a tree. At the other extreme, when there is no limit on the budget, the best network structure is clearly a complete graph over all terminals, where every pair of terminals are connected through a straight edge. Such a network ensures the shortest possible travel distance between any pair of terminals. What if, however, the budget falls between the two extremes?

To address the question, we propose the *budgeted Steiner network* (`BSN`) problem/model. As a natural generalization of `STP`, `BSN` seeks the best network structure for a given length budget to connect three or more terminals, which reside in $\mathbb{R}^d$ for some $d \geq 1$, such that the sum of the (weighted) distances between pairs of nodes are minimized. In this work, we mainly focus on the case of three terminals with $d = 2$ (for three terminals, $d = 2$ is the same as $d \geq 2$).

The generalization immediately leads to rich and interesting structures, even when only three terminals are involved. As the budget increases, the network structure changes continuously between a Steiner tree and a complete graph over the terminals, a few snapshots of which are illustrated in Fig. 2.

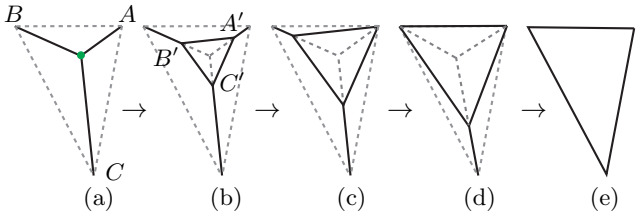


Figure 2: A spectrum of optimal Euclidean `BSN` network structures (solid lines) for three terminals in a typical setup, as the allowed budget increases.

As a summary of the full evolutionary pathway, if all internal angles of a $\triangle ABC$ are smaller than $2\pi/3$, the Steiner tree over terminals A, B, and C has a Steiner point that is internal to the triangle (e.g., the green dot in Fig. 2). In this case, for a generic $\triangle ABC$ (that is, $\triangle ABC$ is not an isosceles triangle), as the budget increases past the length of the Steiner tree, an equilateral triangle $\triangle A'B'C'$ will "grow" out of the Steiner point

*Department of Computer Science, Rutgers University, the State University of New Jersey, {szegedy, jingjin.yu}@cs.rutgers.edu

(Fig. 2(b)) and continues to expand until one vertex of $\triangle A'B'C'$ meets one of the terminals, say A. Past this point, $\triangle A'B'C'$ continues to expand as an isosceles triangle with $A' = A$ fixed (Fig. 2(c)) as the budget continues to increase, until another vertex meets B or C, say B. $\triangle A'B'C'$ then continue to expand with $A' = A$ and $B' = B$ fixed, and C' moving toward C, until it fully coincides with $\triangle ABC$. If $\triangle ABC$ has one angle equal to or larger than $2\pi/3$, the evolutionary pathway is similar but shortened; the corresponding BSN does not have an initial phase containing an equilateral triangle.

The main contribution of this work is the rigorous characterization of the precise evolution pathway of a BSN as the available budget increases, for three arbitrarily located terminals. The analysis also implies an efficient algorithm for computing the optimal BSN structure for any given budget.

2 Related Work

BSN problems are closely related to STPs [8,9,16], which is a broad term covering a class of network optimization problems. An STP seeks a minimal network that connects a set of terminals (in Euclidean space or on graphs that are possibly edge/vertex weighted). There are four main cases: Euclidean, rectilinear, discrete/graph-theoretic [6, 11], and phylogenetic [9]. Considering the paper's scope, we provide a brief literature review of Euclidean STPs.

The Euclidean STP asks the following question: given n terminals in 2D or 3D, find a network that connects all n points with the minimum total length (the discussion from now on will be limited to the 2D case). Obviously, the resulting network is a tree and may only have straight line segments; it may also require additional intermediary nodes to be added. These added nodes are called *Steiner points*. The study of Euclidean STP bears with it a long history; the initial mathematical study of the subject may be traced back to at least 1811 [3]. According to [12], key properties of Euclidean STP have been established in (as early as) the 1930s by Jarník and Kössler [10]. An interconnecting network T is called a Steiner tree if it satisfies the following conditions [9]:

(a) T is a tree,

(b) Any two edges of T meet at an angle of at least $2\pi/3$, and

(c) Any Steiner point cannot be of degree 1 or 2.

These conditions turn out to be also relevant in our study of the BSN problem. The solution to an Euclidean STP must be a Steiner tree. Note that (b) implies a node of the network has a maximum degree of 3. Together, (b) and (c) imply that three edges must meet at a Steiner point forming angles of $2\pi/3$ in a pairwise manner (see Fig. 1). Because Euclidean Steiner trees assume minimal energy configurations, they also appear in nature. Indeed, it is possible to employ related natural phenomena (e.g., using rubber bands and soap film) to "compute" Euclidean Steiner trees [5, 7, 14].

Our study, which focuses on the case of three terminals with equal path weights, bears similarity with a recent study [4] which examines a related problem of characterizing the minimum dilation spanners on three terminals for a given budget. Whereas there exists a mild degree of similarity, we note that we independently developed our results, which provides an exact analysis of the full evolution pathway between the Steiner tree and the complete graph. On the other hand, the analytical result of [4] is mostly limited to the initial stage of the evolution.

Computing an Euclidean STP is NP-hard, although there is a polynomial time approximation scheme (PTAS) for solving it [2]. On the more practical side, fast methods including the GeoSteiner algorithm [15,17] have been developed building on the Melzak construction [13]. An open source implementation of the GeoSteiner algorithm is maintained [1].

3 Preliminaries

Let there be $n \geq 3$ *terminals* $N = \{v_1, \ldots, v_n\}$, distributed in some way in a d-dimensional unit cube, $d > 0$. For each pair of terminals v_i and v_j, $1 \leq i < j \leq n$, let $w_{ij} \in (0, 1]$ denote the (relative) *weight* or *importance* of the route connecting v_i to v_j. In practice, w_{ij} may model the expected traffic flow from v_i to v_j, for example. In an Euclidean *budgeted Steiner tree* (BSN) problem, straight line segments are to be added for connecting the n terminals so that some or all of the terminals are connected. Similar to Steiner trees, intermediate nodes other than $v_1, \ldots, v_n$, which we call *anchors*, may be added. The terminals, anchors, and the straight line segments then form a graph containing one or more connected components. Under the constraint that the total length of the line segments does not exceed a *budget* L, the BSN problem seeks a network structure that minimizes the objective

$$J(L) = \sum_{1 \leq i < j \leq n} w_{ij} d_{ij}, \tag{1}$$

in which d_{ij} denotes the shortest distance between v_i and v_j on the network. If no path exists between v_i and v_j, let d_{ij} be some very large number.

In the current work, we examine the case of $n = 3$ and $w_{ij} = 1$ for all $1 \leq i, j \leq 3$, $i \neq j$, i.e., paths between pairs of terminals are equally important. Let the three terminals be A, B, and C, we are looking for a BSN minimizing the sum $d_{AB} + d_{BC} + d_{AC}$ subject to the budget L. For a fixed L, let $N(L)$ denote the optimal

BSN structure. Let L_{ST} be the budget L when $N(L)$ is a Steiner tree. For convenience, let $N_{\text{ST}} := N(L_{\text{ST}})$.

4 Anchor Structures and Steiner Triangles

4.1 Basic Properties of Anchors

First, we note each anchor must have degree three.

Lemma 1 (Degree of Anchors) *For three terminals, any anchor must have degree exactly three.*

Proof. Each anchor must connect at least three line segments; otherwise, the anchor point and the involved line segments only cause increases to the objective $d_{AB}+d_{BC}+d_{AC}$. An anchor's degree also cannot be four or larger when there are only three terminals, because each outgoing edge from an anchor must be on a shortest path to a unique terminal, if we are to minimize Eq. (1). But there are only three terminals. □

We analyze what happens when $L = L_{\text{ST}}+\varepsilon$ for small $\varepsilon > 0$, for the case where the Steiner point lies inside $\triangle ABC$, which happens when all angles of $\triangle ABC$ are smaller than $2\pi/3$. Due to continuity, the resulting structure that minimizes Eq. (1) must be a perturbation of N_{ST} (e.g., Fig. 1(b)). This means that $N(L_{\text{ST}}+\varepsilon)$ must start "growing" at the Steiner point. We want to understand how $N(L_{\text{ST}}+\varepsilon)$ evolves for small ε. This raises the following questions: (1) how many line segments are in $N(L = L_{\text{ST}}+\varepsilon)$ and (2) how do they come together? We note that $N(L_{\text{ST}}+\varepsilon)$ must contain more than three straight line segments. Otherwise, $N(L_{\text{ST}}+\varepsilon)$ will still be a tree but with $d_{AB}+d_{BC}+d_{AC} = 2(L_{\text{ST}}+\varepsilon) > 2L_{\text{ST}}$, i.e., $J(L) > J(L_{\text{ST}})$.

To answer above-mentioned questions, we start with establishing essential properties of anchors, concerning their locations, degrees, and numbers. It is clear that anchors must always fall within $\triangle ABC$; otherwise, an outside anchor (on the convex hull of all terminals and anchors) can be "retracted" toward the boundary of $\triangle ABC$ to reduce both the budget and the objective function value. In fact, anchors cannot reside on the boundary of $\triangle ABC$, as shown in the following lemma.

Lemma 2 (Interiority of Anchors) *For terminals A, B, and C, any anchor must fall in the interior of $\triangle ABC$, excluding its perimeter.*

Proof. Consider the setting illustrated in Fig. 3 where only a portion of $\triangle ABC$ is drawn. Suppose that D is the only anchor on AC and the horizontal line segment passing through D and D' is part of an optimal network structure. For the setup, DD' must be part of the shortest path on the optimal network that connects A to B as well as C to B; the entire AC must also be part of the network that connects A and C.

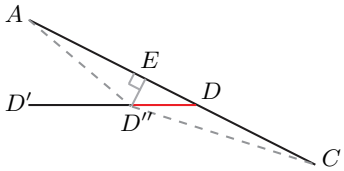


Figure 3: Moving C' along $C'C$ for a small amount.

We claim that such a configuration cannot be optimal. To see this, retract D along DD' by some small distance of $|DD''|$. This reduces the budget by $\Delta L = |DD''|+(|AC|-|AD''|-|CD''|)$. At the same time, the cost reduction is $\Delta J = 2|DD''|+(|AC|-|AD''|-|CD''|)$.

Let $E \in AC$ be a point such that $D''E \perp AC$. It is straightforward to derive that $|ED''| \gg |CD''|-|CE|$ and $|ED''| \gg |AD''|-|AE|$ for sufficiently small $|ED''| > 0$. Therefore, $|DD''| \geq |ED''| > (|AD''|+|CD''|-|AC|)$. This means that for small $|DD''|$, both ΔL and ΔJ are positive, i.e., we can reduce budget and at the same time reduce the cost by retracting D along DD' to D''. This means that D cannot be an anchor. If D is not the only anchor on AC, the same proof works assuming D is the lowest anchor. □

Building on Lemmas 1 and 2, we show that there can be at most three anchors for three terminals.

Lemma 3 (Number of Anchors in $N(L_{\text{ST}}+\varepsilon)$) *When all angles of $\triangle ABC$ are below $2\pi/3$, for small $\varepsilon > 0$, $N(L_{ST}+\varepsilon)$ contains three anchors that forms a triangle inside $\triangle ABC$.*

Proof. By Lemma 1, all anchors have degree three. If there is only a single anchor that is not the Steiner point, then $N(L_{\text{ST}}+\varepsilon)$ still has a tree structure. This tree is different from N_{ST} which is minimal, so the new tree must have a larger objective function value which cannot be optimal.

If there are two anchors, each with degree three, then both of them cannot be connected to all of A, B, and C; there must be exactly five line segments in $N(L_{\text{ST}}+\varepsilon)$, one of which connects the two anchors. This leaves four line segments connected to the three terminals, which means that two of these line segments must reach the same terminal. This will induce a total budget that cannot be an arbitrarily small amount above L_{ST} when the Steiner point is inside $\triangle ABC$. That is, this is impossible with a budget $L_{\text{ST}}+\varepsilon$ for small $\varepsilon > 0$.

There cannot be more than three anchors when there are only three terminals. To establish this, we note that a shortest path between any two terminals, when there are three terminals in total, can make at most two "turns" due to path sharing. To see this, consider the shortest path P_{AB} between terminals A and B. P_{AB} may bend at most two times, once to share with a path

from A to C and once to share with a path from B to C. If P_{AB} bends once, say at an anchor A', then both AA' or $A'B$ must be on a shortest path to B and we must have a tree. This is not possible under the assumption that ε is small, so there can only be one edge coming out of a terminal. Therefore, each shortest path between two terminals must bend exactly twice at two anchors. The three shortest paths then have a total of six anchors. Because each anchor is shared by two shortest paths, there can only be three unique anchors that form a triangle. □

4.2 Steiner Triangle for Three Anchors

Having shown that there are three anchors, let the anchor closest to A, B and C be A', B', and C', respectively. This suggest that $N(L_{\mathtt{ST}} + \varepsilon)$ contains six line segments AA', BB', CC', $A'B'$, $A'C'$, and $B'C'$. We call $\triangle A'B'C'$ that "grows" out of the Steiner point a *Steiner triangle*. Next, we establish that $\triangle A'B'C'$ is an equilateral triangle, starting with showing that its three internal angles are bisected by AA', BB' and CC'. The objective Eq. (1), $d_{AB} + d_{BC} + d_{AC}$ for the current setting, translates to

$$\begin{aligned} J(L_{\mathtt{ST}} + \varepsilon) =& 2|AA'| + 2|BB'| + 2|CC'| \\ &+ |A'B'| + |A'C'| + |B'C'|. \end{aligned} \tag{2}$$

Lemma 4 (Bisector of Steiner Triangle) *For terminals A, B, and C with a Steiner point, let $N(L_{ST}+\varepsilon)$ be composed of the Steiner triangle $\triangle A'B'C'$ and segments AA', BB' and CC'. Then an angle of $\triangle A'B'C'$ is bisected by the line passing the corresponding anchor and the terminal the anchor is connected to.*

Proof. See the Appendix for the technical proof based on infinitesimal analysis. □

Before moving on to showing that $\triangle A'B'C'$ is equilateral, we note that Lemma 4 does not depend on ε being small. Moreover, the result continues to hold if there are one or two anchors, which can be readily verified.

Lemma 5 (Anchor Bisector) *For terminals A, B, and C, suppose C' is an internal anchor connected to C in an optimal network structure $N(L)$. Then CC' bisects the angle formed by the other two outgoing edges from C'.*

We now prove a key structural property of `BSN` for three terminals involving three anchors.

Theorem 6 (Steiner Triangle for Three Anchors) *For terminals A, B, and C with a Steiner point, assume that $N(L_{ST} + \varepsilon)$ is composed of the Steiner triangle $\triangle A'B'C'$ and segments AA', BB' and CC'. Then $\triangle A'B'C'$ is equilateral with its center being the Steiner point of the terminals. The center of $\triangle A'B'C'$ is the intersection point of AA', BB' and CC'.*

Proof. See the Appendix. □

From Theorem 6, we can draw the following conclusion. For three terminals with a Steiner point, as the budget L goes just beyond $L_{\mathtt{ST}}$, an equilateral triangle will "grow" out the Steiner point toward the terminals. Moreover, whenever there are three anchors, they must form an equilateral triangle. All such equilateral triangles have their vertices lying on the line segments formed by the terminals and the Steiner point, as illustrated in Fig. 4. We have not yet show, however, that as L grows, the anchors cannot go from three to fewer and then become three again. We delay this after the structures with fewer anchors are characterized.

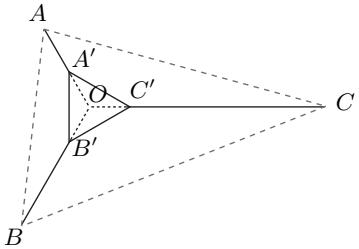


Figure 4: For three terminals with a Steiner point (which is always internal), when there are three anchors, they always form an equilateral triangle.

4.3 One and Two Anchors

If there are two anchors, they must both be connected to one shared terminal, say A, and each connecting to a unique terminal in B and C. Let the anchors be B' and C'. $N(L)$ then consists of five segments AB', AC', BB', CC', and $B'C'$. It can be shown that $\triangle AB'C'$ is an isosceles triangle (see, e.g., Fig. 5).

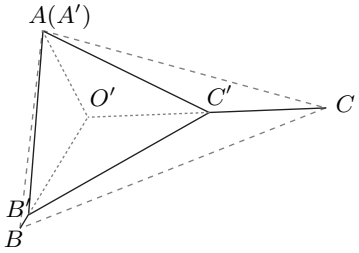


Figure 5: For three terminals with a Steiner point, when there are two anchors, they always form an isosceles triangle with one of the terminals.

Proposition 1 (Steiner Triangle for Two Anchors) *For terminals A, B, and C with a Steiner point, if the optimal network $N(L)$ has two anchors B', C', then these two anchors form an isosceles triangle with one of the terminals, e.g., A. $AB' = AC'$.*

Proof. See the Appendix. □

Following the same line of reasoning, when there is a single anchor in an optimal network $N(L)$, e.g., C' that is connected to A, B, and C, if C' is not the Steiner point, $N(L)$ must contain one of AB, BC, and AC. Suppose $N(L)$ contains AB, then all we know is that CC' must bisect $\angle AC'B$. See Fig. 2(d) for an example.

5 Evolution of the Budgeted Steiner Network

5.1 With Steiner Point

Having established the optimal configuration when there are 1-3 anchors, we now piece them together to understand the evolution of the network. Intuitively, as the budget L increases, the evolution of the optimal network $N(L)$ would look like that shown in Fig. 2, going from Steiner tree to having three anchors, then two, then one, and finally becoming the triangle of the three terminals. To show this is the actual network evolution pathway, however, we must show that there cannot be discrete jumps in `BSN` structures, e.g., going from three anchors to two anchors and then back to three anchors.

We proceed to show that the sequence in Fig. 2 is indeed how $N(L)$ evolves as L increases by analyzing how $J(L)$ changes as L changes, i.e., $\frac{dJ}{dL}$.

Lemma 7 (Rate of Change at Anchors) *For terminals A, B, and C, let C' be an anchor connected to C. Let the angle formed by the other two edges emanating from C' other than CC' be 2α. As C' moves closer to C, the rate of change to the objective function $\frac{dJ}{dL}$ due to the change to CC' is*

$$\frac{dJ}{dL} = \frac{2\cos\alpha - 2}{2\cos\alpha - 1}. \tag{3}$$

Proof. Fig. 6 shows the setting where C' is moved along $C'C$ for a small amount. By the bisector Lemma 5, the addition of length (in green) to the two edges coming out of C' that are not CC' is $2|EC'|$ while the reduction of length to $|CC'|$ is $|C'E|/\cos\alpha$ (the red segment). Therefore, the change to the budget due to this is $\Delta L = 2|C'E| - |C'E|/\cos\alpha$.

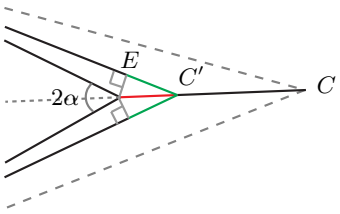


Figure 6: Moving C' along $C'C$ for a small amount.

On the other hand, the change to the objective function value is $\Delta J = -(2|C'E|/\cos\alpha - 2|C'E|)$ because $C'C$ contributes to two shortest paths. Dividing ΔJ over ΔL yields Eq. 3. □

Proposition 1 (Range of Change, Three Anchors) *For three terminals, when there are three anchors,*

$$\frac{dJ}{dL} = \frac{1-\sqrt{3}}{2}. \tag{4}$$

Proof. For three anchors, α in Eq. (3) is $\pi/6$. We then have $dJ/dL = (\sqrt{3}-2)/(\sqrt{3}-1) = (1-\sqrt{3})/2$. □

Proposition 2 (Range of Change, 1-2 Anchors) *For three terminals, when there are one of two anchors, let the angle formed at the anchor belonging to the triangle structure of the network be 2α, then,*

$$\frac{dJ}{dL} = \frac{2\cos\alpha - 2}{2\cos\alpha - 1}. \tag{5}$$

Since $0 < 2\alpha \le \pi/2$, $\alpha \in (0, \pi/4]$. Let $\cos\alpha = x$, $x \in [\frac{\sqrt{2}}{2}, 1)$. Eq. 3 becomes $g(x) = \frac{2x-2}{2x-1}$. It is straightforward to derive (using derivatives) that $g(x)$ is negative on the given range of x and monotonically increases to 0 as $x \to 1$. This means, with reference to Fig. 6, that the magnitude of $\frac{dJ}{dL}$ becomes smaller as C' gets closer to C (α decreases). This allows us to show that $J(L)$ decreases faster when there are more anchors. We begin with showing that internal angles at anchors cannot exceed $\pi/3$.

Lemma 8 (Feasible Anchor Configurations) *For three terminals and an optimal Steiner network, the internal angles of the triangular structure of the network at non-terminal anchors are always no more than $\pi/3$.*

Proof. For three anchors, we have shown they must assume an equilateral triangle configuration. Suppose that in a two-anchor network configuration, the optimal network has internal angles at non-terminals anchors larger than $\pi/3$. For example, suppose that in Fig. 5, $\angle AB'C' = \angle AC'B' > \pi/3$. This requires that $\angle B'AC' < \pi/3$. Now, suppose we push down the triangle $A'B'C'$ along AA' by a small $\delta > 0$ and retract along $B'B$ and $C'C$ so that L remains unchanged. Because $0 > \frac{dJ}{dL}|_{A'} > \frac{dJ}{dL}|_{B'} = \frac{dJ}{dL}|_{C'}$, this means that J will actually decrease due to the change. Therefore, the configuration cannot be optimal.

The same argument also applies to the single anchor case: if the internal angle at the single anchor is larger than $\pi/3$, the at least one of the two other internal angles must be smaller than $\pi/3$. □

We are now ready to establish the evolution pathway of the optimal Steiner network for three terminals with Steiner points.

Theorem 9 (BSN Evolution, with Steiner Point) *For terminals A, B, and C with a Steiner point O, as the budget $L > L_{ST}$ increases, the optimal Steiner network $N(L)$ will first grow an equilateral triangle, $\triangle A'B'C'$, out of O toward the three terminals. The internal angles of $\triangle A'B'C'$ are bisected by AA', BB' and CC'. The growth continues until one of the anchors, say A', reaches terminal A, corresponding to the largest internal angle of $\triangle ABC$. Then, an isosceles triangle continuous to grow in place of the equilateral triangle, with its two internal angles $\angle AB'C'$ and $AC'B'$ bisected by BB' and CC', respectively, until one of the two anchors B' reaches a second terminal, say B, that corresponds to the second largest angle of $\triangle ABC$. Finally, the network grows as C' finally reaches C, with CC' always bisecting $\angle AC'B$.*

Proof. Without loss of generality, assume that $\angle BAC \geq \angle ABC \geq \angle ACB$. By Lemma 3 and Theorem 6, the initial optimal network when $L = L_{\mathtt{ST}} + \varepsilon$ has an equilateral triangle $A'B'C'$ growing out of the Steiner point O, with AA', BB', and CC' bisecting $\angle B'A'C'$, $\angle A'B'C'$, and $\angle A'C'B'$, respectively. By Lemma 8, before $\triangle A'B'C'$ reaches A as an equilateral triangle (AA' is shorter than than BB' and CC' when $\angle BAC$ is the largest angle of $\triangle ABC$), it cannot happen that the optimal network jumps to a configuration where one anchor disappears. To see that this is the case, suppose the network jumps to a configuration where A' merges with A. This would force $\triangle A'B'C'$ to have $\angle B'A'C' < \pi/3 < \angle A'B'C' = \angle A'C'B'$, which is not possible. The situation gets worse if B' merges with B or C' merges with C. Using a similar argument, we can show that it is also not possible for the optimal network to jump from three anchors to having a single anchor without the equilateral $\triangle A'B'C'$ reaching its maximum girth. Using the same approach, we can also show that it is not possible to "jump" from a two-anchor configuration to a single anchor configuration without the anchor B' reaching B, as the isosceles triangle expands. □

5.2 No Steiner Point

When an angle of $\triangle ABC$, say $\angle BAC$, is larger than $2\pi/3$, A acts as a "Steiner" point. In this case, it becomes impossible for the optimal network $N(L)$ to have three internal anchors.

Lemma 10 (Anchor Multiplicity) *For three terminals without a Steiner point, the optimal network $N(L)$ for any L cannot have three anchors.*

Proof. If there are three anchors, Theorem 6 must hold. However, this is impossible if one of the angles formed by the terminals is equal to or larger than $2\pi/3$. Referring to Fig. 4, suppose that $\angle BAC \geq 2\pi/3$. However, also by Theorem 6, $\angle BOC = 2\pi/3$, which is not possible. □

Following similar reasoning used for establishing the case where the Steiner point is in the interior of $\triangle ABC$, the evolution of the optimal network for the current setting goes through the following phases (assuming terminals A, B, and C, and $\angle BAC \geq 2\pi/3$):

1. The budget L is sufficient to cover the shortest edge of $\triangle ABC$ but less than $L_{\mathtt{ST}}$. In this case, $N(L)$ contains one edge of $\triangle ABC$
2. The budget L equal to $L_{\mathtt{ST}}$. In this case, $N(L)$ is the Steiner tree comprised of AB and AC.
3. For $L = L_{\mathtt{ST}} + \varepsilon$ for small positive ε, a small isosceles triangle grows out from A, producing a configuration as shown in Fig. 7(a). The network satisfies the bisector requirement given by Lemma 1. As L increases, the isosceles triangle expands with the bisector structure in place, until one of the vertex of the triangle hits a terminal (B).
4. As one of the two anchors merge with a terminal, the other anchor will continue to march toward the last terminal (C) as L increases, eventually merge with that terminal. A snapshot of this process is given in Fig. 7(b).

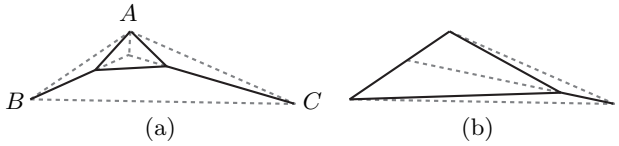


Figure 7: A spectrum of optimal Euclidean BSN network structures (solid lines) for three terminals in a typical setup where $\angle BAC \geq 2\pi/3$, as the allowed budget increases.

6 Conclusion and Discussions

In this work, we propose the *budgeted Steiner network* (BSN) problem to study shortest path structures among multiple terminals under a path length budget. We establish the precise evolution of the BSN structure for three arbitrarily located terminals where paths between each pair of terminals have equal importance. It is clear that the characterization yields efficient algorithms for computing optimal BSN structures for any given 3-terminal setup and length budget.

The current work just begins to scratch the surface of the study of BSN; it is natural to study the case where the weights are not equal as well as the case of more terminals. It is also interesting to explore how BSN structures are affected by obstacles. Finally, as an alternative to analytical approaches, it is interesting to explore establishing BSN structures using numerical methods.

References

[1] *GeoSteiner:Software for Computing Steiner Trees*, 2017 (accessed Aug 31, 2017).

[2] S. Arora. Polynomial-time approximation schemes for Euclidean TSP and other geometric problems. *Journal of the ACM*, 45(5):753–782, 1998.

[3] Marcus Brazil, Ronald L Graham, Doreen A Thomas, and Martin Zachariasen. On the history of the euclidean steiner tree problem. *Archive for history of exact sciences*, 68(3):327–354, 2014.

[4] Kevin Buchin, Herman J Haverkort, and Hidde Koerts. Restricted-weight minimum-dilation spanners on three points. In *CCCG*, pages 240–248, 2020.

[5] RC Clark. Communication networks, soap films and vectors. *Physics Education*, 16(1):32, 1981.

[6] Stephen A Cook. The complexity of theorem-proving procedures. In *Proceedings of the third annual ACM symposium on Theory of computing*, pages 151–158. ACM, 1971.

[7] EN Gilbert and HO Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.

[8] Mathias Hauptmann and Marek Karpiński. *A compendium on Steiner tree problems*. Inst. für Informatik, 2013.

[9] Frank K Hwang and Dana S Richards. Steiner tree problems. *Networks*, 22(1):55–89, 1992.

[10] Vojtěch Jarník and Miloš Kössler. O minimálních grafech, obsahujících n daných bodů. *Časopis pro pěstování matematiky a fysiky*, 63(8):223–235, 1934.

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

[12] Bernhard Korte and Jaroslav Nešetřil. Vojtěch jarník's work in combinatorial optimization. *Discrete Mathematics*, 235(1-3):1–17, 2001.

[13] Zdzislaw Alexander Melzak. On the problem of steiner. *Canad. Math. Bull*, 4(2):143–148, 1961.

[14] William Miehle. Link-length minimization in networks. *Operations research*, 6(2):232–243, 1958.

[15] David M Warme, Pawel Winter, and Martin Zachariasen. Exact algorithms for plane steiner tree problems: A computational study. In *Advances in Steiner trees*, pages 81–116. Springer, 2000.

[16] Pawel Winter. Steiner problem in networks: a survey. *Networks*, 17(2):129–167, 1987.

[17] Pawel Winter and Martin Zachariasen. Euclidean steiner minimum trees: An improved exact algorithm. *Networks*, 30(3):149–166, 1997.

7 Appendix

Proof. [Proof of Lemma 4] Assume that for a given budget $L = L_{\mathtt{ST}} + \varepsilon$, the optimal network $N(L_{\mathtt{ST}} + \varepsilon)$ has corresponding optimal objective $J(L_{\mathtt{ST}} + \varepsilon)$ as given in Eq. 2. We show that CC' is a bisector of $\angle A'C'B'$ by analyzing the local changes to L and $J(L_{\mathtt{ST}} + \varepsilon)$ if we perturb C'.

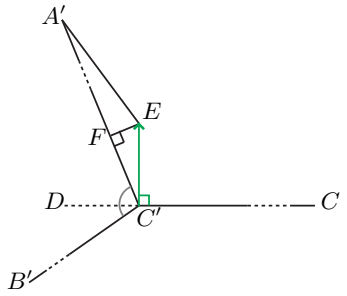


Figure 8: Perturbing C' in an assumed optimal configuration for the three-terminal Euclidean `BSN` problem. The figure zooms in around C' without showing A and B. The drawing intentionally avoids assuming that $\triangle A'B'C'$ is an equilateral triangle.

Referring to Fig. 8, let D be a point on the extension of $\overrightarrow{CC'}$. A point E is introduced that shifts C' up vertically (i.e., $C'E \perp C'C$) by the amount $|C'E|$, as a small perturbation to C'. Now draw a line EF such that $EF \perp A'C'$ with $F \in A'C'$. Because $|C'E|$ is small, $|A'F| \approx |A'E|$ (this is a second order approximation). As C' is moved to E, the length change of $A'C'$ is given by $|A'E| - |A'C'|$, which is approximately $|A'F| - |A'C'| = -|FC'| = -|C'E| \cos \angle A'C'E = -|C'E| \sin \angle A'C'D$.

Following a similar analysis procedure, the length change of $B'C'$, $|B'E| - |B'C'|$, is approximately $|C'E| \sin \angle B'C'D$. Because $C'E \perp C'C$ and $|C'E|$ is small, $|CC'| \approx |CE|$ (also a second order approximation). Relating the length changes due to moving C' up to the change of the budget L, the net change to L is $|C'E|(\sin \angle B'C'D - \sin \angle A'C'D)$ (i.e., $B'C'$ becomes longer and $A'C'$ becomes shorter with CC' unchanged, as a second order approximation). The change to the objective $J(L_{\mathtt{ST}} + \varepsilon)$ is the same since CC' is unaffected by $C'E$.

Because the changes to L and $J(L_{\mathtt{ST}} + \varepsilon)$ are exactly the same, if $\angle A'C'D \neq \angle B'C'D$, then either $\overrightarrow{C'E}$ or a perturbation in the direction of $\overrightarrow{EC'}$ will cause both

$|A'C'|+|B'C'|+|C'C|$ and $|A'C'|+|B'C'|+2|C'C|$ to decrease, which means that L and $J(L_{\text{ST}}+\varepsilon)$ can be simultaneously reduced. This contradicts the assumption that L is the smallest budget for which the current objective $J(L_{\text{ST}}+\varepsilon)$ is possible. Since this cannot happen, it must be the case that $\angle A'C'D = \angle B'C'D$ in an optimal network configuration. That is, CC' is a bisector of $\angle A'C'B'$. By symmetry, BB' is a bisector of $\angle A'B'C'$ and AA' is a bisector of $\angle B'A'C'$. □

Proof. [Proof of Theorem 6] Again assuming an optimal solution, extend line segments AA', BB', and CC' so that they intersect (see Fig. 9).

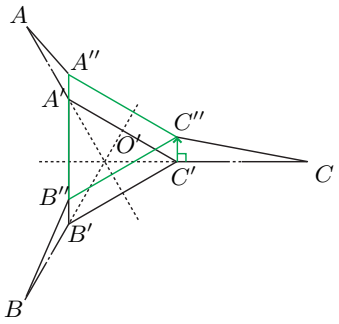


Figure 9: Applying a perturbation to $\triangle A'B'C'$ that lifts it vertically along CC', which keeps the length of CC' unchanged in a first order approximation.

Because they are bisectors of $\triangle A'B'C'$, by Lemma 4, they must meet at the same point O'. For this setting, we again apply a perturbation argument used in proving Lemma 4, this time lift the entire $\triangle A'B'C'$ in a direction perpendicular to CC'. Let the perturbed triangle be $\triangle A''B''C''$. Using the same argument, this time applied to the length changes of AA' and BB', we can reach the conclusion that the line CC' must be a bisector of $\angle AO'B$. In other words, shifting AA' and BB' synchronously will not reduce the objective function only if CC' bisects $\angle AO'B$.

Similarly, AA' must be a bisector of $BO'C$ and BB' must be a bisector of $AO'C$. Using that CC' bisects $AO'B$ and $A'C'B'$, it can be derived that $\angle O'A'C' = \angle O'B'C'$, which in turn shows that $\angle B'A'C' = \angle A'B'C'$. By symmetry, it can then be concluded that $\triangle A'B'C'$ is an equilateral triangle. This further shows that $\angle A'O'B' = \angle A'O'C' = \angle B'O'C' = 2\pi/3$, implying that O', the center of $\triangle A'B'C'$, is the Steiner point O of the terminals. □

Proof. [Proof of Proposition 1] By Lemma 5, BB' bisects $\angle AB'C'$ and CC' bisects $\angle AC'B'$. Let the extensions of BB' and CC' meet at O' (see Fig. 5). Then AO' bisects $\angle B'AC'$. Using the perturbation argument from the proof of Theorem 6, applied to perturb the lengths of BB' and CC', we can show that AO' is also a bisector of $\angle B'O'C'$ (we do this by "rotating" $\triangle AB'C'$ with center A slightly). This means that $\angle B'O'A = \angle C'O'A$, which in turn implies that $\angle AB'O' = \angle AC'O'$ and further implies $\angle AB'C' = \angle AC'B'$. Therefore, $\triangle AB'C'$ is an isosceles triangle and $AB' = AC'$. □

Author Index