CCCG 2017

29th Canadian Conference on Computational Geometry

Proceedings

July 26–28, 2017
Carleton University
Ottawa, Ontario, Canada

Sponsored by

Foreword

The Twenty-Ninth Canadian Conference on Computational Geometry took place on July 26–28, 2017 at Carleton University in Ottawa, Ontario. This annual conference attracts researchers in computational geometry from around the world to Canada for an open exchange of ideas and results. This volume contains 39 contributed papers as well as 3 invited talks. These proceedings are available online at `http://2017.cccg.ca/` and at the central CCCG web site `http://www.cccg.ca`. The organizing committee would like to thank the invited speakers Erin Chambers, David Eppstein (Paul Erdős Memorial Lecture), and Stefan Langerman (Ferran Hurtado Memorial Lecture). We also thank all of those who contributed their papers to the conference. Thanks go out to the program committee and the local arrangements committee for their organizational assistance.

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

Joachim Gudmundsson and Michiel Smid

Invited Speakers

David Eppstein	University of California, Irvine
Erin Wolf Chambers	Saint Louis University
Stefan Langerman	Université Libre de Bruxelles

Program Committee

Luis Barba	ETH Zürich
Ahmad Biniaz	Carleton University
Paz Carmi	Ben-Gurion University of the Negev
Mohammad Farshi	Yazd University
Robin Flatland	Siena College
Joachim Gudmundsson (Co-chair)	University of Sydney
Mark Keil	University of Saskatchewan
Elena Khramtcova	Université Libre de Bruxelles
Christian Knauer	University of Bayreuth
Matias Korman	Tohoku University
Irina Kostitsyna	Université Libre de Bruxelles
Marc van Kreveld	Utrecht University
Anna Lubiw	University of Waterloo
Wolfgang Mulzer	Freie Universität Berlin
Yoshio Okamoto	The University of Electro-Communications
Vera Sacristan	Universitat Politecnica de Catalunya
Stefan Schirra	University of Magdeburg
Michiel Smid (Co-chair)	Carleton University
Csaba Tóth	California State University Northridge
Jan Vahrenhold	Universität Münster
Sander Verdonschot	Carleton University
Birgit Vogtenhuber	Graz University of Technology
Yusu Wang	The Ohio State University
Norbert Zeh	Dalhousie University

Local Organizers

Ahmad Biniaz	Carleton University
Prosenjit Bose	Carleton University
Jean-Lou De Carufel	University of Ottawa
Vida Dujmović	University of Ottawa
Anil Maheshwari	Carleton University
Pat Morin	Carleton University
Michiel Smid	Carleton University
Sander Verdonschot (Chair)	Carleton University

Additional Reviewers

Mohammad Ali Abam, Oswin Aichholzer, Carlos Alegra-Galicia, Andrei Asinowski, Davood Bakhshesh, Michael Biro, Prosenjit Bose, Maike Buchin, Siu-Wing Cheng, Man Kwun Chiu, Yago Diez Donoso, Adrian Dumitrescu, Patrick Eades, Ruy Fabila-Monroy, Fabrizio Frati, Bernd Gärtner, Sariel Har-Peled, Mahdieh Hasheminezhad, Gregorio Hernandez, Takashi Horiyama, Kolja Junginger, Akitoshi Kawamura, Michael Kerber, Sudeshna Kolay, Jason S. Ku, Maarten Löffler, Aleksandar Markovic, Saeed Mehrabi, Tillmann Miltzow, Pat Morin, Joseph O'Rourke, Peter Palfrader, Irene Parada, Jeff Phillips, Alexander Pilz, Roman Prutkin, Domotor Palvolgyi, Benjamin Raichel, Marcel Roeloffzen, Ignaz Rutter, Maria Saumell, Lena Schlipf, Christiane Schmidt, Patrick Schnider, André Schulz, Carlos Seara, Don Sheehy, Luise Sommer, Frank Staals, Fabian Stehn, William Steiger, Ana Paula Tom'as, Ryuhei Uehara, Arthur van Goethem, André van Renssen, Giovanni Viglietta, Carola Wenk, Alexander Wolff, Alireza Zarei, Pawel Zylinski.

Contents

Open Problem Session

Thursday July 27

Session 3A

Session 3B

Session 4A

Session 4B

Session 5A

Session 5B

Friday July 28

Session 6A

Session 6B

Forbidden Configurations in Discrete Geometry

David Eppstein *

We review and classify problems in discrete geometry that depend only on the order-type or configuration of a set of points, and that can be characterized by a family of forbidden configurations. These include the happy ending problem, no-three-in-line problem, and orchard-planting problem from classical discrete geometry, as well as Harborth's conjecture on integer edge lengths and the construction of universal point sets in graph drawing. We investigate which of these properties have characterizations involving a finite number of forbidden subconfigurations, and the implications of these characterizations for the computational complexity of these problems.

*Computer Science Department, University of California, Irvine, eppstein@ics.uci.edu. Supported by NSF grant CCF-1228639, CCF-1618301, and CCF-1616248.

Power domination on triangular grids

Prosenjit Bose * Claire Pennarun† Sander Verdonschot‡

Abstract

The concept of *power domination* emerged from the problem of monitoring electrical systems. Given a graph G and a set $S \subseteq V(G)$, a set M of monitored vertices is built as follows: at first, M contains only the vertices of S and their direct neighbors, and then each time a vertex in M has exactly one neighbor not in M, this neighbor is added to M. The *power domination number* of a graph G is the minimum size of a set S such that this process ends up with the set M containing every vertex of G. We here show that the power domination number of a triangular grid T_k with hexagonal-shape border of length $k-1$ is exactly $\left\lceil \frac{k}{3} \right\rceil$.

1 Introduction

Power domination is a problem that arose from the context of monitoring electrical systems [10, 1], and was reformulated in graph terms by Haynes et al. [9].

Given a graph G and a set $S \subseteq V(G)$, we build a set M as follows: at first, M is the closed neighborhood of S, i.e. $M = N[S]$, and then iteratively a vertex u is added to M if u is the only neighbor of a monitored vertex v that is not in M (we say that v *propagates* to u). At the end of the process, we say that M is the set of vertices *monitored* by S. We say that G is *monitored* when all its vertices are monitored. The set S is a *power dominating set* of G if $M = V(G)$, and the minimum cardinality of such a set is the *power domination number* of G, denoted by $\gamma_P(G)$.

Power domination has been particularly well studied on regular grids and their generalizations: the exact power domination number has been determined for the square grid [6] and other products of paths [3], for the hexagonal grid [7], as well as for cylinders and tori [2]. These results are particularly interesting in comparison with the ones on the same classes for (classical) domination: for example, the problem of finding the domination number of grid graphs $P_n \times P_m$ was a difficult problem which was solved only recently [8]. They also rely heavily on propagation: it is generally sufficient to monitor (with adjacency alone) a small portion of the graph in order to propagate to the whole graph.

We here continue the study of power domination in grid-like graphs by focusing on triangular grids with hexagonal-shaped border.

A *triangular grid* T_k has vertex set $V(T_k) = \{(x, y, z) \mid x, y, z \in [0..2k-2], x - y + z = k - 1\}$. Two vertices (x, y, z) and (x', y', z') are adjacent if and only if $|x' - x| + |y' - y| + |z' - z| = 2$. The graph T_k has a regular hexagonal shape, and k is the number of vertices on each edge of the hexagon. Figure 1 shows the two triangular grids T_2 and T_3. Note that T_k appears as a subgraph of T_{k+1} (where $(1, 1, 1)$ has been added to the coordinates of each vertex in T_k).

We prove the following theorem:

Theorem 1 *For $k \in \mathbb{N}^*$, $\gamma_P(T_k) = \left\lceil \frac{k}{3} \right\rceil$.*

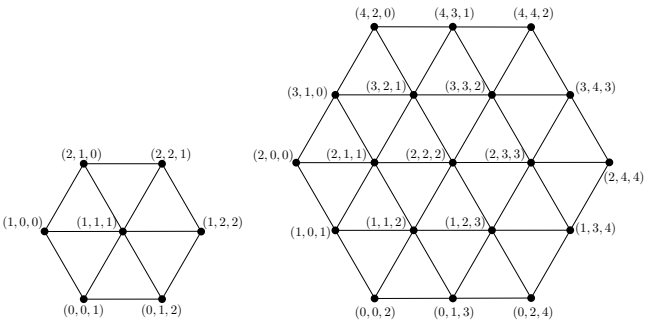


Figure 1: The graphs T_2 and T_3, along with the coordinates of the vertices.

An inner vertex $v \in V(T_k)$ with coordinates (x, y, z) has 6 neighbors with the following coordinates: $(x, y + 1, z+1)$, $(x-1, y, z+1)$, $(x-1, y-1, z)$, $(x, y-1, z-1)$, $(x + 1, y, z - 1)$ and $(x + 1, y + 1, z)$ (see Figure 2a). The coordinates of a vertex v are denoted by (v_1, v_2, v_3). The *line* $l_{v_j=i}$ is the set of vertices $\{(v_1, v_2, v_3) \mid v_j = i\}$ (see Figure 2b).

One interesting property of the triangular grids is that if an equilateral triangle having one side of the hexagonal border as base is monitored, then the border allows the propagation until the whole graph is monitored. For example, it suffices to monitor the set

*School of Computer Science, Carleton University, Ottawa ON, Canada. Research supported in part by NSERC. jit@scs.carleton.ca

†Univ. Bordeaux, claire.pennarun@labri.fr

‡School of Computer Science, Carleton University, Ottawa ON, Canada. Research supported in part by NSERC. sander@cg.scs.carleton.ca

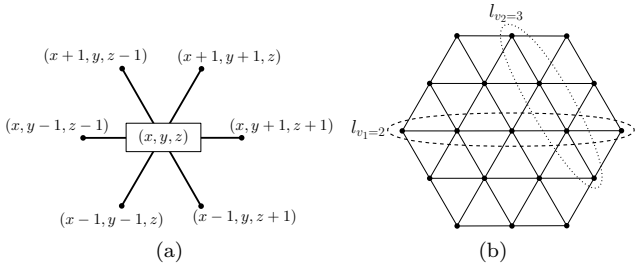


Figure 2: (a) The coordinates of the neighbors around an inner vertex $v = (x, y, z)$. (b) The lines $l_{v_1=2}$ and $l_{v_2=3}$ in T_3.

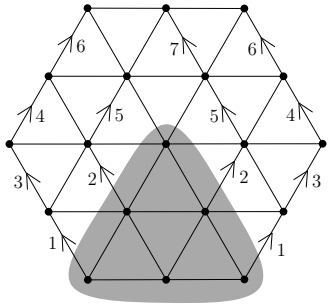


Figure 3: The propagation steps to monitor T_3 once the set $\mathcal{T}$ (in the gray area) is monitored. Propagation steps indexed by the same number can be done in parallel.

$\mathcal{T} = \{v = (v_1, v_2, v_3) \in V(G) \mid 0 \leq v_1, v_2 \leq k-1, k-1 \leq v_3 \leq 2k-2\}$ to monitor T_k (see Figure 3).

We assume throughout the section that $k \geq 4$: observe that if $k \leq 3$, then $\gamma_P(T_k) = 1$, with $S = \{(k-2, k-2, k-1)\}$ (for $k = 2, 3$).

2 Upper bound

We begin by giving a construction for the upper bound:

Lemma 2 *For $k \in \mathbb{N}^*$, $\gamma_P(T_k) \leq \left\lceil \frac{k}{3} \right\rceil$.*

Proof. Let $i = \left\lfloor \frac{k}{3} \right\rfloor$, and $d = k - i - 1$ if $k \equiv 0, 1 \mod 3$, $d = k - i - 2$ otherwise. Let S' be the following set of vertices (see Figure 4): $S' = \{(1+3\ell, d+\ell, k+d-2-2\ell), 0 \leq \ell \leq i-1\}$. In other words, S' contains the vertex $v = (1, d, k+d-2)$ and vertices whose coordinates are obtained by adding $(3, 1, -2)$ up to $i-1$ times to the coordinates of v. If $k \not\equiv 0 \mod 3$, $S = S' \cup \{(k-1, k-1, k-1)\}$. Otherwise, $S = S'$. Then we have, depending on the value of k modulo 3:

- $k = 3i$: $|S| = i = \left\lceil \frac{3i}{3} \right\rceil$.
- $k = 3i+1$: $|S| = i+1 = \left\lceil \frac{3i+1}{3} \right\rceil$.
- $k = 3i+2$: $|S| = i+1 = \left\lceil \frac{3i+2}{3} \right\rceil$.

In each case, S is a set with cardinality $\left\lceil \frac{k}{3} \right\rceil$, and S progressively power dominates the whole triangular grid T_k. □

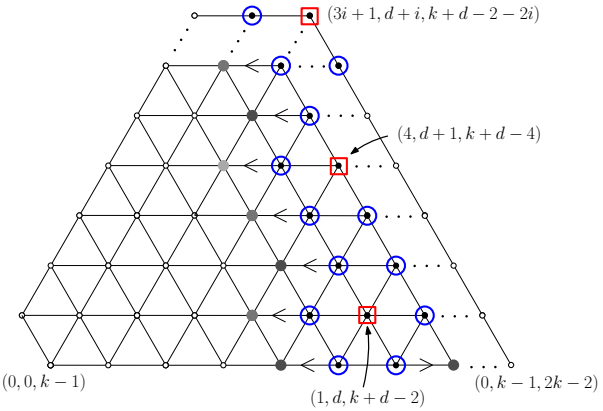


Figure 4: Construction and propagation of the set S': $d = k - i - 1$ if $k \equiv 0, 1 \mod 3$, $d = k - i - 2$ if $k \equiv 2 \mod 3$. Red square-framed vertices are in S', blue circle-framed vertices are in $N[S']$. Dark gray vertices are monitored in the first propagation round, gray ones in the second round, and the light gray one in the third round. Observe how the pattern of monitored vertices repeats.

3 Lower bound

Let $A \subset V(T_k)$ be a set of vertices of the graph. We define the *border* $\mathcal{B}_A \subseteq A$ of A as follows: $\mathcal{B}_A = \{v \in A, N(v) \setminus A \neq \emptyset\}$. Let $A_{v_j=i}$ denote the set of vertices of A in a given line $l_{v_j=i}$. We define the *j-shifted set* $A' = A^{(j)}$ of A as follows (see Figure 5): $|A'| = |A|$, and for each line $l_{v_j=i}$, A' contains the $|A_{v_j=i}|$ vertices with smallest coordinates v_{j+1} (for example, the 1-shifted set of A contains only left-most vertices on each horizontal line). More formally,

$$A'_{v_j=i} = \{(v_1, v_2, v_3) \mid v_j = i, v_{j+1} = \ell+\alpha, 0 \leq \ell < |A_{v_j=i}|\},$$

with $\alpha = 0$ if $0 \leq i \leq k-1$, and $\alpha = i - (k-1)$ if $k \leq i \leq 2k-2$.

Lemma 3 *Let A' be the j-shifted set of A. Then $|\mathcal{B}_{A'}| \leq |\mathcal{B}_A|$.*

Proof. In this proof, since j is fixed, we simplify the notation $l_{v_j=i}$ into l_i. Let a_i be the number of vertices in A (and in A') in line l_i and b_i (resp. b'_i) be the number of vertices in $\mathcal{B}_A$ (resp. $\mathcal{B}_{A'}$) in line l_i. We show that $b_i \geq b'_i$ for every line l_i, $0 \leq i \leq 2k-2$. We consider three cases depending on the value of i (when $0 \leq i < k-1$, when $i = k-1$ and when $k \leq i \leq 2k-2$):

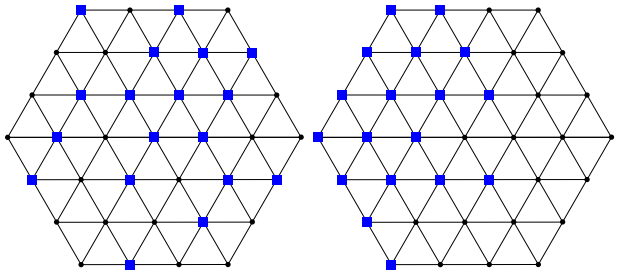


Figure 5: (Left) Blue-square vertices are in the set A. (Right) Blue-square vertices are in the 1-shifted set A' of A: the left-most vertices of each line $l_{v_1=i}$ are in A'.

- $0 \leq i < k-1$: we thus have $|l_{i+1}| = |l_i| + 1$ and $|l_i| = |l_{i-1}| + 1$. Let us consider vertices in line l_i which are in A but not in the border of A: there are $a_i - b_i$ such vertices. By definition, we have $a_i - b_i \leq a_i$. Their neighbors (if they exist) in l_{i-1} and l_{i+1} are in A. We have thus both $a_i - b_i \leq a_{i+1} - 1$, and $a_i - b_i \leq a_{i-1}$. Hence $a_i - b_i \leq \min\{a_{i+1} - 1, a_{i-1}, a_i\}$ for $1 \leq i < k-1$ (for $i = 0$, we have $a_i - b_i \leq \min\{a_{i+1} - 1, a_i\}$). We can apply the same reasoning to the vertices that are in A' but not in the border of A': since the vertices of A' are consecutive on lines l_{i-1}, l_i and l_{i+1}, we get that $a_i - b'_i = \min\{a_{i+1} - 1, a_{i-1}, a_i\}$ (for $i = 0$, we have $a_i - b'_i = \min\{a_{i+1} - 1, a_i\}$). Note that the inequalities we get for A turn into equalities on A'. Then $a_i - b_i \leq a_i - b'_i$, and thus $b_i \geq b'_i$.

- We have a similar proof when $k-1 < i \leq 2k-2$, for which we have $|l_{i+1}| = |l_i| - 1$ and $|l_i| = |l_{i-1}| - 1$: in that case, we get $a_i - b'_i = \min\{a_{i-1} - 1, a_{i+1}, a_i\} \geq a_i - b_i$.

- $i = k-1$: we thus have $|l_{i+1}| = |l_{i-1}| = |l_i| + 1$. As for the previous case, first consider vertices which are in A but not in the border of A: by definition $a_i - b_i \leq a_i$, and we have $a_{i+1} \geq a_i - b_i$ and $a_{i-1} \geq a_i - b_i$. Thus $a_i - b_i \leq \min\{a_{i+1}, a_{i-1}, a_i\}$. Similarly, we get that $a_i - b'_i = \min\{a_{i+1}, a_{i-1}, a_i\}$. Thus $a_i - b_i \leq a_i - b'_i$, and so $b_i \geq b'_i$.

□

We define the *shifting process* of a set $A \subset V(T_k)$ as the following iterative process: $A_{\ell+1} = ((A_\ell^{(1)})^{(2)})^{(3)}$, with $A_0 = A$. In other words, we successively apply 1-shift, 2-shift and 3-shift to the set A until a fixed point A_{ℓ^*} is reached. We show that this fixed point exists and that the vertices of the resulting set form a particular shape:

Lemma 4 (i) *This shifting process stops, i.e. there exists ℓ^* such that $A_{\ell^*+1} = A_{\ell^*}$.*

(ii) *Let $A^* = A_{\ell^*}$. If $v = (x, y, z) \in A^*$, then all vertices $v' = (x', y', z')$ with $y' \leq y$ and $z' \leq z$ are also in A^* (see Figure 6).*

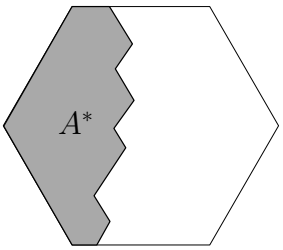


Figure 6: The set A^* has a staircase shape.

Proof. (i) We define the *weight* in A of a vertex as follows: $w_A(v) = v_1 + 2v_2 + 2v_3$ if $v \in A$, $w_A(v) = 0$ otherwise. Similarly, the weight of a set S relatively to A is $w_A(S) = \sum_{v \in S} w_A(v)$. For simplicity, we denote by w_A the *global weight* of the set A: $w_A = w_A(T_k)$.

Let A' be the j-shifted set of A. We show that if $A' \neq A$, then $w_{A'} < w_A$.

Recall that for every vertex $v = (v_1, v_2, v_3)$ of T_k, $v_1 - v_2 + v_3 = k - 1$. We first show that if v and v' are two vertices with $v_j(v') = v_j(v)$ and $v_{j+1}(v') < v_{j+1}(v)$, then $w(v') < w(v)$:

- $j = 1$: $v_1(v') = v_1(v)$ and $v_2(v') < v_2(v)$, so $v_3(v') = k - 1 - v_1(v') + v_2(v') = k - 1 - v_1(v) + v_2(v') < v_3(v)$. Thus $w(v') < w(v)$.

- $j = 2$: $v_2(v') = v_2(v)$ and $v_3(v') < v_3(v)$. Since $v_1(v) - v_2(v) + v_3(v) = v_1(v') - v_2(v') + v_3(v')$, we get $v_1(v) + v_3(v) = v_1(v') + v_3(v')$. Thus $w(v) - w(v') = v_1(v) + 2v_2(v) + 2v_3(v) - v_1(v') - 2v_2(v') - 2v_3(v') = v_3(v) - v_3(v')$. So $w(v') < w(v)$.

- $j = 3$: $v_3(v') = v_3(v)$ and $v_1(v') < v_1(v)$, so $v_2(v') = v_1(v') + v_3(v') - k + 1 = v_1(v') + v_3(v) - k + 1 \leq v_2(v)$. Thus $w(v') < w(v)$.

By definition on a j-shifted set, for each line $l_{v_j=i}$,

$$w_{A'}(l_{v_j=i}) - w_A(l_{v_j=i}) = \sum_{v' \in A' \setminus A} w(v') - \sum_{v \in A \setminus A'} w(v)\,,$$

and either $A_{v_j=i} = A'_{v_j=i}$, and this sums to 0, or $A_{v_j=i} \neq A'_{v_j=i}$, and it is strictly negative. Therefore $A' \neq A$ implies $w_{A'} < w_A$. Since the global weight of any set is always positive, this directly concludes the proof of item (i).

(ii) Let $v = (v_1, v_2, v_3)$ be a vertex in A^*. The vertices $u_1 = (v_1 + 1, v_2, v_3 - 1)$, $u_2 = (v_1, v_2 - 1, v_3 - 1)$ and $u_3 = (v_1 - 1, v_2 - 1, v_3)$ (i.e. the north-west, west and south-west neighbors of v) are also in A^*: otherwise, we could again shift the set A^* and get the set $A^* - \{v\} +$

$\{u_i\}$, which has less weight than A^*, a contradiction. Since this is true for every vertex of A^*, the proposition holds. □

We can now prove the lower bound:

Lemma 5 *For $k \in \mathbb{N}^*$, $\gamma_P(T_k) \geq \dfrac{2k-1}{6}$.*

Proof. Let S be a power dominating set of T_k. If $|S| > \frac{k}{3}$, then the result holds. Thus we assume $|S| \leq \lceil \frac{k}{3} \rceil$. In power domination, propagation from a set S is done by rounds. We decide of an arbitrary order on the vertices monitored by S during each round. This defines a (non-unique) total order $m_1, \ldots, m_{|V(G)\setminus N[S]|}$ on the vertices of $V(G) \setminus N[S]$. We then define the set $M[t]$ as follows: $M[0] = N[S]$, and $M[t+1] = M[t] \cup \{m_{t+1}\}$.

The key idea of this proof is to consider the size of the sets $\mathcal{B}_{M[t]}$, to bound it and to deduce a bound on $|S|$. It is a classical way to prove lower bounds for power domination in regular lattices (see for example the lower bound proof on strong products [3]). However, on the contrary to what happens in other cases, the size of the sets $\mathcal{B}_{M[t]}$ is not globally bounded from below: at the end of the propagation, no vertices belong to the border of the monitored set. We thus "stop" the propagation in the middle of the process and reason from there.

Claim 1. For any $0 \leq i \leq |V(G) \setminus N[S]|$, we have $|\mathcal{B}_{M[i]}| \leq 6|S|$.

Proof. We prove it by induction on i: $|\mathcal{B}_{M[0]}| = |\mathcal{B}_{N[S]}| \leq 6|S|$ by definition. If the vertex m_{i+1} becomes monitored by propagation from a vertex v in $\mathcal{B}_{M[i]}$, then v is not in $\mathcal{B}_{M[i+1]}$, and at most one vertex (m_{i+1}) is added to $\mathcal{B}_{M[i+1]}$. Thus $|\mathcal{B}_{M[i+1]}| \leq |\mathcal{B}_{M[i]}|$. Using the induction hypothesis, we conclude that $|\mathcal{B}_{M[i+1]}| \leq 6|S|$. (□)

Let M be the set $M[t]$ containing $\frac{|V(T_k|}{2}$ vertices (as soon as $k \geq 3$, we get $\frac{|V(T_k|}{2} = \frac{3k^2-3k+1}{2} \geq \frac{7(k+1)}{3} \geq 7|S| \geq |M[0]|$, and so M exists), and let M^* be the set defined from M by Lemma 4(i).

Claim 2. We have $2k-1 \leq |\mathcal{B}_{M^*}|$.

Proof. We now prove that for every index $0 \leq i \leq 2k-2$, the line $l_{v_1=i}$ contains at least one vertex of $\mathcal{B}_{M^*}$.

Suppose there exists an index $0 \leq i \leq 2k-2$ such that all vertices of the line $l_{v_1=i}$ are in M^*. If $0 \leq i \leq k-1$, then the vertex $w = (i, k+i-1, 2k-2)$ (i.e. the right-most vertex of the line $l_{v_1=i}$) is in M^*, and so by Lemma 4(ii), all vertices of the set $\{(v_1, v_2, v_3) \mid v_2 \leq k+i-1\}$ are also in M^* (see Figure 7a). Since $k+i-1 > k-1$, then strictly more than half of the vertices of T_k are in M^*, and so M^* has strictly more than the required number of vertices, a contradiction. Similarly, if $k-1 < i \leq 2k-2$: the vertex $w = (i, 2k-2, 3k-3-i)$ (i.e. the right-most vertex of the line $l_{v_1=i}$) is in M^*, and thus by Lemma 4(ii), all vertices of the set $\{(v_1, v_2, v_3) \mid v_3 \leq 3k-3-i\}$ are also in M^*. Since $3k-3-i > k-1$, then strictly more than half of the vertices of T_k are in M^*, a contradiction. Thus every line $l_{v_1=i}$ contains at least one vertex not in M^*.

Suppose now that one of the lines $l_{v_1=i}$ contains no vertex of M^*. If $0 \leq i \leq k-1$ (see Figure 7b), then the vertex $w = (i, 0, k-1-i)$ (i.e. the left-most vertex of the line $l_{v_1=i}$) is not in M^*. By the contrapositive of Lemma 4(ii), the line $l_{v_3=k-1-i}$ also contains no vertices of M^*, and so all vertices of M^* are included in the set $\{(v_1, v_2, v_3) \mid v_3 < k-1-i\}$ (they are all on the left and above line $l_{v_3=k-1-i}$). Thus M^* contains strictly less than the half of the vertices of T_k, a contradiction. Similarly, if $k-1 < i \leq 2k-2$, then the vertex $w = (i, i-k+1, 0)$ is not in M^*. By the contrapositive of Lemma 4(ii), the line $l_{v_2=i-k+1}$ also contains no vertices of M^*, and so all vertices of M^* are included in the set $\{(v_1, v_2, v_3) \mid v_2 < i-k+1\}$ (they are all on the left and below line $l_{v_2=i-k+1}$). Since in that case $i-k+1 < k-1$, then again, $|M^*| = |M| < \frac{|V(T_k)|}{2}$ vertices, a contradiction.

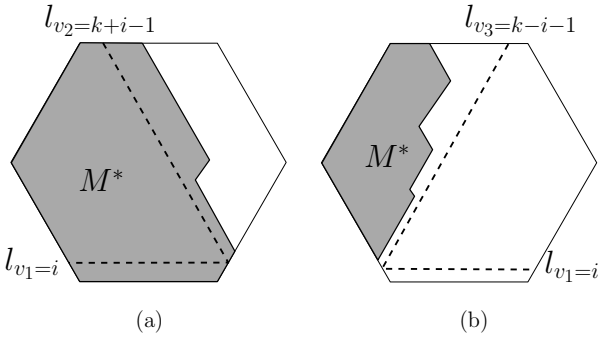


Figure 7: (a) If all vertices of a line $l_{v_1=i}$ are in M^* ($0 \leq i \leq k-1$), then all vertices of T_k with $v_2 \leq k+i-1$ are also in M^*. (b) If the line $l_{v_1=i}$ contains no vertices of M^* ($1 \leq i \leq k-1$), then all vertices of M are above and left of line $l_{v_3=k-i-1}$.

We thus get that each line $l_{v_1=i}$ contains at least one vertex of M^* and not all its vertices are in M^*. Thus each line contains at least one vertex of $\mathcal{B}_{M^*}$, and so $2k-1 \leq |\mathcal{B}_{M^*}|$. (□)

By Lemma 3, $|\mathcal{B}_{M^*}| \leq |\mathcal{B}_M|$, hence $2k-1 \leq |\mathcal{B}_M|$. Using Claim 1, we get $2k-1 \leq |\mathcal{B}_M| \leq 6|S|$, and so $|S| \geq \dfrac{2k-1}{6}$, which concludes the proof. □

We know that $\gamma_P(T_k)$ is an integer. Since there is no integer between $\frac{2k-1}{6} = \frac{k}{3} - \frac{1}{6}$ and $\lceil \frac{k}{3} \rceil$, then Lemma 5 directly implies that $\lceil \frac{k}{3} \rceil \leq \gamma_P(T_k)$.

This then gives our global result:

$$\gamma_P(T_k) = \left\lceil \frac{k}{3} \right\rceil \quad ,$$

concluding the proof of Theorem 1.

4 Discussion

We carried on with the study of power domination in regular lattices, and examined the value of $\gamma_P(G)$ when G is a triangular grid with hexagonal-shaped border. We showed that in that case, $\gamma_P(G) = \lceil \frac{k}{3} \rceil$.

The process of propagation in power domination led to the development of the concept of propagation radius, i.e. the number of propagation steps necessary in order to monitor the whole graph [4]. It would be interesting to study the propagation radius of our constructions (in particular in the case of triangular grids) and to try and find a power dominating set minimizing this radius.

It seems that the border plays an important role in the propagation when the grid has an hexagonal shape, and so the next step in the understanding of power domination in triangular grids would be to look into grids with non-hexagonal shape. For example, what is the power domination number of a triangular grid with triangular border?

Finally, the relation of our results with the ones presented for hexagonal grids by Ferrero et al. [7] has to be noted: they show (with techniques different from the ones used in this paper) that $\gamma_P(H_n) = \lceil \frac{2n}{3} \rceil$, where n is the dimension of the hexagonal grid H_n, and so $\gamma_P(H_n) = \gamma_P(T_{2n})$. Moreover, it is interesting to remark that H_n is an induced subgraph of T_{2n}. We already know [5] that in general, the power domination number of an induced subgraph can be either smaller or arbitrarily large compared to the power domination number of the whole graph. It would then be very interesting to investigate further under which conditions induced subgraphs have the same power dominating number as the whole graph.

References

[1] T. L. Baldwin, L. Mili, M. B. Boisen and R. Adapa. Power system observability with minimal phasor measurement placement. *IEEE Transactions on Power Systems*, 8(2):707–715, 1993.

[2] R. Barrera and D. Ferrero. Power domination in cylinders, tori, and generalized Petersen graphs. *Networks*, 58(1):43–49, 2011.

[3] P. Dorbec, M. Mollard, S. Klavžar and S. Špacapan. Power domination in product graphs. *SIAM Journal on Discrete Mathematics*, 22(2):554–567, 2008.

[4] P. Dorbec and S. Klavžar. Generalized power domination: propagation radius and Sierpiński graphs. *Acta Applicandae Mathematicae*, 134(1):75–86, 2014.

[5] P. Dorbec, S. Varghese and A. Vijayakumar. Heredity for generalized power domination. *Discrete Mathematics & Theoretical Computer Science*, 18(3), 2016.

[6] M. Dorfling and M. A. Henning. A note on power domination in grid graphs. *Discrete Applied Mathematics*, 154(6):1023–1027, 2006.

[7] D. Ferrero, S. Varghese and A. Vijayakumar. Power domination in honeycomb networks. *Journal of Discrete Mathematical Sciences and Cryptography*, 14(6):521–529, 2011.

[8] D. Gonçalves, A. Pinlou, M. Rao and S. Thomassé. The domination number of grids. *SIAM Journal on Discrete Mathematics*, 25(3):1443–1453, 2011.

[9] T. W. Haynes, S. M. Hedetniemi, S. T. Hedetniemi and M. A. Henning. Domination in graphs applied to electric power networks. *SIAM Journal on Discrete Mathematics*, 15(4):519–529, 2002.

[10] L. Mili, T. Baldwin and R. Adapa. Phasor measurement placement for voltage stability analysis of power systems. *Proceedings of the 29th IEEE Conference on Decision and Control*, 3033–3038, 1990.

Monochromatic Plane Matchings in Bicolored Point Set

A. Karim Abu-Affash* Sujoy Bhore† Paz Carmi‡

Abstract

Motivated by networks interplay, we study the problem of computing monochromatic plane matchings in bicolored point set. Given a bicolored set P of n red and m blue points in the plane, where n and m are even, the goal is to compute a plane matching M_R of the red points and a plane matching M_B of the blue points that minimize the number of crossing between M_R and M_B as well as the longest edge in $M_R \cup M_B$. In this paper, we give asymptotically tight bound on the number of crossings between M_R and M_B when the points of P are in convex position. Moreover, we present an algorithm that computes bottleneck plane matchings M_R and M_B, such that there are no crossings between M_R and M_B, if such matchings exist. For points in general position, we present a polynomial-time approximation algorithm that computes two plane matchings with linear number of crossings between them.

1 Introduction

Let P be a set of n points in the plane, where n is even. A *perfect matching* of P is a perfect matching in the complete Euclidean graph induced by P. A *bottleneck matching* M of P is a perfect matching of P that minimizes the length of the longest edge of M. Let λ_M denote the bottleneck of M, i.e., the length of the longest edge in M. A *plane matching* is a matching with non-crossing edges. Matching problems have been studied extensively, see, e.g., [3, 5, 6, 9, 12, 13].

In this paper, we study the bottleneck plane matching problem in bicolored point set. Let P be a bicolored set consisting of n red and m blue points, such that n and m are even. Let R denote the set of the red points of P, and let B denote the set of the blue points of P. An *L-monochromatic matching* M of P is a matching of P, such that (i) $M = M_R \cup M_B$, where M_R and M_B are plane perfect matchings of R and B, respectively, and (ii) $\lambda_M \leq L$. Let $cr(P, L)$ denote the minimum number of intersection present in any L-monochromatic matching of P. In this paper, we investigate the value $cr(P, L)$ for different bicolored point sets, and study the problem of computing an L-monochromatic matching.

1.1 Related work

The bottleneck plane matching problem for general point set has been first studied by Abu-Affash et al. [2]. They showed that the problem is NP-hard and presented a $2\sqrt{10}$-approximation algorithm. In [1], the authors showed how to compute a plane matching of size at least $n/5$, whose edges have length at most λ^* in $O(n \log^2 n)$ time, and a plane matching of size at least $2n/5$, whose edges have length at most $(\sqrt{2} + \sqrt{3}) \cdot \lambda^*$ in $O(n \log n)$ time, where λ^* is the length of the longest edge of a bottleneck plane matching. Carlsson and Armbruster [4] proved that the bipartite (red/blue) version of the bottleneck plane matching problem is NP-hard, and gave an $O(n^3 \log n)$-time algorithm that solves the problem when the points are on convex position, and an $O(n^4 \log n)$-time algorithm that solves the problem when the red points lie on a line l and the blue points lie above (or below) l.

For bicolored inputs, Merino et al. [10] obtained a tight bound on the number of intersections in monochromatic minimum weight matchings for bicolored point sets. Tokunaga [11] examined non-crossing spanning trees of the red points and the blue points and found a tight bound on the minimum number of intersections between the red and blue spanning trees. Joeris et al. [7] studied the number of intersections for monochromatic planar spanning cycles. In each of the above works, points could have only one of two colors and number of intersection points were proved to be asymptotically linear on total number of points. Kano et al. [8] considered the case of more than two colors and studied the number of intersections for monochromatic spanning trees.

1.2 Our results

In Section 2, we consider the case when the points of P are in convex position. We give a tight bound on $cr(P, L)$ in this case. Moreover, we give an algorithm that computes in $O(|P|^3 + |P|)$ time an L-monochromatic matching of P, where L is the minimum real number such that the edges of the matching do not cross each other, if such a matching exists. In Section 3, we give a polynomial-time approximation algorithm for points in general position.

*Software Engineering Department, Shamoon College of Engineering, Beer-Sheva 84100, Israel, abuaa1@sce.ac.il.

†Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel, sujoy.bhore@gmail.com.

‡Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel, carmip@cs.bgu.ac.il. The research is partially supported by the Lynn and William Frankel Center for Computer Science.

2 Monochromatic Matching of Points in Convex Position

In this section, we consider the case where the points of $P = R \cup B$ are in convex position, i.e., the points of P form vertices of a convex polygon. Let M_R and M_B denote the bottleneck plane matchings of the sets R and B, respectively. Let λ_R and λ_B denote the bottlenecks of M_R and M_B, respectively, and let $\lambda = \max\{\lambda_R, \lambda_B\}$. Notice that $M_R \cup M_B$ is not necessarily a plane matching. Notice also that in any L-monochromatic matching of P, the length of the longest edge is at least λ. Let $cr(P, \lambda)$ be the minimum number of intersection present in any L-monochromatic matching of P with $L = \lambda$. In the following, we give a tight bound of $cr(P, \lambda)$.

2.1 Lower bound on $cr(P, \lambda)$

In Figure 1, we show a set of bicolored points P in convex position, such that the number of crossings between M_R and M_B cannot be better than linear.

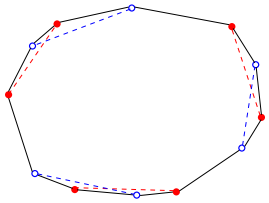


Figure 1: Any L-monochromatic matching of P has linear number of crossings between the red and the blue matchings.

2.2 Upper bound on $cr(P, \lambda)$

For an edge (a, b) and a point w, let $\|(a, b) - w\|$ denote the minimum Euclidean distance between w and any point on (a, b); see Figure 2. For two non-intersecting edges (a, b) and (c, d), such that $\{a, c, d, b\}$ are in convex position and (a, c) does not intersect (b, d), let $\|(a, b) - (c, d)\|$ denote $\max\{|ca|, |db|\}$; see Figure 3. In the following theorem, we give an upper bound on $cr(P, \lambda)$.

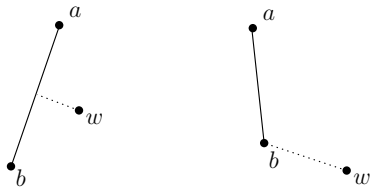


Figure 2: $\|(a, b) - w\|$ is the minimum distance between w and any point on (a, b).

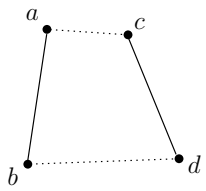


Figure 3: $\|(a, b) - (c, d)\|$ is $\max\{|ca|, |db|\}$.

Theorem 1 *Let $P = R \cup B$ be a set of points in convex position. Then $cr(P, \lambda) \leq \frac{9k}{2}$, where $k = \min\{n, m\}$, $n = |R|$, and $m = |B|$.*

Proof. Assume, w.l.o.g., that $n \leq m$, i.e., $k = n$. For the sake of contradiction, suppose that $cr(P, \lambda) \geq \frac{9n}{2} + 1$. Let M_P be a λ-monochromatic matching of P with minimum number of intersections and $M_P = M_R \cup M_B$. Thus, the number of intersection in M_P is at least $\frac{9n}{2}+1$. Since $|M_R| = \frac{n}{2}$, by pigeonhole principle there is at least one edge $(x, y) \in M_R$ intersected by at least 10 edges from M_B. Assume, w.l.o.g., that x and y are on the X-axis, and notice that $|xy| \leq \lambda$. We define the region $U = \{w \in \mathbb{R}^2 : \|(x, y) - w\| \leq \lambda\}$; see Figure 4. Observe that the total perimeter of the region U is at most $2(\pi + 1)\lambda \approx 8.29\lambda$.

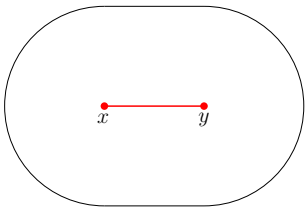


Figure 4: The set of all points of distance at most λ from (x, y).

Let $E = \{(p_1, q_1), (p_2, q_2), \ldots, (p_j, q_j)\}$, where $j \geq 10$, be the set of edges of M_B that intersect (x, y), such that $p_1, p_2, \ldots, p_j$ are above (x, y), $q_1, q_2, \ldots, q_j$ are below (x, y), and, for each $1 \leq i < j$, the intersection point of (p_i, q_i) with (x, y) is to the left of the intersection point of (p_{i+1}, q_{i+1}) with (x, y).

Lemma 2 *There is at least two edges (p_i, q_i), (p_{i+1}, q_{i+1}) in E, such that $\|(p_i, q_i) - (p_{i+1}, q_{i+1})\| \leq \lambda$.*

Proof. For the sake of contradiction, suppose that, for each $1 \leq i < j$, we have $\|(p_i, q_i) - (p_{i+1}, q_{i+1})\| > \lambda$. Thus, for each $1 \leq i < j$, either $|p_i p_{i+1}| > \lambda$ or $|q_i q_{i+1}| > \lambda$. Therefore, the total perimeter of the convex polygon S formed by $\{p_1, p_2, \ldots p_j, q_j, q_{j-1}, \ldots, q_1\}$ is at least $\sum_{i=1}^{j-1} |p_i p_{i+1}| + |q_i q_{i+1}| > 9\lambda$, since $j \geq 10$. This contradicts the fact that S is contained inside the convex region U whose perimeter is at most 8.29λ. □

By Lemma 2, there are two adjacent edges $(p_i, q_i), (p_{i+1}, q_{i+1}) \in E$, such that $|p_i p_{i+1}| \leq \lambda$ and $|q_i q_{i+1}| \leq \lambda$. We replace the edges (p_i, q_i) and (p_{i+1}, q_{i+1}) in M_B by the edges (p_i, p_{i+1}) and (q_i, q_{i+1}) to obtain a new bottleneck plane matching M'_B of B of bottleneck at most λ. Let $M'_P = M_R \cup M'_B$. Clearly, M'_P is a λ-monochromatic matching of P. In the following lemma, we show that the new edges (p_i, p_{i+1}) and (q_i, q_{i+1}) do not increase the number of intersections in M'_P.

Lemma 3 *If (p_i, p_{i+1}) or (q_i, q_{i+1}) intersects an edge (u, v) in M_R, then either (p_i, q_i) or (p_{i+1}, q_{i+1}) intersects (u, v) in M_P.*

Proof. Assume, w.l.o.g, that (p_i, p_{i+1}) intersects an edge (u, v) in M_R. Let Q be the quadrangle obtained by points x, p_i, p_{i+1} and y. Clearly, Q is empty of points of P, since P in convex position. Moreover, since the edges of M_R do not intersect each other, (u, v) and (x, y) do not intersect. Thus, (u, v) intersects (p_i, q_i) or (p_{i+1}, q_{i+1}) ; see Figure 5. □

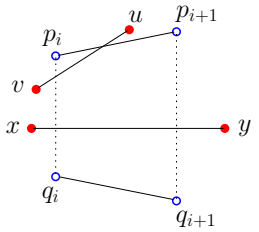


Figure 5: An illustration of Lemma 3.

By Lemma 3, the edges (p_i, p_{i+1}) and (q_i, q_{i+1}) do not increase the number of intersections in M'_P. However, (p_i, p_{i+1}) and (q_i, q_{i+1}) decrease the number of intersections in M'_P, since they do not intersect (x, y). This implies that the number of intersections in M'_P is less than that in M_P, which contradicts the assumption that M_P is a λ-monochromatic matching with minimum number of intersections. This completes the proof of Theorem 1. □

Remark. Abu-affash et al. [1] computed a bottleneck plane matching of a set of points in convex position in $O(|P|^3)$ time. We use their algorithm to compute bottleneck plane matchings of R and of B, separately. Then in additional $O(|P|)$ time, we compute a λ-monochromatic matching M_P of P by considering each edge (x, y) of M_R separately, and if there are two adjacent edges $(p_i, q_i), (p_{i+1}, q_{i+1}) \in M_B$ intersecting (x, y), such that $|p_i p_{i+1}| \leq \lambda$ and $|q_i q_{i+1}| \leq \lambda$, then we replace them by the edges (p_i, p_{i+1}) and (q_i, q_{i+1}). By the above lemmas, the number of intersections in M_P is at most $\frac{9k}{2}$, where $k = \min\{n, m\}$.

2.3 Monochromatic plane matching

Let L be the minimum value such that there exists an L-monochromatic matching of P with no crossings, if such a matching exists. That is, L is the minimum value such that $cr(P, L) = 0$, if such a value exists. An *optimal plane matching* of P is an L-monochromatic matching of P with no crossings. In this section, we present a polynomial-time algorithm that computes an optimal plane matching of P, if such a matching exists,

Let $P = R \cup B$ be a set of points in convex position, where R contains n even number of red points and B contains m even number of blue points. Recall that M_R (resp., M_B) is a bottleneck plane matching of R (resp., B) of bottleneck λ_R (resp., λ_B) and $\lambda = \max\{\lambda_R, \lambda_B\}$. Thus, the length of the longest edge in any L-monochromatic matching of P is at least λ. Our algorithm computes the minimum value L, such that $cr(P, L) = 0$ (if exists), and constructs an optimal plane matching of P.

Let us assume that $R = \{r_1, \ldots, r_n\}$, $B = \{b_1, \ldots, b_m\}$, and $P = \{p_1, \ldots, p_{n+m}\}$, such that $p_i \in R \cup B$. Let $P = R \cup B$ denote the vertices of the convex polygon, and are ordered in clockwise order; see Figure 6. Assume, w.l.o.g., that $p_1 = r_1$. Notice that, a bottleneck plane matching M_R of R and a bottleneck plane matching M_B of B can be computed separately in polynomial time [2]. However, the edges of M_R may cross the edges of M_B. Let M_P denote an optimal plane matching of P. We first state the following observation.

Observation 1 *For each edge (p_i, p_j) in M_P,*
(i) p_i and p_j have the same color,
(ii) (p_i, p_j) partitions the points of P into two disjoint point sets, such that the number of red points and blue points contained in each set is even, and
(iii) $i + j$ is odd; see Figure 6.

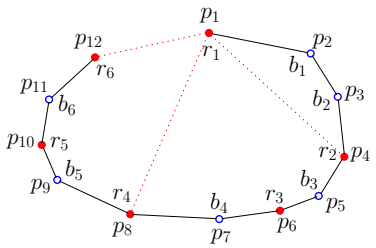


Figure 6: The convex polygon that is obtained from P. p_1 can be matched to the points p_4, p_8, p_{12}.

Based on this observation, we define a weight function for each pair of points in P as follows. For each $1 \leq i < j \leq n+m$, we define $w_{i,j} = |p_i p_j|$, if p_i and p_j are of the same color, $i + j$ is odd, and the edge (p_i, p_j) partitions the points in P into two disjoint point sets such that the number of red points and blue points contained in each

set is even. Otherwise, $w_{i,j} = \infty$. Let $P[i,j]$ denote the set of points $\{p_i, p_{i+1}, \ldots, p_j\}$. Let $\ell_{i,j}$ be the minimum value, such that $cr(P[i,j], \ell_{i,j}) = 0$. Hence, $L = \ell_{1,n+m}$.

Let p_k be the point matched to p_1 in M_P. Thus, $L = \ell_{1,n+m} = \max\{w_{1,k}, \ell_{2,k-1}, \ell_{k+1,n+m}\}$. Therefore, to compute $\ell_{1,n+m}$, we compute $\max\{w_{1,k}, \ell_{2,k-1}, \ell_{k+1,n+m}\}$, for each even k between 2 and $n+m$, and take the minimum over these values. In general, for each $1 \leq i < j \leq n+m$, such that $i+j$ is odd, we compute $\ell_{i,j}$ by

$$\min_{k=i+1,i+3,\cdots,j} \begin{cases} w_{i,k} & \text{, if } k = i+1 = j \\ \max\{w_{i,k}, \ell_{k+1,j}\} & \text{, if } k = i+1 \\ \max\{w_{i,k}, \ell_{i+1,k-1}\} & \text{, if } k = j \\ \max\{w_{i,k}, \ell_{i+1,k-1}, \ell_{k+1,j}\} & \text{, otherwise.} \end{cases}$$

We compute $L = \ell_{1,n+m}$ using dynamic programming. The dynamic programming table T contains $(n+m)$ rows and $(n+m)$ columns and the entry $T[i,j]$ corresponds to a solution of the problem for the set $P[i,j]$. Notice that, each entry $T[i,j]$ is computed by processing $O(n+m)$ entries that are already computed. Hence, we can compute $L = \ell_{i,j} = T[1, n+m]$ in $O((n+m)^3)$ time. Thus, we have the following theorem.

Theorem 4 *Given a set P of n red and m blue points in convex position, where n and m are even, one can compute in $O((n+m)^3)$ time the minimum value L, such that $cr(P, L) = 0$, and if L is finite, then an optimal plane matching can be computed in $O((n+m)^3)$ time.*

3 Monochromatic Matching of Points in General Position

Let $P = R \cup B$ be a set of bicolored points in the plane, such that R contains n red points, B contains m blue points, and n and m are even. In this section, we present a polynomial-time approximation algorithm that computes an L-monochromatic matching of P with at most $O(|P|)$ crossings.

Let M_R and M_B be bottleneck plane matchings of R and B, respectively. Let λ_R and λ_B be the bottlenecks of M_R and M_B, respectively, and let $\lambda = \max\{\lambda_R, \lambda_B\}$. Recall that, in any L-monochromatic matching of P, the length of the longest edge is at least λ. In [2], Abu-affash et al. proved that computing a bottleneck plane matching of a set of points in general position in the plane is NP-Hard. This implies that it is NP-Hard to compute a λ-monochromatic matching of P since $\lambda = \max\{\lambda_R, \lambda_B\}$. However, in this section we give a polynomial time approximation algorithm that computes a $(2\sqrt{10}\lambda)$-monochromatic matching of P with linear number of intersections. That is, we compute an L-monochromatic matching M_P with $L = 2\sqrt{10}\lambda$, such that total number of intersections between the red and the blue edges in M_P is $O(|P|)$.

Let M_R^* and M_B^* be bottleneck matchings of R and B that may have crossings, respectively. Let λ_R^* and λ_B^* be the bottlenecks of M_R^* and M_B^*, respectively. Since M_R^* and M_B^* can be computed in polynomial-time [5], we assume, w.l.o.g., that $\lambda_B^* \leq \lambda_R^*$. Thus, since $\lambda_B^* \leq \lambda_B$ and $\lambda_R^* \leq \lambda_R$, we have $\lambda_R^* \leq \lambda$. In the rest of this section, we prove the following theorem.

Theorem 5 *Let $P = R \cup B$ be a set of bicolored points in the plane, such that contains n red points, B contains m blue points, and n and m are even. Then, there is a polynomial-time approximation algorithm that computes a $(2\sqrt{10}\lambda_R^*)$-monochromatic matching M_P of P, such that the number of intersections in M_P is $O(|P|)$.*

Proof. Let M_P^* be $M_R^* \cup M_B^*$. We begin by laying a grid of side length $2\sqrt{2}\lambda_R^*$. Assume, w.l.o.g. that no point of P lies on the boundary of a grid cell. Each edge of M_P^* is either contained in a grid cell or connects two points from two adjacent cells (i.e., two cells sharing a side or corner). For an edge e in M_P^*, we say that e is an *internal* edge if it is contained in a grid cell, and an *external* edge otherwise. An external edge can be of two types: *straight* external edge (s-edge for short) connects between two points in two grid cells that share a side, and *diagonal* external edge (d-edge for short) connects between two points in two grid cells that share a corner. Let C be a grid cell. The degree of C is denoted by $deg(C)$ and it is equal to the number of external edges of M_P^* with an endpoint in C.

Our algorithm consists of two stages. In Stage 1, we convert M_P^* into a new matching M_P' of P, such that $deg(C) \leq 8$, for each grid cell C, and, in Stage 2, we construct $(2\sqrt{10}\lambda_R^*)$-monochromatic matching M_P based on M_P'.

Stage 1

In this stage, which is taken from [2], we convert M_R^* (resp., M_B^*) to a new perfect matching M_R' (resp., M_B') of R (resp., of B). The conversion is done by applying a sequence of rules on M_R^* and on M_B^* separately. Each rule is applied as long as there is an instance in the current matching to which it can be applied. When there are no more such instances, we move to the next rule in the sequence.

Rule 1: If there are two d-edges (a,b) and (c,d) associated with the same corner, such that a and c in the same cell and b and d in the same cell, then these edges are replaced by two internal edges (a,c) and (b,d); see Figure 7(a).

Rule 2: If there are two d-edges (a,b) and (c,d) associated with the same corner, such that a, b, c, and d are in different cells, then these edges are replaced by two s-edges (a,c) and (b,d); see Figure 7(b).

For a d-edge (a,b) that connects between two cells C and C' and associated with a corner r, we define a

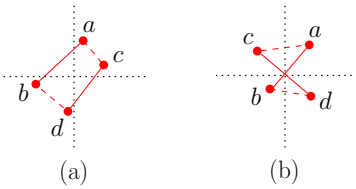


Figure 7: (a) Two d-edges (a, b) and (c, d) are replaced by two internal edges (a, c) and (b, d), and (b) two d-edges (a, b) and (c, d) are replaced by two s-edges.

a *danger zone* of (a, b) in each of the two other cells sharing r as an isosceles right triangle; see Figure 8(a). The length of its sides is $\sqrt{2}\lambda_R^*$ and it is semi-open (i.e., it does not include the hypotenuse of length $2\lambda_R^*$).

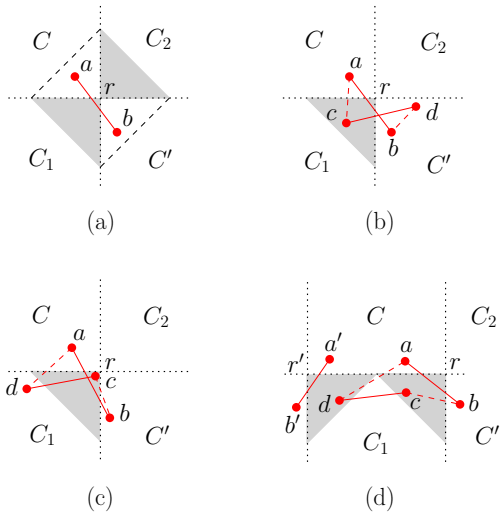


Figure 8: (a) The danger zone defined by the edge (a, b) in C_1 and C_2, (b) (c, d) is an s-edge, (c) (c, d) is an internal edge and d is not in another danger zone in C_1, and (d) (c, d) is an internal edge and d is in another danger zone in C_1.

Rule 3: This rule is applied on a d-edge (a, b) and an edge (c, d) with an endpoint c inside a danger zone defined by (a, b); see Figure 8(b–d).

- If (c, d) is an s-edge, then its other endpoint d is in one of the cells C_1 or C_2; see Figure 8(b). In this case, we replace (a, b) and (c, d) by an internal edge (b, d) and an s-edge (a, c).
- If (c, d) is an internal edge, then consider the other endpoint d of (c, d). If d is not in a danger zone in C_1 defined by another d-edge, then we replace (a, b) and (c, d) by two s-edges (a, d) and (b, c); see Figure 8(c). If d is in a danger zone in C_1 defined by another d-edge (a', b'), then (a', b') is associated with one of the two corners of C_1 adjacent to the corner r; see Figure 8(d). Therefore, either C or C' contains an endpoint of both (a, b) and (a', b'). We replace (a, b) and (c, d) by two s-edges (a, d) and (b, c).

Rule 4: This rule is applied on a d-edge (a, b) and an s-edge (c, d), such that a and c are in the same cell, and b and d are in two adjacent cells that share a side; see Figure 9(a). We replace (a, b) and (c, d) by an internal edge (a, c) and an s-edge (b, d).
Rule 5: This rule is applied on two s-edges (a, b) and (c, d), such that a and c are in the same cell, and b and d are in the same cell; see Figure 9(b). We replace (a, b) and (c, d) by two internal edge (a, c) and (b, d).

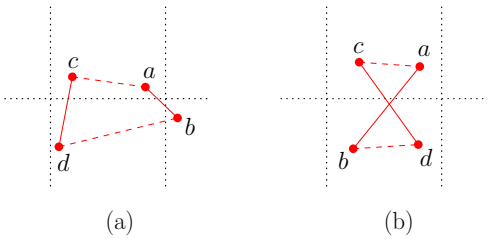


Figure 9: (a) Rule 4, and (b) Rule 5.

Lemma 6 (Lemma 3.2 in [2]) *Let M'_P be $M'_R \cup M'_B$. Then, M'_P has the following properties.*

1. *An edge is either contained in a single cell, or connects between a pair of points in two adjacent cells.*
2. *A corner of the grid has at most one d-edge of M'_R and at most one d-edge of M'_B associated with it.*
3. *A d-edge in M'_P is of length at most λ_R^*.*
4. *The two danger zones defined by a d-edge of M'_R (resp., M'_B) are empty of points of R (resp., of B).*
5. *Each cell C contains at most 4 external edges of M'_R and 4 external edges of M'_B, and $deg(C) \leq 8$.*
6. *If a d-edge in M'_R (resp., in M'_B) connecting between cells C_1 and C_2, and C is a cell sharing a side with both C_1 and C_2, then there is no s-edge in M'_R (resp., in M'_B) connecting between C and either C_1 or C_2.*

Stage 2

In this stage, we construct a $(2\sqrt{10}\lambda_R^*)$-monochromatic matching M_P of P based on $M'_P = M'_R \cup M'_B$. We consider each cell separately. Let C be a non-empty grid cell and let R_C (resp., B_C) be the set of points of R (resp., B) lying in C. Recall that there are at most 4 external edges of M'_R and at most 4 external edges of M'_B associated with C. We use the same procedure of [2] to select the points in R_C (resp., in B_C) that will serve as endpoints of external edges of M'_R (resp., of M'_B), such that the external edges of M'_R (resp., of M'_B) that will be connected to these point do not cross each other. For each external edge e of M'_R (resp., of

M'_B) connecting between two cells C_1 and C_2, let a and b be the points that were chosen as the endpoint of e in C_1 and C_2, respectively. We add the edge (a, b) to M_R (resp., to M_B). It has be proved in [2] that the length of (a, b) is at most $2\sqrt{10}\lambda^*_R$.

Let $R^E_C \subseteq R_C$ (resp., $B^E_C \subseteq B_C$) be the set of points of R_C (resp., of B_C) that were chosen as endpoints of external edges, and let $R^I_C = R_C \setminus R^E_C$ (resp., $B^I_C = B_C \setminus B^E_C$). By the way we select the points of R^E_C (resp., of B^E_C), the points in R^I_C (resp., B^I_C) are contained in a convex polygon $X_R \subseteq C$ (resp., $X_B \subseteq C$), such that any external edge of M'_R (resp., M'_B) with endpoint in R^E_C (resp., in R^E_C) does not intersect the interior of X_R (resp., X_B). For each cell C, we compute a minimum weight matching $M(R^I_C)$ (resp., $M(R^I_C)$) of the points in R^I_C (resp., in B^I_C) and we add it to M_R (resp., to M_B). Clearly, the edges of $M(R^I_C)$ do not cross each other and do not cross the external edges that are connected to points from R^E_C, and the edges of $M(B^I_C)$ do not cross each other and do not cross the external edges that are connected to points from B^E_C. Moreover, the length of each edge in $M(R^I_C)$ and in $M(B^I_C)$ is at most $4\lambda^*_R$.

Let M_P be $M_R \cup M_B$. Since M_R (resp., M_B) is a plane matching and each edge in M_R (resp., in M_B) is of length at most $2\sqrt{10}\lambda^*_R$, M_P is a $(2\sqrt{10}\lambda^*_R)$-monochromatic matching of P. We now show that the number of intersections in M_P, i.e., between the edges of M_R and the edges of M_B, is $O(|P|)$.

We bound the number of intersections for each cell separately. In each cell C, there would be three types of intersections: intersection between an external edge of M_R and an external edge of M_B, intersection between an external edge of M_R and an internal edge of M_B (and vice versa), and intersection between an internal edge of M_R and an internal edge of M_B. We bound the number of intersection of each type separately.

Recall that the number of external edges of M_R (resp., of M_B) that have one endpoint in R^E_C (resp., in B^E_C) is at most 4. This implies that each external edge of M_R can be intersected by at most 4 external edges of M_B, and thus the total number of intersections between the external edges of M_P that have endpoint in C is at most $4|R^E_C|$. Moreover, the total number of intersections between the external edges of M_P that have endpoint in C and the internal edges of $M(R^I_C) \cup M(B^I_C)$ is at most $2(|R^I_C| + |B^I_C|)$. Finally, the total number of intersections between the internal edges of $M(R^I_C)$ and the internal edges of $M(B^I_C)$ is at most $(|R^I_C| + |B^I_C|)/2$, by Theorem 1 in [10]. Thus, the number of intersections produced by the points in C is at most $4(|R_C| + |B_C|)$. Therefore, the number of intersections in M_P is at most $4(|R| + |B|) = 4|P|$. This completes the proof. □

References

[1] A. K. Abu-Affash, A. Biniaz, P. Carmi, A. Maheshwari, and M. H. M. Smid. Approximating the bottleneck plane perfect matching of a point set. *Comput. Geom.*, 48(9):718–731, 2015.

[2] A. K. Abu-Affash, P. Carmi, M. J. Katz, and Y. Trabelsi. Bottleneck non-crossing matching in the plane. *Comput. Geom.*, 47(3):447–457, 2014.

[3] G. Aloupis, J. Cardinal, S. Collette, E. D. Demaine, M. L. Demaine, M. Dulieu, R. Fabila-Monroy, V. Hart, F. Hurtado, S. Langerman, M. Saumell, C. Seara, and P. Taslakian. Matching points with things. In *LATIN, volume 6034 of LNCS*, pages 456–467, 2010.

[4] J. G. Carlsson, B. Armbruster, S. Rahul, and H. Bellam. A bottleneck matching problem with edge-crossing constraints. *Int. J. Comput. Geometry Appl.*, 25(4):245–262, 2015.

[5] M. S. Chang, C. Y. Tang, and R. C. T. Lee. Solving the euclidean bottleneck matching problem by k-relative neighborhood graphs. *Algorithmica*, 8(1–6):177–194, 1992.

[6] A. Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.

[7] B. Joeris, I. Urrutia, and J. Urrutia. Geometric spanning cycles in bichromatic point sets. *Graphs and Combinatorics*, 31(2):453–465, 2015.

[8] M. Kano, C. Merino, and J. Urrutia. On plane spanning trees and cycles of multicolored point sets with few intersections. *Inf. Process. Lett.*, 93(6):301–306, 2005.

[9] L. Lovász and M. D. Plummer. *Matching Theory*. Elsevier Science Ltd, 1986.

[10] C. Merino, G. Salazar, and J. Urrutia. On the intersection number of matchings and minimum weight perfect matchings of multicolored point sets. *Graphs and Combinatorics*, 21(3):333–341, 2005.

[11] S. Tokunaga. Intersection number of two connected geometric graphs. *Inf. Process. Lett.*, 59(6):331–333, 1996.

[12] P. M. Vaidya. Geometry helps in matching. *SIAM J. Comput.*, 18(6):1201–1225, 1989.

[13] K. R. Varadarajan. A divide-and-conquer algorithm for min-cost perfect matching in the plane. In *FOCS*, pages 320–331, 1998.

Bottleneck Bichromatic Full Steiner Trees

A. Karim Abu-Affash* Sujoy Bhore† Paz Carmi‡ Dibyayan Chakraborty§

Abstract

Given two sets of points in the plane, Q of n (terminal) points and S of m (Steiner) points, where each of Q and S contains bichromatic points (red and blue points), a full bichromatic Steiner tree is a Steiner tree in which all points of Q are leaves and each edge of the tree is bichromatic (i.e., connects a red and a blue point). In the bottleneck bichromatic full Steiner tree (BBFST) problem, the goal is to compute a bichromatic full Steiner tree T, such that the length of the longest edge in T is minimized. In k-BBFST problem, the goal is to find a bichromatic full Steiner tree T with at most $k \leq m$ Steiner points from S, such that the length of the longest edge in T is minimized. In this paper, we present an $O((n+m)\log m)$ time algorithm that solves the BBFST problem. Moreover, we show that k-BBFST problem is NP-hard and we give a polynomial-time 9-approximation algorithm for the problem.

1 Introduction

Given a weighted graph $G = (V, E)$ with $V = Q \cup S$, where Q and S are sets of terminal and Steiner points, respectively, a Steiner tree is an acyclic connected subgraph of G spanning all vertices of Q. Informally, Steiner points are new auxiliary nodes that can be added to the network to improve its performance. In the classical *Steiner tree* problem, the goal is to find a Steiner tree T, such that the length of the edges of T is minimized. This problem has been shown to be NP-complete [6, 16], and for arbitrary weighted graphs, many approximation algorithms have been proposed [8, 18, 19].

In the geometric context, i.e., Q and S are disjoint sets of points in the plane, G is the complete graph over $V = Q \cup S$, and the weight of each edge (p, q) in G is the Euclidean distance between p and q. Arora [4] showed that the geometric Steiner tree problem can be efficiently approximated close to optimal.

A Steiner tree is *full* if all terminals are leaves of the tree. In the bottleneck full Steiner tree problem (BFST), the goal is to compute a full Steiner tree with minimum bottleneck (i.e., the length of the longest edge). The k-BFST problem is a restricted version of the BFST problem, for which, in addition to the sets Q and S, we are given a positive integer k, and the goal is to compute a full Steiner tree T with at most k Steiner points such that the bottleneck of T is minimized. Abu-Affash [1] gave a $O((n+m)\log^2 m)$ algorithm for the BFST problem and showed that the k-BFST problem is NP-hard but admits a polynomial-time 4-approximation algorithm. Later, Biniaz et al [10] gave an $O((n+m)\log m)$ algorithm for the BFST problem.

We consider the BFST and the k-BFST problems in bichromatic point sets. Given two sets of points in the plane; a set Q of n red and blue terminals and a set S of m red and blue Steiner points, the goal in the bottleneck bichromatic full Steiner tree (BBFST) problem is to find a full Steiner tree T such that each edge in T connects a red and a blue point and the bottleneck of T is minimized. We refer to this tree as a bichromatic full Steiner tree. In the k-BBFST problem, the goal is to compute a bichromatic full Steiner tree T with at most k Steiner points, such that its bottleneck is minimized, where $k \leq m$ is a given positive integer. The bichromatic input appeared in many geometric problems; for example, red-blue intersection [3], red-blue separation [5, 12, 14, 15], and red-blue connection problems [2, 7, 11].

In this paper, we show how to generalize the algorithms in [1] to solve the BBFST problem and to approximate the k-BBFST problem. More precisely, we present an $O((n+m)\log m)$ algorithm that solves the BBFST problem, we show that the k-BBFST problem is NP-hard, and we give a polynomial-time that approximates it within a factor 9.

2 Exact Algorithm for BBFST

Given a set Q of n red and blue terminals and a set S of m red and blue Steiner points in the plane, we present an $O((n+m)\log m)$ time algorithm that computes a bichromatic full Steiner tree of minimum bottleneck. We refer to such a tree as an optimal bichromatic

*Software Engineering Department, Shamoon College of Engineering, Beer-Sheva 84100, Israel, abuaa1@sce.ac.il.

†Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel, sujoy.bhore@gmail.com. The research is partially supported by the Lynn and William Frankel Center for Computer Science.

‡Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel, carmip@cs.bgu.ac.il. The research is partially supported by the Lynn and William Frankel Center for Computer Science.

§Advanced Computing and Microelectronics Unit, Indian Statistical Institute, Kolkata, India, dibyayancg@gmail.com.

full Steiner tree of Q.

Let Q_R and Q_B be the sets of red and blue terminal points of Q, respectively. Similarly, let S_R and S_B be the sets of red and blue Steiner points of S, respectively. We assume that neither S_R nor S_B is empty. Let $MST(S)$ be a minimum-weight bichromatic spanning tree of S (i.e., of the complete bichromatic graph of S_R and S_B). Let $S(T)$ be the set of Steiner points in a bichromatic full Steiner tree T.

Lemma 1 *There exists an optimal bichromatic full Steiner tree T^* of Q, such that $MST(S(T^*))$ is a subtree of $MST(S)$.*

Proof. Let T be an optimal bichromatic full Steiner tree of Q. Let $e = (p_r, p_b)$ be an edge in $MST(S(T))$ but not in $MST(S)$. Let P be the path between p_r and p_b in $MST(S)$. We know that, each edge in P is of length at most $|p_r p_b|$. Moreover, if $T \cup P$ creates a cycle, then this cycle contains e. We add the edges of P to T and we break the produced cycles (by removing the longest edge from each cycle) to obtain a new optimal bichromatic full Steiner tree. By repeating this process for each edge $e \in MST(S(T)) \setminus MST(S)$, we obtain an optimal bichromatic full Steiner tree T^* satisfying the lemma. □

Let $e_1, e_2, \ldots, e_{m-1}$ be the edges of $MST(S)$ sorted in non-decreasing order by their length. For an edge $e_i \in MST(S)$, let $\mathcal{T}_i$ be the forest obtained from $MST(S)$ by deleting all edges of length greater than $|e_i|$ from $MST(S)$. By Lemma 1, there exists an optimal bichromatic full Steiner tree T^* of Q such that $MST(S(T^*))$ is a tree of $\mathcal{T}_i$, for some edge $e_i \in MST(S)$. Thus, by performing binary search on the lengths of edges of $MST(S)$, we can find a forest $\mathcal{T}_i$ that contains a tree T, such that, by connecting each point in Q to its closest point of opposite color in T, we obtain an optimal bichromatic full Steiner tree of Q.

Let λ be the bottleneck of the optimal bichromatic full Steiner tree. For an edge $e_i \in MST(S)$, we decide in $O(n+m)$ time whether $|e_i| > \lambda$ or $|e_i| \leq \lambda$, using the procedure of [10]. (In order to handle the case that $\lambda < |e_1|$ or $\lambda > |e_{m-1}|$, we add the values $|e_0| = 0$ and $|e_m| = \infty$ to the search space.) Therefore, we can find an $0 \leq i \leq m-1$, such that $|e_i| < \lambda \leq |e_{i+1}|$ in $O((n+m)\log m)$ time. If $|e_i| < \lambda < |e_{i+1}|$, then the optimal bichromatic full Steiner tree of Q is obtained by a tree T from the forest $\mathcal{T}_i$; see Figure 1(a). If $\lambda = |e_{i+1}|$, then the optimal bichromatic full Steiner tree of Q is obtained by a tree T from the forest $\mathcal{T}_{i+1}$; see Figure 1(b). Thus, in both cases, we can find the tree T in the set $\mathcal{T}_i \cup \mathcal{T}_{i+1}$, such that, by connecting each terminal in Q to its closest point of opposite color in T, we obtain an optimal bichromatic full Steiner tree of Q. We conclude by the following theorem.

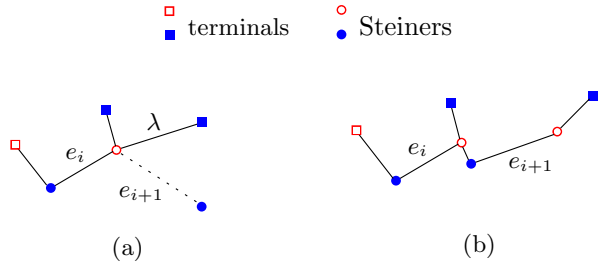


Figure 1: The optimal full bichromatic Steiner tree is obtained (a) from $\mathcal{T}_i$, when $|e_i| < \lambda < |e_{i+1}|$ and (b) from $\mathcal{T}_{i+1}$, when $\lambda = |e_{i+1}|$.

Theorem 2 *The BBFST problem can be solved in $O((n+m)\log m)$ time.*

3 Approximation Algorithm for k-BBFST

Given two sets of points in the plane; a set Q of n red and blue terminal points, a set S of m red and blue Steiner points, and a positive integer $k \leq m$, the goal in the k-BBFST problem is to compute a bichromatic full Steiner tree with at most k Steiner points from S and its bottleneck is minimized. In this section, we first prove that the k-BBFST problem is NP-hard. Then, we present a polynomial-time approximation algorithm with performance ratio 9.

3.1 Hardness proof

We prove the following theorem.

Theorem 3 *The k-BBFST problem is NP-hard.*

Proof. We adopt that proof of Abu-Affash [1] for the k-BFST problem. The proof is based on a reduction from the problem **Connected vertex cover in planar graphs with maximum degree 4** which is NP-complete [17]. Given a planar graph $G = (V, E)$ with vertex degree at most 4 and an integer k, does there exist a vertex cover V^* for G such that $|V^*| \leq k$ and the subgraph of G induced by V^* is connected?

Given a planar graph $G = (V, E)$ with vertex degree at most 4 and an integer k, we construct, in polynomial time, two sets Q and S and compute an integer k', such that G has a connected vertex cover of size at most k if and only if there exists a bichromatic full Steiner tree T of Q with at most k' Steiner points and bottleneck at most 1.

Let $G = (V, E)$ be a planar graph with vertex degree at most 4 and let k be an integer. Let $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$ be the vertices and the edges of G, respectively. We first embed G into a rectangular grid, with distance at least 4 between adjacent vertices. Each vertex $v_i \in V$ corresponds to some

grid vertex and each edge $e = (v_i, v_j) \in E$ corresponds to a rectilinear path p_e, consisting of some horizontal and vertical elementary grid segments, whose endpoints are the grid vertices corresponding to v_i and v_j. In addition, these paths are pairwise disjoints; see Figure 2. This embedding can be done in $O(n)$ time and the size of the grid is at most $n-2$ by $n-2$; see [20].

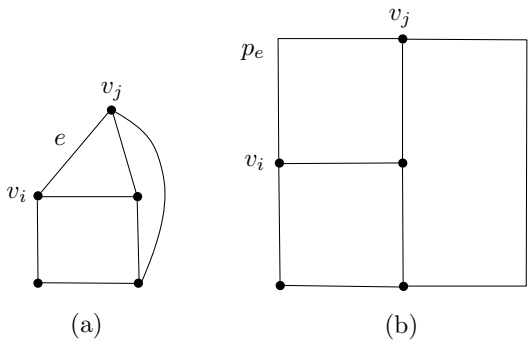


Figure 2: (a) A planar graph $G = (V, E)$, and (b) the embedded graph $G' = (V', E')$ of G.

For each vertex $v_i \in V$ we replace v by a blue Steiner point v'_i; see Figure 3. Let $V' = \{v'_1, v'_2, \ldots, v'_n\}$ be the set of these Steiner points, and let $E' = \{p_{e_1}, p_{e_2}, \ldots, p_{e_m}\}$ be the set of edges (paths) corresponding to the edges of E. We now place two types of points on the interior of each edge $p_e \in E'$. Let $|p_e|$ denote the total length of the grid segments of p_e. We place $|p_e| - 1$ bichromatic Steiner points (red and blue points alternatively) on p_e, such that the distance between any adjacent points is exactly 1, and denote by $s(e)$ this set of Steiner points. Moreover, for each set $s(e)$, we place a red terminal between (in the middle of) every two adjacent points in $s(e)$. Denote by $t(e)$ this set of terminals and notice that $|t(e)| = |p_e| - 2$; see Figure 3. Finally, we set

$$Q = \bigcup_{e \in E} t(e)\,,$$
$$S = V' \cup \bigcup_{e \in E} s(e) \text{ and}$$
$$k' = \sum_{e \in E} |s(e)| - m + 2k - 1\,.$$

For each edge $p_e \in E'$, let $c(e)$ be the set of Steiner points in $s(e)$ except the endpoints, i.e., except the first and the last points. Observe that, connecting every adjacent two Steiner points in $c(e)$ (to form a bichromatic path) and connecting each terminal in $t(e)$ to its closest blue Steiner point in $c(e)$ produces a bichromatic full Steiner tree of $t(e)$ with $|s(e)| - 2$ Steiner points and bottleneck 1. On the other hand, observe that at least $|s(e)| - 2$ Steiner points are necessary to construct a bichromatic full Steiner tree of $t(e)$ with bottleneck at most 1. Denote by T_e such a bichromatic full Steiner tree; see Figure 3.

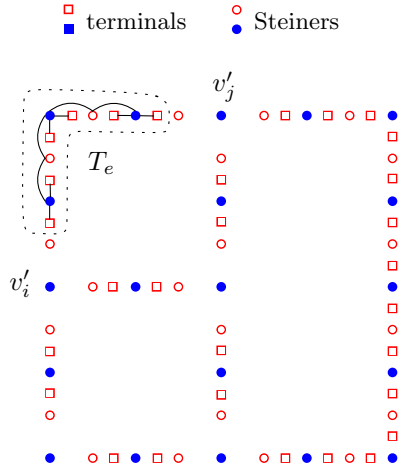


Figure 3: The produced sets: V', $s(e)$, and $t(e)$. T_e is the bichromatic full steiner tree of $t(e)$.

Clearly, the number of points in $Q \cup S$ is $O(n^4)$. Therefore, the reduction can be done in polynomial time. We now prove the correctness of the reduction. Suppose that G has a connected vertex cover V^* with $|V^*| \leq k$. We construct a bichromatic full Steiner tree of Q as follows. For each edge $e \in E$, we construct the tree T_e (as described above). Let T' be any spanning tree of the subgraph of G induced by V^*. This spanning tree exists by the connectivity of V^* and contains $|V^*| - 1$ edges. For each edge $e = (v_i, v_j) \in T'$, we connect the corresponding points $v'_i, v'_j \in S$ (by two edges of length 1) to the tree T_e using their adjacent (first and last) points in $s(e)$. And, for each edge $e = (v_i, v_j) \in E \setminus T'$, we select one endpoint v_i of e that belongs to V^* and we connect v'_i (by an edge of length 1) to the tree T_e using its adjacent red Steiner point in $s(e)$. It is easy to see that the constructed tree is a bichromatic full Steiner tree of Q and it has $|V^*| + \sum_{e \in E}(|s(e)| - 2) + 2(|V^*| - 1) + m - (|V^*| - 1) \leq \sum_{e \in E} |s(e)| - m + 2k - 1 = k'$ Steiner points and bottleneck exactly 1.

Conversely, suppose that there exists a bichromatic full Steiner tree T of Q with at most k' Steiner points and bottleneck at most 1. Let V^* be the subset of points of V' that appear in T, and let T' be the subtree of T spanning V^*. For each subset $t(e) \subseteq Q$, let T_e be the subtree of T spanning the points in $t(e)$. Since the bottleneck of T is at most 1, (i) by the above observation, T_e contains at least $|s(e)| - 2$ Steiner points, and (ii) each tree T_e is connected to at least one point from V^*, which implies that the set of vertices in G corresponding to the points in V^* is a connected vertex cover of G. Moreover, a tree T_e which is also a subtree of T' is connected to two points from V^* via the

endpoints of $s(e)$ (there are $|V^*|-1$ such trees), and a tree T_e which is not a subtree of T' is connected to one point from V^* via one endpoint of $s(e)$ (there are $m-(|V^*|-1)$ such trees). Thus, T contains at least $|V^*|+\sum_{e\in E}(|s(e)|-2)+2(|V^*|-1)+m-(|V^*|-1)$ Steiner points. On the other hand, T contains at most $k'=\sum_{e\in E}|s(e)|-m+2k-1$ Steiner points. This implies that V^* is of size at most k, which completes the proof. □

3.2 Approximation algorithm

We devise a polynomial-time approximation algorithm for computing a bichromatic full Steiner tree with at most k Steiner points (k-BFST for short), such that its bottleneck is at most 9 times the bottleneck of an optimal k-BFST.

Let Q_R and Q_B be the sets of red and blue terminal points of Q, respectively. Similarly, let S_R and S_B be the sets of red and blue Steiner points of S, respectively. We assume that S_R and S_B contains at least one red and one blue point, respectively. Let $G=(V,E)$ be the graph with $V=Q\cup S$ and $E=(Q_R\times S_B)\cup(Q_B\times S_R)\cup(S_R\times S_B)$. We assume, w.l.o.g., that $E=\{e_1,e_2,\cdots,e_l\}$, such that $|e_1|\leq|e_2|\leq\cdots\leq|e_l|$. Notice that, the bottleneck of an optimal k-BFST is a length of an edge from E. For an edge e_i, Let $G_i=(V,E_i)$ be the graph, such that $E_i=\{e_j\in E:|e_j|\leq|e_i|\}$. We devise a procedure which either constructs a k-BFST of Q in G with bottleneck at most 9 times $|e_i|$ or it says that G_i does not contain a k-BFST of Q.

Let G_i^2 be the 2nd power graph of G_i, i.e., G_i^2 has the same set of vertices as G_i and an edge between two vertices if and only if there is a path that contains at most 2 edges between them in G_i. Let $G_i^2(Q)$ be the sub-graph of G_i^2 induced by Q and let Q' be a maximal independent set in $G_i^2(Q)$. Notice that, since all the edges in E are bichromatic, a red terminal and a blue terminal cannot be connected to a same Steiner point in G_i. Hence, a red terminal and a blue terminal cannot be connected to each other in G_i^2. Thus, if $|Q'|=1$, then Q contains points of one color and we can construct a k-BFST of bottleneck at most $3|e_i|$ as follows. Let p be the only point in Q' and assume, w.l.o.g., that p is a red point. We select a blue Steiner point s that is connected to p in G_i and we connect it to all points of Q. Since there is an edge in G_i^2 between p and each other point $q\in Q$, we have $|pq|\leq 2|e_i|$, and therefore, $|sq|\leq 3|e_i|$.

Thus, we assume that $|Q'|>1$. For any two points $p,q\in Q$, let $\delta_i(p,q)$ be the path between p and q in G_i that contains minimum number of Steiner points. Let $G'=(Q',E')$ be the complete graph over Q'. For each edge (p,q) in E', we assign a weight $w(p,q)$ which is equal to the number of Steiner points in $\delta_i(p,q)$. Let $MST(G')$ be the minimum spanning tree of G' under w. We define the normalized weight of $MST(G')$ as $W(MST(G'))=\sum_{e\in MST(G')}\lfloor w(e)/2\rfloor$.

Lemma 4 *If G_i contains a k-BFST of Q', then $W(MST(G'))\leq k$*

Proof. Let T be a k-BFST of Q' in G_i. We construct a spanning tree T' of G' such that $W(T')\leq k$. We start by T and we transform it into T' by an iterative process. We start by selecting an arbitrary Steiner point as the root of T; see Figure 4. In each iteration, we select the deepest leaf p in the rooted tree, which is a terminal, and we connect it to its closest terminal q by an edge (p,q) of weight equal to the number of Steiner points between them. Let s be the lowest common ancestor of p and q. We then remove the Steiner points between p and s. In the last iteration, we remove all of the remaining points.

For example, in Figure 4, we show a construction of T' from T. In iteration 1, we select p_1, connect it to p_2 by an edge of weight 4 and remove the points between p_1 and s_1. In iteration 2, we select p_3, connect it to p_4 by an edge of weight 4, and remove the points between p_3 and s_2. In iteration 3, we select p_6, connect it to p_5 by an edge of weight 3, and remove the points between p_6 and s_3. In iteration 4, we select p_5, connect it to p_4 by an edge of weight 6, and remove the points between p_5 and s_4. In the last iteration, we select p_2, connect it to p_4 by an edge of weight 5, and remove the all the remaining points between p_2 and p_4.

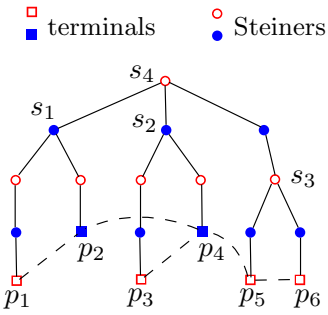


Figure 4: Constructing T' from T.

Since, in each iteration, we select the deepest terminal, we add to T' an edge (p,q) of weight $w(p,q)$, and we remove at least $\lfloor w(p,q)/2\rfloor$ Steiner points from T. Thus, we have $W(T')=\sum_{e\in T'}\lfloor w(e)/2\rfloor\leq k$. Finally, since T' is also a spanning tree of G', we have $W(MST(G'))\leq W(T')\leq k$. □

We now describe the algorithm. For each edge $e_i\in E$ in the sorted order, we construct the graphs G_i, G_i^2, and $G_i^2(Q)$. Then, we compute a maximal independent set Q' in $G_i^2(Q)$. If $|Q'|=1$, then we construct a k-BFST of Q with bottleneck at most 3 times $|e_i|$. Otherwise, we construct the complete graph G' over Q', and we

compute a minimum spanning tree $MST(G')$ of G' with respect to the weight function w. If $W(MST(G')) > k$, then we proceed to the next edge e_{i+1}. Otherwise, we construct a k-BFST of Q with bottleneck at most 9 times $|e_i|$ as follows.

For each edge $(p, q) \in T$, there is a bichromatic path $\delta_i(p, q)$ between p and q in G_i that contains $w(p, q)$ Steiner points. We select $\lfloor w(p, q)/2 \rfloor$ Steiner points on any shortest Steiner path between p and q in G_i by the following procedure.

We select an arbitrary leaf p in $MST(G')$ and we traverse $MST(G')$ starting from p. Let q be the point that is connected to p in $MST(G')$. Set $S' = \emptyset$. We call the recursive procedure $SelectSteiners(p, q, color(p), S')$ (Procedure 1) that selects at most k Steiner points and adds them to S'; see also Figure 5.

Procedure 1 $SelectSteiners(p, q, color, S')$

```
1: j ← w(p, q)
2: let s_1, s_2, ..., s_j be the Steiner points in δ_i(p, q)
3: x ← 0
4: if color(s_1) ≠ color then
5:     i ← 1
6: else
7:     i ← 2
8: while i + 3x ≤ j do
       S' ← S' ∪ {s_{i+3x}}
       x ← x + 1
9: for each (q, t) ∈ MST(G'), such that t ≠ p do
       SelectSteiners(q, t, color(s_{i+3(x−1)}), S')
```

It is not hard to see that for each edge (p, q) in $MST(G')$, we add to S' at most $\lfloor w(p, q)/2 \rfloor$ Steiner points. Therefore, $|S'| \leq k$. Next, we construct a minimum-weight bichromatic spanning tree $MST(S')$ of S' (i.e., of the complete bichromatic (Euclidean) graph over S'). Notice that, each edge in $MST(S')$ is of length at most $5|e_i|$; see Figure 5. Finally, we connect each terminal in Q to its nearest opposite color Steiner point in S' to obtain a bichromatic full Steiner tree. This guarantees that each terminal in Q' is connected to a Steiner point with an edge of length at most $7|e_i|$; see Figure 5, and each terminal in $Q \setminus Q'$ is connected to a Steiner point with an edge of length at most $9|e_i|$.

Remark. If Q' contains only one red and one blue points p and q, respectively, $k = 2$, and $MST(G')$ is a path between p and q that contains exactly 2 Steiner points, a blue Steiner point s_1 and a red Steiner point s_2, then we construct a k-BFST by connecting all the points in Q_R to s_1 and all the points in Q_B to s_2. This k-BFST contains exactly 2 Steiner points and its bottleneck is at most $3|e_i|$.

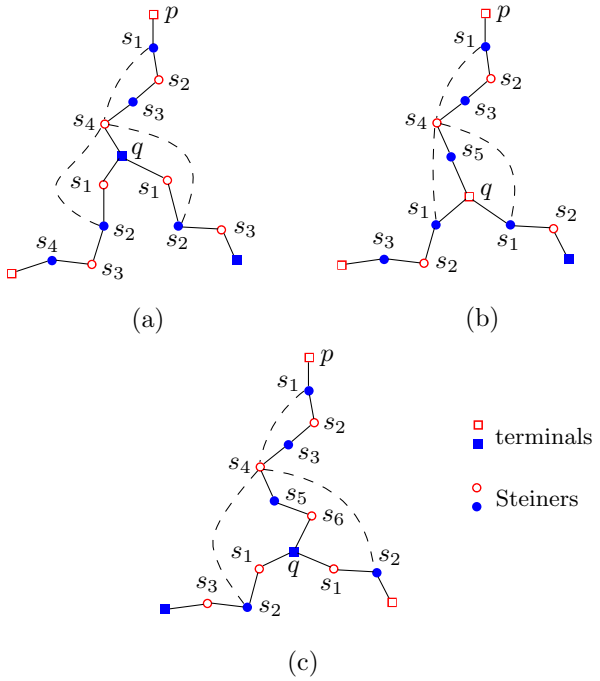


Figure 5: Illustrating the selection of the Steiner points in Procedure 1.

Lemma 5 *Our algorithm constructs a k-BFST of Q with bottleneck at most 9 times the bottleneck of an optimal k-BFST.*

Proof. Let $e_i \in E$ be the first edge satisfying $W(T) \leq k$. Thus, by Lemma 4, the bottleneck of any k-BFST in G is at least $|e_i|$. Therefore, the constructed k-BFST has a bottleneck at most 9 times the bottleneck of an optimal k-BFST. □

Lemma 6 *Our algorithm runs in polynomial time.*

Proof. Notice that, for each edge $e_i \in E$, the third power graph G_i^2 is of size $O((n+m)^2)$. Thus, G_i^2 can be computed from G_i in $O((n + m)^2)$ time, and computing a maximal independent set Q' in $G_i^2(Q)$ also takes $O((n + m)^2)$ time. The construction of G' on Q' can be done in $O((n + m)^3)$ time, by computing the shortest Steiner paths between each pair of points in Q' [13]. Computing a minimum spanning tree of G' can be done in $O(n^2)$ time. Procedure 1 runs in $O(k(n + m))$ time. the construction of the obtained full Steiner tree can be done in $O((n + k) \log k)$. Therefore, the algorithm runs in polynomial time. □

The following theorem summarizes the result of this section.

Theorem 7 *The above algorithm computes a k-BFST with bottleneck at most 9 times the bottleneck of an optimal k-BFST in polynomial time.*

References

[1] A. K. Abu-Affash. The Euclidean bottleneck full steiner tree problem. *Algorithmica*, 71(1):139–151, 2015.

[2] P. K. Agarwal, H. Edelsbrunner, and O. Schwarzkopf. Euclidean minimum spanning trees and bichromatic closest pairs. *Disc. & Comput. Geom.*, 6:407–422, 1991.

[3] P. K. Agarwal and M. Sharir. Red-blue intersection detection algorithms, with applications to motion planning and collision detection. *SIAM J. Comput.*, 19(2):297–321, 1990.

[4] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998.

[5] S. Arora and K. L. Chang. Approximation schemes for degree-restricted MST and red-blue separation problem. In *Proceedings of ICALP*, pages 176–188, 2003.

[6] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.

[7] M. J. Atallah and D. Z. Chen. On connecting red and blue rectilinear polygonal obstacles with nonintersecting monotone rectilinear paths. *Int. J. Comput. Geom. Appl.*, 11(4):373–400, 2001.

[8] P. Berman and V. Ramaiyer. Improved approximations for the steiner tree problem. In *Proceedings of SODA*, pages 325–334, 1992.

[9] A. Biniaz, P. Bose, D. Eppstein, A. Maheshwari, P. Morin, and M. Smid. Spanning Trees in Multipartite Geometric Graphs. *CoRR*, abs/1611.01661, 2016.

[10] A. Biniaz, A. Maheshwari, and M. Smid. An optimal algorithm for the Euclidean bottleneck full steiner tree problem. *Comput. Geom.*, 47(3):377–380, 2014.

[11] A. Biniaz, A. Maheshwari, and M. Smid. Bottleneck bichromatic plane matching of points. In *Proceedings of CCCG*, 2014.

[12] J.-D. Boissonnat, J. Czyzowicz, O. Devillers, J. Urrutia, and M. Yvinec. Computing largest circles separating two sets of segments. *Int. J. Comput. Geom. Appl.*, 10(1):41–53, 2000.

[13] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms, 3rd edition*. The MIT Press, 2009.

[14] E. D. Demaine, J. Erickson, F. Hurtado, J. Iacono, S. Langerman, H. Meijer, M. H. Overmars, and S. Whitesides. Separating point sets in polygonal environments. *Int. J. Comput. Geom. Appl.*, 15(4):403–420, 2005.

[15] H. Everett, J.-M. Robert, and M. J. van Kreveld. An optimal algorithm for the (<= k)-levels, with applications to separation and transversal problems. *Int. J. Comput. Geom. Appl.*, 6(3):247–261, 1996.

[16] M. R. Garey, R. L. Graham, and D. S. Johnson. The complexity of computing steiner minimal trees. *SIAM J. Appl. Math.*, 32(4):835–859, 1977.

[17] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM J. Appl. Math.*, 32(4):826–834, 1977.

[18] M. Karpinski and A. Zelikovsky. New approximation algorithms for the steiner tree problems. *J. Comb. Opt.*, 1(1):47–65, 1997.

[19] H. J. Prömel and A. Steger. A new approximation algorithm for the steiner tree problem with performance ratio 5/3. *Journal of Algorithms*, 36(1):89–101, 2000.

[20] W. Schnyder. Embedding planar graphs on the grid. In *Proceedings of SODA*, pages 138–148, 1990.

Exploring Increasing-Chord Paths and Trees

Yeganeh Bahoo* Stephane Durocher*§ Sahar Mehrpour† Debajyoti Mondal‡§

Abstract

A straight-line drawing Γ of a graph $G = (V, E)$ is a drawing of G in the Euclidean plane, where every vertex in G is mapped to a distinct point, and every edge in G is mapped to a straight line segment between their endpoints. A path P in Γ is called increasing-chord if for every four points (not necessarily vertices) a, b, c, d on P in this order, the Euclidean distance between b, c is at most the Euclidean distance between a, d. A spanning tree T rooted at some vertex r in Γ is called increasing-chord if T contains an increasing-chord path from r to every vertex in T. We prove that given a vertex r in a straight-line drawing Γ, it is NP-complete to decide whether Γ contains an increasing-chord spanning tree rooted at r, which answers a question posed by Mastakas and Symvonis [9]. We also shed light on the problem of finding an increasing-chord path between a pair of vertices in Γ, but the computational complexity question remains open.

1 Introduction

In 1995, Icking et al. [6] introduced the concept of a self-approaching curve. A curve is called *self-approaching* if for any three points a, b and c on the curve in this order, $|bc| \leq |ac|$, where $|xy|$ denotes the Euclidean distance between x and y. A path P in a straight-line drawing Γ is called increasing-chord if for every four points (not necessarily vertices) a, b, c, d on P in this order, the inequality $|bc| \leq |ad|$ holds. Γ is called an *increasing-chord drawing* if there exists an increasing-chord path between every pair of vertices in Γ.

Alamdari et al. [1] examined the problem of recognizing increasing-chord drawings, and the problem of constructing such a drawing on a given set of points. They showed that it is NP-hard to recognize increasing-chord drawings in $\mathbb{R}^3$, and asked whether it is also NP-hard in $\mathbb{R}^2$. They also proved that for every set of n points P in $\mathbb{R}^2$, one can construct an increasing-chord drawing Γ with $O(n)$ vertices and edges, where P is a subset of the vertices of Γ. In this case, Γ is called a *Steiner network of* P, and the vertices of Γ that do not belong to P are called Steiner points. Dehkordi et al. [4] proved that if P is a convex point set, then one can construct an increasing-chord network with $O(n \log n)$ edges, and without introducing any Steiner point. Mastakas and Symvonis [8] improved the $O(n \log n)$ upper bound on edges to $O(n)$ with at most one Steiner point. Nöllenburg et al. [11] examined the problem of computing increasing-chord drawings of given graphs. Recently, Bonichon et al. [3] showed that the existence of an angle-monotone path of width $0 \leq \gamma < 180°$ between a pair of vertices (in a straight-line drawing) can be decided in polynomial time, which is very interesting since angle-monotone paths of width $\gamma \leq 90°$ satisfy increasing chord property.

Nöllenburg et al. [10] showed that partitioning a plane graph drawing into a minimum number of increasing-chord components is NP-hard, which extends a result of Tan and Kermarrec [12]. They also proved that the problem remains NP-hard for trees, and gave polynomial-time algorithms in some restricted settings. Recently, Mastakas and Symvonis [9] showed that given a point set S and a point $v \in S$, one can compute a rooted minimum-cost spanning tree in polynomial time, where each point in $S \setminus \{v\}$ is connected to v by a path that satisfies some monotonicity property. They also proved that the existence of a monotone rooted spanning tree in a given geometric graph can be decided in polynomial time, and asked whether the decision problem remains NP-hard also for increasing-chord or self-approaching properties.

2 Technical Background

Given a straight line segment l, the *slab of* l is an infinite region lying between a pair of parallel straight lines that are perpendicular to l, and pass through the endpoints of l. Let Γ be a straight-line drawing, and let P be a path in Γ. Then the *slabs* of P are the slabs of the line segments of P. We denote by $\Psi(P)$ the arrangement of the slabs of P. Figure 1(a) illustrates a path P, where the slabs of P are shown in shaded regions. Let A be an arrangement of a set of straight lines such that no line in A is vertical. Then the *upper envelope* of A is a polygonal chain $U(A)$ such that each point of $U(A)$ belongs to some straight line of A, and they are visible from the point $(0, +\infty)$. The upper envelope of a set

*Department of Computer Science, University of Manitoba, Winnipeg, Canada, {bahoo,durocher}@cs.umanitoba.ca

†School of Computing, University of Utah, Utah (UT), USA, mehrpour@cs.utah.edu

‡Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada, dmondal@uwaterloo.ca

§Work of the author is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

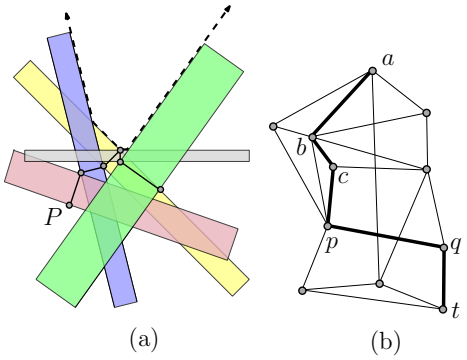


Figure 1: (a) Illustration for $\Psi(P)$, where the upper envelope is shown in dashed line. (b) An increasing-chord extension of $a, b, \ldots, p$ is shown in bold.

of slabs is the upper envelope of the arrangement of lines corresponding to the slab boundaries, as shown in dashed line in Figure 1(a). Let t be a vertex in Γ and let $Q = (a, b, \ldots, p)$ be an increasing-chord path in Γ. A path $Q' = (a, b, \ldots, p, \ldots, t)$ in Γ is called an *increasing-chord extension of* Q if Q' is also an increasing-chord path, e.g., see Figure 1(b).

Observation 1 (Icking et al. [7]) *A polygonal path P is increasing-chord if and only if for each point v on the path, the line perpendicular to P at v does not properly intersect P except possibly at v.*

A straightforward consequence of Observation 1 is that every polygonal chain which is both x- and y-monotone, is an increasing-chord path. We will use Observation 1 throughout the paper to verify whether a path is increasing-chord. Let v be a point in $\mathbb{R}^2$. By the *quadrants of* v we refer to the four regions determined by the vertical and horizontal lines through v.

3 Increasing-Chord Rooted Spanning Trees

In this section we prove the problem (IC-Tree) of computing a rooted increasing-chord spanning tree of a given straight-line drawing to be NP-hard.

Theorem 1 *Given a vertex r in a straight-line drawing Γ, it is NP-complete to decide whether Γ admits an increasing-chord spanning tree rooted at r.*

We reduce the NP-complete problem 3-SAT [5] to IC-Tree. Let $I = (X, C)$ be an instance of 3-SAT, where X and C are the set of variables and clauses. We construct a straight-line drawing Γ and choose a vertex r in Γ such that Γ contains an increasing-chord spanning tree rooted at r if and only if I admits a satisfying truth assignment. Here we give an outline of the hardness proof and describe the construction of Γ. A detailed reduction is given in the full version [2].

Assume that $\alpha = |X|$, and $\beta = |C|$. Let l_h be the line determined by the X-axis. Γ will contain $O(\beta)$ points above l_h, one point t on l_h, and $O(\alpha)$ points below l_h, as shown in Figures 2(a)–(b). Each clause $c \in C$ with j literals, will correspond to a set of $j+1$ points above l_h, and we will refer to the point with the highest y-coordinate among these $j+1$ points as the *peak* t_c of c. Among the points below l_h, there are 4α points that correspond to the variables and their negations, and two other points, i.e., s and r. In the reduction, the point t and the points below l_h altogether help to set the truth assignments of the variables.

We will first create a straight-line drawing H such that every increasing-chord path between r and t_c, where $c \in C$, passes through s and t. Consequently, any increasing-chord tree T rooted at r (not necessarily spanning), which spans the points t_c, must contain an increasing-chord path $P = (r, s, \ldots, t)$. We will use this path to set the truth values of the variables.

The edges of H below l_h will create a set of thin slabs, and the upper envelope of these slabs will determine a convex chain W above l_h. Each line segment on W will correspond to a distinct variable, as shown in Figure 2(b). The points that correspond to the clauses will be positioned below these segments, and hence some of these points will be 'inaccessible' depending on the choice of the path P. These literal-points will ensure that for any clause $c \in C$, there exists an increasing-chord extension of P from t to t_c if and only if c is satisfied by the truth assignment determined by P.

By the above discussion, I admits a satisfying truth assignment if and only if there exists an increasing-chord tree T in H that connects the peaks to r. But H may still contain some vertices that do not belong to this tree. Therefore, we construct the final drawing Γ by adding some new paths to H, which will allow us to reach these remaining vertices from r. We now describe the construction in details.

Construction of H: We first construct an arrangement $\mathcal{A}$ of 2α straight line segments. The endpoints of the ith line segment L_i, where $1 \le i \le 2\alpha$, are $(0, i)$ and $(2\alpha - i + 1, 0)$. We now extend each L_i downward by scaling its length by a factor of $(2\alpha+1)$, as shown in Figure 3(a). Later, the variable x_j, where $1 \le j \le \alpha$, and its negation will be represented using the lines L_{2j-1} and L_{2j}. Let l_v be a vertical line segment with endpoints $(2\alpha+1, 2\alpha)$ and $(2\alpha+1, -5\alpha^2)$. Since the slope of a line in $\mathcal{A}$ is in the interval $[-2\alpha, -1/(2\alpha)]$, each L_i intersects l_v. Since the coordinates of the endpoints of L_i and l_v are of size $O(\alpha^2)$, and all the intersection points can be represented using polynomial space.

By construction, the line segments of $\mathcal{A}$ appear on $U(\mathcal{A})$ in the order of the variables, i.e., the first two segments (from right) of $U(\mathcal{A})$ correspond to x_1 and $\overline{x_1}$, the next two segments correspond to x_2 and $\overline{x_2}$, etc.

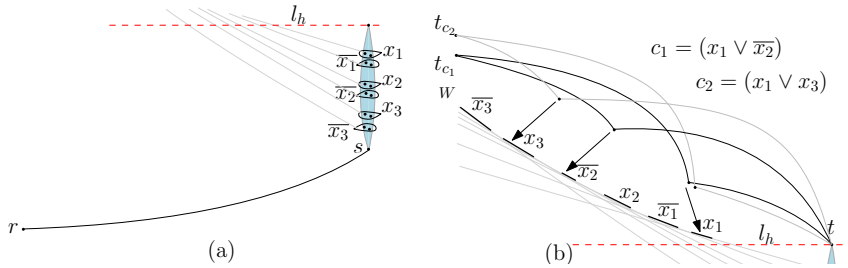


Figure 2: A schematic representation of Γ: (a) Points below l_h, (b) Points above l_h. The points that correspond to c_1 and c_2 are connected in paths of black, and gray, respectively. The slabs of the edges of H that determine the upper envelope are shown in gray straight lines. Each variable and its negation correspond to a pair of adjacent line segments on the upper envelope of the slabs. See the full version [2] for a better illustration.

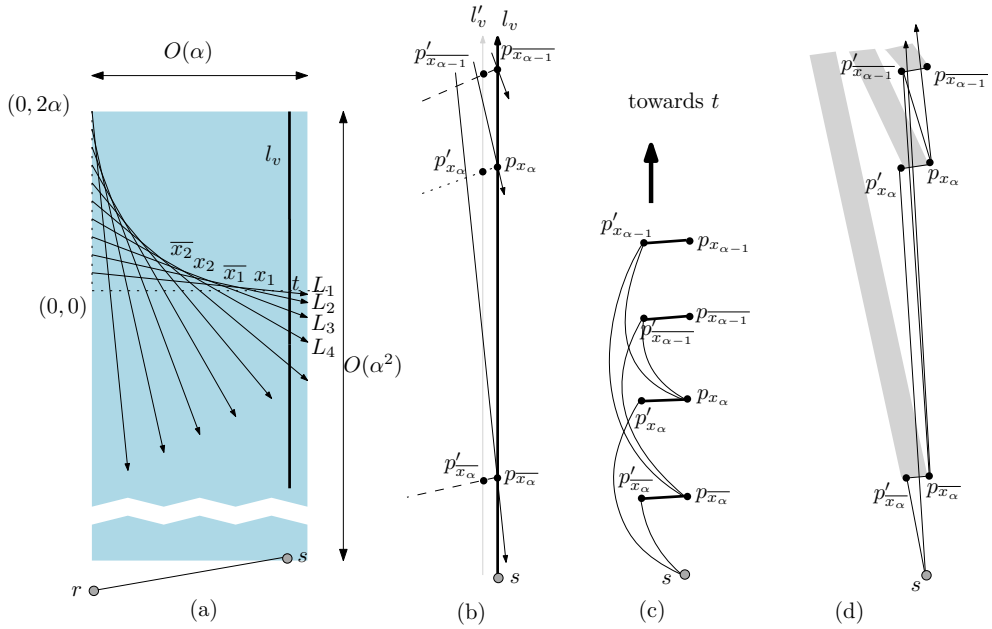


Figure 3: (a) Construction of $\mathcal{A}$. (b)–(c) Construction of the vertices and edges of H_b. (d) Illustration for the straight line segments of H_b, and the slabs corresponding to the needles.

Variable Gadgets: We denote the intersection point of l_h and l_v by t, and the endpoint $(2\alpha+1, -5\alpha^2)$ of l_v by s. We now create the points that correspond to the variables and their negations. Recall that L_{2j-1} and L_{2j} correspond to the variable x_j and its negation $\overline{x_j}$, respectively. Denote the intersection point of L_{2j-1} and l_v by p_{x_j}, and the intersection point of L_{2j} and l_v by $p_{\overline{x_j}}$, e.g., see Figure 3(b). For each p_{x_j} ($p_{\overline{x_j}}$), we create a new point p'_{x_j} ($p'_{\overline{x_j}}$) such that the straight line segment $p_{x_j}p'_{x_j}$ ($p_{\overline{x_j}}p'_{\overline{x_j}}$) is perpendicular to L_{2j-1} (L_{2j}), as shown using the dotted (dashed) line in Figure 3(b). We may assume that all the points p'_{x_j} and $p'_{\overline{x_j}}$ lie on a vertical line l'_v, where l'_v lies ε distance away to the left of l_v. The value of ε would be determined later. In the following we use the points p_{x_j}, $p_{\overline{x_j}}$, p'_{x_j} and $p'_{\overline{x_j}}$ to create some polygonal paths from s to t.

For each j from 1 to α, we draw the straight line segments $p_{x_j}p'_{x_j}$ and $p_{\overline{x_j}}p'_{\overline{x_j}}$. Then for each k, where $1 < k \leq \alpha$, we make p_{x_k} and $p_{\overline{x_k}}$ adjacent to both $p'_{x_{k-1}}$ and $p'_{\overline{x_{k-1}}}$, e.g., see Figure 3(c). We then add the edges from s to p'_{x_α} and $p'_{\overline{x_\alpha}}$, and finally, from t to p_{x_1} and $p_{\overline{x_1}}$. For each x_j ($\overline{x_j}$), we refer to the segment $p_{x_j}p'_{x_j}$ ($p_{\overline{x_j}}p'_{\overline{x_j}}$) as the *needle* of x_j ($\overline{x_j}$). Figure 3(c) illustrates the needles in bold. Let the resulting drawing be H_b.

Recall that l'_v is ε distance away to the left of l_v. We choose ε sufficiently small such that for each needle, its slab does not intersect any other needle in H_b, e.g., see Figure 3(d). The upper envelope of the slabs of all the straight line segments of H_b coincides with $U(\mathcal{A})$. Since the distance between any pair of points that we created on l_v is at least $1/\alpha$ units, it suffices to choose $\varepsilon = 1/\alpha^3$. Note that the points p'_{x_j} and $p'_{\overline{x_j}}$ can be represented in polynomial space using the endpoints of

l'_v and the endpoints of the segments L_{2j-1} and L_{2j}. The proof of the following lemma is omitted (see [2]).

Lemma 2 *Every increasing-chord path P that starts at s and ends at t must pass through exactly one point among p_{x_j} and $p_{\overline{x_j}}$, where $1 \leq j \leq \alpha$, and vice versa.*

We now place a point r on the y-axis sufficiently below H_b, e.g., at position $(0, -\alpha^5)$, such that the slab of the straight line segment rs does not intersect H_b (except at s), and similarly, the slabs of the line segments of H_b do not intersect rs. Furthermore, the slab of rs does not intersect any segment L_j, and vice versa. We then add the point r and the segment rs to H_b. Let P be an increasing-chord path from r to t. The upper envelope of $\Psi(P)$ is determined by the needles in P, which selects some segments from the convex chain W, e.g., see Figure 2(b). For each x_j, P passes through exactly one point among p_{x_j} and $p_{\overline{x_j}}$. Therefore, for each variable x_j, either the slab of x_j, or the slab of $\overline{x_j}$ appears on $U(P)$. Later, if P passes through point p_{x_j} ($p_{\overline{x_j}}$), then we will set x_j to false (true). Since P is an increasing-chord path, by Lemma 2 it cannot pass through both p_{x_j} and $p_{\overline{x_j}}$ simultaneously. Therefore, all the truth values will be set consistently.

Clause Gadgets: We now complete the construction of H by adding clause gadgets to H_b. For each clause c_i, where $1 \leq i \leq \beta$, we first create the peak point t_{c_i} at position $(0, 2\alpha + i)$. For each variable x_j, let λ_{x_j} be the interval of L_{2j-1} that appears on the upper envelope of $\mathcal{A}$. Similarly, let $\lambda_{\overline{x_j}}$ be the interval of L_{2j} on the upper envelope of $\mathcal{A}$. For each c_i, we construct a point q_{x_j,c_i} ($q_{\overline{x_j},c_i}$) inside the cell of $\mathcal{A}$ immediately below λ_{x_j} ($\lambda_{\overline{x_j}}$). We will refer to these points as the *literal-points of c_i*. The full version [2] depicts these points in black squares. We assume that for each variable, the corresponding literal-points lie on the same location. One may perturb them to remove vertex overlaps. For each variable $x \in c_i$, we create a path (t, x, t_c). In the reduction, if at least one of the literals of c_i is true, then we can take the corresponding path to connect t_c to t. Let the resulting drawing be H.

Construction of Γ: Let q be a literal-point in H. We now add an increasing-chord path $P' = (r, a, q)$ to H in such a way that P' cannot be extended to any larger increasing-chord path in H. We place the point a at the intersection point of the horizontal line through q and the vertical line through r, the full version [2] contains the details. We refer to the point a as the *anchor* of q. By the construction of H, all the neighbors of q that have a higher y-coordinate than q lie in the top-left quadrant of q. Let q' be the first neighbor in the top-left quadrant of q in counter clockwise order. Since $\angle aqq' < 90°$, P' cannot be extended to any larger increasing-chord path (r, a, q, w) in H, where the y-coordinate of w is higher than q. On the other hand, every literal-point w in H with y-coordinate smaller than q intersects the slab of ra. Therefore, P' cannot be extended to any larger increasing-chord path.

For every literal-point q in H, we add such an increasing-chord path from t to q. To avoid edge overlaps, one can perturb the anchors such that the new paths remain increasing-chord and non-extensible to any larger increasing-chord paths. This completes the construction of Γ. We refer the reader to the full version [2] for the formal details of the reduction.

4 Increasing-Chord Paths

In this section we attempt to reduce 3-SAT to the problem of finding an increasing-chord path (IC-Path) between a pair of vertices in a given straight-line drawing. We were unable to bound the coordinates of the drawing to a polynomial number of bits, and hence the computational complexity question of the problem remains open. We hope that the ideas we present here will be useful in future endeavors to settle the question.

Here we briefly describe the idea of the reduction. Given a 3-SAT instance $I = (X, C)$, the corresponding drawing $\mathcal{D}$ for IC-Path consists of straight-line drawings $\mathcal{D}_{i-1}$, where $1 \leq i \leq \beta$, e.g., see Figure 4(a). The drawing $\mathcal{D}_{i-1}$ corresponds to the each clause c_i. We will refer to the bottommost (topmost) point of $\mathcal{D}_{i-1}$ as $t_{c_{i-1}}$ (t_{c_i}). We will choose t_{c_0} and t_{c_β} to be the points t and t', respectively, and show that I admits a satisfying truth assignment if and only if there exists an increasing-chord path P from t to t' that passes through every t_{c_i}. For every i, the subpath P_{i-1} of P between $t_{c_{i-1}}$ and t_{c_i} will correspond to a set of truth values for all the variables in X. The most involved part is to show that the truth values determined by P_{i-1} and P_i are consistent. This consistency will be ensured by the construction of $\mathcal{D}$, i.e., the increasing-chord path P_{i-1} from $t_{c_{i-1}}$ to t_{c_i} in $\mathcal{D}_{i-1}$ will determine a set of slabs, which will force a unique increasing-chord path P_i in $\mathcal{D}_i$ between t_{c_i} and $t_{c_{i+1}}$ with the same truth values as determined by P_{i-1}.

Construction of $\mathcal{D}$: The construction of $\mathcal{D}_{i-1}$ depends on an arrangement of lines $\mathcal{A}^{i-1}$. The construction of $\mathcal{A}^0$ is the same as the construction of arrangement $\mathcal{A}$, which we described in Section 3. Figure 4(c) illustrates $\mathcal{A}^0$ in dotted lines. For each variable x_j, where $1 \leq j \leq \alpha$, there exists an interval $\lambda^0_{x_j}$ of L_{2j-1} on the upper envelope of $\mathcal{A}^0$. Similarly, for each $\overline{x_j}$, there exists an interval $\lambda^0_{\overline{x_j}}$ of L_{2j} on the upper envelope of $\mathcal{A}^0$.

We now describe the construction of $\mathcal{D}_0$. Choose t_{c_0} (t_{c_1}) to be the bottommost (topmost) point of $\lambda^0_{x_1}$ ($\lambda^0_{\overline{x_\alpha}}$). We then slightly shrink the intervals $\lambda^0_{x_1}$ and $\lambda^0_{\overline{x_\alpha}}$ such that t_{c_0} and t_{c_1} no longer belong to these segments. Assume that c_1 contains δ literals, where $\delta \leq 3$, and let $\sigma_1, \ldots, \sigma_{2^\delta - 1}$ be the satisfying truth assignments for c_1. We construct a graph G_{c_1} that corresponds to these sat-

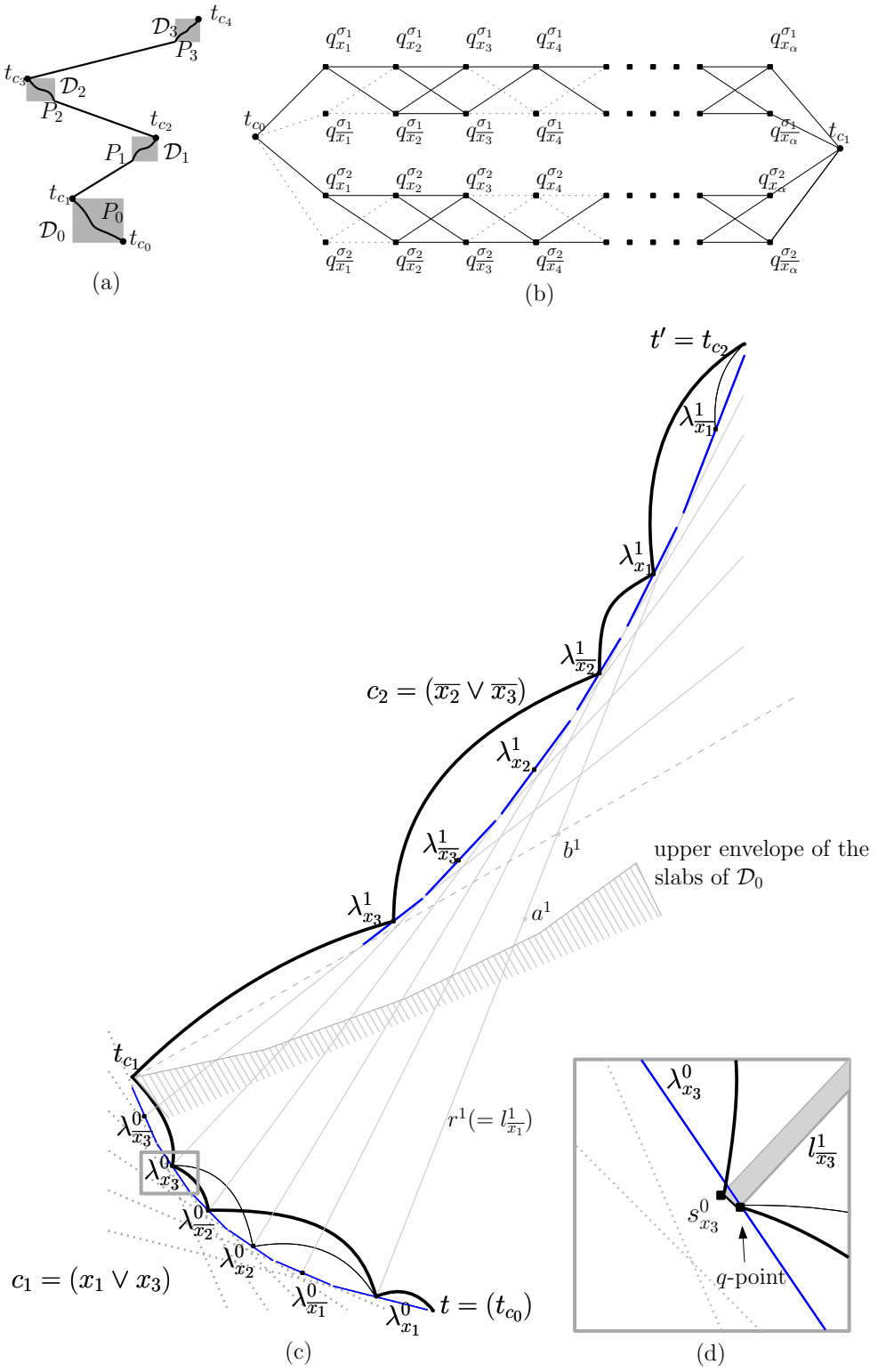


Figure 4: (a) Idea for the reduction. (b) The graph corresponding to the truth assignment satisfying $c_1 = (x_1 \vee x_4)$. Only the construction for the truth assignments $\sigma_1 = \{x_1 = true, x_4 = true\}$ and $\sigma_2 = \{x_1 = true, x_4 = false\}$ are shown. (c) A schematic representation for $\mathcal{D}$, where $I = (x_1 \vee x_3) \wedge (\overline{x_2} \vee \overline{x_3})$. An increasing-chord path is shown in bold, and the corresponding truth value assignment is: $x_1 = true$, $x_2 = false$, $x_3 = true$. (d) An s-segment.

isfying truth assignments, e.g., see Figure 4(b) and the full version [2] for formal details. The idea is to ensure that any path between t_{c_0} and t_{c_1} passes through exactly one point in $\{q_{x_j}^{\sigma_k}, q_{\overline{x_j}}^{\sigma_k}\}$, for each truth assignment σ_k, which will set the truth value of x_j. In $\mathcal{D}_0$, the point $q_{x_j}^{\sigma_k}$ ($q_{\overline{x_j}}^{\sigma_k}$) is chosen to be the midpoint of $\lambda_{x_j}^{i-1}$ ($\lambda_{\overline{x_j}}^{i-1}$). Later, we will refer to these points as *q-points*, e.g., see Figure 4(c). We may assume that for each x_j, the points $q_{x_j}^{\sigma_k}$ lie at the same location. One may later perturb them to remove vertex overlaps.

By Observation 1, any y-monotone path P' between t_{c_0} and t_{c_1} must be an increasing-chord path. If P' passes through $q_{x_j}^{\sigma}$, then we set x_j to true. Otherwise, P' must pass through $q_{\overline{x_j}}^{\sigma}$, and we set x_j to false. In the following we replace each q-point by a small segment. The slabs of these segments will determine $\mathcal{A}^1$. Consider an upward ray r^1 with positive slope starting at the q-point on λ_{x_1}, e.g., see Figure 4(c). Since all the edges that are currently in $\mathcal{D}_0$ have negative slopes, we can choose a sufficiently large positive slope for r^1 and a point a^1 on r^1 such that all the slabs of $\mathcal{D}_0$ lie below a^1. We now find a point b^1 above a^1 on r^1 with sufficiently large y-coordinate such that the slab of $t_{c_1}b^1$ does not intersect the edges in $\mathcal{D}_0$. Let $l_{\overline{x_1}}^1$ be the line determined by r^1. For each x_j and $\overline{x_j}$ (except for $j = 1$), we now construct the lines $l_{\overline{x_j}}^1$ and $l_{x_j}^1$ that pass through their corresponding q-points and intersect r^1 above b^1. The lines $l_{x_j}^1$ and $l_{\overline{x_j}}^1$ determine the arrangement $\mathcal{A}^1$. Observe that one can construct these lines in the decreasing order of the x-coordinates of their q-points, and ensure that for each $l_{x_j}^1$ ($l_{\overline{x_j}}^1$), there exists an interval $\lambda_{x_j}^1$ ($\lambda_{\overline{x_j}}^1$) on the upper envelope of $\mathcal{A}^1$. Note that the correspondence is inverted, i.e., in $\mathcal{A}^1$, $\lambda_{\overline{x_j}}^1$ corresponds to $\lambda_{x_j}^0$, and $\lambda_{x_j}^1$ corresponds to $\lambda_{\overline{x_j}}^0$.

For each j, we draw a small segment $s_{x_j}^0$ ($s_{\overline{x_j}}^0$) perpendicular to $l_{x_j}^1$ ($l_{\overline{x_j}}^1$) that passes through the q-point and lies to the left of q, e.g., see Figure 4(d). The construction of $\mathcal{D}_i$, where $i > 1$, is more involved. The upper envelope of $\mathcal{A}^{i+1}$ is determined by the upper envelope of the slabs of the s-segments in $\mathcal{D}_{i-1}$. For each i, we construct the q-points and corresponding graph G_{c_i}. The full version [2] includes the formal details.

In the reduction we show that any increasing-chord path P from t to t' contains the points t_{c_i}. We set a variable x_j true or false depending on whether P passes through $s_{x_j}^0$ or $s_{\overline{x_j}}^0$. The construction of $\mathcal{D}$ imposes the constraint that if P passes through $s_{x_j}^{i-1}$ ($s_{\overline{x_j}}^{i-1}$), then it must pass through $s_{x_j}^i$ ($s_{\overline{x_j}}^i$). Hence the truth values in all the clauses are set consistently. By construction of G_{c_i}, any increasing-chord path between $t_{c_{i-1}}$ to t_{c_i} determines a satisfying truth assignment for c_i. On the other hand, if I admits a satisfying truth assignment, then for each clause c_i, we choose the corresponding increasing-chord path P_i between $t_{c_{i-1}}$ and t_{c_i}. The union of all P_i yields the required increasing-chord path P from t to t'. The full version [2] presents the construction in details, and explains the challenges of encoding $\mathcal{D}$ in a polynomial number of bits.

5 Open Problems

The most intriguing problem in this context is to settle the computational complexity of the increasing-chord path (IC-Path) problem. Another interesting question is whether the problem IC-Tree remains NP-hard under the planarity constraint; a potential attempt to adapt our hardness reduction could be replacing the edge intersections by dummy vertices.

References

[1] S. Alamdari, T. M. Chan, E. Grant, A. Lubiw, and V. Pathak. Self-approaching graphs. In *Proc. of GD*, volume 7704 of *LNCS*, pages 260–271. Springer, 2013.

[2] Y. Bahoo, S. Durocher, S. Mehrpour, and D. Mondal. Exploring increasing-chord paths and trees. *CoRR*, abs/1702.08380, 2017.

[3] N. Bonichon, P. Bose, P. Carmi, I. Kostitsyna, A. Lubiw, and S. Verdonschot. Gabriel triangulations and angle-monotone graphs: Local routing and recognition. In *Proc. of GD*, volume 9801 of *LNCS*, pages 519–531. Springer, 2016.

[4] H. R. Dehkordi, F. Frati, and J. Gudmundsson. Increasing-chord graphs on point sets. *Journal of Graph Algorithms and Applications*, 19(2):761–778, 2015.

[5] M. R. Garey and D. S. Johnson. *Computers and intractability.* Freeman, San Francisco, 1979.

[6] C. Icking and R. Klein. Searching for the kernel of a polygon - A competitive strategy. In *Proc. of SoCG*, pages 258–266. ACM, 1995.

[7] C. Icking, R. Klein, and E. Langetepe. Self-approaching curves. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 125, pages 441–453. Cambridge Univ Press, 1999.

[8] K. Mastakas and A. Symvonis. On the construction of increasing-chord graphs on convex point sets. In *Proc. of IISA*, pages 1–6. IEEE, 2015.

[9] K. Mastakas and A. Symvonis. Rooted uniform monotone minimum spanning trees. In *Proc. of CIAC*, volume 10236 of *LNCS*, pages 405–417. Springer, 2017.

[10] M. Nöllenburg, R. Prutkin, and I. Rutter. Partitioning graph drawings and triangulated simple polygons into greedily routable regions. In *Proc. of ISAAC*, volume 9472 of *LNCS*, pages 637–649. Springer, 2015.

[11] M. Nöllenburg, R. Prutkin, and I. Rutter. On self-approaching and increasing-chord drawings of 3-connected planar graphs. *JoCG*, 7(1):47–69, 2016.

[12] G. Tan and A. Kermarrec. Greedy geographic routing in large-scale sensor networks: A minimum network decomposition approach. *IEEE/ACM Trans. Netw.*, 20(3):864–877, 2012.

Angle-monotone Paths in Non-obtuse Triangulations

Anna Lubiw* Joseph O'Rourke†

Abstract

We reprove a result of Dehkordi, Frati, and Gudmundsson: every two vertices in a non-obtuse triangulation of a point set are connected by an angle-monotone path—an xy-monotone path in an appropriately rotated coordinate system. We show that this result cannot be extended to angle-monotone spanning trees, but can be extended to boundary-rooted spanning forests. The latter leads to a conjectural edge-unfolding of sufficiently shallow polyhedral convex caps.

1 Introduction

The central result of this paper is to offer an alternative—and we believe simpler—proof of a result of Dehkordi, Frati, and Gudmundsson [DFG15] (henceforth, DFG):

> "**Lemma 4**. Let G be a Gabriel triangulation on a point set P. For every two points $s, t \in P$, there exists an angle θ such that G contains a θ-path from s to t."

We first explain this result, using notation from [BBC+16], before detailing our other contributions. First, we use S for the point set and β instead of θ. A Gabriel triangulation as defined by DFG is a triangulation of S where "every angle of a triangle delimiting an internal face is acute." Because neither they nor we need any of the various properties of Gabriel triangulations except the angle property, and we only need non-obtuse rather than strict acuteness, we define G to be a plane geometric graph that is a non-obtuse triangulation of S.

Define the *wedge* $W(\beta, v)$ to be the region of the plane bounded by rays at angles $\beta \pm 45°$ emanating from v. W is closed along (i.e., includes) both rays, and has angular *width* of 90°. (Later we generalize to widths different from 90°.) A polygonal path $(v_0, \ldots, v_k)$ consisting of edges of G is called *β-monotone* (for short, a *β-path*) if the vector of every edge (v_i, v_{i+1}) lies in $W(\beta, v_0)$. These are the θ-paths of DFG. Note that if $\beta = 45°$, then a β-monotone path is both x- and y-monotone with respect to a Cartesian coordinate system. A path that is β-monotone for some β is called *angle-monotone*.

Our phrasing of the DFG result is:

Theorem 1 *In a non-obtuse triangulation G, every pair of vertices is connected by an angle-monotone path.*

Other Contributions. We extend Theorem 1 to wedges of any width γ—if a plane geometric graph that includes the convex hull of S has all angles at most γ, then there is an angle-monotone path of width γ between any two vertices. Of necessity, $60° \leq \gamma < 180°$. One significance of angle-monotone paths of width $\gamma < 180°$ is that they have a spanning ratio of $1/\cos\frac{\gamma}{2}$ [BBC+16]. We do not pursue that aspect here. Instead, we investigate angle-monotone spanning trees. These were studied independently in [MS16], which addressed recognition and construction, but not existence—our focus. We show that Theorem 1 does not extend to angle-monotone spanning trees, but does extend to boundary-rooted spanning forests. Then, in Section 5 we make a novel connection to edge-unfolding polyhedra.

2 Proof

We prove Theorem 1 by showing that there is an angle-monotone path from an arbitrary fixed vertex s to every other vertex. The proof uses an angular sweep around s, which by convention we place at the origin. We first consider a fixed but arbitrary angle β and investigate which vertices are reached by β-paths from s. Let ∂G be the boundary of G, i.e., the convex hull of S. Our proof relies on two properties of vertices v not on ∂G: (1) the wedge $W(\beta, v)$ includes at least one edge incident to v; (2) if the wedge has only one edge incident to v then that edge does not lie along a bounding ray of the wedge. In order to avoid dealing with boundary vertices as a separate case, we will augment G so that conditions (1) and (2) hold for boundary vertices as well. At every vertex v on ∂G add a finite set of rays that subdivide the exterior angle at v into angles of at most 90°. Call the result G^+. By construction, properties (1) and (2) now hold for every vertex v if we consider both edges

*School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada. alubiw@uwaterloo.ca.

†Department of Computer Science, Smith College, Northampton, MA, USA. jorourke@smith.edu.

and rays incident to v. Note that no added ray crosses an edge of G. In the special case of widths $\geq 90°$, the rays can be chosen so that they do not intersect one another. When we generalize to smaller widths, the rays will necessarily intersect each other but this will not influence our proof. In our figures, the rays are drawn short as a reminder that only their angles at the convex hull are relevant.

A β-path starting at s is *maximal* if there is no edge or ray that can be added at the end of the path while keeping it a β-path. In particular, a path ending with a ray is maximal. DFG proved that every maximal β-path terminates on ∂G, which in our terms becomes:

Lemma 2 *Any maximal β-path ends with a ray.*

Proof. Consider a β-path that ends at a vertex v. The wedge $W(\beta, v)$ must include an edge or ray of G^+ by property (1) so the path can be extended. □

We will see later (in Fig. 3(b)) that it is possible for a β-path to include edges of ∂G, return to interior edges, and then later again include edges of ∂G. For a fixed β, let $P(\beta)$ be the set of all maximal β-paths starting from s. Let $V(\beta)$ and $E(\beta)$ be the set of vertices and edges/rays in $P(\beta)$.

Let $U(\beta) \in P(\beta)$ be the *upper envelope* of $P(\beta)$, defined as follows. Starting from vertex $v = s$, grow $U(\beta)$ by always following the most counterclockwise edge/ray from v falling within $W(\beta, v)$. $U(\beta)$ is necessarily a maximal β-path, so it ends with a ray. By property (2) above, we have:

Observation 1 *$U(\beta)$ does not include any edge/ray along the lower ray of $W(\beta, s)$, at angle $\beta - 45°$.*

$L(\beta)$ is similarly the *lower envelope*, the most clockwise path. Note that "upper" and "lower" are to be interpreted as most counterclockwise and most clockwise respectively, not in terms of y-coordinates.[1] Finally, define $R(\beta)$ to be the region of the plane whose boundary is $L(\beta)$, $U(\beta)$. Fig. 1 illustrates these definitions.

Lemma 3 *Every vertex in $R(\beta)$ is in $V(\beta)$, i.e., every vertex in $R(\beta)$ can be reached from s via a β-path.*

Proof. Wlog assume $\beta = 45°$, so that the wedge rays are at 0 and 90°. Let $v \in V(\beta)$ be the leftmost *inaccessible* vertex, i.e., the leftmost vertex not reached by a β-path from s. If there are ties for leftmost, let v be the lowest. Consider the backward wedge $\overline{W}(\beta, v)$ at v. Note that s lies in $\overline{W}(\beta, v)$. Consider the line segment sv. It lies in $\overline{W}(\beta, v)$ and inside the convex hull of S. Imagine rotating sv clockwise or counterclockwise about

[1] These notions are analogous but not equivalent to DFG's "high" and "low" paths.

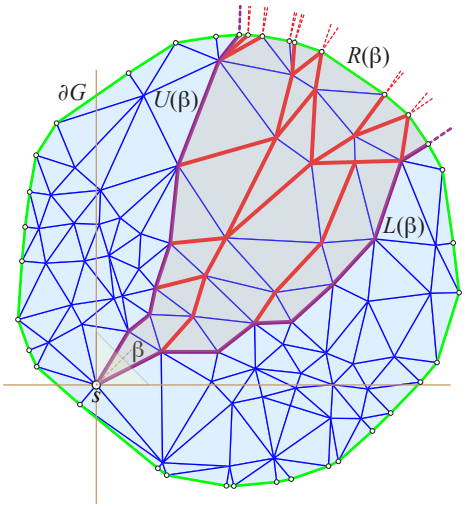


Figure 1: $P(\beta)$ edges are marked; $\beta = 45°$. $L(\beta)$ and $U(\beta)$ delimit the region $R(\beta)$. Rays shown only for $V(\beta)$ hull vertices.

v while remaining inside $\overline{W}(\beta, v)$ and inside the convex hull of S in a small neighborhood of v. Since there are no obtuse angles at v, rotating in one direction or the other must result in an edge or ray in $\overline{W}(\beta, v)$. Furthermore, the result cannot be a ray since we never leave the convex hull. Thus we have identified an edge $e = (u, v)$ in $\overline{W}(\beta, v)$.

Suppose first that u is in $R(\beta)$. Because v is the leftmost lowest inaccessible vertex, u must be accessible (note that u must be at the same y-height or lower than v). But now v lies in $W(\beta, u)$, and so v is accessible after all, a contradiction. Instead suppose u lies outside $R(\beta)$. Then e must cross the boundary of $R(\beta)$. But that boundary is composed of edges/rays of G^+, and e cannot cross an edge of G^+ without the two sharing a vertex, which would lie on the boundary of $R(\beta)$, not the exterior, again a contradiction. □

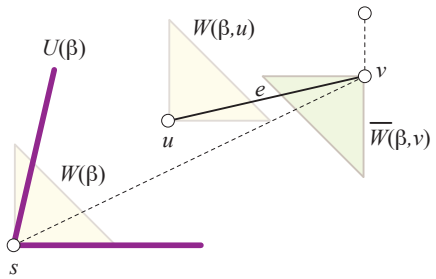


Figure 2: No vertex in $R(\beta)$ is inaccessible: all are reached by a β-path from s.

2.1 Critical angles β_i

We now analyze the relationships between $P(\beta_i)$ and $P(\beta_{i+1})$, where β_{i+1} is the next "relevant" angle after β_i, with the goal of showing that all vertices in G are "covered" over all β_i, i.e., belong to $P(\beta_i)$ for some β_i. Throughout, fix the source s, and let $W(\beta) = W(\beta, s)$. Define an angle β to be *critical* if $P(\beta + \varepsilon)$ or $P(\beta - \varepsilon)$ differs from $P(\beta)$, for an arbitrarily small $\varepsilon > 0$. At a critical angle β, one or both rays of $W(\beta)$ are parallel to one or more edges of $P(\beta)$ If $P(\beta + \varepsilon)$ differs from $P(\beta)$, one or more edges parallel to the $\beta - 45°$ ray drop out of $P(\beta)$. If $P(\beta - \varepsilon)$ differs from $P(\beta)$, one or more edges parallel to the $\beta + 45°$ ray enter $P(\beta)$. Fig. 3 illustrates two adjacent critical angles.

Let $\beta_1, \beta_2, \ldots, \beta_i, \beta_{i+1}, \ldots$ be the critical angles, sorted counterclockwise. For every β strictly between two adjacent critical angles, $\beta_i < \beta < \beta_{i+1}$, $P(\beta)$ is the same. We use the notation $P(\beta_{i'})$ to represent this intermediate set, which differs from $P(\beta_i)$ or $P(\beta_{i+1})$ or both.

In the transition from $P(\beta)$ to $P(\beta + \varepsilon)$, edges can drop out of $P(\beta)$. In particular, any edge $e = (u, v)$ that lies along the $\beta - 45°$-ray of $W(\beta, u)$ will drop out of $P(\beta)$. Furthermore, when edges drop out of $P(\beta)$ this may cause vertices to drop out of $P(\beta)$, and any edge originating at a dropped vertex also drops out of $P(\beta)$.

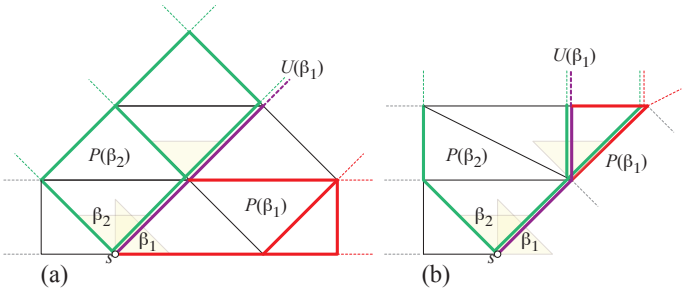


Figure 3: $P(\beta_1)$ (red), $U(\beta_1)$ (purple), and $P(\beta_2)$ (green). (a) $P(\beta_1)$ includes some but not all ∂G edges. (b) $U(\beta_1)$ includes an edge of ∂G before an internal edge.

The next lemma shows that no vertices fall strictly "between" $P(\beta_i)$ and $P(\beta_{i+1})$, where they would escape being spanned.

Lemma 4 *$U(\beta_i)$ is a path in $P(\beta_{i+1})$.*

Proof. Since edges may only enter, not leave, in the transition from $P(\beta_{i+1} - \varepsilon)$ to $P(\beta_{i+1})$, the lemma is equivalent to the claim that $U(\beta_i)$ is a path in $P(\beta_i + \varepsilon) = P(\beta_{i'})$. Let $\beta = \beta_i$, and $\beta' = \beta_{i'}$. Then we aim to prove that $U(\beta) \subseteq P(\beta')$. This requires showing that no edge $e \in U(\beta)$ drops out from $P(\beta)$ to $P(\beta')$, as β increases to β'. As usual, assume that $\beta = 45°$.

Suppose to the contrary that some edge drops out, and let $e = (u, v)$ be the leftmost lowest edge with $e \in U(\beta)$ but $e \notin P(\beta')$. Equivalently, e is the first edge of $U(\beta)$ that is not in $P(\beta')$. Because u is in $P(\beta')$, the only reason for e to drop out is that it lies along the lower, horizontal ray of the wedge $W(\beta, u)$. But by Observation (1), $U(\beta)$ does not include any edge along the lower ray of the wedge. □

In analogy with the definition of $R(\beta)$, define $R(\beta_i, \beta_j)$ for $j > i$ to be the region bound by $L(\beta_i)$, $U(\beta_j)$, and the portion of ∂G between those lower and upper envelope endpoints.

Lemma 5 *$R(\beta_i, \beta_{i+1}) = R(\beta_i) \cup R(\beta_{i+1})$. Informally, no vertices are "orphaned" between $P(\beta_i)$ and $P(\beta_{i+1})$.*

Proof. The lemma essentially says that no vertices are "orphaned" between $P(\beta_i)$ and $P(\beta_{i+1})$, and this follows immediately from the fact that $U(\beta_i) \subseteq P(\beta_{i+1})$ as established in Lemma 4. □

Now we can prove Theorem 1, the key result of [DFG15]:

Proof. [of Theorem 1] Fix s and construct $\bigcup_i P(\beta_i)$. By Lemma 5, this is a spanning graph of G, and so must include a path from s to v. □

Our arguments extend to wedges of any width γ, thus proving that if a plane geometric graph that includes the convex hull of S has all internal angles at most γ, then there is an angle-monotone path of width γ between any two vertices. This answers a question raised in [BBC+16].

3 Spanning Tree

Now that Theorem 1 has established that there is a graph spanning all of G with angle-monotone paths from any source s, it is natural to wonder if the claim can be strengthened to the existence of an *angle-monotone spanning tree* for any s: a tree rooted at s with an angle-monotone path from s to any $v \in G$. The answer is NO, but we canvass a few positive results before detailing a counterexample for spanning trees. Throughout, we let s be an arbitrary vertex of G. First, within a fixed β region, $P(\beta)$ can be easily spanned:

Lemma 6 *$P(\beta)$ includes an angle-monotone tree that spans the same vertices, $V(\beta)$.*

Proof. By Lemma 3, $P(\beta)$ reaches every vertex in $V(\beta)$. For each vertex $v \in V(\beta)$, in any order, delete all but one incoming edge to v. Because an incoming edge remains to each v, v is spanned. Because eventually no v has more than one incoming edge, no cycles can remain. See Fig. 4. □

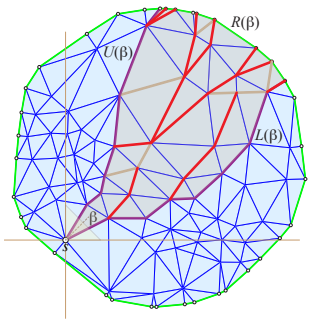


Figure 4: $P(\beta)$ spanning tree. Light-brown edges have been deleted.

We now consider triangulations with special angles.

Lemma 7 *Let G_{45° have edges only at multiples of 45°. Then there is an angle-monotone spanning tree rooted at any source vertex s.*

Proof. Let β_1 and $\beta_2 = \beta_1 + 45^\circ$ be two consecutive critical angles. We argue that $U(\beta_1)$ and $U(\beta_2)$ may share an initial portion of a path, but then diverge and do not rejoin before reaching their terminal rays.

By Observation (1), $U(\beta)$ only includes edges at angles β_i or $\beta_i + 45^\circ$. So the most counterclockwise edge in $U(\beta_1)$ is $\beta_1 + 45^\circ$, and the most clockwise edge in $U(\beta_2)$ is $\beta_2 = \beta_1 + 45^\circ$. Thus, $U(\beta_1)$ and $U(\beta_2)$ can have parallel edges, but once they separate, they can never rejoin.

Now it is easy to create a spanning tree between $U(\beta_1)$ and $U(\beta_2)$ that retains all edges in these two envelopes, by deleting all but one incoming edge to each vertex between the envelopes. □

The problematic possibility avoided in such G_{45° graphs is $U(\beta_1)$ and $U(\beta_2)$ joining, separating, and rejoining. Already in a graph G_{30° that has edges only at multiples of 30°, the divergence of upper envelopes used in Lemma 7 is no longer guaranteed, and thwarts that proof.

3.1 Spanning Tree Counterexample

Fig. 5 shows a graph G that does not have an angle-monotone spanning tree rooted at s. The construction allows two angle-monotone paths to each of $\{C, D, E, F\}$, one of which is marked green in the figure. But vertices A and B are shifted slightly toward one another, which breaks the symmetry and, as we shall argue below, results in a unique angle-monotone path to each. The union of those two unique paths contains the cycle (s, a, x, b). Thus there is no angle-monotone spanning tree from s.

We now argue that there is no angle-monotone path to A other than $saxA$. This is simply a matter of checking that any other path to A contains two *spread-apart* edges whose vectors do not lie in a 90° wedge. In particular, the path $sbxA$ contains spread-apart edges sb and xA, and the path $sawA$ contains spread-apart edges aw and wA. Other paths can be checked similarly. Similar reasoning constrains the (symmetric) paths to B.

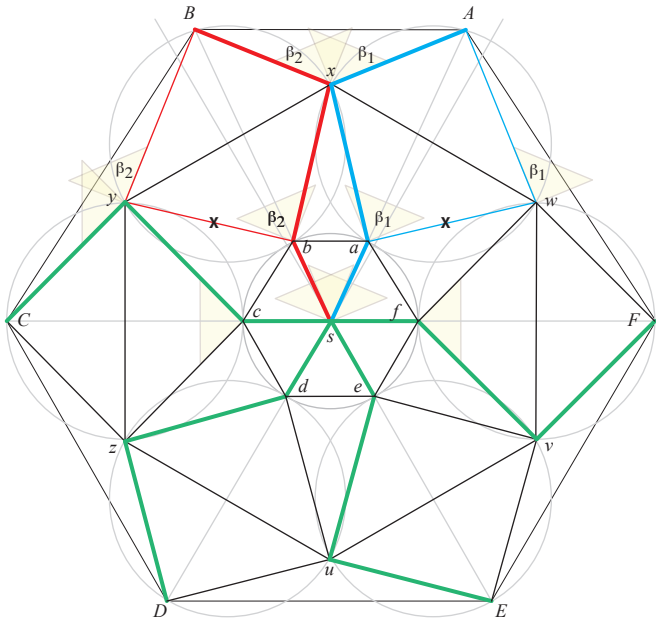


Figure 5: (s, a, x, A) is the unique angle-monotone path to A, and (s, b, x, B) is the unique angle-monotone path to B, forming the cycle (s, a, x, b).

The outer ring of six circles in the construction make clear that various diameter-spanning angles are 90°, but points $\{a, A, \ldots, f, F\}$ could be moved slightly exterior to those circles, rendering those angles $< 90^\circ$, while retaining the properties that force the (s, a, x, b) cycle. So the counterexample is "robust" in this sense.

4 Spanning Forest

For the unfolding application discussed in the next section, it is useful to span G by a boundary-rooted forest $\mathcal{F}$: A set of disjoint angle-monotone trees, each with its root on ∂G, and spanning every interior vertex of G. This can be achieved with β-monotone trees for just four β values.

With a Cartesian coordinate system centered on vertex s, define the quadrants Q_0, Q_1, Q_2, Q_3 as follows. Q_0 is the quadrant coincident with $W(\beta, s)$ when $\beta = 45^\circ$, closed along the x-axis and open along the y-axis, and includes the origin s. Q_i, $i > 0$ are defined analogously, except those quadrants do not include the origin. Thus

the quadrants are pairwise disjoint and together cover the plane.

We construct separate spanning forests for each quadrant, following Algorithm 1, which grows paths from vertices interior to G to ∂G. See Fig. 6.

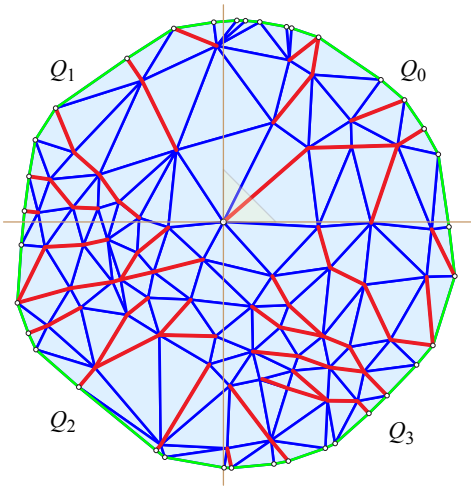


Figure 6: Spanning forest resulting from Algorithm 1.

Algorithm 1: Algorithm to construct spanning forest $\mathcal{F}$

Input : Non-obtuse triangulation G
Output: Spanning forest $\mathcal{F}$ composed of β-monotone paths

`// Quadrants` Q_j`, each corresponding to`
// $\beta_j = 45° + j \cdot 90°$, $j = 0, 1, 2, 3$.

foreach *Quadrant* Q_j, $j = 0, 1, 2, 3$ **do**
  $F_j \leftarrow \varnothing$
  `// Grow forest` F_j `inside` Q_j.
  foreach $v \in Q_j$ **do**
    if $v \notin F_j$ **then**
      Grow β_j-path p from v.
      Stop when p reaches a vertex in F_j, or reaches ∂G.
    end
    $F_j \leftarrow F_j \cup p$
  end
end
$\mathcal{F} = F_0 \cup F_1 \cup F_2 \cup F_3$
return $\mathcal{F}$.

Lemma 8 *Algorithm 1 outputs a boundary-anchored spanning forest, each tree of which is composed of β-monotone paths, for four β's: $\beta_j = 45° + j \cdot 90°$, $j = 0, 1, 2, 3$.*

Proof. Observe that a β_j-path grown from $v \in Q_j$ remains in Q_j. So all the trees in F_j are composed of β_j-paths. All vertices of each quadrant are spanned, because the inner loop of Algorithm 1 runs over all $v \in Q_j$. No cycles can be created because the algorithm only grows a path from v if v is not yet in F_j. So v becomes a leaf of some tree in F_j when its path reaches that tree. □

5 Unfolding

Now we discuss an application of Algorithm 1 and Lemma 8 to edge-unfolding nearly flat convex caps. We only sketch the argument, as several steps need considerable elaboration, and other steps rely on definitions and details in an unpublished report [O'R16]. So this section will end with a conjecture rather than a theorem. At a high level, the construction depends on two claims: (1) angle-monotone paths are "radially monotone paths," a concept introduced in [O'R16], but known before as backwards "self-approaching curves" [IKL99]. (2) Theorem 2 of [O'R16] concludes that the unfolding of a particular "medial" cut path M on a polyhedron is radially monotone and so does not self-cross when unfolded (if certain angle conditions are satisfied).

Let P be a convex polyhedron, and let $\phi(f)$ for a face f be the angle the normal to f makes with the z-axis. Let H be a halfspace whose bounding plane is orthogonal to the z-axis, and includes points vertically above that plane. Define a *convex cap* C of angle Φ to be $C = P \cap H$ for some P and H, such that $\phi(f) \leq \Phi$ for all f in C. We will only consider $\Phi < 90°$, which implies that the projection $C_\perp$ of C onto the xy-plane is one-to-one.

Say that a convex cap C is *acutely triangulated* if every angle of every face is strictly acute. Note that P being acutely triangulated does not always imply that $C = P \cap H$ is acutely triangulated, but it is known that any polyhedron can be acutely triangulated [Bis16]. We will need this lemma.

Lemma 9 *Let a triangle in $\mathbb{R}^3$, whose face normal makes angle ϕ with the z-axis, have one angle α, which projects to $\alpha_\perp$ on the xy-plane. Then the maximum value of $\Delta = |\alpha - \alpha_\perp|$ is a monotonically increasing function, as plotted in Fig. 7.*

We only need that $\Delta \to 0$ as $\phi \to 0$, so we will not calculate the function explicitly. For example, for $\phi < 10°$, $\Delta < 1°$.

For a triangulated convex cap C, let $\alpha_{\max}$ be the maximum of any triangle angle. Using Lemma 9, we can guarantee that an acutely triangulated cap C will project to a non-obtuse plane graph $C_\perp$ by choosing Φ so that $\Delta < 90° - \alpha_{\max}$.

Now we apply Algorithm 1 and Lemma 8 to obtain an angular-monotone spanning forest $\mathcal{F}_\perp$ of $C_\perp$. We then

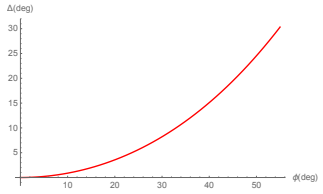


Figure 7: The maximum face angle change resulting from projection with normal at angle ϕ.

lift the trees in $\mathcal{F}_\perp$ to cut trees $\mathcal{F}$ on C in $\mathbb{R}^3$. Again Lemma 9 ensures this can be accomplished without any turn angle in any path in any tree of $\mathcal{F}$ exceeding 90°. Now finally we invoke a version of Theorem 2 as mentioned previously, which guarantees that the cut paths unfold without local overlap. We leave it a claim that the angle conditions for that theorem are satisfied by selecting Φ small enough. The conclusion is that the lifted paths are "radially monotone," which is the condition that implies unfolding without overlap. The end result is this:

Conjecture 1 *For an acutely triangulated convex cap C with sufficiently small Φ bounding face normals, the spanning forest $\mathcal{F}_\perp$ resulting from Algorithm 1 lifts to a cut forest $\mathcal{F}$ that edge-unfolds C without overlap.*

We have implemented this construction. Fig. 8 shows a convex cap with $\Phi \approx 27°$, and Fig. 9 shows the corresponding unfolding.[2]

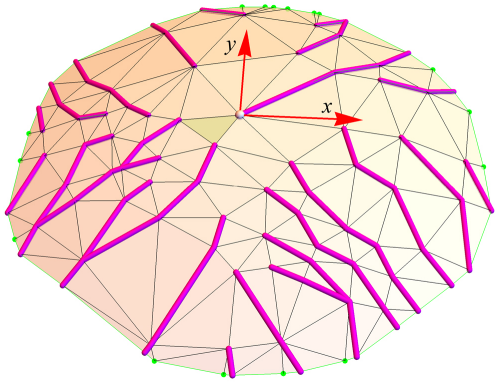


Figure 8: The cut forest $\mathcal{F}$ resulting from lifting $\mathcal{F}_\perp$ to the convex cap. (The marked face is the root of the dual unfolding tree.)

Acknowledgements. We thank Debajyoti Mondal for observing that our proof of Theorem 1 works for widths other than 90°.

[2]The forest in Fig. 8 is slightly different than that shown in Fig. 6, due to different ordering choices of $v \in Q_j$.

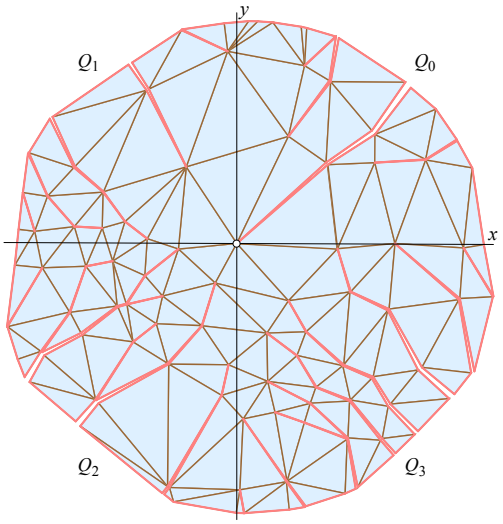


Figure 9: The edge-unfolded convex cap. The origin and quadrants used in Algorithm 1 are indicated.

References

[Bis16] C.J. Bishop. Nonobtuse triangulations of PSLGs. *Discrete & Computational Geometry*, 56(1):43–92, 2016.

[BBC+16] N. Bonichon, P. Bose, P. Carmi, I. Kostitsyna, A. Lubiw, and S. Verdonschot. Gabriel triangulations and angle-monotone graphs: Local routing and recognition. In: *24th Graph Drawing and Network Visualization* (GD). Lect. Notes Comput. Sci., 9801. Springer, 2016, pages 519–531.

[DFG15] H.R. Dehkordi, F. Frati, and J. Gudmundsson. Increasing-chord graphs on point sets. *J. Graph Algorithms Applications*, 19(2):761–778, 2015.

[IKL99] C. Icking, R. Klein, and E. Langetepe. Self-approaching curves. *Math. Proc. Camb. Phil. Soc.*, 125:441–453, 1999.

[MS16] K. Mastakas and A. Symvonis. Rooted uniform monotone minimum spanning trees. *Internat. Conf. Algorithms Complexity* (CIAC), Athens, Greece, 2017. Lect. Notes Comput. Sci. 10236, Springer.

[O'R16] J. O'Rourke. Unfolding convex polyhedra via radially monotone cut trees. arXiv:1607.07421 [cs.CG], 2016.

A General Algorithm for the Maximum Span of Fixed-Angle Chains

David Goering* Ronald Gentle†

Abstract

Fixed-angle chains have been used to model protein backbones [4] and robotic arm motions [5]. Benbernou and O'Rourke proved several structural theorems for finding the maximum 3D span of fixed-angle chains: the largest distance achievable between the two endpoints [1][2]. Borcea and Streinu used different methods to develop an algorithm which computes this span for any chain in polynomial time, and for chains with equal angles greater than or equal to $\pi/3$ in linear time [6]. We use Benbernou and O'Rourke's most general structural theorem to develop a new algorithm which also computes the maximum span for any fixed-angle chain and the configuration in which this is achieved. Our algorithm is purely geometric in nature, meaning that it consists of only straight-edge and compass constructions together with some list-keeping. The algorithm has complexity $O(n^2)$ for any chain with equal angles, known also as α-chains. We do not claim that it runs in polynomial time for all chains but discuss why it will do so for those likely to be used in any modeling application.

1 Introduction

Fixed-angle chains consist of serially connected line segments, each attached to its predecessor at an angle $0° < \alpha_i < 180°$ but capable of spinning at the joint while the angle between the two segments remains constant. Soss proved that finding the maximum span of flat configurations of the chain is NP-hard, but showed that the 3D maxspan is not always achieved by a flat configuration [3]. Benbernou and O'Rourke primarily focused on the maximum 3D span for restricted classes of chains. They conjectured that all α-chains are solvable in quadratic time and our results verify this (it is possible that Borcea and Streinu also show this for chains with $\alpha < \pi/3$ but we are not aware of this result). Our algorithm directly depends on their n-Chain Partition Theorem which we state after introducing notation, most of which is consistent with [2].

Let a chain C have vertices $(v_0, v_1, \ldots, v_n)$. The fixed joint angle is $\alpha_i = \angle v_{i-1}v_iv_{i+1}$. We denote link i (the line segment $v_{i-1}v_i$) as L_i. A *flat* configuration for C is one in which all vertices lie in the same plane. The *zigzag* or *trans-configuration* is the flat configuration in which the direction of the joint turns alternates. The chain C is in *maxspan configuration* when it is positioned to maximize the distance $|v_0v_n|$. We refer to the position of v_n in maxspan configuration as the *maxpt*.

Theorem 1 (n-Chain Partition Theorem) [2] *The planar partition for an n-chain C (described below) in maxspan configuration has the following two properties:*

1. *The vertices shared between adjacent planar sections all lie along the line L through v_0v_n.*
2. *The last planar section cannot contain just one link $v_{n-1}v_n$.*

This implies that in maxspan configuration the vertices v_0, v_1, v_2 and v_n all lie in the same plane. Furthermore if the maxspan configuration is not flat, then the vertices can be partitioned as follows: "Group $v_0, \ldots v_i$ into one section if they lie in plane Π_1, but v_{i+1} does not lie in this plane. Then group $v_{i+1}, \ldots v_j$ into a second section if they lie in plane $\Pi_2 \neq \Pi_1$, and v_{j+1} does not lie in Π_2. And so on" [2]. The vertices v_0, v_i, v_j and v_n all lie on the same line, and therefore all lie in the plane Π_1. See Fig. 1.

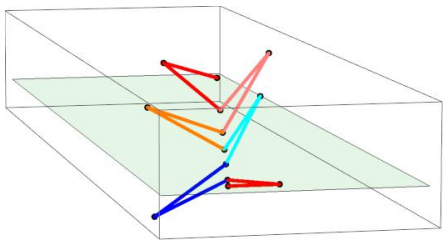


Figure 1: A 12-chain in maxspan configuration. Here vertices v_0, v_2, v_4, v_6, v_8, v_{10} and v_{12} are collinear.

Our search for the maxpt begins by laying out C in the zigzag configuration mentioned above. Note that some chains will be self-crossing when laid out this way and may possibly be self-crossing in the maxspan configuration as well. While these chains may not be of practical importance our method does not exclude this possibility.

The idea behind our algorithm is to search for the maxpt by systematically allowing the links to rotate out

*Mathematics Department, Eastern Washington University, ret., dkgoering@comcast.net

†Department of Mathematics, Eastern Washington University, rgentle@ewu.edu

of Π_1 beginning with L_n, then L_{n-1}, etc. At each step we observe which points in Π_1 are reachable by v_n under all possible rotations of the free links. We then identify those points which could conceivably be the maxpt for C and exclude the rest from further consideration. At each step new points reachable by v_n will be found, and points found in previous iterations may be excluded.

2 The subchain $C_{n-4} = (v_{n-4}, v_{n-3}, v_{n-2}, v_{n-1}, v_n)$

We begin by allowing L_n to rotate out of Π_1 about L_{n-1} while maintaining a constant angle α_{n-1} at the point of attachment v_{n-1}. As it does so, the locus of v_n is a circle orthogonal to Π_1 centered at the projection of v_n onto L_{n-1} extended. This circle intersects Π_1 in the original position of v_n and in it's reflection about L_{n-1}. We denote this reflection $v_n(n-1)$. See the circle in Fig. 2. Note that for the subchain $C_{n-3} = (v_{n-3}, v_{n-2}, v_{n-1}, v_n)$ the maxspan is $|v_{n-3}v_n|$ and is achieved in the trans-configuration [1].

Now allow L_{n-1} to rotate similarly about L_{n-2} while also allowing L_n to rotate about L_{n-1}. The points traced by v_n in this process comprise a partial sphere with center v_{n-2} and radius $|v_{n-2}v_n|$. The intersection of this partial sphere with Π_1 is two circular arcs, each also centered at v_{n-2} with radius $|v_{n-2}v_n|$. See Fig. 2.

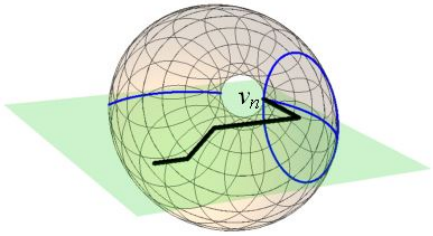


Figure 2: The circle generated by v_n from the rotation of L_n, the partial sphere generated by v_n from the rotations of L_n and L_{n-1}, and the arcs which are the traces of the partial sphere in Π_1.

The endpoints of the two arcs merit further discussion. These are reached in a flat configuration of the chain while the interior points are reached when L_n and L_{n-1} are rotated out of Π_1.

Observation 1 *Assuming again that C is in trans-configuration there are two cases.*

1. **The points v_n and $v_n(n-1)$ lie on the same side of L_{n-2} extended.** Then this line does not pass through the circle created by the rotation of L_n and these are the arc endpoints on one side. Their reflections about L_{n-2} are the arc endpoints on the other side which we denote $v_n(n-2)$ and $v_n(n-1, n-2)$.

2. **The points v_n and $v_n(n-1)$ lie on opposite sides of L_{n-2}.** Then L_{n-2} extended goes through the circle created by the rotation of L_n. In this case the arc endpoints on one side are $v_n(n-1, n-2)$ and v_n, with endpoints $v_n(n-1)$ and $v_n(n-2)$ on the other.

We illustrate these cases in Fig. 3.

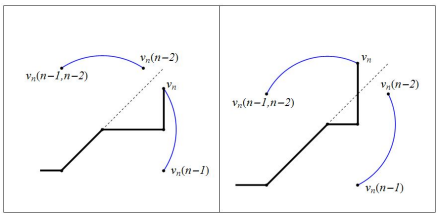


Figure 3: The two cases for arc endpoints following rotations of the last two links in the chain.

We are now in a position to find the maxspan of C_{n-4}. Note that we can do so without taking into account the points generated by the rotation of L_{n-2} at v_{n-3}. This rotation would move any point on the partial sphere in a circle about L_{n-3}, with each point on the circle remaining equidistant from any point on that line, specifically v_{n-4}. So none of these new points would be farther from v_{n-4} than the points on the two arcs.

To find the point on the arcs farthest from v_{n-4} we use the following basic facts.

Lemma 2 *Let C be a circle with center B, let A be an arc on C, and let P be a point in the plane of C other than B. The line PB intersects C in two points. Let Q be the farther of these points from P and let S be the closer. Then*

1. *the farthest point on C from P is Q and the closest such point is S.*
2. *if Q is on A then Q is the farthest point on A from P. If Q is not on A then the point on A closest to Q is the farthest point on A from P.*
3. *let f be a distance function from P to the points on A, traversed from one endpoint to the other. Then f is either*
   (a) *Decreasing with a maximum at the starting endpoint.*
   (b) *Increasing to a maximum then decreasing.*
   (c) *Increasing with a maximum at the terminal endpoint.*

Lemma 3 *Let l be a line, P and Q two points not on l, and Q' the reflection of Q across l. If P and Q are on the same side of l, then $|PQ| < |PQ'|$, otherwise $|PQ| > |PQ'|$.*

We now use these to find the maxpt of C_{n-4}, the farthest point on the arcs from v_{n-4}. Since one of these arcs is on the same side of L_{n-2} as v_{n-4}, all points on this arc can be eliminated by Lemma 3. Now consider the ray $v_{n-4}v_{n-2}$. If this ray intersects the remaining arc then the point of intersection is the maxpt by Lemma 2, with maxspan equal to $|v_{n-4}v_{n-2}|+|v_{n-2}v_n|$. Otherwise the maxpt is an arc endpoint. If the ray passes to the left of the arc (when viewed from L_{n-2}) the maxpt is the left-hand endpoint, otherwise the right, again by Lemma 2. The second case is illustrated in Fig. 4. The chain shown is a 4-chain so this is the final step in the algorithm.

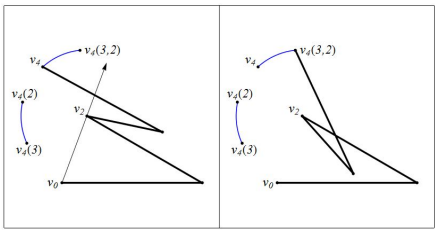


Figure 4: The ray v_0v_2 passes to the right of the upper arc so the maxpt is $v_4(3,2)$. The maxspan configuration is flat and is achieved by first reflecting L_3 and L_4 about L_2, then L_4 about L_3. The maxspan is $|v_0v_4(3,2)|$.

To illustrate the first case start with the chain on the left in Fig. 4 but with L_1 a bit shorter so that the ray v_0v_2 intersects the upper arc. This intersection is the maxpt and the maxspan configuration will occur with L_3 and L_4 rotated out of Π_1. See Fig. 5.

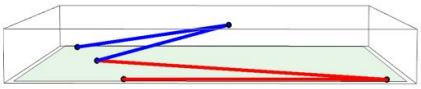


Figure 5: The maxspan configuration with links L_3 and L_4 rotated out of Π_1. The maxspan is $|v_0v_2| + |v_2v_4|$.

3 The subchain C_{n-5}

We have seen that the set of points reachable by v_n under all rotations of the last two links is a partial sphere whose trace in Π_1 is two arcs symmetric about L_{n-2}. We now wish to describe the points in Π_1 generated by the additional rotation of L_{n-3}. Referring again to Fig. 2 the partial sphere consists of a set of circles orthogonal to Π_1 centered on L_{n-2} extended. When rotated about L_{n-3} each of these circles will generate another partial sphere whose trace on Π_1 is again two arcs, this time symmetric about L_{n-3}. The result is an envelope of circular arcs. We will refer to this set of points on Π_1 as R_{n-3}. Some of the arcs in this envelope are shown in Fig. 6.

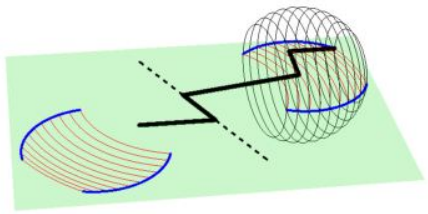


Figure 6: The envelope of arcs generated by the rotation of L_{n-2} about L_{n-3}.

The shape of these arc envelopes is determined by repeatedly applying Observation 1. If P and P' are on the original two arcs and symmetric about L_{n-2}, then the arcs generated by the circle containing these points depend on whether P and P' are on the same or different sides of L_{n-3}. For the chain in Fig. 6 all points on both of the original two arcs are on the same side of L_{n-3} extended. When this is not the case R_{n-3} can take on different appearances as in Fig. 7 below.

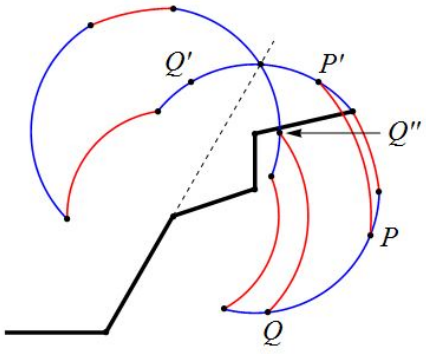


Figure 7: The boundary of an envelope of arcs when L_{n-3} extended intersects one of the original arcs. P and P' lie on the same side of L_{n-3} so are on the same arc. Q and Q' do not so Q and Q'' are on the same arc.

Regardless of appearance in every case R_{n-3} has the following properties:

1. R_{n-3} is symmetric about L_{n-3} extended.
2. R_{n-3} consists of two regions on either side of L_{n-3}, each closed and bounded by circular arcs.
3. Each of the eight points in Π_1 reachable by v_n is an endpoint for an arc on the boundary of R_{n-3}. If the line containing L_{n-3} intersects one of the original arcs there may be additional arc endpoints on this line.

Looking ahead we observe that when L_{n-4} is allowed to rotate it will result in R_{n-4}, the "envelope of an envelope" of arcs, with $R_{n-3} \subset R_{n-4}$. As the complexity

of R_{n-k} increases with each subsequent link rotation we need a way to keep our search for the maxpt as simple as possible. The next result is dedicated to this purpose.

3.1 Trimming

In this section we characterize the points in R_{n-3}, $n \geq 5$, that could possibly be a maxpt. For this purpose we define the "outer" boundary arcs (or arc portions) of R_{n-3}. Let T be a line orthogonal to L_{n-3} which intersects R_{n-3}. By symmetry there exist two points on T in R_{n-3}, one on each side, farthest from L_{n-3}. The set of all such points form the outer boundary arcs which we denote O_{n-3}. The arcs in O_{n-3} are shown in bold colors in Fig. 8. The next theorem allows us to exclude all points in R_{n-3} except those in O_{n-3} from further consideration.

Theorem 4 *Only those points on O_{n-3} can be a maxpt for n-chains with $n \geq 5$. Furthermore, only these points can generate arcs via subsequent link rotations that could possibly contain a maxpt.*

Proof. Let Q be a point on the interior of R_{n-3}. Then there exists a circle C centered at Q in R_{n-3} as well. Since the line v_0Q intersects C at two points, one of which is farther from v_0 than Q, Q is not a maxpt. Now let Q' be the reflection of Q about L_{n-3}, Q'' the reflection of Q' about L_{n-4}, and D the disk bounded by C. The link rotation about L_{n-4} will generate arcs with endpoints Q and Q' or Q and Q'' in R_{n-4}. Arcs will be generated for all points in D in a similar manner. So if A is a point on one of these arcs there will exist a disk centered at A which consists of points belonging to the corresponding arcs with endpoints in D. Therefore A will be an interior point of R_{n-4} and can therefore not be a maxpt by the preceding argument.

Now let Q be a point on the boundary of R_{n-3} but not in O_{n-3}. If v_0 is on the same side of L_{n-3} as Q, then Q cannot be a maxpt by Lemma 3. Otherwise let T be the line orthogonal to L_{n-3} that contains Q. Then T also contains a point S in O_{n-3} farther from L_{n-3} than Q. Now $|v_0Q| < |v_0S|$ by the triangle inequality and Q is not a maxpt. So only points in O_{n-3} can be a maxpt. Finally again let A be a point on an arc with endpoints in D as above. Then a line through A perpendicular to L_{n-4} will contain a point in R_{n-4} farther from v_0 than A as in the preceding argument and A can not be a maxpt.

□

These arguments generalize immediately to the outer boundary arcs of R_{n-k} for $3 \leq k \leq n-1$. So in each iteration we can confine our search for the maxpt to points on these outer boundary arcs.

The process for finding the endpoints of the outer boundary arcs is illustrated in Fig. 8. If a line through the center of a boundary arc parallel to L_{n-3} intersects the arc, then this intersection becomes a new arc endpoint. Portions of the arc below this line are excluded as are any arcs completely below such a line. We refer to this process as *trimming* the boundary arcs.

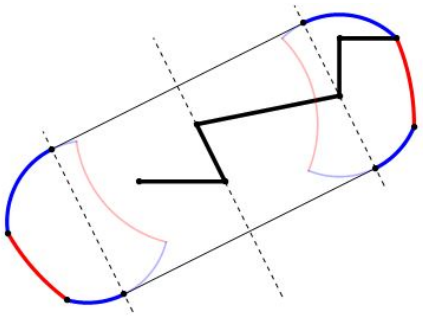


Figure 8: The only points in R_{n-3} which can be a maxpt for any chain containing C_{n-5} lie on the outer boundary arcs O_{n-3}.

3.2 Finding the maxpt on O_{n-3}

We now turn our attention to the task of locating the maxpt on the set of trimmed boundary arcs O_{n-3}. This is simplified by a result which, for $k = 3$ is a direct consequence of Lemma 2. A general proof for arbitrary k is omitted due to lack of space.

Theorem 5 *Let O_{n-3} be the set of outer boundary arcs described above and let P be a point in Π_1. Define a distance function f from P to the points on O_{n-3} (on the side of L_{n-3} opposite P), traversed from one endpoint to the other. Then f is either*

1. *Decreasing with a maximum at the starting endpoint.*
2. *Increasing to a maximum then decreasing.*
3. *Increasing with a maximum at the terminal endpoint.*

This result is used to create a simple algorithm for locating the farthest point on O_{n-3} from any point P in Π_1. If the farthest point from P on any arc A in O_{n-3} is on the interior of A then this is the farthest point from P in O_{n-3}. If the farthest point from P for two consecutive arcs is a shared endpoint then this is the farthest point from P in O_{n-3}. Otherwise the farthest such point is the starting or terminal arc endpoint in O_{n-3}.

As discussed in Section 2 the farthest point from P on each arc A can be determined by drawing a ray from P through the center of A. If the ray intersects A then this intersection is the point farthest from P. Otherwise

the arc endpoint closest to the ray is the point farthest from P.

Algorithm 1. **Maxpt Algorithm**

Input: *A point P in Π_1 and the set of connected arcs O_{n-k}, $2 \leq k \leq n-1$, including arc centers, endpoints and radii.*

Output: *The point M in O_{n-k} farthest from P.*

1. If the farthest point on Arc 1 from P is the LH endpoint then stop. This is M.
2. If the farthest point on Arc 1 from P is on the interior of the arc then stop. This is M.
3. Go to Step 1 and repeat with the next arc. If there are no more arcs then the RH endpoint of the last arc is M.

As an example we use the algorithm to find the maxpt of the 5-chain shown in Fig. 9. We work with the arcs on the side of L_2 opposite v_0. Begin with the leftmost arc as seen from L_2. Its center is v_3 and the closest point to ray v_0v_3 on this arc is the RH endpoint v_5. The center of the next (red) arc is v_2 and the closest point to ray v_0v_2 on this arc is the LH endpoint v_5. Thus v_5 is the maxpt M.

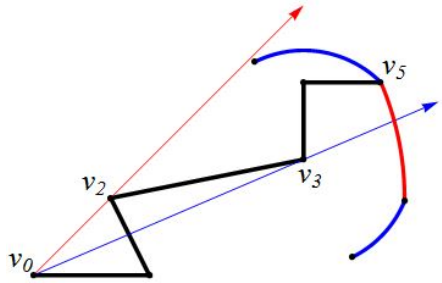


Figure 9: The farthest point from v_0 on Arc 1 is the RH endpoint v_5. The farthest such point on Arc 2 is the LH endpoint, also v_5. So v_5 is the maxpt and the maxspan is $|v_0v_5|$ achieved in the trans-configuration.

We are now in a position to describe our general algorithm. Start with O_{n-2}, the arcs from the rotations of L_n and L_{n-1}. In each iteration the rotation of link L_{i+1} about L_i creates an envelope from which we find the new set of outer boundary arcs O_i. Continue until O_2 is found. The farthest point M on O_2 from v_0 is the maxpt and $|v_0M|$ is the maxspan.

4 Boundary Arc Creation

There is one part of this process which has not yet been well described. The question is how to determine the new set of outer boundary arcs O_i from those in O_{i+1}.

Assume that O_{i+1} is known and we wish to find O_i. The situation is like that shown in Fig. 6 except that there are now multiple connected arcs symmetric about L_{i+1} instead of just one. Each symmetric pair of arcs is the trace of a partial sphere. Each pair then generates its own arc envelope when L_{i+1} is rotated about L_i. The outer boundary arcs of this union of envelopes is O_i which can be found via the following algorithm. Its justification is given in the Appendix.

Algorithm 2. **Boundary Arc Creation Algorithm**

Input: *O_{i+1}, the set of trimmed boundary arcs (endpoints, centers and radii) symmetric about L_{i+1}, and v_i.*

Output: *The trimmed boundary arcs O_i.*

1. Use the Maxpt Algorithm to find M_{i+1} and M'_{i+1}, the points on O_{i+1} farthest from v_i.
2. **Case 1: M_{i+1} and M'_{i+1} are on the same side of L_i.** The arc centered at v_i with M_{i+1} and M'_{i+1} as endpoints is on O_i. Arcs or arc portions between this new arc and L_i are deleted, all other arcs are kept. Trim the remaining arcs on each side of this new arc with respect to L_i, then reflect all about L_i. This collection of arcs is O_i.
3. **Case 2: M_{i+1} and M'_{i+1} are on opposite sides of L_i.** Reflect M'_{i+1} across L_i and call this point M''_{i+1}. The arc centered at v_i with endpoints M_{i+1} and M''_{i+1} is on O_i. If M''_{i+1} is to the left of M_{i+1} as seen from L_{i+1} then reflect all arcs and arc portions to the left of M_{i+1} on O_{i+1} first across L_{i+1}, then L_i. These reflected arcs belong to O_i as do those to the right of M_{i+1}. If M''_{i+1} is to the right of M_{i+1} then the process is identical except with arcs to the right of M_{i+1}. Trim the remaining arcs on each side of the new arc with respect to L_i, then reflect all about L_i. This collection of arcs is O_i.

In each case only one "new" boundary arc in O_i is created on each side of L_i. It is the arc of largest radius in the entire envelope. The remaining arcs were either already on O_{i+1} or are their reflections from the other side of L_{i+1} about L_i. Case 2 of this algorithm is illustrated in Fig. 10.

5 The Maxspan Algorithm

We now give the entire algorithm.

Algorithm 3. **Maxspan Algorithm**

Input: A chain $C = (v_0, v_1, \ldots, v_n)$ in flat zigzag configuration.

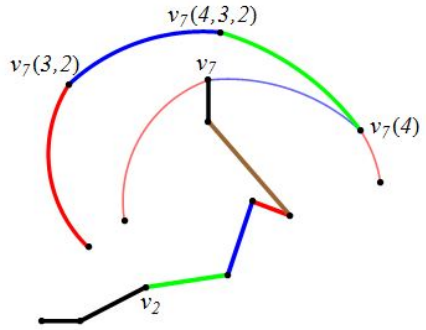


Figure 10: Creation of O_2 (thick arcs) from O_3 (thin arcs) as described in Case 2. The farthest point from v_2 on O_3 is $v_7(4)$. Its reflection about L_3 is on the opposite side of L_2, so the other new arc endpoint is a double reflection $v_7(4,3,2)$. Arcs on O_3 to the left of $v_7(4)$ are also reflected across both L_3 and L_2, then deleted. The rightmost arc on O_2 remains as part of O_3. Trimming with respect to L_2 then occurs (separately) on both sides of the new (green) arc but is not shown.

Output: The maximum span of C expressed in the form $|v_0v_i()| + |v_i()v_j()| + \cdots + |v_k()v_n()|$.

1. **Initialize.** Find O_{n-2}. Record the center, radius, and endpoints. Let $i = n - 3$.
2. **Find O_i, the new (trimmed) outer boundary arcs.** Use the boundary arc creation algorithm. Record their centers, radii, and endpoints. Decrement i.
3. **If $i > 2$ Go to Step 2.**
4. **Find M, the farthest point on O_2 from v_0.** Use the maxpt algorithm. This is the maxpt of C.
5. **Find the maxspan.** If M is an arc endpoint $v_n()$ then the maxspan is $|v_0v_n()|$ and the maxspan configuration is flat. Otherwise M is on an arc centered at $v_i()$ and the maxspan is $|v_0v_i()|$ plus the radius of this arc. The maxspan configuration will have one or more planar sections rotated out of Π_1.

6 Computational Complexity

The operations fundamental to each step of the algorithm (reflecting a point about a line, determining if a ray intersects an arc, etc.) are all constant time operations. The complexity of each step is then strictly a function of the number of boundary arcs in each iteration, so as steps are repeated the complexity of the algorithm as a whole depends on the rate of growth of the number of boundary arcs. This is difficult to determine in general since the number of boundary arcs may increase or decrease in each iteration. The number of arcs on each side of O_i may be one more than double the number in O_{i+1} but may also be reduced to just one.

For α-chains the number of boundary arcs increases linearly. We sketch the proof as follows: In Case 2 of the boundary arc creation algorithm the maximum number of new boundary arcs per iteration (prior to trimming) is two for *all* chains, not just α-chains. Generally in Case 1 the number of new boundary arcs in O_i can be double plus one the number in O_{i+1}. However these arcs are symmetric about L_{i+1} and all remaining vertices $v_0, v_1, \ldots, v_{i-1}$ are on the same side of L_{i+1}. So the arcs on the same side of L_{i+1} as the remaining vertices cannot contain M by Lemma 3 and can therefore be trimmed. In this case at most one new arc is added in each iteration. This gives a total of $k\sum_{i=1}^{n}\sum_{j=1}^{i} O(1) = O(n^2)$ operations for any α-chain.

More generally the number of boundary arcs can increase exponentially until trimming is required in some iteration of the boundary arc algorithm, after which the growth rate tends to be linear. For any given n it is possible to create an n-chain C with exponential boundary arc growth, though this can only be done with link lengths that grow exponentially, fixed angles approaching 180°, or both. These would not likely be present in any modeling environment. As links are repeatedly added to any given subchain trimming will eventually occur. So as $n \to \infty$ the complexity tends to $O(n^2)$.

References

[1] Nadia Benbernou and Joseph O'Rourke. On the Maximum Span of Fixed-angle Chains. In *Proc. 18th Conf. Comput. Geom.*, pages 93-96, 2006.

[2] Joseph O'Rourke. How to Fold It: The Mathematics of Linkages, Origami, and Polyhedra. *Cambridge University Press*, Thm. 3.2, p. 46, 2011.

[3] Michael Soss. Geometric and Computational Aspects of Molecular Reconfiguration. *Ph.D. thesis, School of Comput. Sci., McGill Univ.*, 2001.

[4] M. Soss and G. T. Toussaint. Geometric and computational Aspects of Polymer Reconfiguration. *J. Math. Chemistry*, 27(4):303-318, 2000.

[5] Ciprian S. Borcea and Ileana Streinu. Extremal Configurations of Manipulators with Revolute Joints. *Reconfigurable Mechanisms and Robots. ASME/IFToMM International Conference on. IEEE*, 2009.

[6] Ciprian S. Borcea and Ileana Streinu. Exact Workspace Boundary by Extremal Reaches and Extremal Reaches in Polynomial Time. *Proceedings of the twenty-seventh annual symposium on Computational geometry. ACM*, 2011.

Covering Points: Minimizing the Maximum Depth

Subhas C. Nandy*† Supantha Pandit*‡§ Sasanka Roy*¶

Abstract

We study a variation of the geometric set cover problem. Let P be a set of n points and T be a set of m objects in the plane. We find a cover $T' \subseteq T$ such that each point is covered by T' and the depth of T' is minimum. By ***depth*** of a cover T', we mean the maximum number of objects in T' which contain a point in P. We prove this problem to be NP-complete in $\mathbb{R}^2$ where the objects are unit squares or unit disks. More precisely, we show that it is NP-complete to decide whether this problem has a cover of depth 1. We present an $O((n+m)\log n + m \log m)$ time algorithm for the problem where the points are on a real line and objects are unweighted intervals. For weighted intervals, this problem can be solved in $O(nm \log n)$ time.

1 Introduction

Set cover is a well studied problem in the algorithm literature and has applications in diverse settings. Many real world problems can be modeled as instances of the set cover problem. In the set cover problem, we are given a universe U and a collection C of subsets of U, and the objective is to find a subcollection $C' \subseteq C$ which covers all the elements in U. In geometric setting, the universe U becomes a set of points and the set C becomes a set of geometric objects.

We consider an interesting variation of the geometric set cover problem. Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of points and $T = \{t_1, t_2, \ldots, t_m\}$ be a set of geometric objects on the plane. A subset $T' \subseteq T$ is a ***cover*** of P, if every point in P is contained in some object in T' i.e., $\forall p \in P,\ p \in \bigcup_{t_i \in T'} t_i$. The ***depth of a point*** $p \in P$ is the number of objects in T' which contain p. The ***depth*** of a cover T' is the depth of a point whose depth is maximum among all the points in P.

> **Min-max-coverage:** Given a set P of points and a set T of objects on the plane, the goal is to find a cover $T' \subseteq T$ for P such that the depth of T' is minimum among all possible covers $T'' \subseteq T$ for P.

*Indian Statistical Institute, Kolkata, India.
†nandysc@isical.ac.in
‡This work is partially supported by Grant No. PDF/2016/002490 for the National Post-doctoral Fellowship of the Science & Engineering Research Board, Department of Science and Technology, Government of India.
§pantha.pandit@gmail.com
¶sasanka@isical.ac.in

This problem has applications in the wireless coverage problem, where the objective is to choose the number of signal sending stations active such that each receiving station is connected with at least one of the sending station, and the number of sending stations from which a receiver station receives the signal is as small as possible to minimize interference.

The set cover problem is NP-complete and it is NP-hard to get an approximation algorithm with better than logarithmic factor [15]. A variation of the set cover problem is the maximum coverage problem. Here a universe U, a collection C of subsets of U, and an integer k is given, the goal is to find at most k sets from C which cover maximum number of elements from C. This problem is also NP-complete and has a $O(1 + \frac{1}{e})$ factor approximation algorithm [7]. The geometric set cover problem in $\mathbb{R}^2$ is NP-complete for several simple objects like disks [6], squares [6], etc. However, the same on a real line $\mathbb{R}$ is solvable in $O(n \log n)$ time. It needs to mention that, PTAS exists for geometric set cover problem with unit disks and unit squares as objects [13].

Another variation of the set cover problem is the unique cover problem. Given a set P of points and a set T of objects on the plane, the objective is to find a subset $T' \subseteq T$ of objects such that T' cover maximum number of points whose depth is exactly 1. This problem is NP-hard for both unit disks and unit squares [5]. For unit disks, a 4.31 factor approximation algorithm is available for the unique cover problem [8], and for unit squares PTAS exists [9].

Recently, Mehrabi [11] considered a variation of the set cover problem, called the unique set cover problem. Here also the input is a set P of points and a set T of objects on the plane; the goal is to find a subset $T' \subseteq T$ of objects such that the number of points whose depth is exactly 1 is maximized. He showed that, this problem is NP-complete for unit disks and units squares in the plane. Further, for unit squares he designed a PTAS using mod-one transformation trick of Chan and Hu for the red-blue set cover problem [4].

Another related problem is the weighted depth problem [3, 1, 2], where the input is a set P of points and a set T of n weighted box, the goal is to find a point whose depth is maximum. In $\mathbb{R}^d$, the time complexity of this problem is $O(n^d)$ [3].

In this paper, we present the following results.

- *Min-max-coverage-for-unit-square* problem is NP-complete (Section 2).

- *Min-max-coverage-for-unit-disk* problem is NP-complete (Section 3).
- *Min-max-coverage-for-unweighted-interval* problem can be solved in $O(n \log n)$ time (Section 4).
- *Min-max-coverage-for-weighted-interval* problem can be solved in $O(n^2 \log n)$ time (Section 5).

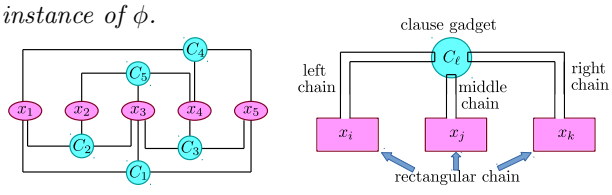
2 Covering by unit squares is NP-complete

In this section, we prove that ***Min-max-coverage-for-unit-square*** problem is NP-complete. We prove the NP-completeness of the decision version of this problem.

> ***Min-max-coverage-for-unit-square* decision problem:** Given a set of points P and a set of unit squares S, test whether there exists a cover S' of S that covers all the points in P and the depth of S' is 1.

We give a reduction from the ***rectilinear-positive-planar-one-in-3-SAT (RPP1in3SAT)*** problem [12] to the *Min-max-coverage-for-unit-square* decision problem. *RPP1in3SAT* problem is defined as follows:

Definition 1 (*RPP1in3SAT* [12]) *We are given a 3SAT formula ϕ with n variables $\{x_1, x_2, \ldots, x_n\}$ and m clauses $\{C_1, C_2, \ldots, C_m\}$. The variables are placed on a horizontal line. Each clause contains exactly 3 positive literals. Each three legged clauses whose three legs are connected to the three variables present in that clause. The clauses are connected with the variables from either above or below, and the clause-variable connection graph is planar. The goal is to find a truth assignment to the variables of ϕ such that exactly one literal in each clause of the formula ϕ is true. See Figure 1(a) for an instance of ϕ.*



(a) (b)

Figure 1: (a) Embedding of a *RPP1in3SAT* formula. (b) Schematic diagram of the construction of *Min-max-coverage-for-unit-square* problem.

We now construct an instance $\beta = (P, S)$ of *Min-max-coverage-for-unit-square* decision problem from an instance of ϕ of *RPP1in3SAT* as follows. We describe the construction for the clauses connected to the variables from above. Similar construction holds for the clauses connecting to the variables from below. Let x_i, x_j, and x_k be the left to right order of the variables which are connected from C from above. Then C connects x_i by ***left leg***, x_j by ***middle leg***, and x_k by ***right leg***. For example, In Figure 1(a) C_4 connects x_1, x_4, and x_5 by left, middle, and right legs.

Variable gadgets: Let δ be the maximum number of clause legs connected to a single variable either from above or from below. Each variable gadget may consist of four different types of ***chains***, a ***rectangular*** chain, and at most δ number of ***left*** (⌜-shaped), or ***middle*** (⊥-shaped), or ***right*** (⌝-shaped) chains. The left, middle, and right chains corresponding to the left, middle, and right clause legs respectively connecting to x_i from clauses from above.

For each variable x_i, the rectangular chain consists of $2q$ points (we fix the value of q later) $1, 2, \ldots, 2q$ placed in a rectangular fashion (see Figure 2). Consider $2q$ unit squares $s_1, s_2, \ldots, s_{2q}$ such that for $1 \leq i \leq 2q-1$, s_i contains i-th and $(i+1)$-th point, s_{2q} contains $2q$-th and 1-st point, and $\cup_{i=1}^{2q} s_i$ covers all the points. The $q-2$ points $1, 2, \ldots, q-2$ and $q-3$ squares $s_1, s_2, \ldots, s_{q-3}$ are responsible for the clauses which connects x_i from above. The $q-2$ points $q, q+1, \ldots, 2q-2$ and $q-3$ squares $s_{q+1}, s_{q+2}, \ldots, s_{2q-3}$ are responsible for the clauses which connects x_i from below.

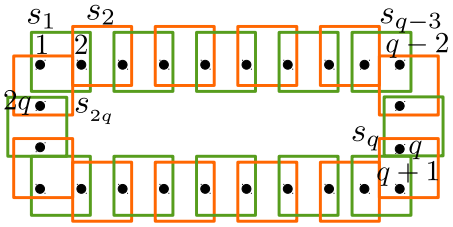


Figure 2: Rectangular chain.

The ***left***, ***middle***, and ***right*** chains are demonstrated in Figures 3(a), 3(b), and 3(c) respectively. Observe that, there are two special squares s' and s'' at the bottom of the vertical arrangements of squares in all the chains. These two squares establish connection with the rectangular chain.

We now describe how the left chain connects with the rectangular chain. The description is similar for the other chains. Let $1, 2, \ldots, \tau$ be the order of the clauses which connect to the variable x_i through their vertical legs. Let the ℓ-th clause leg in this order is a left leg which connects clause C_ℓ. We remove the square $s_{4\ell-1}$ from the rectangular chain. The square s' covers the point $4\ell-1$ and the square s'' covers the point 4ℓ in the rectangular chain. No other point from the rectangular chain can be covered by these two squares. See Figure 4(a) for this construction.

At most 2δ chains (at most δ from either above or below) combined with the rectangular chain together form a ***big-chain***. This big-chain corresponds to the variable gadget. Note that each big-chain contains even number of squares. It is easy to observe that, in the variable gadget of x_i there are two sets of unit squares, namely (i) all odd-numbered squares and (ii) all even-numbered squares, such that each set covers all the points and the depth of each set is $d = 1$.

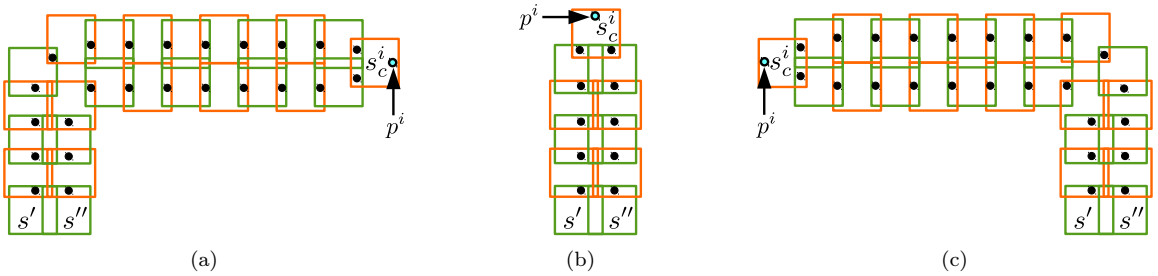


Figure 3: (a) A left chain. (b) A middle chain. (c) A right chain.

We choose the value of q as $4\delta + 4$. From the construction, each left, middle, or right chain from above connects the rectangular chain at a square $s_{4\tau-1}$, for $1 \leq \tau \leq \delta$. Similar construction is made for the chains from below which connect the rectangular chain. In each rectangular chain an even number of squares are required to produce exactly two optimal solutions.

Notice that, in each chain there is a unit square s_c^i such that: (i) s_c^i is an even-numbered square, and (ii) s_c^i contains an extra point p^i called the ***clause point*** and this point is a part of the clause gadget (described in next paragraph) not the variable gadget. The reason for choosing s_c^i even is that, if x_i is true then we select the even-numbered squares from the variable gadget of x_i and the clause point p^i is covered by this solution.

Clause gadgets: Let $C_\ell = (x_i \vee x_j \vee x_k)$ be a clause. The clause gadget for C_ℓ consists of 4 points $\{p^i, p^j, p^k, p^t\}$ and 3 unit squares called the ***clause squares***. There are 3 clause points $\{p^i, p^j, p^k\}$ and one ***special*** point p^t. Each of the clause square covers this special point p^t. In addition, each clause square covers exactly 2 of the 3 clause points (see Figure 4(b)).

We now describe the placement of the clause points and squares with respect to the variable points and squares. Let C_ℓ be a clause that contains three positive literals x_i, x_j, and x_k with the three chains, namely left, middle, and right, associated with it (see Figure 1(b)). The placement of the clause points and clause squares are shown in Figure 4(b).

This completes the construction. Clearly, the number of points and squares in β is a polynomial function on the number of clauses and variables in ϕ. Hence the construction can be done in polynomial time.

Theorem 1 *Min-max-coverage-for-unit-square decision problem is* NP*-complete.*

Proof. Clearly, this problem is in NP. Next we prove that, exactly one literal is true for every clause in ϕ if and only if β has a solution with depth $d = 1$.

Forward direction: Assume that, exactly one literal is true for every clause in ϕ. Let $A : \{x_1, x_2, \ldots, x_n\} \to \{0, 1\}$ be a truth assignment of ϕ. Select all even-numbered (orange colored) squares if $A(x_i) = 1$. Otherwise, select all odd-numbered (green colored) squares.

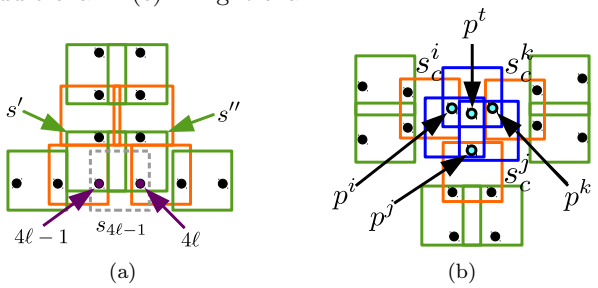
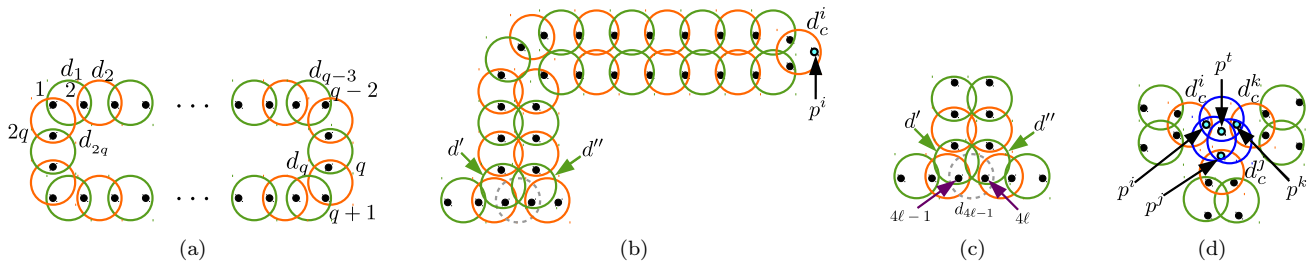


Figure 4: (a) Connection of rectangular chain with the vertical arrangement of any other chain. (b) Clause gadget and variable clause interaction. Blue colored squares are clause squares.

Consider a clause C whose exactly one of the literals is true. Then exactly one of the clause points is covered by the even-numbered square of the satisfied variable. The remaining two clause points can be covered by exactly one clause square. Clearly, the resulting solution covers all the points and the value of d is 1.

Reverse direction: Assume that β has a solution with depth $d = 1$. Note that, in any variable gadget there are 2α unit squares and 2α points for some integer α and each square covers exactly 2 consecutive points along any chain. Since the depth of the given cover for β is 1, these 2α points can be covered by exactly two sets of squares; either all even-numbered or all odd-numbered. We set variable x_i to *true*, if even-numbered squares are selected. Otherwise, set variable x_i to *false*. We now show that this assignment results in exactly one literal to be true for each clause. There are two cases. First, consider any clause C which contains no true literal. This implies that none of the clause point is covered by any variable gadget. Further, to cover the special point of C, one square from the clause gadget of C is required. This square covers 2 of the three clause points of C. To cover the remaining uncovered clause point one extra square is required. This square also covered an already covered point. This makes the depth of the cover at least 2. This is a contradiction. Second, assume that more than 2 literals are true for clause C. Here, at least two of the 3 clause points for C is covered by variable gadgets. To cover the special point of C, one clause square is required. This square makes the depth of the cover at least 2, a contradiction due to our assumption. □



(a) (b) (c) (d)

Figure 5: (a) A rectangular chain. (b) A left chain. (c) Connection of rectangular chain with the vertical arrangement of any other chain. (d) Clause gadget and variable clause interaction.

3 Covering by unit disk is NP-complete

We prove that *Min-max-coverage-for-unit-disk* problem is NP-complete by a reduction from the *RPP1in3SAT* problem [12] as in Section 2. From an instance of *RPP1in3SAT*, construct an instance $\beta = (P, D)$ of *Min-max-coverage-for-unit-disk* problem as follows.

Variable gadgets: The variable gadgets are analougous to the variable gadgets described in Section 2. A rectangular chain is shown in Figure 5(a). We only show the placement of the disks and points for the left chain in Figure 5(b). The construction of the middle and right chains are similar. In Figure 5(c), we show how the left chain is connected to the rectangular chain to form a big chain. Clearly, as in Section 2, there are two sets of unit disks, all odd-numbered or all even-numbered, such that each of this set covers all the points with $d = 1$.

Clause gadgets: Similar to clause gadgets description in Section 2, here also each clause gadget consists of 4 points and 3 unit disks (see Figure 5(d)).

This completes the construction. Since the construction is similar to the construction in Section 2, the number of points and disks in β is polynomial and hence the construction can be done in polynomial time. We now have the following theorem, whose proof is similar to that of Theorem 1.

Theorem 2 *Min-max-coverage-for-unit-disk problem is* NP*-complete.*

4 Minimizing the maximum coverage of points on real line by unweighted intervals

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points and $I = \{i_1, i_2, \ldots, i_m\}$ be a set of m intervals on the real line. Without loss of generality, assume $m \leq n$. For any point $p \in P$, let $I^p \subseteq I$ be the set of intervals which cover the point p. Let $i^p_{left} \in I^p$ be the interval whose left end-point is leftmost among all intervals in I^p, and i^p_{right} be the interval in I^p whose right end-point is rightmost among all intervals in I^p.

Lemma 3 *If a cover exists for an instance of Min-max-coverage-for-unweighted-interval problem, then it has a cover of depth at most* 2.

Proof. Assume that, there exists a cover I' with depth more than 2. Then there must be some point(s) in P which is covered by more than 2 intervals. Let $p \in P$ be such a point which is covered by k intervals $I^p = \{i^p_1, \ldots, i^p_k\}$. We select two intervals i^p_{left} and i^p_{right} from I^p. Note that, i^p_{left} and i^p_{right} may be the same interval. We modify I' as $I' \setminus I^p \cup \{i^p_{left}, i^p_{right}\}$. Clearly, the modified set I' is still a cover for P, where p is covered by at most 2 intervals. The same process is repeated if there exists any other point covered by more than two intervals in the modified I'. Finally, the resulting I' satisfies the lemma. □

Let us now consider the following problem:

> ***Min-max-coverage-interval-1:*** Is there a cover for the *Min-max-coverage-for-unweighted-interval* problem with depth exactly 1 ?

We formulate this problem as the ***Maximum Weight Independent Set (MWIS)*** problem with weighted intervals on real line as follows. For each interval $i = [a, b] \in I$, if it does not contain any point in P, remove it from set I; otherwise truncate the interval i to $i^* = [p^*_\ell, p^*_r]$ where $p^*_\ell, p^*_r \in P$ are the leftmost and rightmost point covered by i. Also assign a weight $w(i^*)$ = the number of points in P that are covered by i. Thus, we have a set I^* of n weighted intervals on real line, and the objective is to compute a subset $I' \subseteq I^*$ of pairwise non-intersecting intervals of maximum weight. Let $A(I^*)$ be a flag variable obtained from solving the algorithm for the *MWIS* problem on I^*. If $\sum_{i \in I'} w(i) = |P|$, then I' is the solution of the *Min-max-coverage-for-unweighted-interval* problem with depth $d = 1$ and $A(I^*) =$ **YES**. Otherwise, $A(I^*) =$ **NO**, and we execute Algorithm 1.

Theorem 4 *The proposed method solves the Min-max-coverage-for-unweighted-interval problem correctly in* $O((n+m)\log n + m \log m)$ *time.*

Proof. If $A(I^*) =$ **YES**, then $d = 1$ is confirmed, and this process requires $O((n+m)\log n + m \log m)$ time. Otherwise, it is easy to verify that there does not exist a cover of depth 1. In this case we run Algorithm 1. If Algorithm 1 returns 0 then there exists a point

Algorithm 1: *Min-max-coverage-for-unweighted-interval* problem

Input : An instance (P, I)
Output: 2 if (P, I) has a cover of depth 2; 0 otherwise.

```
1  Sort the members of P from left to right;
2  Sort the members of I from left to right with respect to
     their left end-points;
3  I'' = ∅; Q = ∅;
4  while P ≠ ∅ do
5      Let p be the first element in the sorted list P;
6      Compute i^p_right;
7      if i^p_right = ∅ (* i.e. no element is found in I to
         cover p *) then
8          return 0
9      end
10     else
11         I'' = I'' ∪ {i^p_right};
12         Q = Q ∪ {p_i};
13         remove all points from P which are contained
             in i^p_right;
14     end
15 end
16 return 2.
```

$p \in P$ that is not covered by any interval of I. If Algorithm 1 returns 2, we need to show that I'' is a cover of depth exactly 2. Let us consider the points in $Q = \{q_1, q_2, \ldots, q_k\}$ in left to right order. Observe that $i_{right}^{q_{i+1}}$ does not contain the point q_i. Otherwise, $i_{right}^{q_{i+1}}$ becomes $i_{right}^{q_i}$ as the right end-point of $i_{right}^{q_{i+1}}$ is to the right of the right end-point of $i_{right}^{q_i}$ (see Figure 6). Therefore, the points in P which are in between q_i

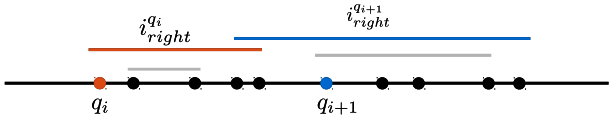


Figure 6: Proof of Theorem 4.

and q_{i+1} are covered by at most 2 intervals of I'' and the points in Q are covered by exactly 1 interval of I''. Thus the optimality of the algorithm follows. The running time follows from sorting. After the sorting of P and I, the algorithm performs a linear scan through the elements of both P and I. □

5 Minimizing the maximum coverage of points on real line by weighted intervals

In this section, we show that *Min-max-coverage-for-weighted-interval* problem can also be solved in polynomial time. Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points and $I = \{i_1, i_2, \ldots, i_m\}$ be a set of m weighted intervals on a real line $\mathcal{L}$. Each interval $i \in I$ is associated with a weight $w(i)$. Let $I' \subseteq I$ be a cover of the points in P. The ***weighted depth of a point*** $p \in P$ is the sum of the weights of all the intervals in I' containing p, and the ***weighted depth*** of I' is the maximum of weighted depth of all the points in P. It is easy to observe that Lemma 3 holds for weighted intervals also, i.e., if for every point $p \in P$ there exists an interval in I containing p, then there exists a cover of P having depth[1] at most 2. We design a dynamic programming based algorithm for the *Min-max-coverage-for-weighted-interval* problem, where the objective is to find a cover I' of P with minimum weighted depth.

Without loss of generality, we assume that the points $P = \{p_1, p_2, \ldots, p_n\}$ are sorted in right to left order, and the intervals $I = \{i_1, i_2, \cdots, i_m\}$ are sorted from right to left according to their left end-points. We process the intervals from I in order in an iterative manner. While processing $i_k = [a, b] \in I$, we solve the subproblem $Q_k = (P_k, I_k)$, where P_k is the subset of P that lie to the right of the point a (if a coincides with some $p_\alpha \in P$, then include p_α in P_k), and I_k be the subset of I with their left end-points to the right of the point a including i_k, and the objective is to compute a subset $I'_k \subseteq I_k$ such that the maximum weighted depth of points in P_k is minimized for I'_k among all feasible solutions of this problem. The weighted depth of the points covered by I'_k are stored in a balanced binary leaf-search tree A_k whose leaf nodes are points in P along with their weighted depth (sum of weights of all intervals in I'_k containing each point $p \in P_k$; the entries for $P \setminus P_k$ are 0), and each internal node contains the maximum weighted depth in the subtree rooted at that node. Initially $I'_0 = \emptyset$ and A_0 contains 0 for each element $\{p_1, p_2, \ldots, p_n\}$. While processing Q_k, we assume that the problems $Q_j, j = 0, 1, \ldots, k-1$ are already solved and the solutions $(I'_j, A_j), j = 0, 1, \ldots k-1$ are available. Algorithm 2, stated below, solves Q_k.

Lemma 5 *The solution A_k, produced by Algorithm 2 is optimum for the problem Q_k, and it needs $O(n + k \log n)$ time to compute.*

Proof. The optimality of Algorithm 2 follows from a recursive argument. We assumed that the solutions for $Q_j, j = k-1, \ldots, 1$ are optimum. If $P_{k-1} = P_k$, then (I'_{k-1}, A_{k-1}) is a feasible solution for Q_k, and we initialize MIN_MAX with the max of A_{k-1} (Steps 3–4).

Otherwise, we start with $MIN\_MAX = \infty$ (Step 7). Let Π_k = the set of points in $P_k \setminus P_{k-1}$ that are not covered by i_k. $\Pi_k \neq \emptyset$ (then min will be set to 0) implies there does not exists any feasible solution of Q_k (see Step 12). Otherwise we consider all possible solutions including i_k and considering $I'_j, j = k-1, \ldots, 1$, and have chosen the one that produces the minimum value of max (see Step 14). Time complexity follows from the following argument.

[1] not the weighted depth

Algorithm 2: *Min-max-coverage-for-weighted-interval* problem

Input : An instance (P_k, I_k) along with I'_j, A_j for all $j = 0, 1, \ldots, k-1$
Output: (I'_k, A_k); or 0 if infeasible

1 Initialization (* Check whether (I'_{k-1}, A_{k-1}) is feasible for Q_k *)
2 **if** $P_k = P_{k-1}$ **then**
3 Set MIN_MAX = max weighted depth in A_{k-1}, $A_k = A_{k-1}$, $I'_k = I'_{k-1}$;
4 **end**
5 **else**
6 Set $MIN\_MAX = \infty$, $MIN\_MAX\_ptr = 0$;
7 **end**
8 **for** $j = 0, \ldots, k-1$ **do**
9 $max = min = w(i_k)$ (* the points in P_k that are only covered by i_k has weighted cover $w(i_k)$ *);
10 Increase the weight of all points in P_j that are covered by i_k in the A_j data structure by an amount $w(i_k)$;
11 Compute the maximum weighted depth in A_j, if it is greater than max, then update max;
12 Compute the minimum weighted depth in A_j, if it is less than min, then update min;
13 **if** $min > 0$ *&* $max < MIN\_MAX$ **then**
14 (* $min > 0$ indicates that $I'_j \cup \{i_k\}$ is a feasible solution *)
15 update MIN_MAX with max, and
16 store j in MIN_MAX_ptr;
17 **end**
18 (* Get back to the original A_j for the computations in the subsequent iterations *)
19 Decrease the weight of all points in P_j that are covered by i_k in the A_j data structure by an amount $w(i_k)$;
20 **end**
21 **if** $MIN\_MAX\_ptr = 0$ **then**
22 The problem instance (P_k, I_k) is infeasible
23 **end**
24 **else**
25 (* Create the data structure A_k *)
26 Copy A_j in A_k, where $j = MIN\_MAX\_ptr$;
27 Increment the weighted depth of the points P_k (leaf entries of A_k) that are covered by i_k;
28 Set $I'_k = I'_j \cup \{i_k\}$;
29 **end**

- While working with (I_j, A_j), incrementing the relevant elements of A_j (Step 11) by $w(i_k)$ needed $O(\log n)$ time and then computing max and min took $O(\log n)$ time since A_j is maintained as a balanced leaf-search binary tree [14].
- Finally, copying $A_{MIN\_MAX\_ptr}$ in A_k, and then incrementing the relevant elements of $A_{MIN\_MAX\_ptr}$ by $w(i_k)$ (Steps 27,28) needed $O(n)$ time.
- In the k-th step, we need to consider (I_j, A_j) for all $j = 1, \ldots, k-1$. □

Finally, we report (I'_m, A_m) as the optimum solution.

Theorem 6 *The time and space complexities of the Min-max-coverage-for-weighted-interval problem are $O(nm \log n)$ and $O(nm)$ respectively.*

Proof. Both the optimality and time complexity follow from Lemma 5. The space complexity follows from the fact that we need to maintain (I'_j, A_j) for all $j = 1, 2, \ldots, m$, and each of them is of size $O(n)$. □

References

[1] A. Backurs, N. Dikkala, and C. Tzamos. Tight hardness results for maximum weight rectangles. In *ICALP*, pages 81:1–81:13, 2016.

[2] J. Barbay, T. M. Chan, G. Navarro, and P. Pérez-Lantero. Maximum-weight planar boxes in $O(n^2)$ time (and better). *IPL*, 114(8):437 – 445, 2014.

[3] T. M. Chan. Klee's measure problem made easy. In *FOCS*, pages 410–419, Oct 2013.

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

[5] T. Erlebach and E. J. van Leeuwen. Approximating geometric coverage problems. In *SODA*, pages 1267–1276, 2008.

[6] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *IPL*, 12(3):133 – 137, 1981.

[7] D. S. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Co., Boston, MA, USA, 1997.

[8] T. Ito, S. Nakano, Y. Okamoto, Y. Otachi, R. Uehara, T. Uno, and Y. Uno. A 4.31-approximation for the geometric unique coverage problem on unit disks. In *Theoretical Computer Science*, 544:14–31, 2014.

[9] T. Ito, S. Nakano, Y. Okamoto, Y. Otachi, R. Uehara, T. Uno, and Y. Uno. A polynomial-time approximation scheme for the geometric unique coverage problem on unit squares. *Comput. Geom.*, 51:25–39, 2016.

[10] J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., 2005.

[11] S. Mehrabi. Geometric unique set cover on unit disks and unit squares. In *CCCG*, pages 195–200, 2016.

[12] W. Mulzer and G. Rote. Minimum-weight triangulation is NP-hard. *J. ACM*, 55(2):11:1–11:29, 2008.

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

[14] M. H. Overmars, *Design of Dynamic Data Structures*. Springer-Verlag New York, Inc., 1983.

[15] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *STOC*, pages 475–484, 1997.

On the Planar Spherical Depth and Lens Depth

David Bremner * Rasoul Shahsavarifar *

Abstract

For a distribution function F on $\mathbb{R}^d$ and a point $q \in \mathbb{R}^d$, the *spherical depth* $SphD(q; F)$ is defined to be the probability that a point q is contained inside a random closed hyperball obtained from a pair of points from F. The *lens depth* $LD(q; F)$ is defined analogously using hyperlens instead of hyperball in the definition of spherical depth. The spherical depth $SphD(q; S)$ (*lens depth* $LD(q; S)$) is also defined for an arbitrary data set $S \subseteq \mathbb{R}^d$ and point $q \in \mathbb{R}^d$. This definition is based on counting all of the closed hyperballs (hyperlenses), obtained from pairs of points in S, that contain q. The straightforward algorithm for computing the spherical depth (lens depth) in dimension d takes $O(dn^2)$. The main result of this paper is an optimal algorithm for computing the planar (bivariate) spherical depth. The algorithm takes $O(n \log n)$ time. By reducing the problem of *Element Uniqueness*, we prove that computing the spherical depth (lens depth) requires $\Omega(n \log n)$ time. Some geometric properties of spherical depth (lens depth) are also investigated in this paper. These properties indicate that *simplicial depth* (*SD*) is linearly bounded by spherical depth and lens depth (in particular, $LD \geq SphD \geq \frac{2}{3} SD$). To illustrate these relationships, some experimental results are provided. In these experiments on random point sets, the bounds of $SphD \geq 2\, SD$ and $LD \geq 1.2\, SphD$ are achieved.

1 Introduction

The rank statistic tests play an important role in univariate non-parametric statistics. If one attempts to generalize the rank tests to the multivariate case, the problem of defining a multivariate order statistic will occur. It is not clear how to define a multivariate order or rank statistic in a meaningful way. One approach to overcome this problem is to use the notion of data depth. Data depth measures the centrality of a point in a given data set in non-parametric multivariate data analysis. In other words, it indicates how deep a point is located with respect to the data set.

Over the last decades, various notions of data depth such as *halfspace depth* (Hotelling, 1929, [9, 17]; Tukey, 1975, [19]), *simplicial depth* (Liu, 1990, [11]) *Oja depth* (Oja, 1983, [15]), *regression depth* (Rousseeuw and Hubert, 1999, [16]), and others have emerged as powerful tools for non-parametric multivariate data analysis. Most of them have been defined to solve specific problems in data analysis. They are different in application, definition, and geometry of their central regions (regions with the maximum depth). Some notable research on the algorithmic aspects of planar data depth can be found in [1, 2, 4, 5, 6, 7, 12, 14, 16].

In 2006, Elmore, Hettmansperger, and Xuan [8] defined another notion of data depth named *spherical depth*. It is defined as the probability that point q is contained in a closed random hyperball with the diameter $\overline{x_i x_j}$, where x_i and x_j are two random points from a common distribution function F. These closed hyperballs are known as influence regions of the spherical depth function. In 2011, Liu and Modarres [13], modified the definition of influence region, and defined *lens depth*. Each lens depth influence region is defined as the the intersection of two hyperballs $B(x_i, d(x_i, x_j))$ and $B(x_j, d(x_i, x_j))$. These influence regions of spherical depth (lens depth) are the multidimensional generalization of *Gabriel circles* (lunes) in the definition of the *Gabriel Graph* (*Relative Neighbourhood Graph*) [13, 18]. Spherical depth and lens depth have some nice properties including affine invariance, symmetry, maximality at the centre and monotonicity. All of these properties are explored in [8, 13, 20].

Although we focus on the planar case here, a notable characteristic of the spherical depth (lens depth) is that its time complexity grows linearly in the dimension d while for most other data depths the time complexity grows exponentially. To the best of our knowledge, the current best algorithm for computing the spherical depth (lens depth) is the straightforward algorithm which takes $O(dn^2)$.

In this paper, we present an $O(n \log n)$ algorithm for computing the spherical depth in $\mathbb{R}^2$. Furthermore, we reduce the problem of Element Uniqueness to prove that computing the spherical depth (lens depth) of a query point requires $\Omega(n \log n)$ time. We also investigate some geometric properties of spherical

*Faculty of Computer Science, University of New Brunswick, Fredericton, NB, Canada, {bremner,ra.shahsavari}@unb.ca

depth and lens depth. These properties lead us to bound the simplicial depth, spherical depth, and lens depth of a point in terms of one another. Finally, some experiments are provided to illustrate the relationship between spherical depth, lens depth, and simplicial depth.

2 Spherical Depth and Lens Depth

Definition: The spherical (lens) influence region of x_i and x_j in $\mathbb{R}^d$ is a closed hyperball (hyperlens) defined as follows:

$$Sph(x_i, x_j) = \left\{ t \mid d(t, \frac{x_i + x_j}{2}) \leq \frac{d(x_i, x_j)}{2} \right\}$$

$$L(x_i, x_j) = \{ t \mid \max\{d(t, x_i), d(t, x_j)\} \leq d(x_i, x_j) \},$$

where $d(.,.)$ is the Euclidean distance. Figures 1 and 2 show the $Sph(x_i, x_j)$ and $L(x_i, x_j)$ in $\mathbb{R}^2$, respectively.

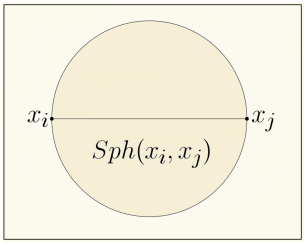


Figure 1: $Sph(x_i, x_j)$

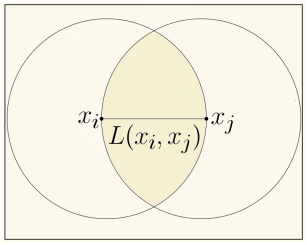


Figure 2: $L(x_i, x_j)$

Definition: For $S = \{x_1, ..., x_n\} \subset \mathbb{R}^d$ and $q \in \mathbb{R}^d$. The spherical (lens) depth of a q with respect to S, is defined as a proportion of $Sph(x_i, x_j)$ ($L(x_i, x_j)$), $1 \leq i < j \leq n$ that contain q. Using the indicator function $\boldsymbol{I}$, these definitions can be represented by (1) and (2).

$$SphD(q; S) = \frac{1}{\binom{n}{2}} \sum_{1 \leq i < j \leq n}^{n} I(q \in Sph(x_i, x_j)) \qquad (1)$$

$$LD(q; S) = \frac{1}{\binom{n}{2}} \sum_{1 \leq i < j \leq n}^{n} I(q \in L(x_i, x_j)) \qquad (2)$$

2.1 Algorithms for Computing the Spherical Depth of a Query Point

The current best algorithm for computing the spherical depth of a point $q \in \mathbb{R}^d$ with respect to a data set $S = \{x_1, x_2, ..., x_n\} \subseteq \mathbb{R}^d$ is the brute force algorithm. This naive algorithm needs to check all of the $\binom{n}{2}$ spherical influence regions obtained from the data points to figure out how many of them contain q. Checking all of the spherical influence regions causes the naive algorithm to take $\Theta(dn^2)$. Instead of counting, we focus on the geometric aspects of the spherical influence regions. These geometric properties lead us to develop an optimal $O(n \log n)$ algorithm for the computation of the spherical depth of q.
A proof of the following lemma which is a generalization of Thales' theorem[1] can be found in the Appendix.

Lemma 1 *For arbitrary points a, b, and t in $\mathbb{R}^2$, $t \in Sph(a, b)$ if and only if $\angle atb \geq \frac{\pi}{2}$.*

Algorithm: Using Lemma 1, we present an algorithm to compute the spherical depth of a query point $q \in \mathbb{R}^2$ with respect to $S = \{x_1, x_2, ..., x_n\} \subseteq \mathbb{R}^2$. This algorithm is summarized in the following steps.

- **Translating the points:** Suppose that T is a translation by $(-q)$. We apply T to translate q and all data points into their new coordinates. Obviously, $T(q) = O$.
- **Sorting the translated data points:** In this step we sort the translated data points based on their angles in their polar coordinates. After doing this step, we have S_T which is a sorted array of the translated data points.
- **Calculating the spherical depth:** Suppose that $x_i(r_i, \theta_i)$ is the i^{th} element in S_T. For x_i, we define the arrays O_i and N_i as follows:

$$O_i = \left\{ j \mid x_j \in S_T \, , \frac{\pi}{2} \leq |\theta_i - \theta_j| \leq \frac{3\pi}{2} \right\} \qquad (3)$$

$$N_i = \{1, 2, ..., n\} \setminus O_i.$$

Thus the spherical depth of q with respect to S can be computed by:

$$SphD(q; S) = SphD(0; S_T) = \frac{1}{2} \sum_{1 \leq i \leq n} |O_i|. \qquad (4)$$

To present a formula for computing $|O_i|$, we define f_i and l_i as follows:

$$f_i = \begin{cases} \min N_i - 1 & \text{if } \frac{\pi}{2} < \theta_i \leq \frac{3\pi}{2} \\ \min O_i & \text{otherwise} \end{cases}$$

$$l_i = \begin{cases} \max N_i + 1 & \text{if } \frac{\pi}{2} < \theta_i \leq \frac{3\pi}{2} \\ \max O_i & \text{otherwise.} \end{cases}$$

[1] Thales' theorem: If a, b, and c are points on a circle where $\overline{ac}$ is a diameter of the circle, then $\angle abc$ is a right angle

Figures 3 and 4 illustrate O_i, N_i, f_i, and l_i in two different cases. Considering the definitions of f_i and l_i,

$$|O_i| = \begin{cases} f_i + (n - l_i + 1) & \text{if } \frac{\pi}{2} < \theta_i \leq \frac{3\pi}{2} \\ l_i - f_i + 1 & \text{otherwise.} \end{cases}$$

This allows us to compute $|O_i|$ using a pair of binary searches.

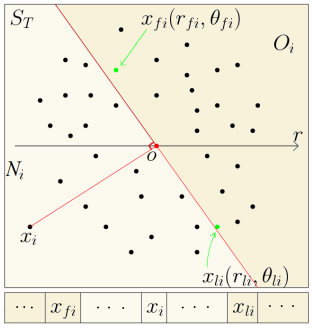


Figure 3: $\theta \in (\frac{\pi}{2}, \frac{3\pi}{2}]$.

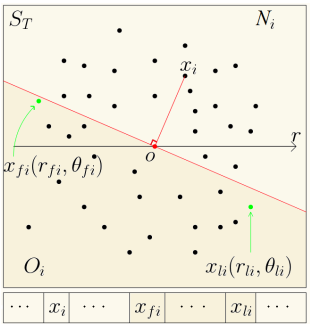


Figure 4: $\theta \notin (\frac{\pi}{2}, \frac{3\pi}{2}]$.

Time complexity: The first procedure in the algorithm takes $O(n)$ to translate q and all data points into the new coordinate system. The second procedure takes $O(n \log n)$ time. In this procedure, the loop iterates n times, and the sorting algorithm takes $O(n \log n)$. Due to using binary search for every O_i, the running time of the last procedure is also $O(n \log n)$. The rest of the algorithm contributes some constant time. In total, the running time of the algorithm is $O(n \log n)$.

Coordinate system: In practice it may be preferable to work in the Cartesian coordinate system. Sorting by angle can be done using some appropriate right-angle tests (determinants). Regarding the other angle comparisons, they can be done by checking the sign of dot products.

2.2 Lower Bound for Computing the the planar Spherical Depth and Lens Depth

We reduce the problem of Element Uniqueness[2] to the problem of computing the spherical depth and lens depth. It is known that the question of Element Uniqueness has a lower bound of $\Omega(n \log n)$ in the algebraic decision tree model of computation [3].

Theorem 2 *Computing the spherical depth of a query point in the plane takes $\Omega(n \log n)$ time.*

[2]Element Uniqueness problem: Given a set $A = \{a_1, a_2, ..., a_n\}$, is there a pair of indices i, j with $i \neq j$ such that $a_i = a_j$?

Proof. We show that finding the spherical depth allows us to answer the question of Element Uniqueness. Suppose that $A = \{a_1, a_2, ..., a_n\}$, for $n \geq 2$ is a given set of real numbers. We suppose all of the numbers to be positive (negative), otherwise we shift the points onto the positive X-axis. For every $a_i \in A$ we construct four points x_i, x_{n+i}, x_{2n+i}, and x_{3n+i} in the polar coordinate system as follows:

$$x_{(kn+i)} = \left(r_i, \theta_i + \frac{k\pi}{2}\right);\ 0 \leq k \leq 3,$$

where $r_i = \sqrt{1 + a_i^2}$ and $\theta_i = \tan^{-1}(1/a_i)$. Thus we have a set S of $4n$ points x_{kn+i}, for $1 \leq i \leq n$. The Cartesian coordinates of the points can be computed by:

$$x_{(kn+i)} = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}^k \begin{pmatrix} a_i \\ 1 \end{pmatrix};\ k = 0, 1, 2, 3.$$

See Figure 5.

We select the query point $q = (0, 0)$, and present an equivalent form of Equation (3) for O_j as follows:

$$O_j = \left\{x_k \in S \mid \angle x_j q x_k \geq \frac{\pi}{2}\right\},\ 1 \leq j \leq 4n, \quad (5)$$

We compute $SphD(q; S)$ in order to answer the Element Uniqueness problem. Suppose that every $x_j \in S$ is a unique element. In this case, $|O_j| = 2n + 1$ because, from (5), it can be figured out that the expanded O_j is as follows:

$$\begin{cases} \{x_{n+1}, .., x_{n+j}, x_{2n+1}, .., x_{3n}, x_{3n+j}, .., x_{4n}\}; & 1 \leq j \leq n \\ \{x_{2n+1}, .., x_{n+j}, x_{3n+1}, .., x_{4n}, x_{j-n}, .., x_n\}; & n < j \leq 2n \\ \{x_{3n+1}, .., x_{n+j}, x_1, .., x_n, x_{j-n}, .., x_{2n}\}; & 2n < j \leq 3n \\ \{x_1, .., x_{j-3n}, x_{n+1}, .., x_{2n}, x_{j-n}, .., x_{3n}\}; & \text{otherwise.} \end{cases}$$

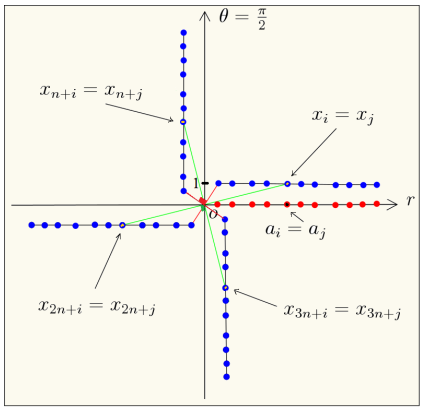


Figure 5: A representation of A, S, and duplications in these sets

Referring to Lemma 1 and Equation (4),

$$SphD(q;S) = \frac{1}{2}\sum_{1\le j\le 4n}(2n+1) = 4n^2+2n.$$

Now suppose that there exist some $i \neq j$ with $x_i = x_j$ in S. In this case, from (5), it can be seen that:

$$|O_{(kn+i) \bmod 4n}| = |O_{(kn+j) \bmod 4n}| = 2n+2,$$

where $k = 0, 1, 2, 3$ (see Figure 5). As an example, for $k = 0$, $|O_j| = |O_i| = 2n+2$ because the expanded form of these two sets is as follows: (without loss of generality, assume $i < j < n$)

$$O_i = O_j = \{x_{n+1}, .., x_{n+j}, x_{2n+1}, .., x_{3n}, x_{3n+i}, x_{3n+j}, x_{3n+j+1}, .., x_{4n}\}.$$

Lemma 1 and Equation (4) imply that:

$$SphD(q;S) \ge \frac{1}{2}(8 + \sum_{1\le j\le 4n}(2n+1)) = 4n^2+2n+4.$$

Therefore the elements of A are unique if and only if the spherical depth of $(0,0)$ with respect to S is $4n^2+2n$. This implies that the computation of spherical depth requires $\Omega(n\log n)$ time. It is necessary to mention that the only computations in the reduction are the construction of S which takes $O(n)$ time. □

Note: Instead of four copies of the elements of A, we could consider two copies of such elements to construct S. However, the depth calculation becomes more complicated in this case.

Theorem 3 *Computing the lens depth of a query point in the plane takes $\Omega(n\log n)$ time.*

Proof. Suppose that $B = \{b_1, b_2, ..., b_n\}$, for $n \ge 2$ is a given set of real numbers. Without loss of generality, let these numbers to be positive (see the proof of Theorem 2). We construct set $S = \{x_i, x_{n+i}\}$ of $2n$ points in the polar coordinate system such that $x_i = (b_i, 0)$ and $x_{n+i} = (b_i, \pi/3)$. See Figure 6. We select the query point $q = (0,0)$, and define L_j as follows:

$$L_j = \{x_k \in S \mid q \in L(x_j, x_k)\}\,,\ 1 \le j \le 2n. \quad (6)$$

Using Equation (6), the unnormalized form of Equation (2) can be presented by:

$$LD_S(q) = \frac{1}{2}\sum_{1\le j\le 2n}|L_j|. \quad (7)$$

We solve the problem of Element Uniqueness by computing $LD_S(q)$. Suppose that every $x_j \in S$ is a unique element. In this case, it can be verified that $L_j = \{x_{(n+j) \bmod 2n}\}$ (see Lemma 9 in the Appendix). Equation (7) implies that:

$$LD_S(q) = \frac{1}{2}\sum_{1\le j\le 2n}1 = n.$$

Now assume that there exists some $i \neq j$ with $x_i = x_j$ in S. In this case, $L_j = L_i = \{x_{(n+i) \bmod 2n}, x_{(n+j) \bmod 2n}\}$ and $L_{n+i} = L_{n+j} = \{x_{i \bmod 2n}, x_{j \bmod 2n}\}$ which means that

$$LD_S(q) = \frac{1}{2}\sum_{1\le j\le 2n}|L_j| = n+2.$$

In fact,

$$LD_S(q) = \frac{1}{2}\sum_{1\le j\le 2n}|L_j| = n+2c, \quad (8)$$

where c is the number of duplications in the elements of S. Therefore the elements of S are unique if and only if $c = 0$ in Equation 8. This implies that the computation of lens depth requires $\Omega(n\log n)$. Note that all of the other computations in this reduction take $O(n)$. □

Note: This technique of reduction can be generalized to prove that computing a generalization of spherical and lens depth called $\beta-$skeleton depth ($\beta > 1$) [20] also requires $\Omega(n\log n)$ time.

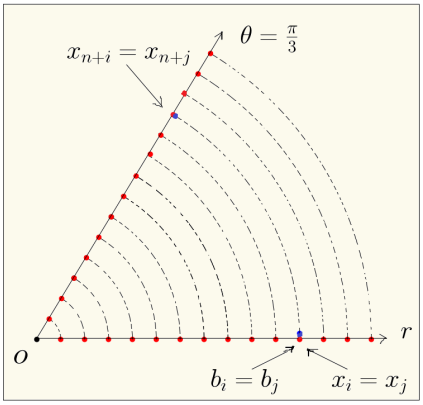


Figure 6: A representation of B, S, and duplications in these sets

3 Relationships Among Spherical Depth, Lens Depth, and Simplicial Depth

Theorem 4 *For $S \subset \mathbb{R}^d$ and $q \in \mathbb{R}^d$, $LD(q;S) \ge SphD(q;S)$.*

Proof. From the definition of the spherical (lens) influence regions of any arbitrary pair of points x_i and x_j

in S, it can be seen that $Sph(x_i, x_j) \subset L(x_i, x_j)$. Hence Equation (9) is sufficient to complete the proof.

$$\begin{aligned} SphD(q;S) &= \sum_{x_i,x_j \in S} I(q \in Sph(x_i,x_j)) \\ &\leq \sum_{x_i,x_j \in S} I(q \in L(x_i,x_j)) = LD(q;S) \end{aligned} \tag{9}$$

□

Definition: The simplicial depth of $q \in \mathbb{R}^d$ with respect to the data set $S = \{x_1, ..., x_n\} \subset \mathbb{R}^d$ is defined by:

$$SD(q;S) = \frac{1}{\binom{n}{d+1}} \sum_{(x_1,...,x_{d+1}) \in S} I(q \in Conv[x_1, ..., x_{d+1}]), \tag{10}$$

where $Conv[x_1, ..., x_{d+1}]$ is a closed simplex formed by $d+1$ points of S [11].

Definition: For a point $q \in \mathbb{R}^2$ and a data set S consisting of n points in $\mathbb{R}^2$, we define $B_{in}(q;S)$ to be the set of all closed sphere areas, out of $\binom{n}{2}$ possible sphere areas, that contain q. We also define $S_{in}(q;S)$ to be the set of all closed simplices, out of $\binom{n}{3}$ possible closed simplices defined by S, that contain q.

Lemma 5 *Suppose that q is a point in a given convex hull H obtained from a data set S in $\mathbb{R}^2$. q is covered by the union of sphere areas defined by S.*

See the Appendix for a proof of this Lemma.

Lemma 6 *Suppose that $S = \{a, b, c\}$ is a set of points in $\mathbb{R}^2$. For every $q \in \mathbb{R}^2$, if $|S_{in}(q;S)| = 1$, then $|B_{in}(q;S)| \geq 2$.*

A proof of this Lemma can also be found in the Appendix. Another form of Lemma 6 is that if $q \in \triangle abc$, then q falls inside at least two sphere areas out of three sphere areas $Sph(a, b)$, $Sph(c, b)$, and $Sph(a, c)$.

Lemma 7 *For $S = \{x_1, ..., x_n\} \subset \mathbb{R}^2$,*

$$\frac{|B_{in}(q;S)|}{|S_{in}(q;S)|} \geq \frac{2}{n-2}.$$

Proof. We suppose that $Sph(x_i, x_j) \in B_{in}(q;S)$ (see Figure 7). There exist at most $(n-2)$ triangles in $S_{in}(q;S)$ such that $x_i x_j$ is an edge of them. Let us consider $\triangle x_i x_j x_k$ from these triangles. Referring to Lemma 6, we know that q falls inside at least one of $Sph(x_i, x_k)$ and $Sph(x_j, x_k)$. It means that there exist at most $(n-2)$ triangles in $S_{in}(q;S)$ such that $x_i x_k$ (respectively $x_j x_k$) is an edge of them. As can be seen, the triangle $\triangle x_i x_j x_k$ is counted at least two times, one time for $Sph(x_i, x_j)$ and one time for $Sph(x_i, x_k)$ (or $Sph(x_j, x_k)$). So, we can say that for every sphere area from $B_{in}(q;S)$, such as $Sph(x_i, x_j)$ there exist at most $\frac{(n-2)}{2}$ distinct triangles, triangles with only one common side, in $S_{in}(q;S)$. Consequently, (11) can be obtained.

$$\frac{|B_{in}(q;S)|}{|S_{in}(q;S)|} \geq \frac{2}{(n-2)} \tag{11}$$

□

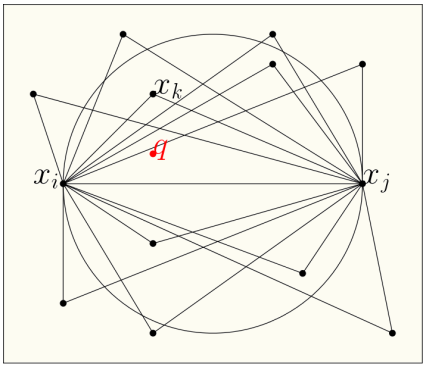


Figure 7: Sphere area $Sph(x_i, x_j)$ contains point q

Theorem 8 *For $q \in \mathbb{R}^2$ and a given data set S which consists of n points in $\mathbb{R}^2$, $SphD(q;S) \geq \frac{2}{3} SD(q;S)$.*

Proof. From the definitions of spherical depth and simplicial depth, it is clear that:

$$\frac{SphD(q;S)}{SD(q;S)} = \frac{\frac{|B_{in}(q;S)|}{\binom{n}{2}}}{\frac{|S_{in}(q;S)|}{\binom{n}{3}}} = \frac{|B_{in}(q;S)|}{|S_{in}(q;S)|} \times \frac{(n-2)}{3}. \tag{12}$$

From (12) and Lemma 7, it can be seen that

$$\frac{SphD(q;S)}{SD(q;S)} \geq \frac{2}{3} \Rightarrow SphD(q;S) \geq \frac{2}{3} SD(q;S).$$

□

4 Experiments

To support Theorem 8 and Theorem 4, we compute the spherical depth, lens depth, and the simplicial depth of the points in three random sets Q_1, Q_2, and Q_3 with respect to data sets S_1, S_2, and S_3, respectively. The elements of Q_i and S_i are some randomly generated points (double precision floating point) within the square $A = \{(x, y) | x, y \in [-10, 10]\}$. The results of our experiments are summarized in Table 1. Every cell in the table represents the corresponding depth of q_i with respect to data set S_i, where $q_i \in Q_i$. The cardinalities

of Q_is and S_is are as follows: $|Q_1| = 100$, $|S_1| = 750$, $|Q_2| = 750$, $|S_2| = 2500$, $|Q_3| = 2500$, $|S_3| = 10000$. As can be seen in Table 1, there are some gaps between experimental bounds for random points and the theoretical bounds. These gaps motivate us to do more research in this area.

	$(t_1; S_1)$		$(t_2; S_2)$		$(t_3; S_3)$	
	Min	Max	Min	Max	Min	Max
SD	0.00	0.25	0.00	0.25	0.00	0.24
$SphD$	0.01	0.50	0.00	0.50	0.00	0.50
LD	0.05	0.61	0.05	0.61	0.04	0.61
$\frac{SphD}{SD}$	2.00	∞	2.00	∞	2.03	∞
$\frac{LD}{SD}$	2.43	∞	2.44	∞	2.44	∞
$\frac{LD}{SphD}$	1.21	8.11	1.22	23.16	1.22	157.16

Table 1: Experimental results

5 Conclusion

In this paper, we developed an optimal $\Theta(n \log n)$ algorithm to compute the spherical depth of a bivariate query point with respect to a given data set in $\mathbb{R}^2$. In addition to the time complexity, the main advantage of this algorithm is it simplicity of implementation. To obtain a lower bound for computing the planar spherical (lens) depth, we reduced the Element Uniqueness problem to the computing of spherical (lens) depth. We also investigated some geometric properties which lead us to find some theoretical relationships (i.e. $SphD \geq \frac{2}{3}SD$ and $LD \geq SphD$) among spherical depth, lens depth, and simplicial depth. Finally, some experimental results (i.e. $SphD \geq 2SD$ and $LD \geq 1.2\,SphD$) are provided. More research on this topic is needed to figure out if the real bounds are closer to the experimental bounds or to the current theoretical bounds.

References

[1] Aloupis, Greg. On computing geometric estimators of location. *PhD thesis, McGill University*, Montreal, Canada, 2001.

[2] Aloupis, Greg and Langerman, Stefan and Soss, Michael and Toussaint, Godfried. Algorithms for bivariate medians and a Fermat–Torricelli problem for lines. *Journal of Computational Geometry*, Vol. 26, 69-79, 2003.

[3] Ben-Or, Michael. Lower bounds for algebraic computation trees. *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, 80-86, 1983.

[4] Bremner, David and Chen, Dan and Iacono, John and Langerman, Stefan and Morin, Pat. Output-sensitive algorithms for Tukey depth and related problems. *Journal of Statistics and Computing*, Springer , Vol. 18, 259-266,2008.

[5] Chan, Timothy M. An optimal randomized algorithm for maximum Tukey depth. *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, 430-436, 2004.

[6] Chen, Dan. Algorithms for Data Depth. *PhD thesis, Carleton University*, Ottawa, Canada, 2013.

[7] Christmann, Andreas. Regression depth and support vector machine. *DIMACS series in discrete mathematics and theoretical computer science*, AMS, Vol. 72, 71, 2006.

[8] Elmore, Ryan T and Hettmansperger, Thomas P and Xuan, Fengjuan. Spherical data depth and a multivariate median. *Journal of DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, AMS, Vol. 72, 87-101,2006.

[9] Hotelling, Harold. Stability in competition. *The Collected Economics Articles of Harold Hotelling*, Springer, 50-63, 1990.

[10] Libeskind, Shlomo. Euclidean and transformational geometry: A deductive inquiry. *Jones & Bartlett Publishers*, 2008.

[11] Liu, Regina Y. On a notion of data depth based on random simplices. *Journal of The Annals of Statistics*, Vol. 18, 405-414, 1990.

[12] Liu, Regina Y and Serfling, Robert Joseph and Souvaine, Diane L. Data depth: robust multivariate analysis, computational geometry, and applications. *American Mathematical Soc.*, Vol. 72, 2006.

[13] Liu, Zhenyu and Modarres, Reza. *Lens data depth and median*. Journal of Nonparametric Statistics, Vol. 23, 1063-1974, 2011.

[14] Matousek, Jirí. Computing the center of planar point sets. *Journal of Computational Geometry: Papers from the DIMACS special year*, AMS, 221-230, 1991.

[15] Oja, Hannu. Descriptive statistics for multivariate distributions. *Journal of Statistics & Probability Letters*, Elsevier, Vol. 1, 327-332, 1983.

[16] Rousseeuw, Peter J and Hubert, Mia. Regression depth. *Journal of the American Statistical Association*, Taylor & Francis Group, Vol. 94, 388-402, 1999.

[17] Small, Christopher G. A survey of multidimensional medians. *Journal of International Statistical Review/Revue Internationale de Statistique*, JSTOR, 263-277,1990.

[18] Supowit, Kenneth J. The relative neighborhood graph, with an application to minimum spanning trees. *Journal of the ACM (JACM)*, Vol 30, N3, 428-448,1983.

[19] Tukey, John W. Mathematics and the picturing of data. *Proceedings of the international congress of mathematicians*. Vol. 2, 523-531,1975.

[20] Yang, Mengta. Depth Functions, Multidimensional Medians and Tests of Uniformity on Proximity Graphs. *PhD Thesis, The George Washington University, USA*, 2014.

Appendix

Lemma 1: For arbitrary points a, b, and t in $\mathbb{R}^2$, $t \in Sph(a,b)$ if and only if $\angle atb \geq \frac{\pi}{2}$.

Proof. If t is on the boundary of $Sph(a,b)$, the *Inscribed Angle Theorem* (Theorem 2.2 in [10]) suffices as the proof in both directions. For the rest of the proof, by $t \in Sph(a,b)$, we mean $t \in int\ Sph(a,b)$.

$\Rightarrow$) For $t \in Sph(a,b)$, suppose that $\angle atb < \pi/2$ (proof by contradiction). We continue the line segment $\overline{at}$ to cross the boundary of $Sph(a,b)$. Let t' be the crossing point (see Figure 8). Since $\angle atb < \frac{\pi}{2}$, then, $\angle btt'$ is greater than $\frac{\pi}{2}$. Let $\angle btt' = \frac{\pi}{2} + \epsilon_1; \epsilon_1 > 0$. From the Inscribed Angle Theorem, we know that $\angle at'b$ is a right angle. The angle $tbt' = \epsilon_2 > 0$ because $t \in Sph(a,b)$. Summing up the angles in $\triangle tt'b$, as computed in (13), leads to a contradiction. So, this direction of proof is complete.

$$\angle tt'b + \angle t'bt + \angle btt' \geq \frac{\pi}{2} + \epsilon_2 + (\frac{\pi}{2} + \epsilon_1) = \pi + \epsilon_1 + \epsilon_2 > \pi \quad (13)$$

$\Leftarrow$) If $\angle atb = \frac{\pi}{2} + \epsilon_1; \epsilon_1 > 0$, we prove that $t \in Sph(a,b)$. Suppose that $t \notin Sph(a,b)$ (proof by contradiction). Since $t \notin Sph(a,b)$, at least one of the line segments $\overline{at}$ and $\overline{bt}$ crosses the boundary of $Sph(a,b)$. Without loss of generality, assume that $\overline{at}$ is the one that crosses the boundary of $Sph(a,b)$ at the point t' (see Figure 9). Considering the Inscribed Angle Theorem, we know that $\angle at'b = \frac{\pi}{2}$ and consequently, $\angle bt't = \frac{\pi}{2}$. The angle $\angle t'bt = \epsilon_2 > 0$ because $t \notin Sph(a,b)$. If we sum up the angles in the triangle $\triangle tt'b$, the same contradiction as in (13) will be implied. □

Lemma 5: Suppose that q is a point in a given convex hull H obtained from a data set S in $\mathbb{R}^2$. q is covered by the union of sphere areas defined by S.

Proof. It can be seen that there is at least one triangle, defined by the vertices of H, that contains q. We prove that the union of the sphere areas defined by such triangle contains q. See Figure 10. We prove this statement by contradiction. Suppose that q is covered by none of $Sph(a,b)$, $Sph(a,c)$, and $Sph(b,c)$. Therefore, Lemma 1 implies that none of the angles $\angle aqb$, $\angle aqc$, and $\angle bqc$ is greater than or equal to $\frac{\pi}{2}$ which is a contradiction because at least one of these angles should be at least $\frac{2\pi}{3}$ in order to get 2π as their sum. □

Lemma 6: Suppose that $S = \{a, b, c\}$ is a set of points in $\mathbb{R}^2$. For every $q \in \mathbb{R}^2$, if $|S_{in}(q;S)| = 1$, then $|B_{in}(q;S)| \geq 2$.

Proof. We prove the lemma by contradiction. By Lemma 5, $B_{in}(q;S) \geq 1$. Suppose that $|B_{in}(q;S)| = 1$. If q is located on the vertices of $\triangle abc$, it clear that $|B_{in}(q;S)| \geq 2$ thus, we suppose that q is not located on the vertices of $\triangle abc$. Without loss of generality, we suppose that q falls inside $Sph(a,b)$. For the rest of the proof, we focus on the relationships among the angles $\angle aqb$, $\angle cqa$, and $\angle cqb$ (see Figure 10). Since q is inside $\triangle abc$, $\angle aqb \leq \pi$. Consequently, at least one of $\angle cqa$ and $\angle cqb$ is greater than or equal to $\frac{\pi}{2}$.

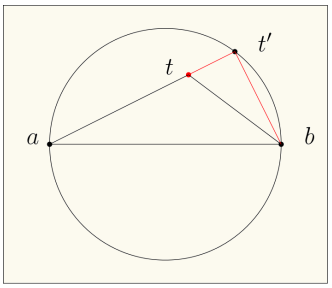


Figure 8: $t \in Sph(a,b)$

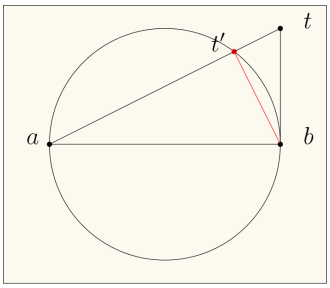


Figure 9: $t \notin Sph(a,b)$

So, Lemma 1 implies that q will fall inside at least one of $Sph(a,c)$ and $Sph(b,c)$. Hence, $|B_{in}(q;S)| = 1$ contradicts $|S_{in}(q;S)| = 1$. This means that the case $|B_{in}(q;S)| \geq 2$. As an illustration, in Figure 10, for the points inside the hatched area $|B_{in}(q;S)| = 3$.

□

Lemma 9 *$L_j = \{x_{(n+j) \bmod 2n}\}$ if every x_j $(1 \leq j \leq 2n)$ is a unique element in S , where $S = \{(b_i, 0), (b_i, \pi/3) \mid b_i > 0, 1 \leq i \leq n\}$, and $L_j = \{x_k \in S \mid q \in L(x_j, x_k)\}$.*

Proof. Suppose that $L_j = \{x_k, x_{(n+j) \bmod 2n}\}$ for some $x_k \in S$ $(k \neq j)$. We prove that such x_k does not exist. If $\angle x_j O x_k = 0$, it is obvious that $O \notin L(x_j, x_k)$ which means that x_k cannot be an element of L_j. For the case $\angle x_j O x_k = \pi/3$, let us assume that $O \in L(x_j, x_k)$ which is equivalent with $d(x_j, x_k) \geq d(O, x_k)$ and $d(x_j, x_k) \geq d(O, x_j)$. From the definitions $d(O, x_k) = b_k$, $d(O, x_j) = b_j$, and from the cosine formula, $d^2(x_j, x_k) = b_j^2 + b_k^2 - 2b_k b_j cos(\pi/3)$. Therefore,

$$d(x_j, x_k) \geq d(O, x_k) \Rightarrow b_j^2 - b_j b_k \geq 0 \Rightarrow b_j - b_k \geq 0 \Rightarrow b_j \geq b_k$$

and

$$d(x_j, x_k) \geq d(O, x_j) \Rightarrow b_k^2 - b_j b_k \geq 0 \Rightarrow b_k - b_j \geq 0 \Rightarrow b_k \geq b_j.$$

□

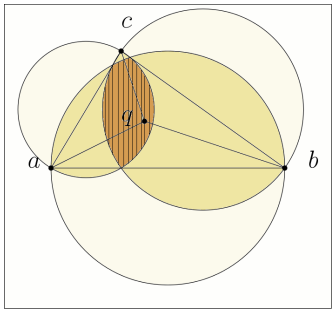


Figure 10: Triangle abc contains point q

Minimum Enclosing Circle Problem with Base Point

Binay Bhattacharya* Lily Li†

Abstract

This article presents a linear time algorithm to solve a variant of the minimum enclosing circle (MEC) problem. The inputs are a point set S of size n, and a point b in the plane called the *free point*. Our goal is to locate a circle center o^* such that the maximum distance of all points in S to o^* divided by the distance from o^* to b is minimized. The original investigation by Qiu et al. [5] found an $O(n \log n)$ algorithm using the furthest point Voronoi diagram of the point set S. This problem can be formulated as a generalized linear programming problem when the domain for the optimal solution is restricted and therefore, can be solved in linear expected time [3]. We describe here a simple deterministic linear time algorithm based on Meggido's prune-and-search solution to the standard problem [4]. We extend our technique to solve similar variants of the MEC problem where the free point is replaced with other geometric objects such as a free line, a free line segment, and a set of free points.

1 Introduction

The classical minimum enclosing circle (MEC) problem takes a point set S of size n and seeks to find a covering circle of smallest radius. Since this enclosing circle is uniquely defined by its center, the problem is equivalent to finding a point o^* to minimize its maximum distance to the points of S. This problem, proposed as early as 1856 by James J. Sylvester, yielded to various techniques [7]. Shamos and Hoey developed an $O(n \log n)$ algorithm [6]. Later, in 1983, Megiddo presented an $O(n)$ algorithm using the prune-and-search technique. By first constructing and solving a restricted problem, he was able to eliminate a fraction of the points at each iteration to solve the general problem optimally [4]. Simple linear time randomized algorithms have been proposed by Matousek et al. [3].

In addition to the search for an optimal algorithm, researchers studied variants of the standard problem by introducing weight to the points of S and by restricting the placement of the circle center [4, 2]. We came across the basic premise of this problem from a paper by Qiu et al. [5]. In previous MEC problems, a cost can be assigned to each point of the point set. The goal of the problems can be restated as minimizing the maximum cost. In the classical MEC problem, the cost of a point is its euclidean distance to the circle center. In the weighted MEC problem the cost of a point is this distance scaled by the weight of the point. Instead of a constant weight associated to each $p \in S$, Qiu et al. proposed a dynamic weight on the distance to a chosen point, known as the *free point*, in the plane. Their paper described an application of this work to target registration error.

Each variant of the classical problem considered here introduces a fixed geometric object to augment the distance to the circle center. The simplest geometric object to consider is a point so, stated formally, the free point variant of the problem is as follows:
Input: A point set S, $|S| = n$, and a free point b.
Goal: If $d(u, v)$ denotes the Euclidean distance between points u and v, then the cost $c_o(p, b)$ for a circle center o and point $p \in S$ with respect to the base point b is defined as:

$$c_o(p, b) = \frac{1}{d\,(b, o)} \cdot d\,(p, o)\,.$$

Thus our objective is to determine o^* such that:

$$\max_{p \in S} \frac{d\,(p, o^*)}{d\,(b, o^*)} = \min_{o \in \mathbb{R}^2} \left(\max_{p \in S} \frac{d\,(p, o)}{d\,(b, o)} \right).$$

As we will see later, we are interested in the distance function $c_o(p, b)$ where o lies in the half-space, delineated by the bisector of p and b, containing p. As a result, the problem under consideration can be formulated by a quasiconvex program. Such programs can be solved in expected linear time by using the randomized algorithm of Matousek et al. [3]. The contribution of this paper is to give a simpler deterministic algorithm using Megiddo's prune-and-search method. Similar algorithms can be designed for other dynamic MEC problems considered here.

In Section 2 we will present a linear time algorithm to solve this problem. Section 3 will explore the MEC problem with dynamic weight with respect to a *free line* and a *free line segment*. These three variants are solved in linear time by extending the algorithm described in Section 2 and by using previously constructed building blocks. Section 4 extends the free point variant by considering a set of free points. We will present an algorithm for this problem which is linear in the number

*School of Computer Science, Simon Fraser University, binay@cs.sfu.ca
†Same as above, xyl9@sfu.ca

of input points given that a *non-trivial* solution — a concept that we will define later — exists.

2 One Free Point

We begin our investigation into the dynamic weighted MEC problem with respect to a free point by considering the range of possible costs. Observe that as the circle center o approaches infinity in any direction, $c_o(p, b) \to 1$ for all $p \in S$ since $d(p, o)$ approaches $d(b, o)$ in value. Let o at infinity be the *trivial solution*. Thus it only makes sense to optimize the placement of o when $c_o(p, b) < 1$ for all $p \in S$.

Lemma 1 *If b is inside the convex hull of S, then for any circle center o, there exists a point $p \in S$ such that $c_o(p, b) \geq 1$.*

Proof. Place o anywhere in the plane and construct the convex hull $CH(S)$ of S. If the segment $\overline{ob}$ is inside or on the convex hull then there exists a point $p \in S$ such that $|\overline{op}| \geq |\overline{ob}|$. Suppose instead that o is outside $CH(S)$. Let L be the line containing b and o. Since b is inside $CH(S)$, L intersects an edge e of $CH(S)$ between o and b and an edge f of $CH(S)$ after b. Let u and v be the end-points of f. Either $d(u, o) \geq d(b, o)$ or $d(v, o) \geq d(b, o)$. If $d(u, o) \geq d(b, o)$ then $c_o(u, b) \geq 1$. If $d(v, o) \geq d(b, o)$ then $c_o(v, b) \geq 1$. See Figure 1. □

By Lemma 1, a nontrivial solution only exists when b is strictly outside the convex hull of S. We can check that this is the case in linear time. In the following, we assume that b is outside the convex hull of S.

The basis of our linear time algorithm will be Megiddo's prune-and-search algorithm for finding the unweighted MEC [4]. Our algorithm proceeds in two steps. First, we solve a restricted version of the problem in linear time. Using this solution as a subroutine, we will address the problem in full generality. At each step of the algorithm, we will be able to prune at least kn points where k is some constant in the interval $(0, 1)$ and n is the number of points remaining.

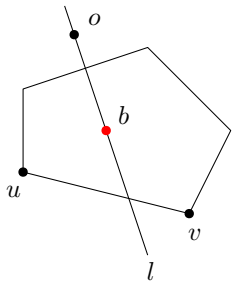


Figure 1: Configuration when base point is inside of the convex hull of point set.

2.1 Restricted Case for a Free Point

The restricted version of the problem is as follows:
Input: A point set S where $|S| = n$, a free point b, and a line L.
Goal: Find the optimum circle center o^* of the general one free point problem if it lies on L. Otherwise determine the side of L containing o^*.

2.1.1 Building Tools

To solve the restricted problem, we must understand the geometry of the optimum circle center o^*. We require that $c_{o^*}(p, b) < 1$ so, intuitively, o^* should be placed closer to the points in S than to b. We formalize our intuition in the following.

Let $L_{u,v}$ be the bisector line of points u and v. For any point p and circle center o, if o is on $L_{b,p}$, then $c_o(p, b) = 1$. Further, $L_{b,p}$ divides the plane in two halves, one side closer to b and the other closer to p. If o is placed on the side closer to p, then $c_o(p, b) < 1$. Thus, to ensure that $c_o(p, b) < 1$ for all points, we must place o on the side of $L_{b,p}$ closest to p for all $p \in S$. This is exactly the cell of b in the furthest point Voronoi diagram $S \cup b$. Since b is an extreme point of the convex hull of $S \cup b$, this cell is non-empty. Call this cell the *feasible region* of S with respect to b. Each point in the interior of the feasible region represents a circle center with cost less than one. Let the intersection of a line L with the feasible region be the *feasible interval* of L.

Lemma 2 *The feasible interval of L with respect to the point set S of size n and a free point b can be found in optimal $O(n)$ time.*

Proof omitted.

Next, focus on one point $p \in S$ and consider how moving o changes $c_o(p, b)$. In particular, we investigate all locations for o which yield the same value of $c_o(p, b)$. Let $c_o(p, b)$ be a constant α with $0 < \alpha < 1$. To simplify the calculation, locate p at the origin. Let $o = (o_x, o_y)$ and $b = (b_x, b_y)$. We determine the coordinates of o which keep the cost constant.

$$\alpha^2 = \left(\frac{d(p, o)}{d(b, o)}\right)^2 \tag{1}$$

$$= \frac{o_x^2 + o_y^2}{(o_x - b_x)^2 + (o_y - b_y)^2} \tag{2}$$

$$o_x^2 + o_y^2 = \alpha^2((o_x - b_x)^2 + (o_y - b_y)^2) \tag{3}$$

$$\alpha^2(b_y^2 + b_x^2) = (1 - \alpha^2)o_x^2 + (1 - \alpha^2)o_y^2 + 2\alpha^2 b_x o_x + 2\alpha^2 b_y o_y \tag{4}$$

Thus $c_o(p, b)$ remains unchanged when o is placed on the circle described by equation 4 with variable in o_x and o_y. Let this circle and its interior be denoted by

$D_\alpha(p,b)$. Observe that if o is located on the boundary (resp. inside, outside) of $D_\alpha(p,b)$, then $c_o(p,b) = \alpha$ (resp. $c_o(p,b) < \alpha$, $c_o(p,b) > \alpha$). See Figure 2. For the restricted case we need to find the value α such that $D_\alpha(p,b)$ is tangent to a line L. The interior of $D_\alpha(p,b)$, in this case, lies entirely on one side of L and this side contains the optimum circle center with respect to b.

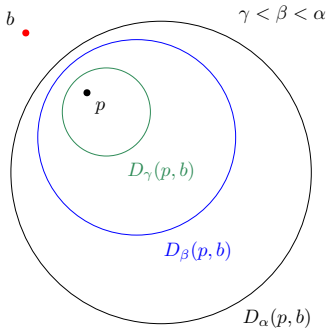


Figure 2: Circles of equal cost about point p with respect to the *free point* b.

Lemma 3 *Given a line L, a point p of the point set, and a free point b, we can find an α such that $D_\alpha(p,b)$ is tangent to L in constant time.*

2.1.2 Solving the Restricted Case for a Free Point

We will use the tools built thus far to solve the restricted problem in linear time.

Lemma 4 *Let a line L be given. If the optimum circle center o^* lies on L, then o^* can be found in linear time. Otherwise, it takes at most linear time to determine the side of L containing o^*.*

Proof. To simplify the explanation, we perform a transformation on the input so that L coincides with the x-axis.

1. Use Lemma 2 to find the feasible interval of L. If the interval is empty, return the side of L which does not contain the free point b. This side contains the optimal solution.

2. Suppose that the feasible interval of L is non-empty. Proceed through the following loop:

   (a) If the point set S contains fewer than two points, then the loop terminates. Otherwise, randomly pair up the points in S. For each pair $\{p,q\}$, find the intersection of the bisector $L_{p,q}$ with L. If $L_{p,q}$ is parallel to L, then the point in the pair closer to L is redundant and can be removed. Let the set of intersection points on L be I.

   (b) Find the median of I using a linear time algorithm [1]. Let this median value be x_m.

   (c) Suppose, without loss of generality, that x_m is to the left of the feasible interval of L. The optimum circle center x^* on L satisfies $x_m < x^*$. For each bisector $L_{p,q}$ which intersect L to the left of x_m, we can remove the rightmost point q of $L_{p,q}$ from S since q is closer to x^* than p. See Figure 3. Start again at step (a) with this modified S.

   (d) Otherwise, x_m falls strictly within the feasible interval. Let E be the subset of points in S farthest from x_m such that $d(p,x_m) = d_m$ for all $p \in E$. Then the cost $c_{x_m}(p,b) = \frac{d(p,x_m)}{d(x_m,b)} = \frac{d_m}{d(x_m,b)}$ for each $p \in E$. Let $\alpha = \frac{d_m}{d(x_m,b)}$.

      i. For every point $p \in E$ find the α-cost disk $D_\alpha(p,b)$ of p as discussed in Section 2.1.1. The boundary of each $D_\alpha(p,b)$ will intersect L at x_m and at most one other point.

      ii. Suppose, without loss of generality, that the boundary of some $D_\alpha(p,b)$ intersects L at a point to the right of x_m. Since $c_o(p,b) < \alpha$ when the circle center o is in the intersection of $D_\alpha(p,b)$ for all $p \in E$, $x_m < x^*$. Prune one point from each bisector intersecting L to the left of x_m as before. After removing the unnecessary points from S, restart from step (a).

      iii. Otherwise all α-cost disks are tangent to L at x_m. Exit the loop. See Figure 4c.

Upon termination, we can find the global optimal circle center o^* if it is on L or determine the side of L containing o^* by constructing the intersection of the $D_\alpha(p,b)$ for the remaining $p \in S$. See Figure 4.

In each iteration we do a linear amount of work and prune away at least one fourth of the remaining points. Thus the running time of the algorithm is linear. □

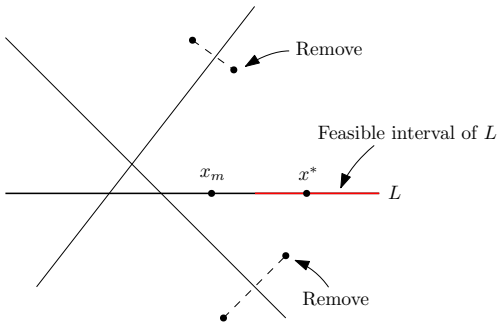
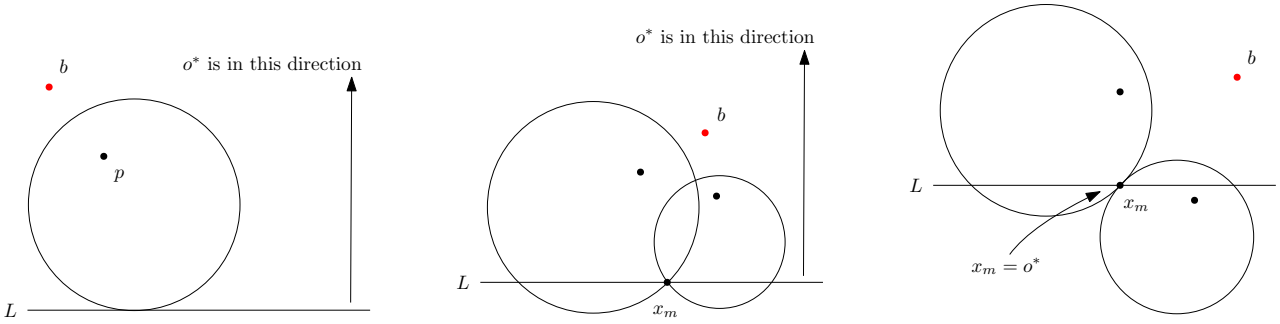


Figure 3: Points to remove if x_m falls outside the feasible interval of L.



(a) Terminate with only one point.

(b) Terminate with multiple points whose equal-cost circle all intersect.

(c) Terminate with multiple points where at least two have equal-cost circle which do not intersect.

Figure 4: Terminating conditions for the general case with one free point and global optimum circle center o^*.

2.2 General Case for a Free Point

We can design the algorithm for the general problem with respect to a free point using the linear time oracle for the restricted case. It is almost identical to Megiddo's general algorithm [4].

Theorem 5 *The optimum circle center o^* to minimize the maximum cost $c_{o^*}(p, b)$ of all points in the point set S with respect to a free point b can be found in linear time with respect to the size of S.*

3 A Free Line and a Free Line Segment

Similar to the dynamic MEC problem involving one free point, we will introduce the problem variant where the cost changes with respect to a free line.
Input: A point set S where $|S| = n$ and a free line B.
Goals: Find a circle center o such that the maximum cost $c_o(p, B)$ for the points $p \in S$ is minimized where the cost $c_o(p, B)$ of a point p is:

$$c_o(p, B) = \frac{1}{\min_{b \in B} d(b, o)} \cdot d(p, o).$$

The problem involving a free line segment is identical except that the input B is a line segment.

3.1 General Case with Free Line

We begin by considering cases where the trivial solution is optimal. In the case of the free point, the base point cannot be inside the convex hull of the point set S.

Lemma 6 *If B intersects the convex hull of S, then there exists a point $p \in S$ such that $c(p, B) \geq 1$.*

Proof. Suppose the free line (resp. free line segment) intersects edge e of the convex hull of S at some point b. Let u and v be the end points of e. Then for any circle center o, $d(o, u) \geq d(o, b)$ or $d(o, v) \geq d(o, b)$ similar to Lemma 1. Without loss of generality suppose $d(o, u) \geq d(o, b)$. If b' is the actual base point closest to o, then $d(o, b) \geq d(o, b')$ so

$$c_o(u, B) = c_o(u, b') = \frac{d(o, u)}{d(o, b')} \geq \frac{d(o, u)}{d(o, b)} \geq 1.$$

□

Thus the trivial solution, which locates the circle center at infinity orthogoal to B, is optimal when B intersects the convex hull of S. Determining if a line intersects the convex hull of a point set S takes linear time. In the forgoing, let the intersection of the free line B and the convex hull of S be empty and, without loss of generality, that all $p \in S$ are to the right of B.

Given a line L, first find the feasible interval of L with respect to B. Instead of intersecting L with the bisectors $L_{b,p}$ for $p \in S$ as is the case for the free point, we intersect L with the parabola of equal distance between p and the line B. Call this the 1-cost parabola of p and denote it by $H_1(p, B)$. Since a parabola intersects a line in at most two places, finding the intersection of L with $H_1(p, B)$ for all $p \in S$ takes linear time. Without too much ambiguity, let this intersection be the feasible interval of L.

Theorem 7 *Let B be a free line. The optimum circle center o^* to minimize the maximum cost $c_{o^*}(p, B)$ of all points in the point set S can be found in linear time.*

The crucial observation is that the intersection of the α-disks behaves as though we have a free point. Choose a point u_m on the line L as the circle center. Let b_m be the closest point on B to u_m. Suppose that the boundary of every $D_\alpha(p, b_m)$ intersect L to the left of u_m. For any u' on L right of u_m, with closest point b' on B, the cost for a point p is:

$$c_{u'}(p, b') = \frac{d(p, u')}{d(b', u')} > \frac{d(p, u')}{d(b_m, u')} \geq c_{u_m}(p, b_m).$$

The first inequality holds since $d(b_m, u') > d(b', u')$ and the second holds since u' is outside $D_\alpha(p, b_m)$.

3.2 A Free Line Segment

We observe that the linear time solution for the free line variant of the MEC problem easily adapts to a solution for a free line segment. Let K be the free line segment. Instead of the 1-cost parabolas considered in Theorem 7, we note that region in the plane equidistant between K and any point p in the point set is composed of constantly many line and parabola parts. Let these be 1-cost curves. Finding the intersection of all 1-cost curves takes linear time just like the 1-cost parabolas. Thus a slight modification to the algorithm in Theorem 7 yields an algorithm for the free line segment.

Corollary 8 *Let K be a line segment and S be a point set of order n. Finding the optimum circle center o^* to minimize the maximum cost $c_{o^*}(p, K)$ overall points $p \in S$ takes $O(n)$.*

4 A Set of Free Points

We extend the one free point problem by using a point set as the fixed geometric object. This variant of the MEC problem is stated formally as:
Input: A point set S where $|S| = n$ and a free point set B where $|B| = m$.
Goals: Find a circle center o^* such that the maximum cost $c_{o^*}(p, B)$ for the points $p \in S$ is minimized where the cost $c_o(p, B)$ of a point p with respect to center o is:

$$c_o(p, B) = \frac{1}{\min_{b \in B} d(b, o)} \cdot d(p, o).$$

Given a circle center o, this slightly altered cost calculation first finds the closest point $b^* \in B$ to o, then divides the distance to p by the distance to this closest free point b^*. More succinctly our objective is to find o^* such that:

$$\max_{p \in S} \left(\frac{d(p, o^*)}{\min_{b \in B} d(b, o^*)} \right) = \min_{o \in R^2} \max_{p \in S} \left(\frac{d(p, o)}{\min_{b \in B} d(b, o)} \right).$$

As before, we begin by considering when the trivial solution is optimal. Suppose that some $b \in B$ falls inside the convex hull, $CH(S)$, of S. For any o in the plane as the circle center, there exists a points $p \in S$ such that $c(p, b) \geq 1$ by Lemma 1. Consider the closest point $b' \in B$ to o. Observe that

$$1 \leq \frac{d(o, p)}{d(o, b)} \leq \frac{d(o, p)}{d(o, b')} = c_o(p, b') = c_o(p, B).$$

Since $d(o, b) \geq d(o, b')$, the trivial solution is optimal when any point in B falls within $CH(S)$. However, even if all points in B are outside the $CH(S)$, a non-trivial solution might still not exist. Possible cases include:

1. If S and B are linearly separable, then the optimal cost is less than one.
2. If S and B are not linearly separable, we have the following sub-cases.
   (a) If there exist a point of B inside $CH(S)$, the optimal cost is trivially one.
   (b) If all points of B lie outside $CH(S)$, but $CH(S)$ and $CH(B)$ intersect, the optimal cost could be one or less than one.

In the next section, we consider the case (1) where the optimal cost is less than one. Again we will solve this problem in two steps. First we build an oracle to solve the restricted problem on a line in linear time. Then we use the oracle to develop an algorithm to solve the general case. This second step is a modified version of the general case of the one free point variant so will only be mentioned briefly.

4.1 Restricted Case for a Set of Free Points

Input: A point set S where $|S| = n$, a free point set B where $|B| = m$ and a line L.
Goals: Find a circle center o^* which minimizes the maximum cost $c_{o^*}(p, B)$ for all $p \in S$ if o^* is on L. Otherwise determine the side of L containing o^*.

Lemma 9 *Let a line L, a point set S of order n, and a set of free points B of order m be given. Assume that B is linearly separable from S. We can find the side of L, possibly including L, containing o^* in $O(n + m)$ time.*

Proof. Perform a transformation of the input so that L coincides with the x-axis.

1. Randomly pair up the points of S. For each pair $\{p, q\}$, find the intersection of the bisector $L_{p,q}$ with L. Do the same with the free points of B. Let the set of all intersections on L be I. Note that $|I| = \lfloor \frac{n}{2} \rfloor + \lfloor \frac{m}{2} \rfloor$.
2. Find the median x-coordinate of the points of I using a linear time algorithm [1]. Let this median value be x_m.
3. Let $E = \{p \in S : d(p, x_m) = \max_{q \in S} d(q, x_m)\}$ be the points in S farthest from x_m and $F = \{b \in B : d(b, x_m) = \min_{c \in B} d(c, x_m)\}$ be the points in B closest to x_m. Further let $d(p, x_m) = u$ for all $p \in E$, $d(b, x_m) = v$ for all $b \in B$, and $u/v = \alpha$.
4. If $\alpha \geq 1$ then x_m is outside the feasible interval of L. We must decide which side of x_m contains the feasible interval of L, if exists. Randomly select one point $p \in E$ and one free point $b \in F$.

(a) Find the bisector $L_{p,b}$ of p and b. Suppose that $L_{p,b}$ intersects L at $x' \geq x_m$. Since $\alpha \geq 1$ and $u \geq v$, we must have b is to the left of $L_{p,b}$. Thus the optimum circle center x^* on L satisfies $x_m \leq x^*$. For bisectors $L_{s,q}$, with $s, q \in S$, intersecting L to the left of x_m, remove the non-dominating point associated with $L_{s,q}$. This point is closer to x^* so has lower cost. For bisectors $L_{r,t}$, with $r, t \in B$, intersecting L to the left of x_m, remove the dominating base point associated with $L_{r,t}$. This base point is farther from x^*. The case where every $D_\alpha(p, b)$ intersects L to the left of x_m can be handled similarly. Restart the loop after removing the redundant points.

5. If $\alpha < 1$ then x_m is in the feasible interval of the line L. Pick any $p \in E$ and for every $b \in F$ then calculate $D_\alpha(p, b)$. The optimum circle center o^* must be in $D_\alpha(p, b)$ for any b since the optimum cost of p is less than or equal to α.

   (a) Suppose there exists some $b \in F$ such that $D_\alpha(p, b)$ intersects L to the right of x_m. Then x^* satisfies $x_m \leq x^*$. Prune one point for every bisector which intersects L to the left of x_m as described above. The case where $D_\alpha(p, b)$ intersect L to the left of x_m can be treated similarly. Restart the loop after removing the redundant points.

   (b) Otherwise $D_\alpha(p, b)$ is tangent to L at x_m for every $b \in F$. Since a non-trivial solution exists, the half-plane of L containing any $D_\alpha(p, b)$ for any $p \in E$ and $b \in F$ contains the optimum circle center. Return this side of L.

The running time analysis is identical to that of the restricted case for one free point in Lemma 4. Here, however, the total size of the input is $|S| + |B| = n + m$. Thus the total running time is $O(n + m)$. □

4.2 Unrestricted Case for a Set of Free Points

Theorem 10 *Let B be a set of free points where $|B| = m$. The optimum circle center o^* to minimize the maximum cost $c_{o^*}(p, B)$ of all points in the point set S can be found in optimal $O(n + m)$ time.*

We can modify the algorithm for one free point to obtain an $O(n + m)$ algorithm for the general case for a free point set B.

We use the oracle presented in Lemma 9 when solving for the restricted problem. Next, since the standard algorithm already handles bisector intersections formed from points of S, we will instead consider bisector intersections formed from points of B. By determining the quadrant containing the optimal solution, a fraction of the base points can be eliminated from further considerations.

5 Conclusion

We considered a variant of the MEC problem where the cost of each point in the input point set S was dynamic with respect to a fixed geometric object such as a point, a line, a line segment, and a point set — with some restrictions. We have presented a deterministic prune-and-search based linear time algorithm to solve each of these problems by using the fact that the distance function is quasiconvex in the domain where the optimal solution could lie. For the case 2b of Section 4, it is not known whether a linear time solution exists.

Future work may consider a convex polygon as the fixed geometric object. The cost of a point p given a circle center o would be the $d(o, p)$ divided by the closest distance between o and a point on the convex polygon.

References

[1] Aho, A., Hopcroft, J., and Ullman, J. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] Hurtado, F., Sacristán, V., et al. Some constrained minimax and maximin location problems. In *Studies in Locational Analysis* (2000), Citeseer.

[3] Malousek, J., Sharir, M., and Welzl, E. A subexponential bound for linear programming. *Algorithmica 16*, 498–516.

[4] Megiddo, N. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing 12*, 4 (1983), 759–776.

[5] Qiu, L., Zhang, Y., and Zhang, L. Minimum enclosing circle of a set of static points with dynamic weight from one free point. *arXiv preprint arXiv:1703.00112* (2017).

[6] Shamos, M. I., and Hoey, D. Closest-point problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on* (1975), IEEE, pp. 151–162.

[7] Sylvester, J. J. A question in the geometry of situation. *Quarterly Journal of Pure and Applied Mathematics 1* (1857).

[8] Welzl, E. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science*. Springer, 1991, pp. 359–370.

Snipperclips: Cutting Tools into Desired Polygons using Themselves

Erik D. Demaine* Matias Korman† André van Renssen‡, § Marcel Roeloffzen‡, §

Abstract

We study *Snipperclips*, a computer puzzle game whose objective is to create a target shape with two tools. The tools start as constant-complexity shapes, and each tool can snip (i.e., subtract its current shape from) the other tool. We study the computational problem of, given a target shape represented by a polygonal domain of n vertices, is it possible to create it as one of the tools' shape via a sequence of snip operations? If so, how many snip operations are required? We show that a polynomial number of snips suffice for two different variants of the problem.

1 Introduction

Snipperclips: Cut It Out, Together! [8] is a puzzle game developed by SFB Games and published by Nintendo worldwide on March 3, 2017 for their new console, Nintendo Switch. In the game, up to four players cooperate to solve puzzles. Each player controls a character[1] whose shape starts as a rectangle in which two corners have been rounded so that one short side becomes a semicircle. The main mechanic of the game is *snipping*: when two such characters partially overlap, one character can *snip* the other character, i.e., subtract the current shape of the first character from the current shape of the latter character; see Figure 1. In addition, a *reset* operation allows a character to restore its original shape. An unreleased 2015 version of this game, *Friendshapes* by SFB Games, had the same mechanics, but supported only up to two players [4].

Puzzles in Snipperclips have varying goals, but an omnipresent subgoal is to form one or more players into desired shape(s), so that they can carry out required actions. In particular, a core puzzle type ("Shape Match") has one target shape which must be (approximately) formed by the union of the character's shapes. When the target shape has one connected component per character, the puzzle is equivalent to the characters reaching a desired set of target shapes, one per character. In this paper, we study when this goal is attainable, and when it is, analyze the minimum number of operations required.

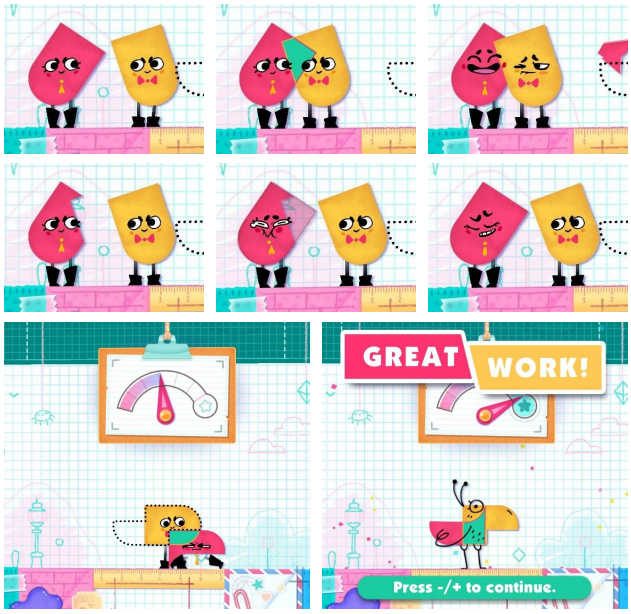


Figure 1: Cropped screenshots of Snipperclips: snipping, resetting, and solving a Shape Match puzzle. Sprites copyright SFB/Nintendo and included here under Fair Use.

2 Problems and Results

For the remainder of the paper we consider the case of exactly two characters or *tools* $\mathcal{T}_1$ and $\mathcal{T}_2$. For geometric simplicity, we assume that the initial shape of both tools is a unit square. Most of the results in this paper work for nice (in particular, fat) constant-complexity initial shapes, such as the rounded rectangle in Snipperclips, but would result in a more involved description.

We view each tool as an open set of points that can be rotated and translated freely.[2] After any rigid transformation, if the two tools have nonempty intersection,

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

†Tohoku University, Sendai, Japan, mati@dais.is.tohoku.ac.jp. Partially supported by MEXT KAKENHI Nos. 12H00855, and 17K12635.

‡National Institute of Informatics, Tokyo, Japan, {andre, marcel}@nii.ac.jp

§JST, ERATO, Kawarabayashi Large Graph Project. Supported by JST ERATO Grant Number JPMJER1305, Japan

[1] The game in fact allows one human to control up to two characters, with a button to switch between which character is being controlled.

[2] In the actual game, the tools' translations are limited by gravity, jumping, crouching, stretching, standing on each other, etc.,

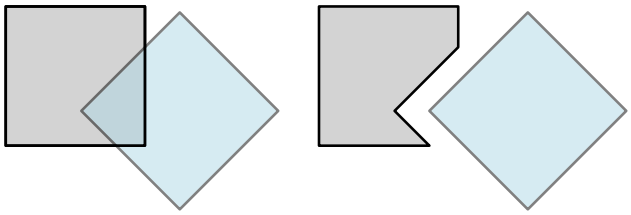


Figure 2: By translating and rotating the two tools we can make them partially overlap (left figure). In the right we see the resulting shape of both tools after the snip operation.

we can *snip* (or *cut*) one of them, i.e., remove from one of the tools the closure of the intersection of the two tools (or equivalently, the closure of the other tool); see Figure 2. (The closure is used to preserve the invariant that both tools remain open sets.) In addition to the snip operation, we can *reset* a tool, which returns it back to its original unit-square shape.

Although we forbid it in our study, the actual game has an additional *undo/redo* operation, allowing each tool to change into its previous shape (before its last snip or undo/redo operation, but not before its last reset operation), effectively implementing an undo stack of size 1. Our positive results are stronger without needing this operation; our negative results (Section 3) are weaker without allowing this operation, and may not hold in the stronger undo/redo model.

After a snip operation, the changed tool could become disconnected. There are two natural variants on the problem of how to deal with disconnection. In the *connected model*, we force each tool to be a single connected component. Thus, if the snip operation disconnects a tool, we can choose which component to keep. In the *disconnected model*, we allow the tool to become disconnected, viewing a tool as a set of points to which we apply rigid transformations and the snip/reset operation. The Snipperclips game by Nintendo follows the disconnected model, but we find the connected model an interesting alternative to consider.

Ideally, given two target shapes P_1 and P_2, we would like to find a sequence of snip/reset operations that transform tool $\mathcal{T}_1$ into P_1 and at the same time transform $\mathcal{T}_2$ into P_2. However, as we show in Observation 1, this is not always possible, even when $P_1 = P_2$. Instead, we consider creating a single target shape P_1 by one of the tools $\mathcal{T}_1$. Because our initial shape is polygonal, and we allow only finitely many snips, the target shape must be a polygonal domain, say of n vertices. The primary goal of this paper is to design an algorithm that, for any target shape P_1, can transform tool $\mathcal{T}_1$ into the desired shape P_1 using as few snip and reset operations as possible. Specifically, our aim is for the number of snip and reset operations to depend only on n (and not depend on other parameters such as the feature size of the target shape).

though in practice this is not a huge limitation. Rotation is indeed arbitrary.

2.1 Results

For negative results (Section 3), we show a pair of shapes that cannot be simultaneously realized by both tools. We also provide a shape that requires $\Omega(n)$ snips when we aim to construct it in a single tool (in both the connected and disconnected models). For positive results, we give constructive algorithms to create any target shape in the connected model using $O(n)$ snips (Section 4) and in the disconnected model using $O(n^2)$ snips (Section 5).

2.2 Related Work

Computational geometry has considered a variety of problems related to cutting out a desired shape using a tool such as circular saw [3], hot wire [5], and glass cutting [6, 7]. The Snipperclips model is unusual in that the tools are themselves the material manipulated by the tools. This type of model arises in real-world manufacturing, for example, when using physical objects to guide the cutting/stamping of other objects—a feature supported by the popular new *Glowforge* laser cutter [1] via a camera system.

Our problem can also be seen as finding the optimal Constructive Solid Geometry (CSG) expression tree, where leaves represent base shapes (in our model, rectangles), internal nodes represent shape subtraction, and the root should evaluate to the target shape, such that the tree can be evaluated using only two registers. Applegate et al. [2] studied a rectilinear version of this problem (with union and subtraction, and a different register limitation).

3 Lower Bounds

We begin with the intuitive observation that not all combinations of target shapes can be constructed.

Observation 1 *In both the connected and disconnected models (without undo/redo), there is a target shape that cannot be realized by both tools at the same time.*

Proof. Consider the target shape shown in Figure 3: a unit square in which we have removed a very thin hole in the middle. First observe that, if we perform no resets, neither tool has space to spare to construct a thin auxiliary needle to carve out the middle section of the other tool. Thus, after we have completed carving one tool, the other one would need to reset. This implies that we cannot have the target shape in both tools at the same time.

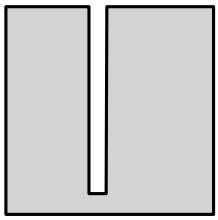


Figure 3: A target shape that cannot be realized by both tools at the same time.

Now assume that we can transform both tools into the target shape by performing a sequence of snips and resets. Consider the state of the tools just after the last reset operation. One of the two shapes is the unit square and thus we still need to remove the thin hole using the other shape. However, because no more resets are executed, the other tool is currently and must remain a superset of the target shape. In particular, it can differ from the square only in the thin hole, so it does not have any thin portions that can carve out the hole of the other tool.

Because the above argument is based solely on the shape of the figure, it holds in both the connected and disconnected model. □

Next we show that constructing a target shape in only one of the two tools may require a linear number of operations in both the connected and disconnected model.

Theorem 2 *There are target shapes that require $\Theta(n)$ snips to construct, both in the connected and disconnected model (without undo/redo).*

Proof. Consider the target shape P_1 to be a set of $n/3$ triangles on a line such that the distance between two consecutive triangles grows exponentially. In the connected model, we add a strip to connect these triangles; see Figure 4. We complete the construction by scaling it so that the width and height of P_1 is unit (and thus fits in either tool).

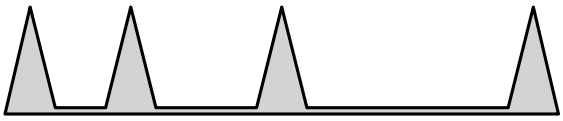


Figure 4: The target shape used in the lower bound.

First observe that it is straightforward to construct P_1 into one of the two tools by making the other tool a rectangle with width smaller to or equal to the shortest segment of P_1. Thus, we now proceed to prove the lower bound.

Consider now a sequence of snip and reset operations that can be used to construct P_1 in one of the tools. If we only have tools that cut out a constant number of edges or we have a linear number of tools, the lower bound follows, so we focus on tools that cut out multiple edges from different triangles. We observe that due to the spacing between the triangles, these tools cannot be reused to cut out multiple edges from any other triangles. Hence, if we can show that the number of snips required to construct these tools is linear in their complexity, we are done.

Let us consider such a tool that cuts out multiple edges from different triangles. Note that this tool has strictly fewer vertices than P_1. Using similar arguments as above, we can assume that each such tool was not constructed using only tools that cut out a constant number of edges nor using a linear number of tools. If we continue this recursion, each time we recurse into strictly smaller tools. Hence, in the final step, we end up with tools that are constructed using tools that cut out a constant number of edges or we have a linear number of tools in total. Regardless, the lower bound follows. □

4 Connected Model

In the connected model, the shapes must remain connected and we enforce this by choosing a connected component whenever a snip breaks the shape into multiple pieces. In this model, we show that $O(n)$ snips suffice to create any polygonal shape of n vertices.

Theorem 3 *We can cut one of the tools into any target polygonal domain P_1 of n vertices using $O(n)$ snip operations (and no reset) in the connected model.*

Proof. The idea is that we can shape $\mathcal{T}_2$ into a very narrow triangle, a *needle*, and use that to cut along the edges of the target shape P_1. Whenever a snip disconnects the shape, we simply keep the one containing the target shape. Initially, we start with a long needle to cut the long edges of $\mathcal{T}_2$ and we gradually shrink the needle to cut the smaller edges.

More formally, let α be the smallest angle between any two adjacent edges of P_1. As it will be seen later, we need α to be small. Thus, if $\alpha > 1°$, we simply lower it to $1°$ instead. Furthermore, let h be the shortest distance between any vertex and a non-adjacent edge. Our needle will be an isosceles triangle, with the two equal-length edges making an angle of at most α and the base edge with length equal to h (if this would cause the equal length segments to have length greater than one we reduce them to unit length, see Figure 5).

Now we group all edges of P_1 into sets based on their length. Let $\mathcal{E}$ denote the full set of edges defining P_1 and let $\mathcal{E}_i$, for $0 \leq i$, be the set of edges whose length is between 2^{-i-1} and 2^{-i}. To cut along the edges of $\mathcal{E}_i$, we use a needle where the equal-length edges have length 2^{-i-2}. Such a needle can construct each edge in $\mathcal{E}_i$ using at most four snips; see Figure 5. For an edge e, its

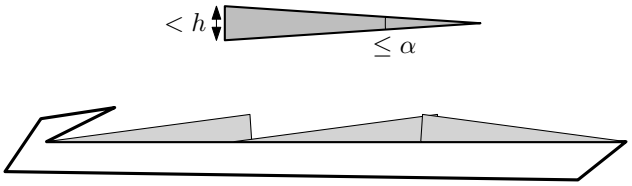


Figure 5: The needle is an equilateral triangle with apex at most α and a base edge of length at most h. The edges of equal length have length at most 1 so that the whole triangle can fit inside a tool.

nearest other features of P_1 are its two adjacent edges, the vertices closest to the edge, and the edges closest to its endpoints. We avoid cutting into the adjacent edges by placing the tip of the needle at the vertex when cutting near a vertex and we cannot cut out non-adjacent vertices and edges, because the base of the needle is at most h.

By making the cuts along the edges in the sets $\mathcal{E}_i$ in increasing order of i the needle has to only shrink, which is easily done by cutting it with any outer edge of the current polygon (all are guaranteed to be at least length h). Making the initial needle requires two snips, cutting each edge requires at most four snips and hence $O(n)$ snips in total, and reducing the needle length requires one snip per nonempty set $\mathcal{E}_i$ of which there are at most $O(n)$. Thus, in total the required number of snips is $O(n)$. □

5 Disconnected Model

Recall that in the disconnected model, we allow the tool to become disconnected, i.e., when a snip disconnects the tool, we keep both components.

In order to carve out a target shape P_1, we virtually fix a location of P_1 inside $\mathcal{T}_1$, pick a corner c of $\mathcal{T}_1$ (say, the lower right one) and consider the set of distances $d_1, \ldots, d_{n'}$ from each of the vertices in the fixed location of the target shape P_1 to c in decreasing order under the L_∞-metric. For simplicity assume that all distances are distinct, and thus $n' = n$ (this can be achieved with symbolic perturbation). We refer to the part of $\mathcal{T}_1$ not in P_1, i.e., $\mathcal{T}_1 \setminus P_1$, as the *free-space*. We will remove the free-space in n steps, where in each step i we remove the free-space from an L-shaped region Q_i that is the intersection of $\mathcal{T}_1$ and an annulus formed by removing the L_∞-ball of radius d_i from the L_∞-ball of radius d_{i-1}. We argue that in each step we will need $O(n)$ snips and resets, thus creating the target shape in $O(n^2)$ operations.

Lemma 4 *The free-space in region Q_i can be removed in $O(n)$ snips and reset operations provided that $\bigcup_{j \leq i} Q_j$ is a square in $\mathcal{T}_1$.*

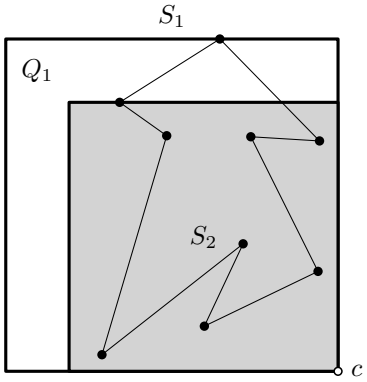


Figure 6: The squares S_1 and S_2 along with L-shaped region Q_1 and corner c.

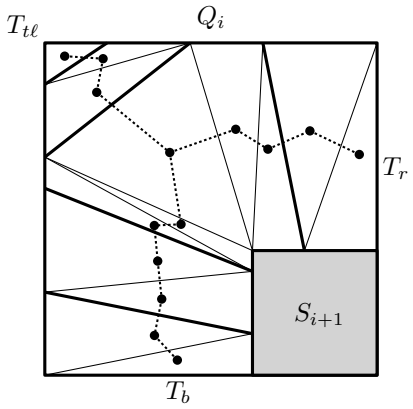


Figure 7: An L-shaped region Q_i, the edges of the target shape that cross it (thick edges) define F_i. We further triangulate each face (thin edges), and consider the corresponding dual graph (dotted edges).

Proof. Let S_i be the bounding square containing Q_i (see Figure 6) and let F_i be the set of faces created when removing the boundary edges of the target shape from Q_i. By definition all vertices of the target shape on Q_i must be on its inner or outer L-shaped boundary and all boundary segments must fully traverse Q_i, i.e., they cannot have an endpoint inside Q_i. It then follows that the set F_i of faces consists of $O(n)$ constant complexity pieces. Now triangulate all faces of F_i and let T_i denote the resulting set of triangles (Figure 7). Note that our aim is to remove some of the triangles of T_i. We will show that we can remove any triangle that fits in $S_i \setminus S_{i+1}$ with a constant number of cuts.

For simplicity in the exposition we first consider the case in which $\boldsymbol{S_{i+1}}$ **is large.** That is, the side length of S_{i+1} is at least half the side length of S_i. Consider a triangle $T \in F_i$ that needs to be removed. To create a cutting tool move $\mathcal{T}_2$ so that its only overlap with $\mathcal{T}_1$ is S_i. Let S_i' denote the area in $\mathcal{T}_2$ corresponding to S_i and let T' be the projection of T on $\mathcal{T}_2$. Our goal

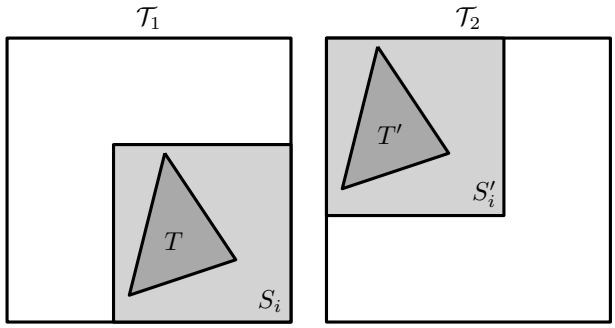


Figure 8: A triangle T in S_i is cut out of $\mathcal{T}_2$ at T'.

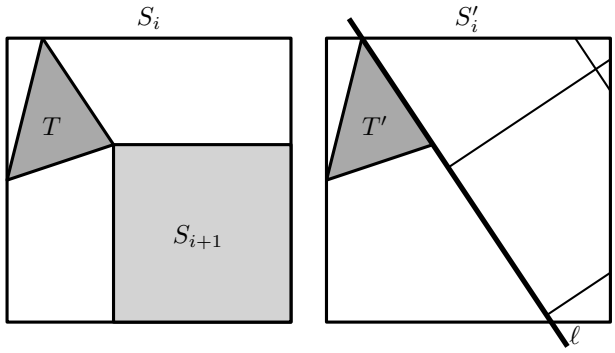


Figure 9: If S_{i+1} is large, we can use it to carve out any desired shape in $\mathcal{T}_2$ with $O(1)$ snips.

will be to remove $S_i' \backslash T'$ from $\mathcal{T}_2$ without affecting T'. Note that we can create a cut where only S_i' overlaps $\mathcal{T}_1$ in S_i, so the shape of $T_2 \backslash S_i'$ does not influence the cut (Figure 8). That means we do not have to cut it away and we do not need to worry about cutting part of it while creating a cutting tool within S_i'.

Consider the halfspace H defined by one of the bounding lines ℓ of T' that does not contain T'. We can remove $H \cap S_i'$ by rotating $\mathcal{T}_1$ so that one of the sides of $\mathcal{T}_1$ along which S_{i+1} is situated aligns with ℓ and repeatedly snip with S_{i+1} in a grid-pattern as shown in Figure 9. Because S_{i+1} is large compared to S_i' we can remove $H \cap S_i'$ in $O(1)$ snips. We then apply the same procedure for the other two halfspaces that should be removed to obtain the cutting tool for T. This means that, under the assumption that S_{i+1} is large, each triangle can be removed in $O(1)$ snips. Since there are $O(n)$ triangles in S_i, the linear bound holds.

It remains to consider the case in which $\boldsymbol{S_{i+1}}$ **is small.** (that is, the side length of S_{i+1} is less than half that of S_i, and potentially much smaller). Although the main idea is the same, we need to remove the triangles in order, and use portions of Q_i that are still solid to create the cutting tools.

Let G_i be the dual graph of T_i. This graph is a tree with at most three leaves. Two leaves correspond to the unique triangles T_b and T_r that share an edge with the lower and right boundary of Q_i respectively and the third exists only if the top-left corner of Q_i is contained in a single triangle $T_{t\ell}$, that is, there is at least one segment contained in Q_i that connects the top and left boundaries; see Figure 7. Finally, we change the coordinate system so that c is the origin, and S_i is a unit square (note that the vertices of this square are $(-1, 1)$, $(-1, 0)$, $(0, 1)$, and $c = (0, 0)$).

We process the triangles in the following order. We first process the *cross-triangles*, triangles with one endpoint on the left boundary and one on the top boundary, (if any exist) starting from $T_{t\ell}$ following G_i until we find a triangle that has degree three in G_i which we do not process yet. The remaining *fan-triangles* form a path in G_i which we process from T_b to T_r.

Cross-triangles. Recall that, by the way in which we nest regions Q_i, there cannot be vertices to the right or below S_i. In particular, cross-triangles have all three vertices in the top and left boundaries of Q_i. Hence, while we have some cross-triangle that has not been processed, the triangle of vertices $(-1, 0)$, $(0, 1)$ and c must be present in $\mathcal{T}_1$. This triangle has half the area of Q_i and can be used to create cutting pieces in the same way as when S_{i+1} is large. Thus, we conclude that any cross-triangle of Q_i can be removed from $\mathcal{T}_1$ with $O(1)$ cuts.

Fan-triangles. We now process the fan-triangles in the path from T_b to T_r in G_i. We treat this sequence in two phases. First consider the triangles that have at least one vertex on the left edge of S_i (that is, we process triangles up to and including the triangle that has degree three in G_i if it exists); out of these triangles, only the triangle of degree three can intersect the triangle of vertices $(0, 1)$, $(0, 3/4)$, and $(-3/4, 3/4)$. This triangle has $1/32$ of the total area of S_i, and as before we can use it as cutting tool to create any desired triangle with $O(1)$ snips.

The remaining triangles have their vertices in the upper edge of S_i and on the upper or left edge of S_{i+1}. In this case we must be more careful as we cannot guarantee the existence of a large square in $\mathcal{T}_1$. However, we do not have to clear the entire space S_i' any longer. Instead it suffices to clear a much smaller area.

Let T denote the next triangle to be removed and let B denote the bounding box of T and c (see Figure 10). As before consider moving $\mathcal{T}_2$ so that the only overlap with $\mathcal{T}_1$ is B, let B' denote this area in $\mathcal{T}_2$ and T' the projection of T onto B'. To create a cutting tool we need only remove the area $B' \backslash T'$.

As before, we look for a region in $\mathcal{T}_1$ that has roughly the area of T to use for carving the desired shape in $\mathcal{T}_2$. Let w be the width of B. Also, let h' the height of S_{i+1}. Note that the height of B is 1, and since S_{i+1} is small, we have $h' < 1/2$. By construction of the bounding box,

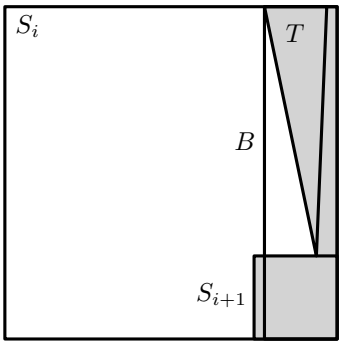


Figure 10: The solid areas (grey) and bounding box B when cutting fan-triangles with no vertices on the left boundary of S_i.

one of the vertices of T will have x coordinate equal to $-w$; let q denote this vertex. The y coordinate y_q of q is either 1 or h' as it must be on the upper edge of S_i or on the upper boundary of S_{i+1}—if T has vertices on left boundary of S_{i+1}, then there is a vertex on the upper boundary of S_i with lower x coordinate. Now consider with vertices $(0,1),(0,h'),q$. This triangle has height at least $1-h' > 1/2$ and width w, and thus its area is at least $1/4$ of the area of B. As in the previous cases, we use this triangle to create a cutting tool from $\mathcal{T}_2$ to remove triangle T from $\mathcal{T}_1$.

Thus, it follows that all free-space triangles can be removed with a cutting tool that is constructed from $\mathcal{T}_2$ in $O(1)$ snips and reset operations, hence we can clear Q_i of free-space in total $O(n)$ operations. □

Because there are at most n distinct distances, we repeat this procedure at most n times, giving us the desired result.

Theorem 5 *We can cut one of the tools into any target polygonal domain P_1 of n vertices using $O(n^2)$ snips and reset operations in the disconnected model.*

6 Open Problems

For cutting one tool into a desired polygonal shape, our results are tight in the connected model ($\Theta(n)$), but the disconnected model (as implemented by the Snipperclips game) still has a gap between $\Omega(n)$ and $O(n^2)$. What is the optimal worst-case number of cuts as a function of n? What about the algorithmic question of cutting out a given shape with the fewest possible cuts for that shape (instead of the worst case)? Is this problem NP-hard, and does it have a constant-factor approximation algorithm?

For cutting two tools (or more tools) simultaneously into desired polygonal shapes, the main open problem is to characterize when this is possible. Is the decision problem NP-hard? How does the problem change if we allow the undo/redo operation described in Section 2?

It would also be interesting to consider the initial shape implemented in the Snipperclips game (instead of the unit squares we used for simplicity), namely, a unit square adjoined with half a unit-diameter disk. This initial shape opens up the possibility of making curved target shapes bounded by line segments and circular arcs of matching curvature. Can all such shapes be made, and if so, by how many cuts?

Acknowledgments

This work was initiated at the 32nd Bellairs Winter Workshop on Computational Geometry held January 2017 in Holetown, Barbados. We thank the other participants of that workshop for providing a fun and stimulating research environment. E. Demaine also thanks Hugo Akitaya, Martin Demaine, Adam Hesterberg, Jason Ku, and Jayson Lynch for helpful discussions about (and games of) Snipperclips. We also thank Hugo Akitaya for taking the screenshots in Figure 1.

References

[1] Glowforge — the 3D laser printer. https://glowforge.com/, 2015.

[2] D. A. Applegate, G. Calinescu, D. S. Johnson, H. Karloff, K. Ligett, and J. Wang. Compressing rectilinear pictures and minimizing access control lists. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1066–1075, New Orleans, Louisiana, 2007.

[3] E. D. Demaine, M. L. Demaine, and C. S. Kaplan. Polygons cuttable by a circular saw. *Computational Geometry: Theory and Applications*, 20(1–2):69–84, October 2001. CCCG 2000.

[4] GDC. European innovative games showcase: Friendshapes. YouTube video, 2015. https://youtu.be/WJGooKIoy1Q.

[5] J. W. Jaromczyk and M. aw Kowaluk. The face-wise continuity in hot wire cutting of polyhedral sets. In *Proceedings of the 16th European Workshop on Computational Geometry*, pages 93–97, Eilat, Israel, March 2000.

[6] M. H. Overmars and E. Welzl. The complexity of cutting paper. In *Proceedings of the 1st Annual ACM Symposium on Computational Geometry*, pages 316–321, Baltimore, Maryland, June 1985.

[7] J. Pach and G. Tardos. Cutting glass. *Discrete & Computational Geometry*, 24:481–495, 2000.

[8] Wikipedia. Snipperclips. https://en.wikipedia.org/wiki/Snipperclips, 2017.

Rep-cubes: Unfolding and Dissection of Cubes

Dawei Xu* Takashi Horiyama† Ryuhei Uehara*

Abstract

Last year, a new notion of *rep-cube* was proposed. A rep-cube is a polyomino that is a net of a cube, and it can be divided into some polyominoes such that each of them can be folded to a cube. This notion was inspired by the notions of *polyomino* and *rep-tile*, which were introduced by Solomon W. Golomb. It was proved that there are infinitely many distinct rep-cubes. In this paper, we investigate this new notion and obtain three new results. First, we prove that there does not exist a regular rep-cube of order 3, which solves an open question proposed in the paper. Next, we enumerate all regular rep-cubes of order 2 and 4. For example, there are 33 rep-cubes of order 2; that is, there are 33 dodecominoes that can fold to a cube of size $\sqrt{2} \times \sqrt{2} \times \sqrt{2}$ and each of them can be divided into two nets of unit cube. Similarly, there are 7185 rep-cubes of order 4. Lastly, we focus on pythagorean triples that consist of three positive integers (a, b, c) with $a^2 + b^2 = c^2$. For each of these triples, we can consider a rep-cube problem that asks whether a net of a cube of size $c \times c \times c$ can be divided into two nets of two cubes of size $a \times a \times a$ and $b \times b \times b$. We give a partial answer to this natural open question by dividing into more than two pieces. For any given pythagorean triple (a, b, c), we construct *five* polyominoes that form a net of a cube of size $c \times c \times c$ and two nets of two cubes of size $a \times a \times a$ and $b \times b \times b$.

1 Introduction

A *polyomino* is a "simply connected" set of unit squares introduced by Solomon W. Golomb in 1954 [7]. Since then, polyominoes have been playing an important role in recreational mathematics (see, e.g., [5]). In 1962, Golomb also proposed an interesting notion called "*rep-tile*": a polygon is a rep-tile of order k if it can be divided into k replicas congruent to one another and similar to the original (see [6, Chap 19]).

From these notions, Abel et al. proposed a new notion [1]; a polyomino is said to be a *rep-cube* of order k if it is a net of a cube (or, it can fold to a cube), and it can be divided into k polyominoes such that each of them can fold to a cube. If all k polyominoes have the same size, we call the original polyomino a *regular* rep-cube of order k. We note that crease lines are not necessarily along the edges of the polyomino. For example, a regular rep-cube of order 2 folds to a cube by folding along the diagonals of unit squares; see Figure 1.

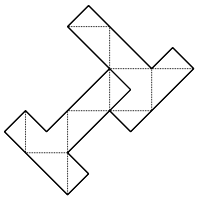


Figure 1: A regular rep-cube of order 2 [1]; each T shape can fold to a cube, and this shape itself can fold to a cube of size $\sqrt{2} \times \sqrt{2} \times \sqrt{2}$ by folding along the dotted lines.

In [1], Abel et al. propose regular rep-cubes of order k for each $k = 2, 4, 5, 8, 9, 36, 50, 64$, and also $k = 36gk'^2$ for any positive integer k' and an integer g in $\{2, 4, 5, 8, 9, 36, 50, 64\}$. In other words, there are infinitely many k that allow regular rep-cube of order k. On the other hand, they left an open problem that asks if there is a rep-cube of order 3. In this paper, we first answer to this question. There are no regular rep-cube of order 3.

Next we enumerate all possible regular rep-cubes of order k for small k. We mention that the following problem is not so easy to solve efficiently; for a given polygon P, determine if P can fold to a cube or not. Recently, Horiyama and Mizunashi developed an efficient algorithm that solves this problem for a given orthogonal polygon, which runs in $O((n + m) \log n)$ time, where n is the number of vertices in P, and m is the maximum number of line segments that appears on a crease line [8]. We remark that the parameter m is hidden and can be huge comparing to n. In our case, P is a polyomino, and this hidden parameter is linear to the number of unit squares in P, and hence our algorithm is simpler.

Finally, we investigate non-regular rep-cube. In [1], Abel et al. also asked if there exists a rep-cube of area 150 that is a net of a cube of size $5 \times 5 \times 5$ and it can be divided into two nets of cubes of size $3 \times 3 \times 3$ and $4 \times 4 \times 4$. This idea comes from a pythagorean triple $(3, 4, 5)$ with $3^2 + 4^2 = 5^2$. We give a partial answer to this question by dividing into more pieces than 2. We give a general way for any pythagorean triple (a, b, c) with $a < b < c$ to obtain five piece solution. That is, for any given pythagorean triple (a, b, c) with $a < b < c$, we construct a polyomino that is a net of a cube of $c \times c \times c$, and it can be divided into 5 pieces such that one of 5 pieces can fold to a cube of $a \times a \times a$, and gluing the

*School of Information Science, Japan Advanced Institute of Science and Technology, Japan. {xudawei,uehara}@jaist.ac.jp

†Graduate School of Science and Engineering, Saitama University, Japan. horiyama@al.ics.saitama-u.ac.jp

remaining 4 pieces, we can obtain a net of a cube of $b \times b \times b$.

Due to lack of space, some proofs are omitted.

2 Nonexistence of regular rep-cubes

The main theorem in this section is the following.

Theorem 1 *There does not exist a regular rep-cube of order* 3.

We first show two lemmas (proofs are omitted):

Lemma 2 *Let Q be a cube and P any development*[1] *of Q. Then P is concave.*

Let P be a polyomino (not necessarily hexomino) that can fold to a cube Q. Then, by Lemma 2, P has no "rolling belt" (see [4] for further details). This fact implies that, when we fold P to Q, each vertex on Q should appear at either the grid point of P or the middle point of a unit edge in P. For these vertices of Q, we state stronger property:

Lemma 3 *Let P be a polyomino that can fold to a cube Q. Let ℓ be the length of an edge of Q. (That is, P is a $6\ell^2$-omino.) Then P can be placed on a grid of size ℓ so that every vertex of Q on P is on a grid point. I.e., not only all vertices on Q appear on the boundary of P, but also they are also aligned on the grid points of size ℓ.*

Now we turn to the proof of Theorem 1. We assume that there exists a regular rep-cube of order 3, and derive contradictions. That is, we assume that there is a polyomino $\hat{P}$ such that $\hat{P}$ can be divided into three polyominoes P_1, P_2, P_3 of the same size, and each of $\hat{P}, P_1, P_2, P_3$ can fold to a cube of certain size. Let $\hat{Q}$ and Q_i denote the cubes folded from $\hat{P}$ and P_i, respectively. We suppose that the length of an edge of Q_i is ℓ. That is, P_i is a $6\ell^2$-omino, and $\hat{P}$ is a $18\ell^2$-omino. We remark that ℓ is not necessarily an integer, but $6\ell^2$ is.

Now we consider the polyomino P_1; that is a $6\ell^2$-omino, and folds to the cube Q_1 of size $\ell \times \ell \times \ell$. Then, by Lemma 3, P_1 can be on the grid of size ℓ so that every vertex of Q_1 is on the grid. We take any two vertices v_1 and v_2 of Q_1 of distance ℓ on the grid. Then the vector $\overrightarrow{v_1 v_2}$ can be represented by (a, b) for some nonnegative integers a and b. That is, $a^2+b^2 = \ell^2$ for some integers a and b. (The same idea can be found in [4, Ch. 5.1.1] and [3].) We can apply the same argument to $\hat{P}$ and $\hat{Q}$, and hence there are some nonnegative integers $\hat{a}$ and $\hat{b}$ such that $\hat{a}^2+\hat{b}^2 = 3\ell^2$. Thus we obtain $\hat{a}^2+\hat{b}^2 = 3(a^2+b^2)$.

Therefore, it is sufficient to show that there are no such integers. To derive a contradiction, we assume that we have $\hat{a}^2 + \hat{b}^2 = 3(a^2 + b^2)$, and they are the minimum integers with respect to the value of $\hat{a}^2 + \hat{b}^2$.

Now, for an integer i, $(3i \pm 1)^2 = 9i^2 \pm 6i + 1$. Therefore, a square number x is either $x = 3x'$ or $3x' + 1$ for some integer x'. Since $\hat{a}^2 + \hat{b}^2 = 3(a^2 + b^2)$ is a multiple of 3, both of $\hat{a}$ and $\hat{b}$ are multiples of 3, say $\hat{a} = 3\hat{a'}$ and $\hat{b} = 3\hat{b'}$. Then, we have $(3\hat{a'})^2 + (3\hat{b'})^2 = 9(\hat{a'}^2 + \hat{b'}^2) = 3(a^2 + b^2)$. Thus we obtain $a^2 + b^2 = 3(\hat{a'}^2 + \hat{b'}^2)$. This contradicts the minimality of the value of $\hat{a}^2+\hat{b}^2$. Therefore, we have no such integers $a, b, \hat{a}, \hat{b}$. This completes the proof of Theorem 1. □

3 Enumeration of regular rep-cubes

In this section, we describe the exhaustive search algorithm for generating all regular rep-cubes of order k (for $k = 2$ and $k = 4$).

Algorithm 1 gives the outline of this algorithm. It works as follows: Let S_i be the set of all $(6 \times i)$-ominoes such that (1) it is composed by i nets of a unit cube, (2) it can cover a part of a cube of size $\sqrt{k} \times \sqrt{k} \times \sqrt{k}$. In the term of search of development, each element in S_i is called a *partial development* of a cube of size $\sqrt{k} \times \sqrt{k} \times \sqrt{k}$ [10]. That is, S_1 is the set of all nets of a unit cube, which consists of 11 hexominoes, and each set S_i with $i > 1$ is a subset of $(6 \times i)$-ominoes that can be computed from S_{i-1}. Let P_i be any polyomino in S_i, e.g., P_1 is one of the 11 hexominoes in S_1.

In Procedure CheckCover, the algorithm checks if P_i can cover the cube of size $\sqrt{k}\times\sqrt{k}\times\sqrt{k}$ without overlap. The details will be described later. Our final goal is to obtain the set S_k that contains all regular rep-cubes of order k from the set S_1.

Algorithm 1: Outline of the exhaustive search algorithm.

```
Input  : Integer k of the order for the rep-cube;
Output: All rep-cubes in S_k;
for i = 2 to k do
    foreach partial development P_{i-1} in S_{i-1} do
        foreach development P_1 in S_1 do
            attach P_1 to P_{i-1} at each possible
            adjacency square on the boundary of
            P_{i-1} to obtain a new polyomino P_i;
            if CheckCover(P_i)==1 then
                store P_i into S_i; // P_i is a partial
                    // development of the box of
                    // size √k × √k × √k
return S_k;
```

The algorithm works in a loop as follows. It picks up a polyomino P_{i-1} in S_{i-1} and a hexomino P_1 in

[1] We use "net" that has no overlap when it is spread out. When we use "development," overlap is not yet considered.

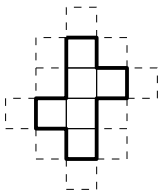


Figure 2: All possible adjacency empty squares on the boundary of a net.

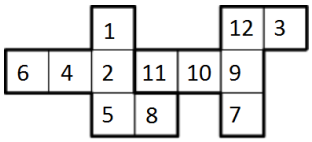


Figure 3: Every square of P is marked with a unique number according to the adjacency list.

S_1, and attaches P_1 by edge-to-edge gluing to P_{i-1} at each possible adjacency empty square on the boundary of P_{i-1} as shown in Figure 2. We note that we have to consider not only the overlap, but also the flip of P_1 if P_1 is not congruent to its mirror image. By this step, it generates a new polyomino P_i, which is a component of i nets of a unit cube. This P_i will be examined whether it can fold to a part of the cube of size $\sqrt{k}\times\sqrt{k}\times\sqrt{k}$ or not. This loop terminates at $i = k$, when the polyomino P_k can fold to a complete cube.

As mentioned in Introduction, we find that the folding lines of the cube of $\sqrt{k}\times\sqrt{k}\times\sqrt{k}$ are not along the edges of unit squares. Since the rep-cubes of order 2 and 4 have different folding ways, we need a universal method to check whether a polyomino is a partial development or not. In [10], the authors proposed an algorithm that checks the positional relationships of unit squares on the polyomino. Consider any polyhedron, e.g., a cube Q, folded from a polyomino P. Then we can obtain an adjacency relationship of unit squares in P on Q. That is, two unit squares share an edge on P only if they share it on Q. Thus any development of Q keeps a part of the same adjacency relationship. Therefore, we can decide if a polyomino P can fold to a cube Q by checking the positional relationship of the unit squares in Procedure CheckCover.

We consider the first development in Figure 5 as an example P (Figure 3). We first mark a unit square with the number 1 as the start point. Then we mark all of its neighbor-squares a number according to the adjacency list of cube of size $\sqrt{2}\times\sqrt{2}\times\sqrt{2}$ as in Table 1 and Figure 4 in all four directions. (For example, the square 1 is surrounded by 12(above), 11(right), 2(below), and 3(left) from the viewpoint of the square 1.) This step is extended to its farther neighbors until every square of P is marked with a number. After this step, if every square in connected P is marked with its unique number, P can wrap the cube of size $\sqrt{k}\times\sqrt{k}\times\sqrt{k}$ with consistency. On the other hand, if (1) one square is marked with different numbers by its neighbors or (2) two or more squares are marked with the same number, then an overlap occurs in this folding way of P. We check all possible start points and directions for each P.

Table 1: Adjacency list of cube of size $\sqrt{2}\times\sqrt{2}\times\sqrt{2}$.

Square ID	Up	Right	Down	Left
1	12	11	2	3
2	1	11	5	4
3	12	1	4	6
4	3	2	5	6
5	4	2	8	7
6	3	4	7	9
7	6	5	8	9
8	7	5	11	10
9	6	7	10	12
10	9	8	11	12
11	10	8	2	1
12	9	10	1	3

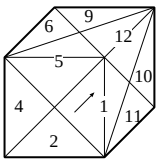


Figure 4: Adjacency relationship of the squares on the cube of size $\sqrt{2}\times\sqrt{2}\times\sqrt{2}$.

Procedure CheckCover(P_i)

Input : Polyomino P_i in S_i;
Output: Whether P_i can wrap up the cube of size $\sqrt{k}\times\sqrt{k}\times\sqrt{k}$ or not;

```
1 foreach square in P_i do
2     mark the square 1 as the start point
3     foreach marked square in P_i do
4         mark its unmarked adjacent squares as the
          adjacency matrix of the cube of size
          √k × √k × √k;
5         if any square of P_i gets marked by two or
          more different numbers then
6             break; // P_i has overlap
7     if every square of P_i is marked by a unique
      number then
8         return 1; // P_i can wrap up the cube
9 return 0;
```

As a result of finding the rep-cube of order 2, by putting two developments of a cube aside, there are 2424 distinct dodecominoes. Among them, there are 33 regular rep-cubes of order 2 that can fold to a cube of size $\sqrt{2}\times\sqrt{2}\times\sqrt{2}$ and each of them can be divided into two nets of a unit cube. As shown in Figures 5 and 6, we can observe that 17 rep-cubes out of 33 consist of two nets of the same shape. We call them *uniform* rep-cubes. Precisely, we say a regular rep-cube of order k is *uniform* if its all k nets are the same shape.

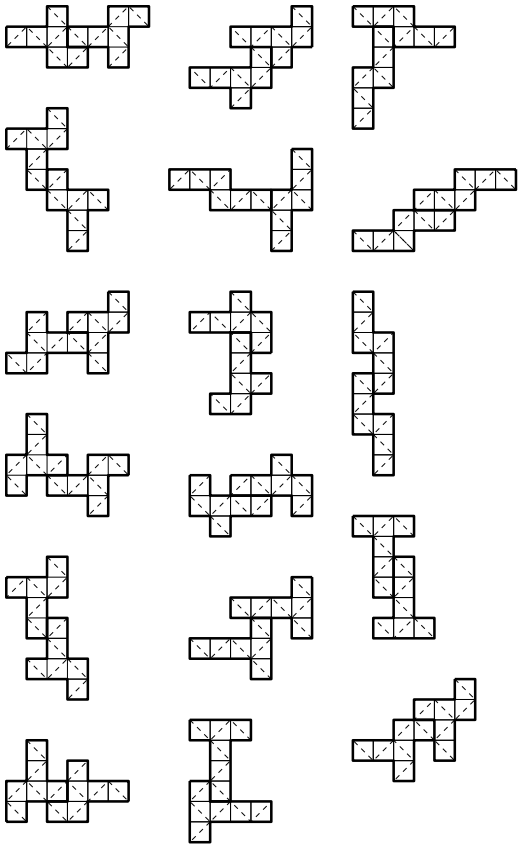


Figure 5: All 17 *uniform* rep-cubes of order 2.

For the case of finding the regular rep-cube of order 4, we also implement this algorithm. As a result, we got the amount of partial developments of i pieces as in Table 2, which means there are 7185 regular rep-cubes of order 4. Among them, we also find all uniform rep-cubes of order 4, which are 158 in total. One example of these uniform rep-cubes is shown in Figure 7. Out of 158, 98 of these uniform rep-cubes are made of pieces in shape (b) shown in Figure 8.

Table 2: The number of partial developments of regular rep-cubes of order 4.

Set of partial developments	S_1	S_2	S_3	S_4
Number of developments	11	2345	114852	7185

In Figure 1 of [1], they gave three uniform rep-cubes of order 2 (Figure 1), 4, and 9. On the other hand, in [1], they also show a regular rep-cube of order 50 that contains all kinds of 11 nets of a unit cube. It may worth focusing on these special cases for a larger k.

In the analysis of the results, we found two different patterns of shapes that can make the same rep-cube. As shown in Figure 9, except for the difference in composition, these two rep-cubes have the same contour, the same surface area and the same folding way. Finding this kind of rep-cube can be a interesting topic in the future research.

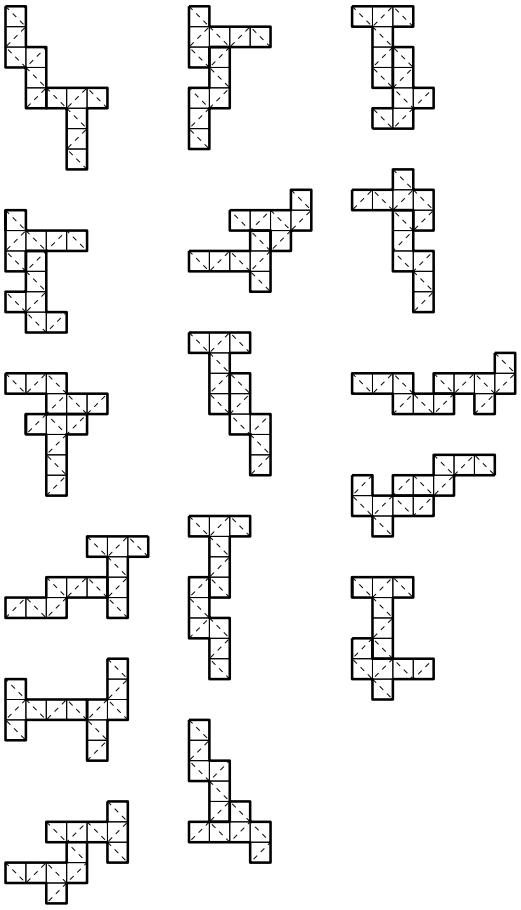


Figure 6: All regular rep-cubes of order 2 that are not *uniform*.

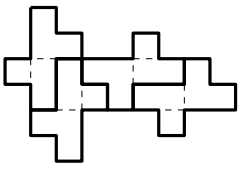


Figure 7: An example of *uniform* rep-cubes of order 4.

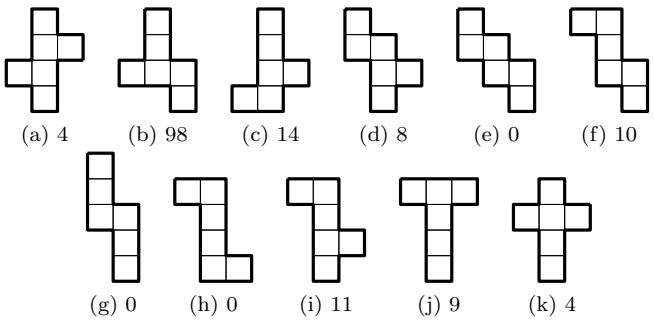


Figure 8: List of the amount of uniform rep-cubes of order 4 made by each of 11 shapes.

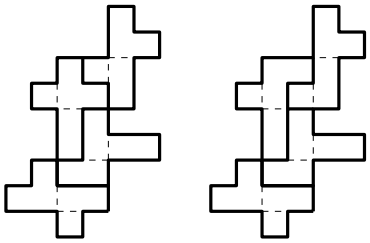


Figure 9: Two different patterns make the same rep-cube of order 4.

4 Rep-cubes based on pythagorean triples

A *pythagorean triple* is a 3-tuple of positive integers that satisfies $a^2 + b^2 = c^2$. In [1], Abel et al. propose an interesting open question related to the pythagorean triple. That is, the question asks whether there is a rep-cube of order 2 of area c^2 such that (1) it folds to a $c \times c \times c$ cube, and (2) it can be divided into two polyominoes so that they fold to a $a \times a \times a$ cube and another $b \times b \times b$ cube. The most famous one is $(3, 4, 5)$ with $3^2 + 4^2 = 5^2$. We note that for any pythagorean triple (a, b, c), for any positive integer k, (ak, bk, ck) is also a pythagorean triple. However, we only consider pythagorean triples with $GCD(a, b, c) = 1$. Then, it is known that a triple (a, b, c) with $GCD(a, b, c) = 1$ is a pythagorean triple if and only if there are two positive integers m, n such that m, n are relatively prime, $0 < n < m$, $m - n$ is odd, and we can obtain a pythagorean triple as $(m^2 - n^2, 2mn, m^2 + n^2)$ for these n and m.

It is trivial that when we divide any net of a $c \times c \times c$ cube into $6c^2$ unit squares, we can make two cubes of size $a \times a \times a$ and $b \times b \times b$. Therefore, we can consider this open problem as an optimization problem to minimize the number of polyominoes that can form both of a net of $c \times c \times c$ cube, and two nets of two cubes of size $a \times a \times a$ and $b \times b \times b$. In this section, we give the following theorem:

Theorem 4 *Let (a, b, c) be any pythagorean triple with $a < b < c$. Then we can construct a set $S(a, b, c)$ of five polyominoes such that (1) the polyominoes can form a net of $c \times c \times c$ cube, and (2) they can form two nets of two cubes of size $a \times a \times a$ and $b \times b \times b$.*

We here show an example in Figure 10 to get the idea. When we choose a pythagorean triple $(3, 4, 5)$, the polyomino in Figure 10(a) folds to a $3 \times 3 \times 3$ cube, and the polyomino in Figure 10(b) folds to a $4 \times 4 \times 4$ cube. It is less intuitive, however, the reader can obtain a $5 \times 5 \times 5$ cube from the polyomino in Figure 10(c) by folding along the dotted lines. Here we give a general construction for any pythagorean triple.

Proof. We first give a brief idea of the construction in Figure 11. The first step is that we open two small cubes

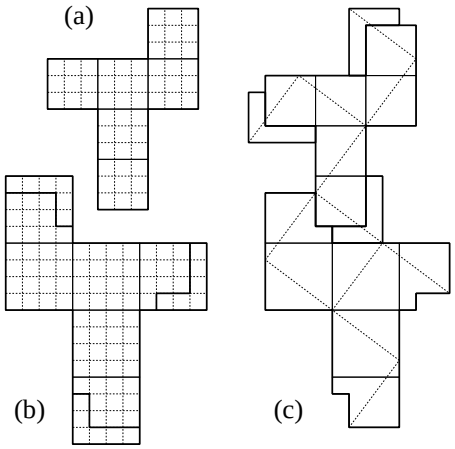


Figure 10: The set $S(3, 4, 5)$ of five polyominoes that folds to (a,b) two cubes of size $3 \times 3 \times 3$ and $4 \times 4 \times 4$, and (c) one cube of size $5 \times 5 \times 5$.

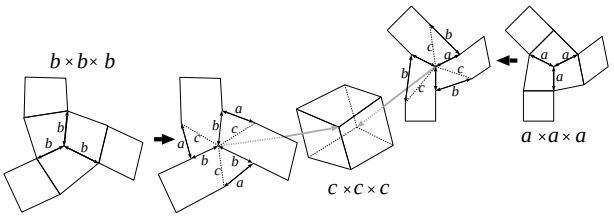


Figure 11: Brief idea of the construction.

of size $a \times a \times a$ and $b \times b \times b$ at their any vertices. We cut along the three lines from the vertex until we have a kind of a triangular-pyramid-like shape; each rectangular face consists of two squares, and these three rectangles are glued like in wind-wheel shape. Then we regard these two triangular pyramids as cone-like shapes, and attach each of apexes to the two opposite vertices of the big cube of size $c \times c \times c$. That is, they are glued to two endpoints of a diagonal of the big cube.

The main trick is that the grids of two small cubes are not aligned to the grid of the big cube; we twist two cones so that their edges (or grid lines) make two edges of pythagorean triangle of length a and b. As a result, three vertices of the big cube are on the boundary of the wind-wheel shape made from the cube of size $b \times b \times b$, and the other three vertices of the big cube are on the boundary of the other wind-wheel shape made from the cube of size $a \times a \times a$. Then, we have two cases depending on the size of these two small cubes.

The first case is that $a < b < 2a$. For example, the most famous pythagorean triple $(3, 4, 5)$ (for $m = 2, n = 1$) satisfies this condition. In this case, the situation is illustrated on the net of the big cube in Figure 12. The outline is the net of the big cube, and three vertices labeled by p form a vertex of the big cube, and the apex of the cone made by the small cube of size $a \times a \times a$

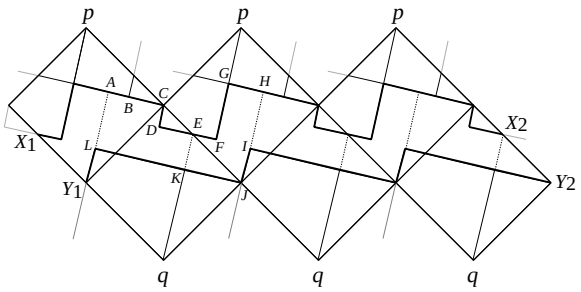


Figure 12: View on the net of the big cube of size $c \times c \times c$.

is attached at the vertex p. In the figure, all squares of this small cube are already depicted, and they are aligned along the zig-zag line joining two points X_1 and X_2. On the other side, three vertices labeled by q form the opposite vertex of the big cube, where the apex of the cone made by the other small $b \times b \times b$ cube is attached to. In the figure, three squares of size $b \times b$ are depicted along the zig-zag line joining two points Y_1 and Y_2. Therefore, out task is to form three more squares of size $b \times b$ by the belt between lines X_1X_2 and Y_1Y_2 with few dissections.

We first extend the grid lines of squares of size $b \times b$ as shown in Figure 12. Then the belt is divided into six parts; three of them are congruent to the hexagon $ACDEKL$, and three of them are congruent to the hexagon $EFGHJK$. Then our claim is that gluing the line $GFEK$ to $ACDE$, we obtain a square $HJKL$ of size $b \times b$. If it works, it is easy to see the theorem holds.

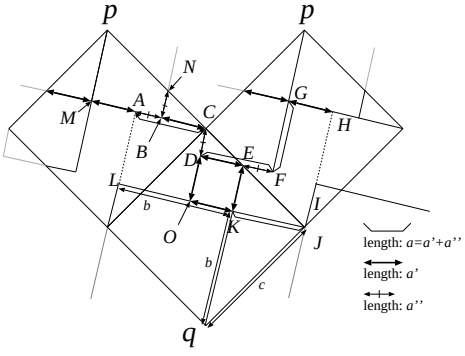


Figure 13: Detailed lengths of polyominoes.

Now we focus on this part (see Figure 13). We first observe that two triangles pMC and JKq are congruent to the right triangle xyz with $|xy| = a$, $|yz| = b$, and $|zx| = c$. We now let $a' = b - a$ and $a'' = a - a' = 2a - b$. Since $|MC| = b$ and $|MB| = a$, we have $|BC| = b - a = a'$. The edges BC and CD make an edge of an $a \times a$ square when it folds to a small cube, hence $|CD| = a - a' = a''$. Since triangle NBC is congruent to CDE, $|DE| = a'$, and hence $|EF| = a''$. Since the triangle COJ is congruent to the right triangle xyz, we obtain $|CO| = a$, $|DO| = a'$, and hence $|EK| = a'$. Since $|EF| = a''$ and $|KJ| = a$, we have $|GH| = |MA| = a'$. Thus $|AC| = b - a' = a$. Therefore, the zig-zag line $ACDE$ can be glued to the zig-zag line $GFEK$ since all lengths are matched and they are orthogonal. By the fact $|LK| = b$, the resulting rectangle $LKJH$ should be square by the area constraint for the belt.

The second case $2a < b$ is omitted, however, a similar idea works. In both cases, we have the theorem. □

By Theorem 4, we have the following immediately.

Corollary 5 *There are infinitely many sets of five polyominoes such that (1) the polyominoes can form a net of $c \times c \times c$ cube, and (2) they can form two nets of two cubes of size $a \times a \times a$ and $b \times b \times b$.*

We remark that it is open that if there are infinitely many distinct non-regular rep-cubes.

References

[1] Z. Abel, B. Ballinger, E. D. Demaine, M. L. Demaine, J. Erickson, A. Hesterberg, H. Ito, I. Kostitsyna, J. Lynch, and R. Uehara. Unfolding and Dissection of Multiple Cubes. In *JCDCG*^{3}, 2016.

[2] J. Akiyama. Tile-Makers and Semi-Tile-Makers. *American Mathematical Monthly*, Vol. 114, pp. 602–609, 2007.

[3] Y. Araki, T. Horiyama, and R. Uehara. Common Unfolding of Regular Tetrahedron and Johnson-Zalgaller Solid. *J. of Graph Algorithms and Applications*, Vol. 20, No. 1, pp. 101–114, 2016.

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

[5] M. Gardner. *Hexaflexagons, Probalitity Paradoxes, and the Tower of Hanoi*. Cambridge, 2008.

[6] M. Gardner. *Knots and Borromean Rings, Rep-Tiles, and Eight Queens*. Cambridge, 2014.

[7] S. W. Golomb. *Polyominoes: Puzzles, Patterns, Problems, and Packings*. Princeton Univ., 1996.

[8] T. Horiyama and K. Mizunashi. Folding Orthogonal Polygons into Rectangular Boxes. *19th Korea-Japan Joint Workshop on Algorithms and Computation*, 2016.

[9] J. Mitani and R. Uehara. Polygons Folding to Plural Incongruent Orthogonal Boxes, *CCCG 2008*, pp. 39-42, 2008.

[10] D. Xu, T. Horiyama, T. Shirakawa, and R. Uehara. Common Developments of Three Incongruent Boxes of Area 30. *COMPUTATIONAL GEOMETRY: Theory and Applications*, Vol. 64, pp. 1–17, 2017.

On the Shortest Separating Cycle

Adrian Dumitrescu*

Abstract

According to a result of Arkin et al. (2016), given n point pairs in the plane, there exists a simple polygonal cycle that separates the two points in each pair to different sides; moreover, a $O(\sqrt{n})$-factor approximation with respect to the minimum length can be computed in polynomial time. Here we extend the problem to geometric hypergraphs, and obtain the following characterization of feasibility. Given a geometric hypergraph on points in the plane with hyperedges of size at least 2, there exists a simple polygonal cycle that separates each hyperedge if and only if the hypergraph is 2-colorable.

We extend the $O(\sqrt{n})$-factor approximation in the length measure as follows: Given a geometric graph $G = (V, E)$, a separating cycle (if it exists) can be computed in $O(m + n \log n)$ time, where $|V| = n$, $|E| = m$. Moreover, a $O(\sqrt{n})$-approximation of the shortest separating cycle can be found in polynomial time. Given a geometric graph $G = (V, E)$ in $\mathbb{R}^3$, a separating polyhedron (if it exists) can be found in $O(m + n \log n)$ time, where $|V| = n$, $|E| = m$. Moreover, a $O(n^{2/3})$-approximation of a separating polyhedron of minimum perimeter can be found in polynomial time.

Keywords: Minimum separating cycle, traveling salesman problem, geometric hypergraph, 2-colorability.

1 Introduction

Given a set of n pairs of points in the plane with no common elements, $\{(p_i, q_i) \mid i = 1, \ldots, n\}$, a Shortest Separating Cycle is a plane cycle (a closed curve, a.k.a. *tour*) of minimum length that contains inside exactly one point from each of the n pairs. The problem was introduced by Arkin et al. [3] motivated by applications in data storage and retrieval in a distributed sensor network. They gave a $O(\sqrt{n})$-factor approximation for the general case and better approximations for some special cases. On the other hand, using a reduction from Vertex Cover, they showed that the problem is hard to approximate for a factor of 1.36 unless P = NP, and is hard to approximate for a factor of 2 assuming the Unique Games Conjecture; see, e.g., [21, Ch. 16] for technical background.

*Department of Computer Science, University of Wisconsin–Milwaukee, USA. Email: dumitres@uwm.edu

The assumption that no point appears more than once, i.e., $|\{p_1, \ldots, p_n\} \cup \{q_1, \ldots, q_n\}| = 2n$, is sometimes necessary for the existence of a separating cycle; i.e., there are instances of sets of pairs with common elements and no separating cycle; see for instance Fig. 1 (left). For convenience, points on the boundary of the cycle are considered inside; it is easy to see that requiring points to lie strictly in the interior or also on the boundary are equivalent variants in regards to the existence of a separating cycle. Moreover, the equivalence is almost preserved in the length measure: given any positive $\varepsilon > 0$, and a separating cycle C for n pairs, enclosing $P = \{p_1, \ldots, p_n\}$ (say, after relabeling each pair, if needed), with some of the points of P on its boundary, a separating cycle of length at most $(1 + \varepsilon)\,\mathrm{len}(C)$ can be constructed, having all points of P in its interior.

In this paper we study the extension of the concept of separating cycle to arbitrary graphs and hypergraphs, and to higher dimensions; in the original version introduced by Arkin et al. [3], the input graph is a matching, i.e., it consists of n edges with no common endpoints. Two instances with 8 and respectively 3 point pairs that do not admit separating cycles are illustrated in Fig. 1.

Interestingly enough, even in instances with pairs where a solution exists, one cannot use the algorithm from [3]. Their algorithm (in [3, Subsec. 3.5]) starts by computing a minimum-size square Q containing at least one point from each pair, and then computes a constant-factor approximation of a shortest cycle (tour) of the points contained in Q, in the form of a simple polygon. In the end, this tour is refined to a separating cycle of the given set of point pairs with only a small increase in length. Here we note that there exist instances (like that in Fig. 1) for which there is no separating cycle confined to Q; moreover, the length of a shortest separating cycle can be arbitrarily larger than any function of $\mathrm{diam}(Q)$ and n, and so a new approach is needed for the general version with arbitrary input graphs, or its extension to hypergraphs; i.e., the current $O(\sqrt{n})$-factor approximation does not carry through to these settings.

We first show that a planar geometric graph $G = (V, E)$ admits a separating cycle (for all its edge-pairs) if and only if it is bipartite. This result can be extended to hypergraphs in $\mathbb{R}^d$. Given a geometric hypergraph on points in $\mathbb{R}^d$ with no singleton edges, there exists a simple polyhedron that separates each hyperedge if and only if the hypergraph is 2-colorable.

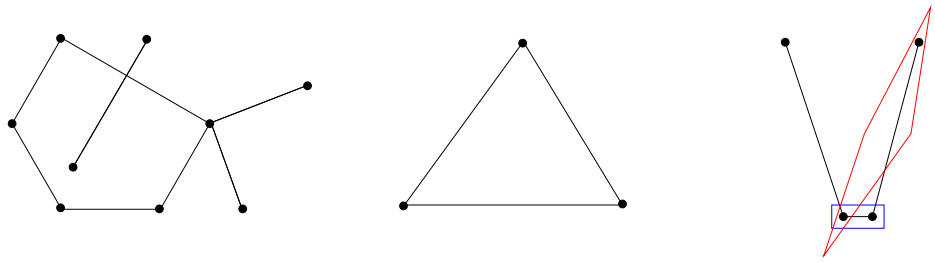


Figure 1: Left and center: instances with no separating cycle. Right: instance where the minimum axis-parallel square (or rectangle) that contains at least one point from each pair does not lead to a solution; a solution is indicated by the red cycle.

Definitions and notations. A *hypergraph* is a pair $H = (V, E)$, where V is finite set of *vertices*, and E is a family of subsets of V, called *edges*. H is said to have *property B*, or be 2-*colorable*, if there is a 2-coloring of V such that no edge is monochromatic; see, e.g., [2, Ch. 1.3].

If C is a (polygonal) cycle, let $\overset{\circ}{C}$ and $\overline{C}$ denote the interior and exterior of C, respectively; let ∂C denote its boundary. Consider a geometric hypergraph $H = (V, E)$ on points in the plane with no singleton edges. A polygonal cycle C is said to be a *separating cycle* for H if (i) C is simple; and (ii) each edge of H has points inside C (in its interior or on its boundary) and points in the exterior of C; that is, for each edge $A \in E$, both $A \cap (\overset{\circ}{C} \cup \partial C)$ and $A \cap \overline{C}$ are nonempty.

A simple polygonal cycle is said to have *zero area*, if $\text{Area}(C) \leq \varepsilon$, for a sufficiently small given $\varepsilon > 0$. Similarly, a polyhedron P is is said to have *zero volume*, if $\text{Vol}(P) \leq \varepsilon$, for a sufficiently small given $\varepsilon > 0$.

Preliminaries and related work. Let S be a finite set of points in the plane. According to an old result of Few [11], the length of a minimum spanning path (resp., minimum spanning tree) of any n points in the unit square is at most $\sqrt{2n} + 7/4$ (resp., $\sqrt{n} + 7/4$). Both upper bounds are constructive; for example, the construction of a short spanning path works as follows. Lay out about $\sqrt{n}$ equidistant horizontal lines, and then visit the points layer by layer, with the path alternating directions along the horizontal strips. In particular, the length of the minimum spanning tree of any n points in the unit square is bounded from above by the same expression. An upper bound with a slightly better multiplicative constant for a path was derived by Karloff [18]. L. Fejes Tóth [10] had observed earlier that for n points of a regular hexagonal lattice in the unit square, the length of the minimum spanning path is asymptotically equal to $(4/3)^{1/4}\sqrt{n}$, where $(4/3)^{1/4} = 1.0745\ldots$. As such, the maximum length of the minimum spanning tree of any n points in the unit square is $\Theta(\sqrt{n})$, for a small constant (close to 1). The bound also holds for points in a convex polygon of diameter $O(1)$, in particular for n points in a rectangle of diameter $O(1)$. In every dimension $d \geq 3$, Few showed that the maximum length of a shortest path (or tree) through n points in the unit cube is $\Theta(n^{1-1/d})$; this upper bound is again constructive and extends to rectangular boxes of diameter $O(1)$.

The topic of "separation" has appeared in multiple interpretations; here we only give a few examples: [1, 6, 7, 12, 14, 15, 16]. Some results on watchman tours relying on Few's bounds can be found in [8]; others can be be found in [4]. For instance, in the problem of finding a separating cycle for a given set of segment pairs, that we study here, it is clear that the edges of the cycle must hit all of the given segments. As such, this problem is related to the classic problem of hitting a set of segments by straight lines [15]. Coloring of geometric hypergraphs has been studied, e.g., in [20].

2 Separating Cycles for Graphs and Hypergraphs

By adapting results on hypergraph 2-colorability to a geometric setting, we obtain the following.

Theorem 1 *Let $H = (V, E)$ be a geometric hypergraph on points in the plane with no singleton edges. Then H admits a separating cycle if and only if H is 2-colorable.*

Proof. For the direct implication, assume that C is a separating cycle: then for each $A \in E$, both $A \cap \overset{\circ}{C}$ and $A \cap \overline{C}$ are nonempty. Color the points in the interior of C by red and those in its exterior by blue. As such, the hypergraph H is 2-colorable.

We now prove the converse implication. Let $V = R \cup B$ be a partition of the points into red and blue points, such that no edge in E is monochromatic. We construct a simple polygonal cycle containing only the red points in its interior. To this end, we first compute a minimum spanning tree T for the points in R; T is non-crossing [19, Ch. 6], however there could be blue points contained in edges of T. Replace each such edge s with a two-segment polygonal path $\widetilde{s}$ connecting the same pair of points and lying very close to the original segment, and so that $\widetilde{s}$ is not incident to any other point.

The resulting tree, $\widetilde{T}$ is still non-crossing and spans all points in R. By doubling the edges of $\widetilde{T}$ and adding

short connection edges, if needed, construct a simple polygonal cycle C of *zero area* that contains it and lies very close to it; as such, C contains all red points and none of the blue points, as required. □

Since hypergraph 2-colorability is NP-complete [13], Theorem 1 yields the following.

Corollary 1 *Given a geometric hypergraph $H = (V, E)$ on points in the plane with no singleton edges, the problem of deciding whether H admits a separating cycle is NP-complete.*

A key fact in our algorithm is the following observation.

Lemma 2 *Let G be connected bipartite graph. Then (apart from a color flip), G admits a unique 2-coloring.*

Proof. Recall that a graph is bipartite if and only if it contains no odd cycle [17, Ch. 3.3]. Consider an arbitrary vertex s and color it red. Then the color of any other vertex, say v, is uniquely determined by the parity of the length of the shortest path from s to v in G: red for even length and blue for odd length. Indeed, the vertices are colored alternately on any path, and since any cycle has odd length, all lengths of paths from s to v have the same parity, as required. □

Let $G = (V, E)$ be the input geometric graph, where $|V| = n$, $|E| = m$, and G has no isolated vertices. Let $G_1, \ldots, G_k$ denote the connected components of G, where $G_i = (V_i, E_i)$, for $i = 1, \ldots, k$.

Theorem 3 (i) *Given a geometric graph $G = (V, E)$, a separating cycle (if it exists) can be computed in $O(m + n \log n)$ time, where $|V| = n$, $|E| = m$.* (ii) *Further, a $O(\sqrt{n})$-approximation of the shortest separating cycle can be found in polynomial time.*

Proof. (i) The graph is first tested for bipartiteness and the input instance is declared infeasible if the test fails (by Theorem 1). This test takes $O(m + n)$ time; see, e.g., [17, Ch. 3.3]. We subsequently assume that G is bipartite, with vertices colored by red and blue. Then the algorithm constructs a plane spanning tree T of the red points (for instance, a minimum spanning tree), and outputs a simple cycle by doubling its edges and avoiding the blue points on its edges by bending those edges as indicated in the proof of Theorem 1. To this end, the following parameters are computed: $\delta_1 > 0$ is the minimum pairwise distance among points in V, found in $O(n \log n)$ time [19, Ch. 5]. For each edge e of T, $\delta_2(e) \geq 0$ is the minimum distance from some blue point to e ($\delta_2(e) = 0$ if e is incident to at least one blue point); and $\delta_3(e) > 0$ is the minimum nonzero distance from a blue point to e ($\delta_3(e) = \infty$ if no blue point is close to e, as described next). The set of values $\delta_2(e), \delta_3(e)$ can be determined using point location for the blue points (as query points) in a planar triangulated subdivision containing the edges of T, all in $O(n \log n)$ time [5, Ch. 6]. The overall time complexity of the algorithm is $O(m + n \log n)$.

(ii) The algorithm above is modified as follows; the first step is the same bipartiteness test. The algorithm 2-colors the vertices in each connected component by red and blue: $V_i = R_i \cup B_i$, for $i = 1, \ldots, k$. By Lemma 2, the 2-coloring of each component is unique (apart from a color flip). The initial coloring of a component may be subsequently subject to a color flip if the algorithm so later decides. Obviously, the coloring of each component is done independently of the others.

Then, the algorithm guesses the diameter of OPT, as determined by one of the $\binom{n}{2}$ pairs of points in V (by trying all such pairs). In each iteration, the algorithm may compute a separating cycle and record its length; the shortest cycle will be output by the algorithm; some iterations may be abandoned earlier, without the need of this calculation.

Consider the iteration in which the guess is correct, with pair $a, b \in V$; we may assume for concreteness that ab is a horizontal segment of unit length; refer to Fig 2. As such, we have that $\text{len}(\text{OPT}) \geq 2|ab| = 2$. In this iteration, the algorithm computes a separating cycle whose length is bounded from above by $O(\sqrt{n})$. First, the algorithm computes a rectangle Q of unit width and height $\sqrt{3}$ centered at the midpoint of ab. By the diameter assumption, OPT is contained in Q. In the next step the algorithm computes a separating cycle C containing only red points in Q in its interior (however, the initial coloring of some of the components may be flipped, as needed). By Lemma 2, the coloring of each component is unique (modulo a color flip) and so for each of the components at least one of its color classes is entirely contained in Q. As such, all points in V not contained in Q can be discarded from further consideration.

Each of the components G_i, $i = 1, \ldots, k$ is checked against this containment condition: if a component is found where neither of its two color classes lies in Q, the algorithm abandons this iteration (and assumed diameter pair, $ab = \text{diam}(\text{OPT})$). For each component G_i: (i) if $R_i \subset Q$, then the coloring of this component remains unchanged, regardless of whether $B_i \subset Q$ or $B_i \not\subset Q$. (ii) if $R_i \not\subset Q$ and $B_i \subset Q$, then the coloring of this component is flipped: $R_i \leftrightarrow B_i$, so that $R_i \subset Q$ after the color flip.

Once the recoloring of components is complete, the algorithm computes a minimum spanning tree T of the red points in Q. Its length is bounded from above by the length of the spanning tree computed by Few's algorithm. Since the number of red points does not exceed n, we have $\text{len}(T) = O(\sqrt{n})$. Finally, T is converted

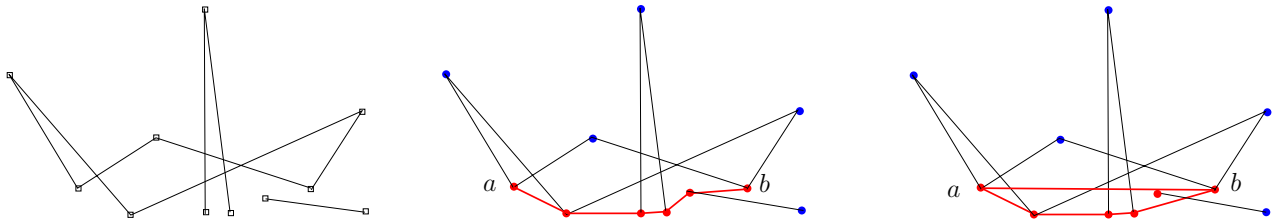


Figure 2: Left: input bipartite graph. Center: a separating cycle can be computed from the MST of the red points (after color flips). Right: a shortest separating cycle.

to a separating cycle C by a factor of at most $2+\varepsilon$ increase in length, for any given $\varepsilon > 0$, as in the proof of part (i). Recalling that $\text{len}(\text{OPT}) \geq 2$, it follows that C is a $O(\sqrt{n})$-factor approximation of a shortest separating cycle. □

3 Remarks

1. If the input consists of a set of pairs so that the corresponding graph is bipartite, then by Theorem 1, it admits a separating cycle. (If the corresponding graph is not bipartite, no separating cycle exists.) Similarly, if the input is a 2-colorable hypergraph, it admits a separating cycle. For illustration, we recall some common instances of 2-colorable hypergraphs. A hypergraph $H = (V, E)$ is called k-uniform if all $A \in E$ have $|A| = k$. A random 2-coloring argument gives that any k-uniform hypergraph with fewer than 2^{k-1} edges is 2-colorable [2, Ch. 1.3]; as such, by Theorem 1, it admits a separating cycle. Slightly better bounds have been recently obtained; see [2, Ch. 3.5]. Similarly, let $H = (V, E)$ be a hypergraph in which every edge has size at least k and assume that every edge $A \in E$ intersects at most Δ other edges, i.e., the maximum degree in H is at most Δ. If $e(\Delta+1) \leq 2^{k-1}$ (here $e = \sum_{i=0}^{\infty} 1/i!$ is the base of the natural logarithm), then by the Lovász Local Lemma, H can be 2-colored [2, Ch. 5.2] and so by Theorem 1, it admits a separating cycle; moreover, if a 2-coloring is given, it can be used to obtain a separating cycle. While testing for 2-colorability can be computationally expensive in a general setting (for certain problem instances), it can be always achieved in exponential time; recall that hypergraph 2-colorability is NP-complete [13].

2. Theorem 3 generalizes to 3-dimensional polyhedra. A polyhedron in 3-space is a simply connected solid bounded by piecewise linear 2-dimensional manifolds. The *perimeter* $\text{per}(P)$ of a polyhedron P is the total length of the edges of P (as in [8]).

For part (i), a method similar to that used in the planar case can be used to construct a separating polyhedron in $\mathbb{R}^3$ (or $\mathbb{R}^d$). However, since computing minimum spanning trees in $\mathbb{R}^3$ is more expensive [9, Ch. 9], we employ a slightly different approach. We may assume a coordinate system so that no pair of points have the same x-coordinate. First, the points in V are colored by red or blue as a result of the bipartiteness test, in $O(m+n)$ time. The algorithm then computes a (spanning tree of the red points in the form of a) x-monotone polygonal path P spanning all the red points; this step takes $O(n \log n)$ time. From P, it then obtains a x-monotone polygonal path $\widetilde{P}$ spanning all the red points and not incident to any blue point ($P = \widetilde{P}$ if no blue points are incident to edges of P); $\widetilde{P}$ is constructed in $O(n \log n)$ time.

To this end, P and all blue points are projected onto the xoy plane. Let $\sigma(\cdot)$ denote the projection function. Note that $\sigma(P)$ is x-monotone and that the projection $\sigma(b)$ of a blue point b can be incident to at most one edge of $\sigma(P)$. Checking the projection points $\sigma(b)$ against corresponding edges of $\sigma(P)$ allows for testing whether the original edges of P are incident to the respective blue points. Further, this test allows replacing each such edge s with a two-segment polygonal path $\widetilde{s}$ connecting the same pair of points and lying very close to the original segment, and so that $\widetilde{s}$ is not incident to any other point. Finally the algorithm computes a polyhedron of *zero volume* that contains $\widetilde{P}$; as such, the polyhedron contains all red points but no blue points; this step takes $O(n \log n)$ time. Some details are omitted.

For part (ii), instead of a rectangle based on segment ab as an assumed diameter pair, the algorithm works with a rectangular box where ab is parallel to a side of the box and is incident to its center. The upper bound on the perimeter of the separating polyhedron follows from Few's bound mentioned in the preliminaries.

Theorem 4 (i) *Given a geometric graph $G = (V, E)$ in $\mathbb{R}^3$, a separating polyhedron (if it exists) can be found in $O(m + n \log n)$ time, where $|V| = n$, $|E| = m$.* (ii) *Further, a $O(n^{2/3})$-approximation of a separating polyhedron of minimum perimeter can be found in polynomial time.*

Acknowledgment. The author is grateful to an anonymous reviewer for his careful reading of the manuscript and pertinent remarks.

References

[1] N. Alon, Z. Füredi, and M. Katchalski, Separating pairs of points by standard boxes, *European Journal of Combinatorics* **6(3)** (1985), 205–210.

[2] N. Alon and J. Spencer, *The Probabilistic Method*, 4th edition, Wiley, Tel Aviv and New York, 2015.

[3] E. Arkin, J. Gao, A. Hesterberg, J. S. B. Mitchell, and J. Zeng, The shortest separating cycle problem, *Proc. 14th International Workshop on Approximation and Online Algorithms* (WAOA 2016), vol. 10138 of Lecture Notes in Computer Science, 2016, pp. 1–13.

[4] E. M. Arkin, J. S. B. Mitchell and C. D. Piatko, Minimum-link watchman tours, *Information Processing Letters* **86** (2003), 203–207.

[5] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd edition, Springer, 2008.

[6] S. Cabello and P. Giannopoulos, The complexity of separating points in the plane, *Algorithmica* **74(2)** (2016), 643–663.

[7] G. Călinescu, A. Dumitrescu, H. Karloff, and P.-J. Wan, Separating points by axis-parallel lines, *International Journal of Computational Geometry & Applications* **15(6)** (2005), 575–590.

[8] A. Dumitrescu and C. D. Tóth, Watchman tours for polygons with holes, *Computational Geometry: Theory and Applications*, **45** (2012), 326–333.

[9] D. Eppstein, Spanning trees and spanners, in J.-R. Sack and J. Urrutia (editors), *Handbook of Computational Geometry*, pages 425–461, Elsevier Science, Amsterdam, 2000.

[10] L. Fejes Tóth, Über einen geometrischen Satz, *Mathematische Zeitschrift* **46** (1940), 83–85.

[11] L. Few, The shortest path and shortest road through n points, *Mathematika* **2** (1955), 141–144.

[12] R. Freimer, J.S.B. Mitchell, and C. Piatko, On the complexity of shattering using arrangements. Extended abstract in *Proc. of the 2nd Canadian Conference on Computational Geometry* (CCCG 1990), 1990, pp. 218–222. Technical Report 91-1197, Dept. of Comp. Sci., Cornell University, April, 1991.

[13] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., New York, 1979.

[14] D. R. Gaur, T. Ibaraki, and R. Krishnamurti, Constant ratio approximation algorithm for the rectangle stabbing problem and the rectilinear partitioning problem, *Journal of Algorithms* **43** (2002), 138–152.

[15] R. Hassin and N. Megiddo, Approximation algorithms for hitting objects with straight lines, *Discrete Applied Mathematics* **30(1)** (1991), 29–42.

[16] F. Hurtado, M. Noy, P. A. Ramos, and C. Seara, Separating objects in the plane with wedges and strips, *Discrete Applied Mathematics* **109** (2001), 109–138.

[17] D. Jungnickel, *Graphs, Networks and Algorithms*, Springer Verlag, Berlin, 1999.

[18] H. J. Karloff, How long can a Euclidean traveling salesman tour be? *SIAM Journal on Discrete Mathematics* **2** (1989), 91–99.

[19] F. P. Preparata and M. I. Shamos, *Computational Geometry*, Springer-Verlag, New York, 1985.

[20] S. Smorodinsky, On the chromatic number of geometric hypergraphs, *SIAM Journal on Discrete Mathematics* **21(3)** (2007), 676–687.

[21] D. Williamson and D. Shmoys, *The Design of Approximation Algorithms*, Cambridge University Press, 2011.

Open Problems from CCCG 2016

Joseph O'Rourke*

The following is a description of the problems presented on August 3, 2016 at the open-problem session of the *28th Canadian Conference on Computational Geometry* held at Simon Fraser University in Vancouver, Canada.

Sofa in Snakey 3D Corridor
Joseph O'Rourke
Smith College
orourke@cs.smith.edu

What is the largest volume object that can pass though a $1 \times 1 \times L$ "snaky" corridor, where L is large enough to be irrelvant, say $L > 6$.

This is a 3D version of the 2D sofa-moving problem, which has been heavily studied. See especially Dan Romik's description [Rom16]. The optimal-area 2D sofa is conjectured to be Gerver's (slight) modification of Hammersley's shape, the latter of which I show in Figure 1, extruded in 3D to fill the corridor. There are two natural candidates,

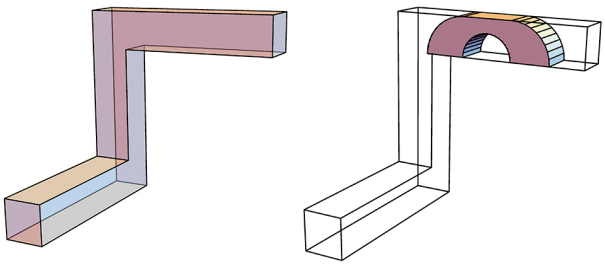


Figure 1: Left: Corridor. Right: Hammersley's shape extruded.

the first of which was suggested at the conference: (1) Slice the extruded 2D optimal shape, in the orthogonal direction, so it can negotiate both turns in the same manner. See Figure 2. (2) Rotate the illustrated shape 90° but shear-off every portion that falls outside the 1×1 corridor. A basic question is: Is either of these the optimal solution, or can one identify some shape that beats both? An even more basic (and easier) question is: Which of (1) or (2) has larger volume?

*Department of Computer Science, Smith College, Northampton, MA 01063, USA. orourke@cs.smith.edu

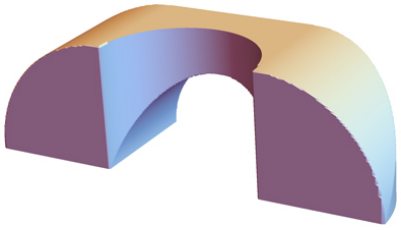


Figure 2: The sofa (1) can pass through the 1×1 corridor. Volume: $4\left(8+\pi^3\right)/(3\pi^3) \approx 1.67735$.

References

[Rom16] Dan Romik. The moving sofa problem. https://www.math.ucdavis.edu/~romik/movingsofa/).

Dimension-descending Tilers
Joseph O'Rourke
Smith College
orourke@cs.smith.edu

The hypercube obviously tiles $\mathbb{R}^4$. The hypercube has 261 "face-unfoldings," each a polycube composed of 8 cubes. The most famous unfolding is the cross used by Salvador Dali's in his painting *Corpus Hypercubus*. It was shown in [DO15] that the Dali cross can tile $\mathbb{R}^3$ (Figure 3), and Stefan Langerman and Andrew Winslow (personal communication) showed there is an unfolding of the Dali cross that tiles $\mathbb{R}^2$. And of course any simply polygon can be cut at a vertex and unfolded to tile R^1.

Is the $\mathbb{R}^5$ hypercube a *dimension-descending tiler* in the same sense? In particular, how many facet-unfoldings of the 5-dimensional cube are there?

References

[DO15] Giovanna Diaz and J. O'Rourke. Hypercube unfoldings that tile $\mathbb{R}^3$ and $\mathbb{R}^2$. arXiv:1512.02086 [cs.CG]. http://arxiv.org/abs/1512.02086, 2015.

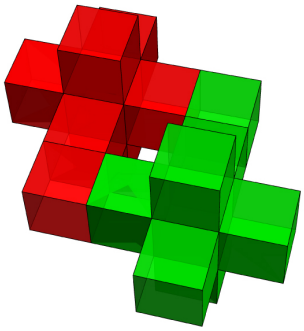


Figure 3: Two copies of the Dali cross, the start of a tiling of $\mathbb{R}^3$.

Random Non-obtuse Inscribed Polyhedra
Joseph O'Rourke
Smith College
orourke@cs.smith.edu

Under some reasonable definition of "random," can one generate arbitrarily large number of random points on a sphere such that the faces of the convex hull have no obtuse angles?

The hull of random points in general includes obtuse triangles; see Figure 4.

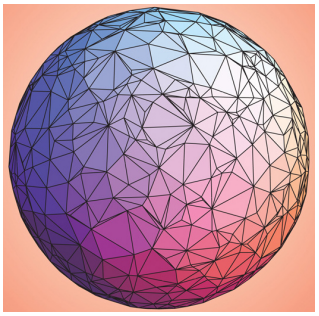


Figure 4: Convex hull of random points on a sphere.

A Floppy Pizza Problem
Don Sheehy
UConn
don.r.sheehy@gmail.com

This problem was inspired by the popular use of Gauss's Theorema Egregium to fold pizza in a way that keeps the tip from flopping (isometric embeddings of surfaces preserve curvature).

A *ruling* of a polygon P is a covering of P with disjoint line segments such that each line segment has both ends on the boundary of P. A ruling corresponds to the lines of 0-sectional curvature in some isometric embedding of P into $\mathbb{R}^3$.

Given a ruling of P, a set of points $S \subset P$ *supports* P if every line segment of the ruling separates at least one pair of point of S. That is, if we cut along any line segment, each of the resulting pieces would contain at least one vertex from S. See Figure 5.

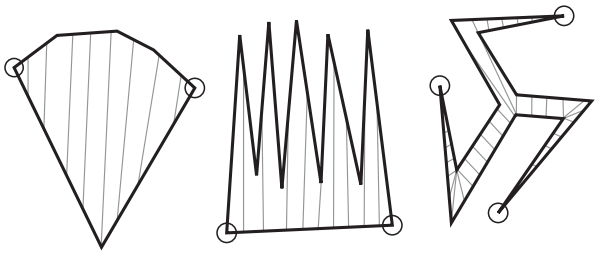


Figure 5: Supporting rulings.

The Algorithmic Problem: Given a polygon P in the plane, find a ruling of P and a supporting set S of minimum size.

The Maximizing Problem: What is the minimum number $f(n)$ such that every n-gon P has a ruling and a set of size $f(n)$ that supports P?

The Minimizing Problem: (From David Eppstein) Characterize those polygons P that can be given a ruling and a supporting set of size 2. David Eppstein conjectures that it is necessary and sufficient for the polygon to have a subdivision (of edges) that admits a Hamiltonian triangulation.

When We Don't Care
W. Randolph Franklin
RPI
mail@wrfranklin.org

This topic arises when computing the union of two polygons, or even more, when finding the area of the union. Although the exact union may be affected by roundoff error, sometimes we don't care. Perhaps all of the possible outputs are acceptable: see Figure 6. The example shows find the union of

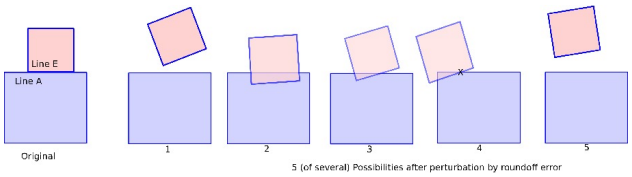


Figure 6: Two aligned squares, perturbed by roundoff error

two squares, who are adjacent on parts of a common edge. After roundoff, those two edges may

intersect or be disjoint in any of several configurations. However, all the possibilities might have the same area within roundoff error. All the possibilities are equally valid as input to a following operation, such as: testing point-containment, computing further boolean operations, etc. All we care is that the output be internally consistent and topologically possible.

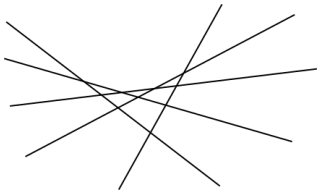


Figure 7: Wanting the envelope of some lines

Figure 7 shows another example, due to Jarek Rossignac. Suppose that we wish to trace the upper envelope, from the top left to the top right. What particular intersections we traverse will depend on roundoff error, but the resulting envelope is the same for many purposes.

Overgeneralizing, this is reminiscent of string theory in physics. At a sufficiently large scale, small geometric details vanish. But at that large scale, we don't need them anyway.

The open problem is to explore this topic.

Recognizing the k-th NN Voronoi Diagram

Carsten Grimm
Otto-von-Guericke-Universität Magdeburg
carsten.grimm@ovgu.de

Suppose we are given a set of n sites in the plane. For $k = 1, 2, \ldots, n-1$, the k-th nearest neighbour Voronoi diagram of S is the subdivision of the plane depending on which site in S is the k-th nearest one. We would like to know how we could recognize whether a given subdivision is the k-th nearest neighbour Voronoi diagram for some set of sites.

We distinguish the following three input models. In each model, we seek a set of sites S such that the k-th nearest neighbour Voronoi diagram of S corresponds to the input, if such a set of sites exists. If an input cannot be realized as a k-th nearest neighbour Voronoi diagram, we would like a certificate.

Geometric Model We are given a subdivision of the plane with coordinates for the vertices and rays representing the unbounded edges.

Ordered Model We are given an ordered graph where every vertex has a fixed cyclic ordering of its neighbours. We have no embedding or coordinates, but some vertices are marked as symbolic endpoints of unbounded edges.

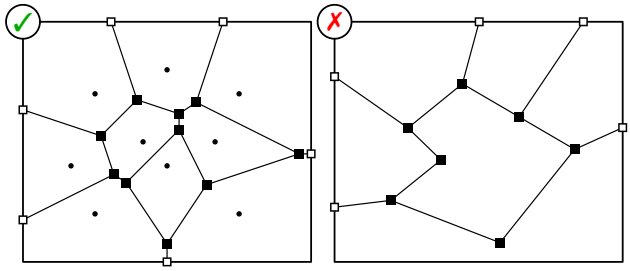


Figure 8: Two subdivisions of the plane. The left one is a Voronoi diagram ($k = 1$), the right one is not. Full squares mark vertices of the subdivison; empty squares mark symbolic endpoints of unbounded edges.

Abstract Model We are given an abstract graph without any further restrictions.

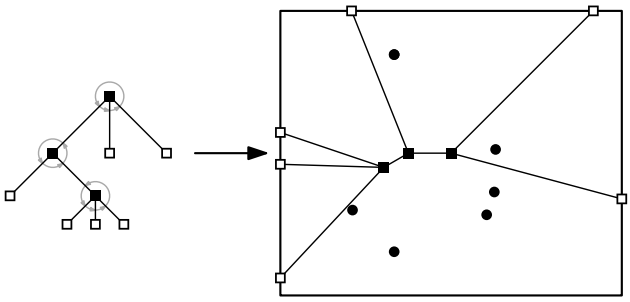


Figure 9: Turning an ordered tree into a farthest-point Voronoi diagram.

Recognizing nearest-neighbour Voronoi diagrams ($k = 1$) has been studied in the geometric model [AB85, Har92, BHH13, AP+13] and in the ordered setting [LM03]. Recognizing farthest-point Voronoi diagrams ($k = n - 1$) has been studied in both the geometric and ordered model [BG+16]. When the input is a geometric or ordered tree T it takes linear time to locate a suitable set of sites to realize T as nearest/farthest-point Voronoi diagram [LM03, BHH13, BG+16].

This question might also be interesting in higher dimensions, with respect to more general metrics, or for other types of sites. Another challenge would be to locate sites with *nice coordinates*, i.e., coordinates with polynomial size.

References

[AB85] P. Ash and E. Bolker. Recognizing Dirichlet tesselations. *Geometriae Dedicata*, 19:175–206, 1985.

[Har92] D. Hartvigsen. Recognizing Voronoi diagrams with linear programming. *ORSA J. Comput.*, 4:369–374, 1992.

[LM03] G. Liotta and H. Meijer. Voronoi drawings of trees. *Comput. Geom.*, 24(3):147–178, 2003.

[BHH13] T. Biedl, M. Held, and S. Huber. Recognizing straight skeletons and Voronoi diagrams and reconstructing their input. In *10th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2013)*, pages 37–46, 2013.

[AP+13] G. Aloupis, H. Pérez-Rosés, G. Pineda-Villavicencio, P. Taslakian, and D. Trinchet-Almaguer. Fitting Voronoi diagrams to planar tesselations. In *Intl. Workshop on Combinatorial Algorithms (IWOCA 2013)*, pages 349–361, 2013.

[BG+16] T. Biedl, C. Grimm, L. Palios, J. Shewchuk, and S. Verdonschot. Realizing Farthest-Point Voronoi Diagrams. In *28th Canadian Conference on Computational Geometry (CCCG 2016)*, pages 48–56, 2016.

Optimal Paving of Disk

Tom Shermer
Simon Fraser Univ.
shermer@sfu.ca

Let D be a unit-radius disk constituting a town. Travel from point to point in the disk off a highway is at walking speed 1. On highways the speed is $s > 1$. The town has a budget of B units of money–call them \$–where 1\$ can build a highway of length 1. Highways are rectifiable curves, with length Euclidean. What network of roads should the town construct, given s and its budget of B\$, to minimize the maximum point-to-point travel time, over all pairs of points in D?

What happens as s becomes large with respect to B? It may be interesting to suppose that $B \leq 2\pi$ and $s \geq \pi$. One might also consider the L_1 metric, or the "heavy luggage" metric.

Burning the Medial Axis

Erin Wolf Chambers*

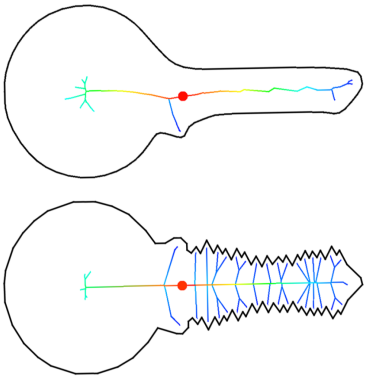


Figure 1: The burn time of a 2d shape (top) and the same shape with a perturbed boundary (bottom).

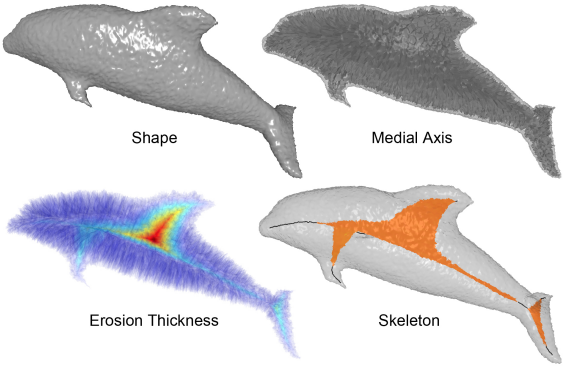


Figure 2: A figure of a dolphin, with the medial axis, erosion thickness, and resulting skeleton.

Initially proposed by Blum in 1967, the medial axis of a shape consists of the union of all centers of maximally inscribed balls. The medial axis is one of the most commonly used tools for understanding shape, as it is homotopy equivalent to the original object, has codimension one, and is centrally located. In addition, it is used as a component in building skeletons that are of smaller dimension than the original object, but which capture the shape in a more compact but still useful representation. However, the medial axis is unstable to perturbations; even small changes in the boundary of the shape result in large changes in the medial axis.

Methods for pruning the medial axis are usually guided by some measure of significance, with considerable work done for both 2 and 3 dimensional shapes. However, the majority of significance measures over the medial axis are locally defined and hence unable to capture more global features, or are difficult to compute and sensitive to perturbations on the boundary. In general, there are no skeletons which provably capture the correct topology, are central to the object, are always result in a curve skeleton for a 3-dimensional input.

In this talk, I will present recent work done in 2d and 3d to compute new significance measures on the medial axis. In 2d, the extended distance function (EDF), also called the burn time, was recently developed by Liu et [3], as well as related measures such as erosion thickness and weighted EDF [1]. See Figure 1 for an illustration of this function. The EDF function was later generalized to the burn time function for 3 dimensional shapes, yielding both a mathematical framework for quantifying shape as well as an algorithm for approximating this function for a union of balls, which are commonly used for surface reconstruction and approximation [4]. In 3d, this also allows us to develop a definition of topologically accurate 1-dimensional skeletons; see Figure 2. These measures give practical methods for differentiating boundary noise from primary features, and can be used for shape alignment and recognition. In addition, there is both practical and theoretical evidence that these measures are robust under certain types of noise in the boundary [2], unlike the medial axis itself.

*Department of Computer Science, Saint Louis University, echambe5@slu.edu

References

[1] K. Leonard, G. Morin, S. Hahmann, and A. Carlier. A 2d shape structure for decomposition and part similarity. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 3216–3221, Dec 2016.

[2] D. Letscher and K. Sykes. On the stability of medial axis of a union of balls in the plane. In *CCCG*, 2016.

[3] L. Liu, E. W. Chambers, D. Letscher, and T. Ju. Extended grassfire transform on medial axes of 2d shapes. *Comput. Aided Des.*, 43(11):1496–1505, Nov. 2011.

[4] Y. Yan, K. Sykes, E. Chambers, D. Letscher, and T. Ju. Erosion thickness on medial axes of 3d shapes. *ACM Trans. Graph.*, 35(4):38:1–38:12, July 2016.

The most-likely skyline problem for stochastic points

Akash Agrawal* Yuan Li† Jie Xue‡ Ravi Janardan§

Abstract

For a set O of n points in $\mathbb{R}^d$, the *skyline* consists of the subset of all points of O where no point is dominated by any other point of O. Suppose that each point $o_i \in O$ has an associated probability of existence $p_i \in (0, 1]$. The problem of computing the skyline with the maximum probability of occurrence is considered. It is shown that in $\mathbb{R}^d$, $d \geq 3$, the problem is NP-hard and that the desired skyline cannot even be well-approximated in polynomial-time unless $P = NP$. In $\mathbb{R}^2$, an optimal $\mathcal{O}(n \log n)$-time and $\mathcal{O}(n)$-space algorithm is given.

1 Introduction

In $\mathbb{R}^d$, a point u *dominates* a point v if each coordinate of u is at least as large as the corresponding coordinate of v, with strict inequality in at least one dimension. The *skyline* of a set of points consists of the subset of all points where no point is dominated by any other point of the set. (See Figure 1a.) The skyline (or *Pareto set* or *maximal vector*) is useful in multi-criteria decision-making as it yields a set of viable candidates for further exploration. It has been well-studied in the database, optimization, and computational geometry literature; e.g., [3, 5, 6].

We investigate skylines in a setting where there is uncertainty associated with the existence of the points. Such stochastic datasets can model, for instance, experimental observations with associated confidence values or physical entities that may not always be available (e.g., sensors whose activity level depends on battery life or hotel rooms where availability depends on demand).

We consider the problem of computing the skyline that has the greatest probability of being present, hence the one that the user is most likely to encounter and explore further. We call this the most-likely skyline. Our results include an optimal algorithm in the plane and hardness results in higher dimensions.

*Dept. of Computer Science and Engg., Univ. of Minnesota-Twin Cities, akash@umn.edu

†Dept. of Computer Science and Engg., Univ. of Minnesota-Twin Cities, lixx2100@umn.edu

‡Dept. of Computer Science and Engg., Univ. of Minnesota-Twin Cities, xuexx193@umn.edu

§Dept. of Computer Science and Engg., Univ. of Minnesota-Twin Cities, janardan@umn.edu

1.1 Problem formulation, contributions, and related work

Let $O = \{o_1, o_2, \ldots, o_n\}$ be a set of points in $\mathbb{R}^d$, where $x_k(o_i)$ denotes the kth coordinate of o_i. Point o_i *dominates* o_j (i.e., $o_i \succ o_j$) if $x_k(o_i) \geq x_k(o_j)$ for $1 \leq k \leq d$, with strict inequality in at least one dimension.

Suppose that each $o_i \in O$ has an associated real $p_i \in (0, 1]$. (The p_i's are known and independent of each other.) We call p_i (resp. $q_i = 1 - p_i$) the *existence* (resp. *non-existence*) *probability* of o_i and call O a *stochastic set*.

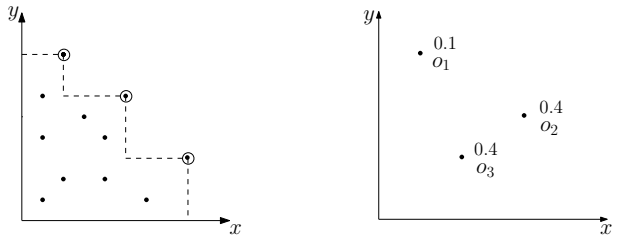
Let $O' \subseteq O$, where no point of O' dominates another of O'; thus, O' itself is the skyline of O'. Now, when is O' also a skyline of O? Let $F(O') \subseteq O \setminus O'$ be the points that are not dominated by any point of O'; intuitively, these are the points "above" the staircase contour defined by O'. Clearly, as long as no point of $F(O')$ is present, O' is also the skyline of O. (The points of $O \setminus O'$ that are dominated by one or more points of O', i.e., the ones "below" the staircase, do not affect the skyline property of O'.) Thus, for O' to be a skyline of O, each point of O' must be present and no point of $F(O')$ should be present. So, the probability that O' is a skyline of O is $PrSky(O') = \prod_{o_i \in O'} p_i \times \prod_{o_i \in F(O')} q_i$.

Our problem is to compute a skyline O' of O for which $PrSky(O')$ is maximum. This skyline, called the *most-likely skyline* of O, is denoted by $MLSky(O)$. (See Figure 1b and Figure 1c for an example.)

Note that the introduction of uncertainty makes our problem challenging as there might be an exponential number of candidate skylines—as many as one for each possible subset of existent points. By contrast, in the non-stochastic setting, there is exactly one skyline for a given set of points.

We make three contributions to the most-likely skyline problem. We prove that computing such a skyline is NP-hard in $\mathbb{R}^3$, hence also in $\mathbb{R}^d$ for $d > 3$ (Section 2). Furthermore, we prove that the most-likely skyline in $\mathbb{R}^d$ ($d \geq 3$) cannot even be well-approximated in polynomial-time unless $P = NP$ (Section 3). We complement these results with an $\mathcal{O}(n \log n)$-time and $\mathcal{O}(n)$-space algorithm to compute the most-likely skyline in $\mathbb{R}^2$ (Section 4), which is optimal in the comparison model due to the known $\Omega(n \log n)$ lower bound for the non-stochastic skyline problem [5].

To our knowledge, this paper is the first to consider skylines in the *unipoint stochastic model*, where



O'	$F(O')$	$PrSky(O')$
$\emptyset$	$\{o_1, o_2, o_3\}$	$(1-0.1)\times(1-0.4)\times(1-0.4) = 0.324$
$\{o_1\}$	$\{o_2, o_3\}$	$0.1\times(1-0.4)\times(1-0.4) = 0.036$
$\{\mathbf{o_2}\}$	$\{\mathbf{o_1}\}$	$\mathbf{(1-0.1)\times 0.4 = 0.36}$
$\{o_3\}$	$\{o_1, o_2\}$	$(1-0.1)\times(1-0.4)\times 0.4 = 0.216$
$\{o_1, o_2\}$	$\emptyset$	$0.1\times 0.4 = 0.04$
$\{o_1, o_3\}$	$\{o_2\}$	$0.1\times(1-0.4)\times 0.4 = 0.024$

(c)

Figure 1: (a) Skyline of a conventional (i.e., non-stochastic) point-set, with skyline points circled. (b) A set, O, of stochastic points and associated existence probabilities. (c) Computing the most-likely skyline for the example in (b); here it consists of just $\{o_2\}$. (Note that if p_2 is decreased to 0.2 and the example is re-worked, then the most-likely skyline is $\emptyset$, with probability 0.432.)

the points have fixed locations and associated existence probabilities. An alternative setting is the *multipoint stochastic model*, where each point is described by discretely-many locations, with associated existence probabilities, or by a continuous probability distribution. Examples of work here include computing skylines whose points have existence probabilities above a threshold [7], computing for each dataset point (or for a query point) the probability that it is not dominated by any other point [1, 2], computing stochastic skyline operators to find a minimum set of candidate points with respect to a certain scoring function [9], etc.

2 NP-Hardness of computing the most-likely skyline in $\mathbb{R}^d$, $d \geq 3$

We give a polynomial-time reduction from the minimum ε-ADR problem in $\mathbb{R}^3$, which is known to be NP-hard [4], to the most-likely skyline problem in $\mathbb{R}^3$. (Here ADR stands for "Approximately Dominating Representatives" [4].)

An instance of the ε-ADR problem in $\mathbb{R}^3$ consists of a set, S, of n (non-stochastic) points and a real $\varepsilon > 0$. An *ε-ADR* of S is a set $S' \subseteq S$ such that every $s \in S$ is dominated by some $s' \in S'$ when s' is boosted by ε, i.e., $(1+\varepsilon)\cdot s' \succ s$. The *minimum ε-ADR problem* seeks the smallest such set S'.

The reduction: Given any ε-ADR instance in $\mathbb{R}^3$, we compute the (conventional) skyline, $Sky(S)$, of S and boost its points by ε to get a set $\hat{S}$. To each point in $Sky(S)$ (resp. $\hat{S}$) we assign an existence probability β (resp. α), where $1/3 < \alpha < 1/2 < \beta < 1$ and $\beta(1-\alpha) < \alpha$; e.g., $\alpha = 0.4$ and $\beta = 0.6$. The set $\bar{S} = Sky(S) \cup \hat{S}$ is an instance of the most-likely skyline problem in $\mathbb{R}^3$. The reduction takes polynomial time. We observe the following:

(i) The probability that the skyline of $\bar{S}$ is empty is the probability that no point of $\bar{S}$ exists, i.e., $(1-\beta)^K(1-\alpha)^K$, where $K = |Sky(S)| = |\hat{S}|$. Since $\beta > 1/2$, the probability of the empty skyline is less than $\beta^K(1-\alpha)^K$. The latter is the probability of that skyline of $\bar{S}$ where the points of $Sky(S)$ exist and those of $\hat{S}$ do not. Thus, the most-likely skyline of $\bar{S}$ is non-empty.

(ii) Consider the skyline of $\bar{S}$ that consists of just $Sky(S)$. Suppose that a point $s \in Sky(S)$ is replaced by a point $\hat{s} \in \hat{S}$ that dominates it. The probability expression for $Sky(S)$ contains terms β and $1-\alpha$, since s is present in it and $\hat{s}$ is not. In the probability expression for the new skyline, the term $1-\alpha$ is replaced by α, since $\hat{s}$ is present and the term β is excluded since s is not present. (The term β is not replaced by $1-\beta$ since s is dominated by $\hat{s}$. Indeed, $\hat{s}$ may also dominate other points of $Sky(S)$, so the term β for each such point is also excluded.) Since $\beta(1-\alpha) < \alpha$, the new skyline has a higher probability than the previous skyline of $\bar{S}$. The replacement process is continued until a skyline consisting of only points drawn from $\hat{S}$ is obtained. Since each replacement yields a skyline of higher probability, it follows that the most-likely skyline of $\bar{S}$ consists of only points from $\hat{S}$. Let $k \leq K$ be the number of such points from $\hat{S}$.

(iii) The points of $\hat{S}$ are mutually non-dominating since their pre-images are in $Sky(S)$. Thus, each point of $\hat{S}$ that is not in the most-likely skyline of $\bar{S}$ con-

tributes $(1-\alpha)$ to the probability of this skyline, so its probability is $\alpha^k(1-\alpha)^{K-k} = (\alpha/(1-\alpha))^k(1-\alpha)^K$. Since $\alpha < 1/2$, we have $\alpha/(1-\alpha) < 1$. Since $(1-\alpha)^K$ is fixed for a given set S and the skyline under consideration has maximum probability, it follows that k is minimum.

Suppose that there is a polynomial-time algorithm for the most-likely skyline problem in $\mathbb{R}^3$. We generate $\bar{S}$ and compute its most-likely skyline, all in polynomial time. These skyline points, which are a subset of $\hat{S}$ of some minimum size k, dominate the points of $Sky(S)$, hence also the points of S. Therefore, the pre-images of these skyline points in S (i.e., prior to boosting) are an ε-ADR of S of minimum size k and can be computed in polynomial time. This contradicts the known NP-hardness of the ε-ADR problem in $\mathbb{R}^3$ and yields the following theorem.

Theorem 1 *The most-likely skyline problem in $\mathbb{R}^d$, $d \geq 3$, is NP-hard.*

3 Inapproximability in $\mathbb{R}^d$, $d \geq 3$

Let $\mathcal{A}$ be an algorithm that computes a skyline whose probability is greater than c times the probability of the most-likely skyline, $0 < c < 1$. We call $\mathcal{A}$ a *c-approximation* algorithm. Our first result is based on the reduction in Section 2.

Theorem 2 *For $c > 1/2$, there exists no polynomial-time c-approximation algorithm, $\mathcal{A}$, for the most-likely skyline problem in $\mathbb{R}^d$, $d \geq 3$, unless $P = NP$.*

Proof. We show that $\mathcal{A}$ cannot exist for $c = \alpha/(1-\alpha)$, where $1/3 < \alpha < 1/2$. (Recall that α is the existence probability assigned to each point of $\hat{S}$ in Section 2.) The theorem follows since $\alpha > 1/3$ implies $c > 1/2$. (A similar result is stated in [8] for the most-likely convex hull problem.)

Suppose $\mathcal{A}$ exists for $c = \alpha/(1-\alpha)$. We run $\mathcal{A}$ on $\bar{S} = Sky(S) \cup \hat{S}$. The probability of the resulting skyline is greater than $\alpha/(1-\alpha)$ times the probability of the most-likely skyline of $\bar{S}$, i.e., greater than $(\alpha/(1-\alpha)) \cdot (\alpha^k(1-\alpha))^{K-k}) = \alpha^{k+1}(1-\alpha)^{K-(k+1)}$. Assume without loss of generality that the computed skyline contains points from $\hat{S}$ only. (If it contains points of $Sky(S)$, then each of these can be replaced, in polynomial time, by the corresponding boosted point in $\hat{S}$ and the probability of the resulting skyline only increases.)

How many points does the computed skyline have? Note that a skyline with $k+1$ points of $\hat{S}$ has probability $\alpha^{k+1}(1-\alpha)^{K-(k+1)}$. For each additional point of $\hat{S}$ that is included in the skyline, the probability gets multiplied by a factor $\alpha/(1-\alpha)$, which is less than 1 since $\alpha < 1/2$. It follows that the skyline computed by $\mathcal{A}$ has fewer than $k+1$ points of $\hat{S}$. Thus, the ε-ADR of S corresponding to the pre-images of the points of the skyline computed by $\mathcal{A}$ has fewer than $k+1$ points. This ε-ADR of S must be a minimum ε-ADR of S, since the latter has size k. This yields a polynomial-time algorithm for computing a minimum ε-ADR, which is not possible unless $P = NP$. □

We can use Theorem 2 and the notion of "product composability" to show that, for any $\delta > 0$, there is not even a polynomial-time $2^{-\mathcal{O}(n^{(1-\delta)})}$-approximation algorithm for the most-likely skyline problem in $\mathbb{R}^d$, $d \geq 3$, unless $P = NP$.

An optimization problem is *product composable* [8] if any given set of problem instances $I_1, \ldots, I_k$ can be combined to yield a new instance I^* whose objective function is expressible as the product of the objective functions of $I_1, \ldots, I_k$. We require that $|I^*| = \sum_{j=1}^{k} |I_j|$, that I^* is constructible in time polynomial in $|I^*|$, and that there exists a polynomial-time computable bijection between the set of feasible solutions of I^* and those of $I_1, \ldots, I_k$.

The following lemma relates product composability to inapproximability.

Lemma 3 ([8]) *If a maximization problem of size n is product composable and cannot be approximated within a constant $c < 1$ in polynomial time, then it has no polynomial-time $2^{-\mathcal{O}(n^{(1-\delta)})}$-approximation algorithm, for any $\delta > 0$.*

A proof of this lemma can be found in Appendix G of [8] (full version).

Intuitively, the lemma is proved by showing that the existence of a $2^{-\mathcal{O}(n^{(1-\delta)})}$-approximation algorithm together with product composability would imply the existence of a c-approximation algorithm. Recall that Theorem 2 has established that, for $1/2 < c < 1$, no c-approximation algorithm exists for the most-likely skyline problem, unless $P = NP$. In fact, the proof of Theorem 2 shows that this is true even for the subset of instances consisting of the set $\bar{S} = Sky(S) \cup \hat{S}$ and the associated probabilities, as defined in Section 2.

So it suffices to show that the most-likely skyline problem consisting of instances $\bar{S} = Sky(S) \cup \hat{S}$ and the aforementioned probabilities is product composable. Specifically, we form the instances $I_1, \ldots, I_k$ by partitioning $\bar{S}$ using the k points on its most-likely skyline.

Let $\hat{S}' = \{\hat{s}_1, \ldots, \hat{s}_k\} \subseteq \hat{S}$ be the points on the most-likely skyline of $\bar{S}$, sorted by non-increasing x_1-coordinates. For each $\hat{s}_i \in \hat{S}'$, we define two sets S_i and $\hat{S}_i$. S_i contains points $s_j \in Sky(S)$ such that $\hat{s}_i \succ s_j$, $\hat{s}_l \not\succ s_j$ for $l < i$ and either $(1+\varepsilon) \cdot s_j = \hat{s}_i$ or $(1+\varepsilon) \cdot s_j \notin \hat{S}'$. $\hat{S}_i$ contains the boosted points of S_i. For $1 \leq i \leq k$, let $I_i = S'_i = S_i \cup \hat{S}_i$. (See Figure 2.) It is easy to verify that $I_1, \ldots, I_k$ can be combined to form a new instance, I^*, in polynomial-time such that,

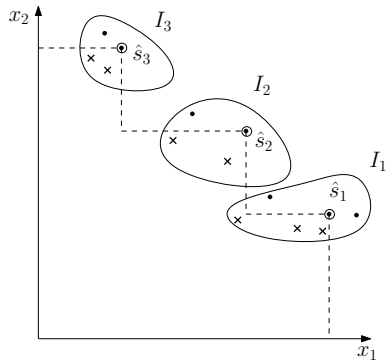


Figure 2: Example of a partition of $\bar{S}$ to form the problem instances I_1, I_2, and I_3. The points in $Sky(S)$ are represented by crosses ($\times$) and the boosted points, i.e., points in $\hat{S}$, are represented by disks ($\bullet$). Points on the most-likely skyline are circled.

given the most-likely skyline for $I_1, \ldots, I_k$, the most-likely skyline for I^* can be computed in polynomial-time, and vice-versa. This establishes product composability. Lemma 3 now yields the following result.

Theorem 4 *For any $\delta > 0$, there is no polynomial-time $2^{-\mathcal{O}(n^{(1-\delta)})}$-approximation algorithm for computing the most-likely skyline of n stochastic points in $\mathbb{R}^d$, $d \geq 3$, unless $P = NP$.*

Finally, we note that there is a simple, but uninteresting, polynomial-time 2^{-n}-approximation algorithm for the most-likely skyline problem: Simply compute the skyline of the points of S whose existence probability is more than $1/2$. (A similar observation appears in [8] for the most-likely convex hull problem.)

4 An efficient algorithm in $\mathbb{R}^2$

We now describe an algorithm to compute the most-likely skyline, $MLSky(O)$, for a set O of n points, in $\mathbb{R}^2$. Our algorithm runs in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space, which is optimal in the comparison model.

We assume w.l.o.g. that all points of O are in the first quadrant and that no two points have the same x_1- or x_2-coordinate. Let the points of O be $o_1, o_2, \ldots o_n$, in decreasing order of their x_1-coordinates. For convenience, we augment O with dummy points $o_0 = (\infty, 0)$ and $o_{n+1} = (0, \infty)$, with $p_0 = p_{n+1} = 1$. The proof of the following lemma is fairly easy, hence omitted.

Lemma 5 *For any subset O' of O, $O' = MLSky(O)$ iff $O' \cup \{o_0, o_{n+1}\} = MLSky(O \cup \{o_0, o_{n+1}\})$.*

Based on Lemma 5, we augment the input set with o_0 and o_{n+1}, and, hereafter, we will focus on finding the most-likely skyline for $\{o_0, o_1, \ldots, o_n, o_{n+1}\}$. For notational convenience, let $O_i = \{o_0, o_1, \ldots, o_i\}$.

We sweep a vertical line from right to left over O, stopping at each point o_i. At o_i we compute the most-likely skyline of O_i subject to the constraint that o_i belongs to this skyline. We denote this optimal skyline by $\mathcal{S}(o_i)$. We initialize $\mathcal{S}(o_0) = \{o_0\}$ and report $\mathcal{S}(o_{n+1}) - \{o_0, o_{n+1}\}$ as $MLSky(O)$.

Let $F(\mathcal{S}(o_i))$ be the set of points of $O_i \setminus \mathcal{S}(o_i)$ that are not dominated by any points in $\mathcal{S}(o_i)$. Then, the probability of $\mathcal{S}(o_i)$ being the skyline of O_i is

$$PrSky(\mathcal{S}(o_i)) = \prod_{o_k \in \mathcal{S}(o_i)} p_k \times \prod_{o_k \in F(\mathcal{S}(o_i))} q_k.$$

Let o_j be the point in $\mathcal{S}(o_i)$ with largest x_2-coordinate smaller than $x_2(o_i)$ and let $R(o_i, o_j) = (x_1(o_i), x_1(o_j)) \times (x_2(o_j), \infty)$. (Figure 3.) Let $F_R(\mathcal{S}(o_i))$ be the subset of $F(\mathcal{S}(o_i))$ lying in $R(o_i, o_j)$. Then,

$$PrSky(\mathcal{S}(o_i)) = p_i \times \left(\prod_{o_k \in F_R(\mathcal{S}(o_i))} q_k \times \prod_{o_k \in (\mathcal{S}(o_i) \setminus \{o_i\})} p_k \times \prod_{o_k \in (F(\mathcal{S}(o_i)) \setminus F_R(\mathcal{S}(o_i)))} q_k \right).$$

Thus, we can write

$$PrSky(\mathcal{S}(o_i)) = p_i \times score_{o_i}(o_j),$$

where $score_{o_i}(o_j)$ is the expression inside the large parentheses in the above equation.

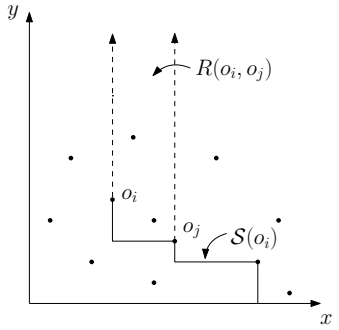


Figure 3: Illustrating $\mathcal{S}(o_i)$ and $R(o_i, o_j)$. (Dummy points o_0 and o_{n+1} are not shown.)

Since p_i is fixed for o_i and $PrSky(\mathcal{S}(o_i))$ is maximum, it follows that $score_{o_i}(o_j)$ must be maximum. For the given pair (o_i, o_j), the first term in $score_{o_i}(o_j)$, i.e., $(\prod_{o_k \in F_R(\mathcal{S}(o_i))} q_k)$, is fixed. Thus, for $score_{o_i}(o_j)$ to be maximum, the product of the second and third terms must be maximum. This product is nothing but the probability of the most-likely skyline of O_j subject to the constraint that o_j belongs to the skyline, i.e., $PrSky(\mathcal{S}(o_j))$. Hence, $score_{o_i}(o_j) = (\prod_{o_k \in F_R(\mathcal{S}(o_i))} q_k) \times PrSky(\mathcal{S}(o_j))$.

Thus the recurrence for $\mathcal{S}(o_i)$ is:

$$\mathcal{S}(o_i) = \begin{cases} \{o_i\}, & \text{if } i = 0, \\ \{o_i\} \cup \mathcal{S}(o_j), & \text{otherwise,} \end{cases}$$

where $o_j = \underset{\substack{o_j \in O_{i-1}; \\ x_2(o_i) > x_2(o_j)}}{\arg\max} \{score_{o_i}(o_j)\}$.

This recurrence leads naturally to a dynamic programming algorithm that can be implemented easily to run in $\mathcal{O}(n^2)$ time, where the run time is dominated by time to maintain the first term, i.e., $(\prod_{o_k \in F_R(\mathcal{S}(o_i))} q_k)$, in $score_{o_i}(o_j)$ for all relevant pairs (o_i, o_j). The run time can be improved to $\mathcal{O}(n \log n)$ through a more careful approach, as follows.

When the sweep is at o_i, we will, for the sake of brevity, refer to $(\prod_{o_k \in F_R(\mathcal{S}(o_i))} q_k)$, $score_{o_i}(o_j)$, and $PrSky(\mathcal{S}(o_j))$ as the *R-value*, *S-value*, and *P-value* of o_j, respectively. Note that the S-value of o_j is the product of the R-value and the P-value of o_j. Note also that the R-value and S-value of o_j depend on o_i, too, but since we refer to these when the sweep is at o_i, we omit the reference to o_i. (Similarly, if the sweep is later at some other point.)

Just after o_i is processed in the right-to-left sweep, let o be some point of O for which we wish to compute $\mathcal{S}(o)$. Let o_j be any point of O to the right of and below o_i. Then, if o is above o_j then o_i will be in the rectangle $R(o, o_j)$ and so q_i will need to be included in the R-value of o_j when the sweep reaches o. Thus, when we have finished processing o_i, we preemptively multiply by q_i the R-value of each o_j to the right of and below o_i. All such points o_j would have already been encountered in the sweep and their y-coordinates will lie in the range $[0, x_2(o_i))$, so the multiplication can be done for all o_j in this range efficiently by grouping them into a small number of sets. This observation along with a suitable data structure is the key to realizing the improved run time.

Let $\mathcal{D}$ be a data structure on O which supports the following operations when the sweepline is at o_i.

- *FindMax_S-value*(o_i): Returns the maximum of the S-values associated with the points whose y-coordinates are in the range $[0, x_2(o_i))$, along with the corresponding point o_j.
- *Set_P-value*(o_i, μ): Sets the P-value of o_i to μ. (In our algorithm, μ will be p_i times the S-value returned by *FindMax_S-value*(o_i), which is executed just before *Set_P-value*(o_i, μ).)
- *RangeMult_R-value*(o_i): Multiplies by q_i the R-values of the points whose y-coordinates are in the range $[0, x_2(o_i))$.

Note that the range $[0, x_2(o_i))$ used in *FindMax_S-value*(o_i) and *RangeMult_R-value*(o_i) may include points that are below and to the left of o_i, hence have not yet been seen in the sweep. However, $\mathcal{D}$ is set up so that the P-value (hence the S-value) of any point that has not yet been seen in the sweep is zero. Thus, such points are effectively ignored when the sweep is at o_i. This approach obviates the need to make $\mathcal{D}$ dynamic.

In Section 4.1 we show that $\mathcal{D}$ can be implemented so that it supports the above operations in $\mathcal{O}(\log n)$ time using $\mathcal{O}(n)$ space. Given this it should be clear that the algorithm runs in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space as it involves doing at each point of $o_i \in O$ (other than at o_0), one *FindMax_S-value*(o_i), one *Set_P-value*(o_i, μ), and one *RangeMult_R-value*(o_i) operation, in that order. (After each *FindMax_S-value*(o_i), $\mathcal{S}(o_i)$ is updated by including o_i in $\mathcal{S}(o_j)$.) This leads to the following conclusion.

Theorem 6 *The most-likely skyline of a set of n stochastic points in $\mathbb{R}^2$ can be computed in $\mathcal{O}(n \log n)$ time using $\mathcal{O}(n)$ space.*

4.1 Implementing the data structure $\mathcal{D}$

We implement $\mathcal{D}$ as a 1-dimensional range tree, i.e., a balanced binary search tree where the x_2-coordinates of the points of O are stored at the leaves, in increasing order from left to right. We maintain several fields at the nodes of $\mathcal{D}$, whose meanings are defined relative to the position of the sweep at the current point o_i.

During the sweep, we need to keep track of the R-value of each point o_j that is to the right of and below o_i. Rather than doing this explicitly for each such o_j, which would be expensive, we accumulate the R-value for o_j as the product of certain real numbers stored at the leaf containing o_j and the ancestors of this leaf. Specifically, let $prod(v)$ be a real-valued field at any node v. Then the R-value of o_j (relative to the current position of the sweep at o_i) is the product of the $prod(\cdot)$ fields at the leaf containing o_j and its ancestors. (Note that when the sweep is at o_i, the R-value is irrelevant for points that are to the right of and above o_i, and undefined for points that are to the left of o_i.) We initialize $prod(v)$ to 1 for all nodes of $\mathcal{D}$.

At each leaf w, besides $prod(w)$, we store three additional fields $pt(w)$, $pval(w)$, and $val(w)$. Here $pt(w)$ is the point whose x_2-coordinate is stored at w, $pval(w)$ is the P-value of the point if it has already been seen in the sweep and zero otherwise, and $val(w)$ is $prod(w) \times pval(w)$. We initialize $pval(w)$ to 1 for the leaf w corresponding to o_0 and to zero for all other leaves.

At each non-leaf v, besides $prod(v)$, we store two additional fields $val(v)$ and $pt(v)$, whose meanings are as follows: Let $\mathcal{D}(v)$ be the subtree of $\mathcal{D}$ rooted at v. Among

the leaves of $\mathcal{D}(v)$, let w be the one for which the product of $pval(w)$ and the $prod(\cdot)$ fields at w and its ancestors, up to and including v, is maximum. Then $val(v)$ stores this maximum product and $pt(v)$ equals $pt(w)$. Thus the maximum S-value among the leaves in $\mathcal{D}(v)$ is the product of $val(v)$ and the $prod(\cdot)$ fields at the proper ancestors of v. (Note that any leaf in $\mathcal{D}(v)$ corresponding to a point that has yet to be seen in the sweep cannot realize the maximum product as its $pval(\cdot)$ field is zero.)

As we will see below, when searching downwards in $\mathcal{D}$ during any of the aforementioned operations, if we are at a non-leaf node v then we will multiply the $prod(\cdot)$ field and the $val(\cdot)$ field of each child of v by $prod(v)$ and then reset $prod(v)$ to 1. This ensures that (a) at any node on the search path, the maximum S-value among the leaves in its subtree is equal to the node's $val(\cdot)$, and (b) the value that was originally in $prod(v)$ will continue to be applied to the points in the subtree of each of v's children. This as-needed, lazy approach to propagating the values in the $prod(\cdot)$ fields allows us to implement the operations efficiently.

We now describe how to do the operations on $\mathcal{D}$.

- *FindMax_S-value*(o_i): We search downwards in $\mathcal{D}$ with $x_2(o_i)$ and identify a set, $\mathcal{C}$, of *canonical nodes*, as follows: Whenever the search at a node v goes to the right child, we include the left child v' in $\mathcal{C}$. Thus, the leaves of the $\mathcal{D}(v')$'s yield a grouping of the points of O lying in the range $[0, x_2(o_i))$ into $\mathcal{O}(\log n)$ subsets.

  During the search down $\mathcal{D}$, when we are at a non-leaf node v, we update $prod(u)$ to $prod(u) \times prod(v)$ (resp. $val(u)$ to $val(u) \times prod(v)$) for each child u of v, and then reset $prod(v)$ to 1. Finally, we return the maximum of the $val(v')$'s, taken over all nodes v' in $\mathcal{C}$.

- *Set_P-value*(o_i, μ): We search downwards in $\mathcal{D}$ with $x_2(o_i)$ to find the leaf w containing o_i. At each non-leaf node v in the search, we update $prod(u)$ to $prod(u) \times prod(v)$ (resp. $val(u)$ to $val(u) \times prod(v)$) for each child u of v, and then reset $prod(v)$ to 1.

  At w, we set $pt(w)$ to o_i, $prod(w)$ to 1, and both $pval(w)$ and $val(w)$ to μ. We then walk back up $\mathcal{D}$ towards the root and, at each node v visited, we update $val(v)$ to the larger of the $val(\cdot)$ of its children and update $pt(v)$ accordingly.

- *RangeMult_R-value*(o_i): We search downwards in $\mathcal{D}$ with $x_2(o_i)$ and identify the set $\mathcal{C}$ of canonical nodes, as we did in *FindMax_S-value*(o_i). At each non-leaf node v in the search, we update $prod(u)$ to $prod(u) \times prod(v)$ (resp. $val(u)$ to $val(u) \times prod(v)$) for each child u of v, and then reset $prod(v)$ to 1. Next, for each node v' in $\mathcal{C}$, we update $prod(v')$ to $prod(v') \times q_i$. Finally, starting at the lowest node in $\mathcal{C}$ we walk back up $\mathcal{D}$ towards the root and, at each node v visited, we update $val(v)$ to the larger of the $val(\cdot)$ of its children and update $pt(v)$ accordingly.

It should be evident from the preceding discussion that $\mathcal{D}$ implements the operations correctly in $\mathcal{O}(\log n)$ time apiece and uses $\mathcal{O}(n)$ space.

5 Conclusion

Given a set of points in $\mathbb{R}^d$, where each point has a fixed probability of existence, we have considered the problem of computing the skyline that has the greatest probability of existing, i.e., the most-likely skyline. For $d > 2$, we have shown that the problem is NP-hard and, moreover, cannot even be well-approximated unless $P = NP$. For $d = 2$, we have given an optimal algorithm which runs in $\mathcal{O}(n \log n)$ time and uses $\mathcal{O}(n)$ space.

Acknowledgement

The research of the first author was supported, in part, by a Doctoral Dissertation Fellowship from the Graduate School of the University of Minnesota.

References

[1] P. Afshani, P. Agarwal, L. Arge, K. Larsen, and J. Phillips. (Approximate) uncertain skylines. In *Proc. 14th Intl. Conf. on Database Theory*, pages 186–196, 2011.

[2] M. Atallah, Y. Qi, and H. Yuan. Asymptotically efficient algorithms for skyline probabilities of uncertain data. *ACM Trans. on Database Sys.*, 36(2):1–28, 2011.

[3] S. Börzsöny, D. Kossmann, and K. Stocker. The skyline operator. In *Proc. 17th Intl. Conf. on Data Engineering*, pages 421–430, 2001.

[4] V. Koltun and C. H. Papadimitriou. Approximately dominating representatives. *Theoretical Computer Science*, 371(3):148–154, 2007.

[5] H.-T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, 1975.

[6] C. Papadimitriou and M. Yannakakis. Multiobjective query optimization. In *Proc. 20th ACM Symp. on Principles of Database Sys.*, pages 52–59, 2001.

[7] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic skylines on uncertain data. In *Proc. 33rd Intl. Conf. on Very Large Data Bases*, pages 15–26, 2007.

[8] S. Suri, K. Verbeek, and H. Yıldız. On the most likely convex hull of uncertain points. In *Proc. 21st European Symposium on Algorithms*, pages 791–802. 2013. (Full version at: https://goo.gl/fFlFbS).

[9] W. Zhang, X. Lin, Y. Zhang, M. Cheema, and Q. Zhang. Stochastic skylines. *ACM Trans. on Database Sys.*, 37(2):14, 2012.

Visibility Testing and Counting for Uncertain Segments

Mohammad Ali Abam * Sharareh Alipour† Mohammad Ghodsi ‡ Mohammad Mahdian §

Abstract

We study two well-known planar visibility problems, namely visibility testing and visibility counting, in a model where there is uncertainty about the input data. The standard versions of these problems are defined as follows: we are given a set $\mathcal{S}$ of n segments in $\mathbb{R}^2$, and we would like to preprocess $\mathcal{S}$ so that we can quickly answer queries of the form: is the given query segment $s \in \mathcal{S}$ visible from the given query point $q \in \mathbb{R}^2$ (for visibility testing) and how many segments in $\mathcal{S}$ are visible from the given query point $q \in \mathbb{R}^2$ (for visibility counting).

In our model of uncertainty, each segment may or may not exist, and if it does, it is located in one of finitely many possible locations, given by a discrete probability distribution. In this setting, the probabilistic visibility testing problem (PVTP, for short) is to compute the probability that a given segment $s \in \mathcal{S}$ is visible from a given query point q and the probabilistic visibility counting problem (PVCP, for short) is to compute the expected number of segments in $\mathcal{S}$ that are visible from a query point q. We first show that PVTP is $\#P$-complete. In the special case where uncertainty is only about whether segments exist and not about their location, we show that the problem is solvable in $O(n \log n)$ time. Using this, together with a few old tricks, we can show that one can preprocess $\mathcal{S}$ in $O(n^5 \log n)$ time into a data structure of size $O(n^4)$, so that PVTP queries can be answered in $O(\log n)$ time. Our algorithm for PVTP combined with linearity of expectation gives an $O(n^2 \log n)$ time algorithm for PVCP. We also give a faster 2-approximation algorithm for this problem.

1 Introduction

Background. Visibility testing and visibility counting are basic problems in computational geometry. Visibility plays an important role in robotics and computer graphics. In robotics, for example, the efficient exploration of an unknown environment requires computing the visibility polygon of the robot or the number of visible objects from the robot or test whether the robot sees a specific object. In some computer graphics applications, also, it is important to identify the objects in a scene that are illuminated by a light source.

Two points $p, q \in \mathbb{R}^2$ are visible from each other with respect to $\mathcal{S}$, if there exists no segment $s \in \mathcal{S}$ intersecting line segment $\overline{pq}$. We say that a segment $\overline{st} \in \mathcal{S}$ is visible from a point p, if a point $q \in \overline{st}$ can be found from which p is visible. In this paper, we consider two planar visibility problems; visibility testing and visibility counting. For a set $\mathcal{S}$ of n segments in $\mathbb{R}^2$ and a point q, in visibility testing problem, we want to test whether q sees a given segment $s \in \mathcal{S}$. In visibility counting problem we want to count the number of segments in $\mathcal{S}$ that are visible from q. For simplicity we assume all the segments are contained in a bounding box.

Uncertain data. It is not surprising that in many real-world applications we face uncertainty about the data. For geometric problems like visibility, this means uncertainty about the location of the input set. There are multiple ways to model such uncertainty. For example, we can assume each object lies inside some region, but not exactly where in that region, and use this assumption to prove bounds on the quantity of interest. Such a model is used in [14]. Alternatively, we can use a discrete probability distribution to model uncertainty. This "stochastic" approach is used in [1, 11]. We choose the latter approach in this paper. In particular, our model of uncertainty is very similar to the model used in [11].

Related work. There is significant prior work on the non-stochastic version of the problems studied in this paper. There are some works dedicated not only to the exact computing [5, 12, 15] of the problem but also to approximate computing [3, 4, 9, 12]. In both, time-space trade-offs haven been considered.

In real application there are situations where we need to model the problems based on uncertain data (See [1, 14, 10]). In [6], they compute visibility between imprecise points among obstacles. This leads us to define the uncertain model of VTP and VCP and propose algorithms to solve them.

*Computer Engineering Department, Sharif University of Technology

†Institute for Research in Fundamental Sciences (IPM) School of Computer Science

‡Computer Engineering Department, Sharif University of Technology and Institute for Research in Fundamental Sciences (IPM) School of Computer Science

§Google research

Problem statement. Suppose we are given a set $\mathcal{S}$ of n uncertain segments. More precisely, we are given a discrete probability distribution for each $s_i \in \mathcal{S}$, that is, we have a set $\mathcal{D}_i = \{s_{i,1}, \cdots, s_{i,m_i}\} \cup \{s_{i,0} = \perp\}$ of possible locations with associated probabilities $p_{i,j}$ such that $\mathbf{Pr}(s_i = s_{i,j}) = p_{i,j}$ and $\sum_j p_{i,j} = 1$. The special segment $\perp$ indicates that the segment s_i does not exist in $\mathcal{S}$. In this setting, the set $\mathcal{S}$ can be seen as a random variable (or random set) as it consists of probabilistic segments. This random variable gets its value from a sample space of size $\Pi_i(m_i + 1)$ with the probability being equal to $\Pi_{s\in\mathcal{S}}\mathbf{Pr}(s)\Pi_{s\notin\mathcal{S}}(1-\mathbf{Pr}(s))$. To this end, assume $z = \max\{1 + m_i\}$, i.e., z denotes the maximum size of the given distributions. A special case that we will pay special attention to is when $z = 2$. This is the case where the uncertainty is only about the existence of the segments, and not about their location.

It is natural to define the probabilistic version of visibility testing and visibility counting problems in the above setting where $\mathcal{S}$ is a random set:

- Probabilistic Visibility Testing Problem (PVTP): compute the probability that a given segment $s \in \mathcal{S}$ is visible from a given query point q.
- Probabilistic Visibility Counting Problem (PVCP): compute the expected number of segments in $\mathcal{S}$ being visible from q.

Our results. We first show that PVTP is $\#P$-complete. We then turn our attention to the special case where $z = 2$. We present an algorithm running $O(n \log n)$ time that answers PVTP. Then, we present a simple way of putting n uncertain segments into a data structure of size $O(n^4)$ such that queries can be answered in $O(\log n)$ time. Finally, we focus our attention to PVCP whose complexity class is unknown to us. Here, we present a polynomial-time 2-approximation algorithm that approximately solves PVCP. We then show how to preprocess $\mathcal{S}$ into a data structure of size $O(n^4)$ in order to approximately answer each query in $O(\log n)$ time.

2 Probabilistic visibility testing

We start by a simple polynomial-time reduction from #perfect-matching problem to PVTP in order to show PVTP is $\#P$-complete. The #perfect-matching problem of computing the number of perfect matching in a given bipartite graph, is known to be $\#P$-complete [13] even for 3-regular bipartite graphs [8]. We next explain the details.

Suppose a bipartite graph $G = (U, V, E)$ is input to #perfect-matching problem where $U = \{u_1, \cdots, u_n\}$ and $V = \{v_1, \cdots, v_n\}$ are vertex parts of G and E is the edge set of G. For the given bipartite graph, we construct an instance of PVTP and introduce a query point q and a query segment s such that each perfect matching uniquely corresponds to one element of the sample space of uncertain segments in which s is not visible from q. Consider n intervals $[i, i + 1]$ on the x-axis where i changes from 0 to $n - 1$. Imagine the interval $[i, i + 1]$ corresponds to the vertex v_i; denoted by $I(v_i)$. For each vertex $u_i \in U$, we define an uncertain segment $\mathcal{D}_i = \{I(v_j) | \{u_i, v_j\} \in E\}$ with the uniform distribution—note that in this instance each uncertain segment always exist. We add one more uncertain segment s consisting of one segment with probability 1 whose endpoints are $(0, -1)$ and $(n, -1)$. To this end, consider the query point q is $(n/2, n)$ (See figure 1).

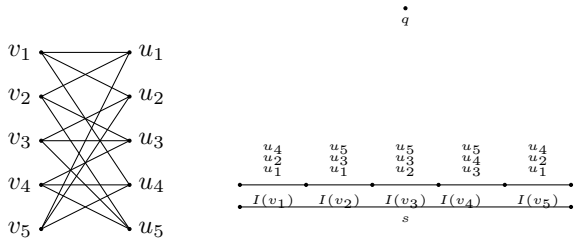


Figure 1: Each matching in the left side corresponds to a set of segments that cover s in the right side and each set of segments that cover s corresponds to a matching.

Segment s is not visible from q iff the interval $[0, n]$ is completely covered by the uncertain segments defined on the x-axis. There are n such uncertain segments and each covers exactly 1 unit of $[0, n]$. Therefore, each uncertain segment must cover exactly one of n unit intervals. So, the number of perfect matchings is equal to the number of ways that s is covered by the uncertain segments. This is the intuition behind one-to-one correspondence between perfect matching and the subset of the sample space in which s is not visible from q. Therefore, we conclude the following theorem.

Theorem 1 *PVTP is $\#P$-complete.*

From now on, we restrict ourself to the special case where $z = 2$, i.e., each uncertain segment either does not exist or exists in only one possible location. Suppose we are given n uncertain segments $s_1, \cdots, s_n$. Let $\mathbf{Pr}(s_i \in \mathcal{S}) = p_i$ which of course implies $\mathbf{Pr}(s_i \notin \mathcal{S}) = 1 - p_i$.

Next, we explain how to compute $\mathbf{Pr}(q \text{ sees } s)$ for the given segment s and point q. If $s \notin \mathcal{S}$, q of course can not see s. Therefore, $\mathbf{Pr}(q \text{ sees } s) = \mathbf{Pr}(q \text{ sees } s | s \in \mathcal{S})\mathbf{Pr}(s \in \mathcal{S})$. This reduces our task to computing of $\mathbf{Pr}(q \text{ sees } s | s \in \mathcal{S})$. Let Δ be a triangle with vertex q and side s. Every other uncertain segment that does not intersect Δ, can not prevent q to see s. Therefore, we can restrict ourself to uncertain segments intersecting Δ. We project these uncertain segments to s with respect to q. Now, as the main ingredient, we must solve

the following problem (See figure 2):

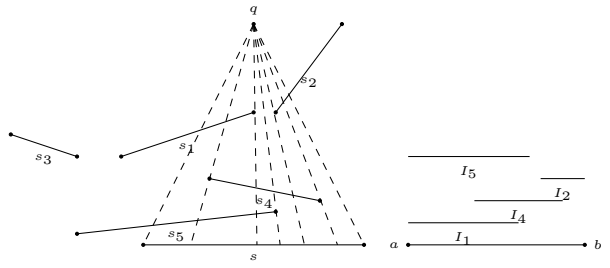


Figure 2: The projection of uncertain segments on s according to q defines four uncertain intervals.

- Suppose we are given n uncertain intervals $I = \{I_1, \cdots, I_n\}$ on the real line; each I_i exists with probability p_i. Compute the probability that the given interval $[a, b]$ is covered by the uncertain intervals, denoted by $\mathbf{Pr}([a, b]$ is covered).

Computing the desired probability seems needs $\Theta(2^n)$ time as the size of the sample space can be $\Theta(2^n)$ in the worst case. But, we next show how the dynamic paradigm helps us to perform the computation in $O(n \log n)$ time. For simplicity, we can assume the intervals have been sorted by their right endpoints and intersection of each I_i with $[a, b]$ is not empty. Let $r(I_i)$ ($l(I_i)$) be the right (left) endpoint of I_i. We present the following recursive formula.

For each point $a' \in [a, b]$, let $sol(a')$ be the probability that $[a', b]$ is covered. So, $sol(a)$ is the probability that $[a, b]$ is covered. Let $S(a') = \{I'_1, ..., I'_l\}$ be the set of intervals that cover a' and they are sorted according to their right endpoints (See figure 3).

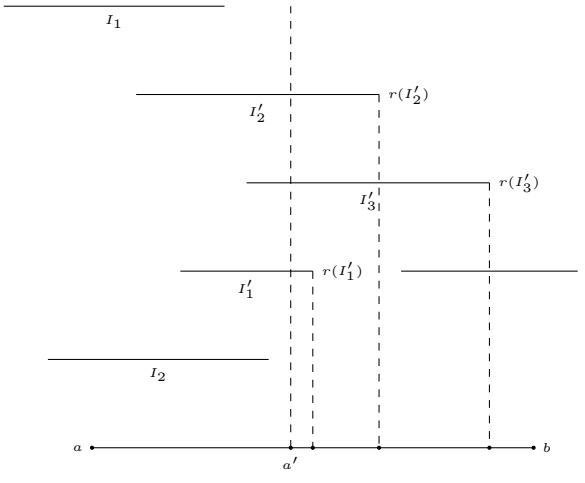


Figure 3: I'_1, I'_2 and I'_3 are the intervals that can cover $[a, b]$, so we have $sol(a') = p'_1 sol(r(I'_1)) + p'_2(1 - p'_1)sol(r(I'_2)) + p'_3(1 - p'_2)(1 - p'_1)sol(r(I'_3))$.

Lemma 2 *We define $sol(b) = 1$, then we have*

$$sol(a') = \sum_{j=1}^{l} p'_j(\prod_{i=1}^{j-1}(1 - p'_i))sol(r(I'_j)).$$

Proof. Suppose that $a' \in [a, b]$, so if $[a', b]$ is covered, then at least one of the segments in $S(a')$ should be chosen. There are l segments that cover a'. Since the segments in $S(a')$ are sorted according to their right endpoints then, the probability that I'_j is the first segment that covers a' is $p'_j \prod_{i'=1}^{j-1}(1-p'_i)$. Recursively $[a', b]$ is covered with the probability of $sol(r(I'_j))$. So, we have

$$sol(a') = \sum_{j=1}^{l} p'_j(\prod_{i'=1}^{j-1}(1 - p'_i))sol(r(I'_j)).$$

□

Each right endpoint of the intervals can be covered by $O(n)$ of the intervals. In the recursive formula, we call each right endpoint at most once. For each $sol(r(I'_j))$ we have to compute $\prod_{i'=1}^{j-1}(1 - p'_i)$, since the segments are sorted according to their right endpoint, for each $sol(r(I'_j))$ we multiply $\prod_{i'=1}^{j-2}(1 - p'_i)$(the value of previous step) by $1-p'_j$, which means we can compute $sol(a)$ in $O(n^2)$ time. Next we propose a faster algorithm.

To fill the array sol, we sweep the endpoints from right to left and keep the track of all intervals intersecting the sweep line in a binary search tree (BST, for short) over the right endpoint of intervals supporting insertion/deletion in $O(\log n)$ time. We augment each node of the BST with extra values in order to expedite our computation as we explain next.

Upon processing a right endpoint, say $r(I_i)$, we compute $sol(r(I_i))$, which is the sum of all the nodes of tree. This can be computed in $O(\log n)$ time. Then, we implicitly multiply all the nodes by $(1 - p_i)$ and then add $r(I_i)$ to the tree with the value of $p_i sol(r(I_i))$. For the left endpoint of an interval, $l(I_i)$, we delete I_i, from the tree and implicitly divide all the right endpoints greater than $r(I_i)$ by $(1-p_i)$. This also can be done in $O(\log n)$ time. There are $O(n)$ endpoints, so the running time is $O(n \log n)$.

Theorem 3 *Given a point and a segment, PVTP can be answered in $O(n \log n)$ time when $z = 2$.*

Now, we preprocess the segments such that for any given query point q, PVTP can be answered in $O(\log n)$ time. First, connect each pair of the endpoints by a line and extend it until it hits the bounding box. These lines will partition the bounding box into $O(n^4)$ regions. For a fixed segment $s \in \mathcal{S}$, the answer to PVTP for all the points in a given region is the same, because the combinatorial order of segments that cover s is the same for all the points inside that region. Therefore, in the preprocessing time we choose a point q_i from each region

r_i and compute $\mathbf{Pr}(q_i \text{ sees } s)$ in $O(n \log n)$ time. So, for a given set of segments $\mathcal{S}$ and a segment $s \in \mathcal{S}$, we preprocess the segments in $O(n^5 \log n)$ time and $O(n^4)$ space such that for any given query point q, we locate the region r_i containing q in $O(\log n)$ time and return $\mathbf{Pr}(q_i \text{ sees } s) = \mathbf{Pr}(q \text{ sees } s)$.

3 Probabilistic visibility counting

In this section we study the probabilistic visibility counting problem. We start with a few notations. For each subset $T \subset \mathcal{S}$, let $m_q(T)$ be the number of segments visible from q when the set of segments is T. So, the expected number of segments visible from q can be written as: $E(m_q) = \sum_{T \subseteq \mathcal{S}} \mathbf{Pr}(T) m_q(T)$, where $\mathbf{Pr}(T)$ denotes the probability that the set of realized segments is T.

Another way to compute $E(m_q)$ is using linearity of expectations: $E(m_q) = \sum_{i=1}^{n} \mathbf{Pr}(q \text{ sees } s_i)$.

For the case $z = 2$, we can use the above identity and the algorithm in the previous section to compute $E(m_q)$ in $O(n^2 \log n)$ time with no preprocessing. Also as in the previous section, we can use preprocessing to reduce query time: the answer of $PVCP$ is the same for all the points in each region in the space partition. So, we can compute this number for all the regions in $O(n^6 \log n)$ preprocessing time and $O(n^4)$ space, such that for any query point q, $E(m_q)$ can be answered in $O(\log n)$ time. Now, we show how to approximately solve this problem more efficiently.

3.1 Approximation of PVCP

In this section we propose a 2-approximation solution for PVCP. First, we present the following theorem

Theorem 4 [4] *Let S be a set of disjoint line segments in the plane and ve_q be the number of visible endpoints of the segments and m_q be the number of visible segments, then we have*

$$m_q \leq ve_q \leq 2m_q$$

Now, we use Theorem 4 to approximate PVCP. Let $m_q(T)$ and $ve_q(T)$ be the number of visible segments and visible endpoints, respectively in $T \subset \mathcal{S}$ w.r.t T, so we have $m_q(T) \leq ve_q(T) \leq 2m_q(T)$. So, we can conclude that,

$$\begin{aligned}\sum_{T \subset \mathcal{S}} \mathbf{Pr}(\mathcal{S} = T) m_q(T) &\leq \sum_{T \subset \mathcal{S}} \mathbf{Pr}(\mathcal{S} = T) ve_q(T) \\ &\leq \sum_{T \subset \mathcal{S}} \mathbf{Pr}(\mathcal{S} = T) 2m_q(T).\end{aligned}$$

Or in other words,

$$E(m_q) \leq E(ve_q) \leq 2E(m_q).$$

So, we compute

$$E(ve_q) = \sum_{i=1}^{n} \mathbf{Pr}(r(s_i) \text{ sees } q) + \mathbf{Pr}(l(s_i) \text{ sees } q).$$

We have

$$\mathbf{Pr}(r(s_i) \text{ sees } q)) = \textstyle\sum_{j=1}^{z} p_{i,j} \mathbf{Pr}(r(s_{i,j}) \text{ sees } q).$$

Let $s_{k,1'}, s_{k,2'}, ..., s_{k,l'}$ be the possible locations of s_k in $\mathcal{D}_k$ that cross $\overline{r(s_{i,j})q}$, the probability that s_k does not intersect $\overline{r(s_{i,j})q}$ is $p_k^{i,j} = (1 - p_{k,1'} - p_{k,2'} - ... - p_{k,l'})$.

$$\mathbf{Pr}(q \text{ sees } r(s_i)) = \textstyle\sum_{j=1}^{z} p_{i,j} p_1^{i,j} p_2^{i,j} .... p_n^{i,j}$$

We have $2nz$ possible locations for the endpoints and we can compute $P(q \text{ sees } r(s_i))$ in $O(zn)$, so $E(ve_q)$ is computed in $O(n^2 z^2)$.

For $z = 2$ we present a faster algorithm. Suppose that $a \in s_i$ is an endpoint of s_i. Let $s'_1, s'_2, ..., s'_k$ be the set of segments that intersect $\overline{aq}$, since the probability of selection of the segments are independent, we have

$$\mathbf{Pr}(q \text{ sees } a) = p_i(1 - p'_1)(1 - p'_2)...(1 - p'_k).$$

Which yields: $E(ve_p) = \sum_{a \in s_i} \mathbf{Pr}(q \text{ sees } a)$.

So, for each endpoint, we need the segments that intersect $\overline{aq}$. We use the following theorem:

Theorem 5 [2, 7] *Let S be a set of n segments in the plane and $n \leq k \leq n^2$, we can preprocess the segments in $O_\epsilon(k)$ such that for a given query segment s, the number of segments crossed by s can be computed in $O_\epsilon(n/\sqrt{k})$. Where $O_\epsilon(f(n)) = O(f(n)n^\epsilon)$ and $\epsilon > 0$ is a constant that can be made arbitrarily small.*

By Theorem 5 we can compute $\mathbf{Pr}(q \text{ sees } a)$ in $O(n/\sqrt{k})$. So, for $2n$ endpoints, $E(ve_p)$ is computed in $n \cdot O(n/\sqrt{k})$. If $k = n^{\frac{4}{3}}$, then we have:

Theorem 6 *Let, S be a set of given segments and q be a given point. If each segment is chosen with probability p_i, then, the expected number of visible endpoints from q can be computed in $O_\epsilon(n^{\frac{4}{3}})$ which is a 2-approximation of $E(m_q)$.*

4 Conclusion

We introduced a probabilistic variant of two well known visibility problems: visibility testing and counting. We proved that visibility testing problem in general case is $\#P$-complete. Then, we proposed a polynomial time for a special case of these problems and then gave an approximation algorithm for the probabilistic visibility counting problem. In future we want to study the complexity of these problems in some other special cases. Also, we want to study algorithms to approximate the answer of probabilistic visibility testing problem.

Acknowledgments

We thank Mahdi Safarnejad for his comments and helps.

References

[1] M. A. Abam, M. de Berg, and A. Khosravi. Piecewise-linear approximations of uncertain functions. In *Algorithms and Data Structures - 12th International Symposium, WADS 2011, New York, NY, USA, August 15-17, 2011. Proceedings*, pages 1–12, 2011.

[2] P. K. Agarwal and M. Sharir. Applications of a new space-partitioning technique. *Discrete & Computational Geometry*, 9:11–38, 1993.

[3] S. Alipour, M. Ghodsi, A. Zarei, and M. Pourreza. Visibility testing and counting. *Inf. Process. Lett.*, 115(9):649–654, 2015.

[4] S. Alipour and A. Zarei. Visibility testing and counting. In *Proceedings of the 5th Joint International Frontiers in Algorithmics, and 7th International Conference on Algorithmic Aspects in Information and Management*, FAW-AAIM'11, pages 343–351, Berlin, Heidelberg, 2011. Springer-Verlag.

[5] T. Asano. An efficient algorithm for finding the visibility polygon for a polygonal region with holes. *IEICE TRANSACTIONS (1976-1990)*, 68(9):557–589, 1985.

[6] K. Buchin, I. Kostitsyna, M. Löffler, and R. I. Silveira. Region-based approximation of probability distributions (for visibility between imprecise points among obstacles). *CoRR*, abs/1402.5681, 2014.

[7] S. Cheng and R. Janardan. Algorithms for ray-shooting and intersection searching. *J. Algorithms*, 13(4):670–692, 1992.

[8] P. Dagum, M. Luby, M. Mihail, and U. V. Vazirani. Polytopes, permanents and graphs with large factors. In *29th Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988*, pages 412–421, 1988.

[9] J. Gudmundsson and P. Morin. Planar visibility: testing and counting. In *Proceedings of the 26th ACM Symposium on Computational Geometry, Snowbird, Utah, USA, June 13-16, 2010*, pages 77–86, 2010.

[10] M. Löffler and M. J. van Kreveld. Largest bounding box, smallest diameter, and related problems on imprecise points. *Comput. Geom.*, 43(4):419–433, 2010.

[11] A. Munteanu, C. Sohler, and D. Feldman. Smallest enclosing ball for probabilistic data. In *30th Annual Symposium on Computational Geometry, SOCG'14, Kyoto, Japan, June 08 - 11, 2014*, page 214, 2014.

[12] S. Suri and J. O'Rourke. Worst-case optimal algorithms for constructing visibility polygons with holes. In *Proceedings of the Second Annual Symposium on Computational Geometry*, SCG '86, pages 14–23, New York, NY, USA, 1986. ACM.

[13] L. G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[14] M. J. van Kreveld and M. Löffler. Approximating largest convex hulls for imprecise points. *J. Discrete Algorithms*, 6(4):583–594, 2008.

[15] G. Vegter. The visibility diagram: a data structure for visibility problems and motion planning. In J. R. Gilbert and R. G. Karlsson, editors, *SWAT*, volume 447 of *Lecture Notes in Computer Science*, pages 97–110. Springer, 1990.

Nearest-Neighbor Search Under Uncertainty

Boris Aronov* John Iacono* Khadijeh Sheikhan*

Abstract

We study the problem of Nearest-Neighbor Searching under locational uncertainty. Here, an uncertain query or site consists of a set of points in the plane, and their distance is defined as distance between the two farthest points within them. In L_∞ metric, we present an algorithm with $O(n \log^2 n + s)$ expected preprocessing time, $O(n \log n)$ space, and $O(\log^2 n + k)$ query time, where s is the total number of site points, n is the number of sites, and k is the size of the query. We also propose a $\sqrt{2}$-approximation algorithm for the L_2 version of the problem.

1 Introduction

In this paper, our focus is on *Nearest-Neighbor (NN) Searching Under Uncertainty*. In the basic version of the NN problem, one wants to preprocess a set of site points in the plane, so that the closest one to a query point can be found efficiently.

For a brief survey on NN searching under uncertainty, refer to [18]. Two models of uncertainty have been considered in the literature. In the *existential model*, which we will not address, a site has a specified location, but it appears with a given probability and otherwise is not present at all. The model that we are interested in is the *locational model*, where a site and/or a query consists of more than one point, for example a region or a finite set of points representing possible locations of the uncertain point. An application of this model is location-based services where the data is imprecise. The distance of an uncertain query from an uncertain site is defined as an aggregate function of distances of points within them, such as maximum, minimum, or average of all possible distances.

For most of this paper, $d(.,.)$ is the L_∞ distance, and sites and queries are uncertain; an uncertain site or query consists of a finite set of possible locations in the plane, i.e., points, and the distance between them is defined as the maximum of all possible distances (see Figure 1).

*NYU Tandon School of Engineering, Brooklyn, NY, 11201, USA, {boris.aronov,iacono,khadijeh}@nyu.edu. Work of B.A. on this paper has been partially supported by NSF Grants CCF-11-17336, CCF-12-18791, and CCF-15-40656, and by BSF grant 2014/170. Work by K.S. on this paper has been supported by NSF Grants CCF-12-18791 and CCF-13-19648. Work of J.I. on this paper has been supported by NSF Grant CCF-13-19648.

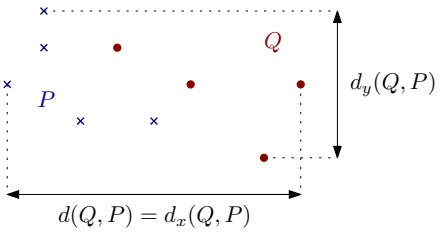


Figure 1: Distance of an uncertain query Q (red dots) from an uncertain site P (blue crosses).

Throughout this paper, for simplicity of presentation, we assume general position, which here means no two points share their x- or y-coordinates.

Problem statement We define the *distance* $d(P,Q)$ between two compact point sets P, Q in the plane by

$$d(P,Q) = \max_{p \in P, q \in Q} d(p,q).$$

Given a set $\mathcal{P} = \{P_1, P_2, ..., P_n\}$ of n uncertain points $P_i \subseteq 2^{\mathbb{R}^2}$, with $s = \sum_{P_i \in \mathcal{P}} |P_i|$, construct a data structure $D(\mathcal{P})$, such that queries of the form $\mathrm{NN}_{D(\mathcal{P})}(Q)$, for $Q \subseteq 2^{\mathbb{R}^2}$, $k = |Q|$, that return $\arg\min_i d(P_i, Q)$, can be answered efficiently.

Motivation Our definition of distance between a query and a site as the largest separation between any two representatives of the respective sets can be motivated by the following slightly artificial example: Each site represents the set of possible locations of an ambulance. The query represents the set of possible locations of a 911 caller. The answer to the query is the ambulance that is closest to the caller, using the worst-case combination of the positions of the ambulance and the caller. In other words, it minimizes the worst-case response time given the uncertain locations.

Our results We first propose an algorithm to find the nearest site in $O(\log n + k)$ time, using quadratic space and $O(n^2 \log n + s)$ expected preprocessing time. Then we improve the space requirement to near linear by adding a logarithmic factor to the query time and propose a data structure smoothly interpolating between the two extremes. The nearest neighbor in L_∞ metric is also a $\sqrt{2}$-approximate solution for the L_2 version of the

problem. Finally, we propose a more efficient algorithm to find a $\sqrt{2}$-approximate answer to the L_2 version.

2 Related Work

The nearest-neighbor problem has been studied extensively in the literature. A well-known approach is subdividing the plane into cells, each consisting of points with the same nearest neighbor. This subdivision of the plane is called a *Voronoi Diagram*. A Voronoi diagram is then preprocessed for point location, in order to answer NN queries. This diagram has been extensively studied for different types of sites such as segments or polygons, using different metrics and also in higher dimensions [15].

NN searching under uncertainty has been studied in a variety of settings. In the case where queries are points but sites are uncertain, modified versions of Voronoi Diagram are proposed. An *Uncertain Voronoi Diagram* is a subdivision of space into regions, so that all the points in each region have the same *set* of possible nearest neighbors [4]. Zhang et al. propose the notion of *Possible Voronoi cell (PV-cell)* [17]. The PV-cell of a site is the region where that site has positive probability of being the nearest neighbor. Evans et al. introduce another version of Voronoi diagram where each cell contains those points guaranteed to be closest to a particular site [6].

When distance from a query point to an uncertain site is defined as the distance to the site's farthest point, it is equal to the *Hausdorff* distance from the query point to the site, so NN searching can be done using Hausdorff Voronoi Diagrams (HVD). Papadopoulou proves that the size of the HVD is $O(n^2)$ where n is the total number of vertices on the convex hulls of the sites. She provides a plane sweep algorithm to construct it [14]. If the convex hulls of sites are disjoint then the HVD's size is $O(n)$ and can be computed in $O(n \log^3 n)$ time [5]. By performing a point-location query in the HVD, the nearest uncertain site can be found in poly-logarithmic time.

Agarwal et al. cover different cases of uncertainty in their work [2]. In their setting a site or query point is specified as a probability density function (pdf) and the goal is to find the *Expected Nearest Neighbor (ENN)* which is the site with minimum expected distance to the query. Under squared Euclidean distance, they prove that if the pdf of each site has description complexity at most k, the *Expected Voronoi Diagram (EVD)* has linear size and can be computed in $O(n \log n + nk)$ time. Thus, an ENN query can be answered in $O(\log n)$ using point location in the EVD. Using rectilinear metrics and assuming each site has a discrete pdf consisting of k points, they provide an algorithm to answer queries in $O(\log^3(kn))$ time by doing a point location query. For the Euclidean distance they construct an ε-approximation of the EVD (ε-EVD), and the ε-approximation of the ENN (ε-ENN) can be reported in $O(\log(n/\varepsilon))$. In another work [1], Agarwal et al. propose an algorithm to find those sites that can be the NN with probability greater than a threshold and an algorithm to report the point that has the maximum probability of being the NN.

Many results on the NN problem use branch-and-bound pruning techniques. These methods mostly use R-trees to index the sites, try to prune nodes, and use heuristics to make the process more efficient, but there is no guarantee that it runs asymptotically faster than linear-time brute-force algorithm, in the worst case.

In *Aggregate Nearest-Neighbor Searching* an aggregate or group query consists of a finite set of points, and distance is an aggregate function of all the distances, such as their sum, maximum, or average. This type of queries can be viewed as an equivalent to our variant of uncertain queries. Dealing with sum version of aggregate queries under Euclidean distance, Papadias et al. use R-trees to create an index on the set of sites [12]. They propose several empirical algorithms using this data structure. They also provide a modification of those algorithms to work efficiently for disk-resident queries. In another work [13], previous algorithms are modified to cover other variants of the aggregate NN problem such as the sum, max, and min versions. Again, input sites are indexed using R-trees, algorithms are evaluated by experiments, and no worst-case analysis is provided.

To answer the aggregate-max NN query on a set of n sites in the plane, Wang provides algorithms for L_1 and L_2 metrics that give exact answers in sub-linear time [16]. For the L_1 version, he builds a linear-size data structure in $O(n \log n)$ time that answers a query of size k in $O(\log n + k)$ time. For L_2, constructing the data structure takes $O(n \log n)$ time and $O(n \log \log n)$ space and a query can be answered in $O(k\sqrt{n} \log^{O(1)} n)$ time. He also proposes another data structure for L_2, which takes $O(n^{2+\varepsilon})$ time and space and guarantees $O(k \log n)$ query time.

As an example of uncertainty for both queries and sites, Lian et al. introduce *Probabilistic Group Nearest Neighbor (PGNN)*, which are aggregate NN queries in uncertain data sets [10]. Their approach is reducing the search space by proposing pruning methods. They demonstrate the efficiency of their algorithm experimentally, with no analysis provided. Our results are closely related to the PGNN problem, except that we provide preprocessing and query time analysis.

3 Exact Nearest Neighbor for L_∞

First we observe that we can find the nearest neighbor using only the axis-parallel bounding box of uncertain sets; the exact position of points within the box are not needed (see Figure 2). Therefore, after computing the bounding boxes, neither preprocessing nor query time will depend on the sizes of sites or queries.

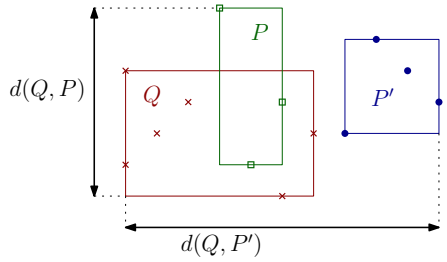


Figure 2: Distance of two uncertain points only depends on their axis-aligned bounding boxes.

Lemma 1 *For any two compact sets P, Q in the plane, $d(P,Q) = d(b(P), b(Q))$, where $b(\cdot)$ denotes the axis-parallel bounding box of a set.*

Proof. Since $P \subset b(P)$, $Q \subset b(Q)$, and the distance between sets is defined as the *longest* interpoint distance, we have $d(b(P), b(Q)) \geq d(P,Q)$.

Now suppose $d(b(P), b(Q)) = |p_x - q_x|$, for $p \in b(P)$, $q \in b(Q)$; p, q must lie on bounding box boundaries, for otherwise $|p_x - q_x|$ can be increased. By definition of a bounding box, there exist $p' \in P$, $q' \in Q$ with $p'_x = p_x$, $q'_x = q_x$. Thus $d(b(P), b(Q)) = |p'_x - q'_x| \leq d(P,Q)$. □

So, our problem is reduced to the nearest-neighbor problem for axis-parallel boxes. For the remainder of this paper, we will assume that a site or query is given by its axis-aligned bounding box, specified by its four coordinates. First we will define properties of a query rectangle based on these four values.

For a given query rectangle $Q(x_1, x_2, y_1, y_2)$, its center is $\odot = \odot(Q) = (x_\odot, y_\odot) = ((x_1{+}x_2)/2, (y_1{+}y_2)/2)$. The width and height are defined as $\Delta_x = \Delta_x(Q) = x_2 - x_1$ and $\Delta_y = \Delta_y(Q) = y_2 - y_1$, respectively, as shown in Figure 3. We also define $\Delta = \Delta(Q) = |\Delta x - \Delta y|$ to be

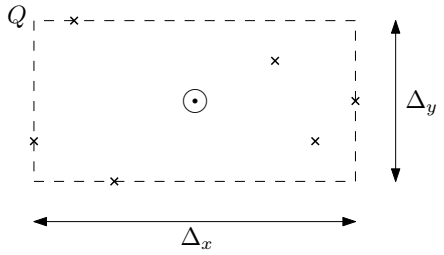


Figure 3: Center, width, and height of a query.

the absolute difference between the height and width of Q.

Without loss of generality, we assume that width of Q is greater than or equal to the height, so $\Delta = \Delta_x - \Delta_y \geq 0$; a symmetric structure will handle the complementary case. We partition the set of sites into left and right subsets, named $\mathcal{P}_\ell$ and $\mathcal{P}_r$, based on the position of their center relative to the vertical line through $\odot$. When $\Delta_x < \Delta_y$, 'right' and 'left' will be replaced by 'top' and 'bottom' in all of the arguments.

Observation 1 *The left edge $e_\ell = e_\ell(P)$ of any rectangle site $P \in \mathcal{P}_\ell$ is sufficient to compute the distance from the query Q; in other words, $d(P,Q) = d(e_\ell, Q)$. A symmetric statement holds for $\mathcal{P}_r$.*

So sites can be replaced by either vertical or horizontal segments, based on the width and height of the query, and their dimension decreases by one. See Figure 4. Next we explain how to effectively decrease the dimension of a query by replacing it with a point.

Lemma 2 *The nearest site in $\mathcal{P}_\ell$ to the rectangle Q, is the same as the nearest site to its center $\odot$ shifted by $\Delta/2$ to the right.*

Proof. First, we compute the distance of a site $P \in \mathcal{P}_\ell$ from a query Q using the query's center, width, and height:

$$\begin{aligned} d(P,Q) &= d(e_l, Q) = \max(d_x(e_l, Q), d_y(e_l, Q)) \\ &= \max(d_x(e_l, \odot) + \frac{\Delta_x}{2}, d_y(e_l, \odot) + \frac{\Delta_y}{2}). \end{aligned}$$

We assumed $\Delta_x \geq \Delta_y$, thus

$$d(P,Q) = \frac{\Delta_y}{2} + \max(d_x(e_l, \odot) + \frac{\Delta}{2}, d_y(e_l, \odot)).$$

Since e_l is to the left of $\odot$, if we shift $\odot$ to the right by $\Delta/2$, to the point $\odot' = (x_\odot + \Delta/2, y_\odot)$, then

$$\begin{aligned} d(P,Q) &= \frac{\Delta_y}{2} + \max(d_x(e_l, \odot'), d_y(e_l, \odot')) \\ &= \frac{\Delta_y}{2} + d(e_l, \odot') \end{aligned}$$

So, the distance from Q differs from the distance from $\odot'$ by $\Delta_y/2$. This proves that the nearest site to the rectangle Q among those in $\mathcal{P}_\ell$, is the same as the nearest one to the point $\odot'$. □

Now we query in $\mathcal{P}_\ell$ and $\mathcal{P}_r$ separately, using an appropriately shifted center of Q, to find the nearest site in each set. Then we can compare their distance from Q to find the real nearest neighbor.

Theorem 3 *The nearest site to a query Q can be found in $O(\log n)$ time, using $O(n^2)$ space and $O(n^2 \log n)$ expected preprocessing time.*

Proof. Given a set of n vertical segments, we need the closest to a point. Assuming general position, the segments are pairwise disjoint and our definition of the distance (a variant of the Hausdorff metric) satisfies the axioms of an *Abstract Voronoi Diagram* [11]. Therefore

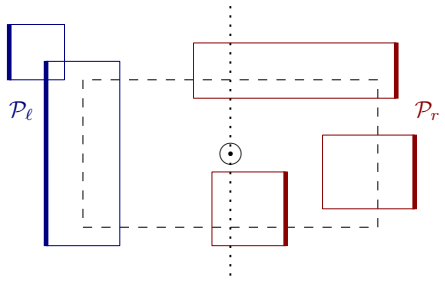


Figure 4: Dividing the sites into sets $\mathcal{P}_\ell$ and $\mathcal{P}_r$. The highlighted edges are sufficient for measuring the distance from the query.

the diagram has linear size and can be constructed in $O(n \log n)$ expected time using linear space [8]. We then preprocess it for logarithmic-time point location.

We focus on finding the nearest site in $\mathcal{P}_\ell$. A site in $\mathcal{P}_\ell$ is a site with the x-coordinate of its center smaller than $\odot_x$. We presort the sites in $\mathcal{P}$ according to the x-coordinate of their centers and construct the Voronoi diagram of every prefix, for a total of n diagrams with total size $O(n^2)$, in expected time $O(n^2 \log n)$. When answering a query, we binary search using $\odot_x$ to find the Voronoi diagram corresponding to $\mathcal{P}_\ell$ and query in the diagram with $\odot'$ to find the nearest site. We repeat this process for $\mathcal{P}_r$, then compare the results in constant time and return the better of the two answers. Answering a query involves a constant number of binary searches and point locations, so the query time is $O(\log n)$. □

Theorem 4 *The nearest site to a query Q can be found in $O(\log^2 n)$ time, using $O(n \log n)$ space and $O(n \log^2 n)$ expected preprocessing time.*

Proof. To answer a query with center $\odot$, we need to find the nearest left (right) edge of sites the x-coordinate of whose center is less (greater) than $\odot_x$, which is a prefix (suffix) of sites sorted by their centers. This problem is an example of a *decomposable searching problem* introduced by Bentley and Saxe [3], i.e., to find NN in some set, we can decompose it into smaller sets, search for NN in each subset, then compare the results to find the real NN.

Using this property we can improve the preprocessing time, by creating a hierarchical data structure of sites sorted by x-coordinate of their centers. We construct a binary tree where each internal node stores a list of sites in its subtree, which is equivalent to sites whose centers belong to some canonical range of x values. At each node we construct the Voronoi diagram of this list, separately for the left and right edges. Given query Q, we perform a query using its center, to obtain the $O(\log n)$ nodes whose subtrees are a decomposition of $\mathcal{P}_\ell$ and $\mathcal{P}_r$. We perform a query with corresponding $\odot'$ in each of these Voronoi diagrams, and compare them to find the real nearest neighbor. This way the preprocessing takes $O(n \log^2 n)$ expected time and $O(n \log n)$ space, but the query time will be $O(\log^2 n)$. □

If we replace the binary tree by an m-ary one in the data structure, for any $2 \le m \le n$, we obtain a trade-off between query and preprocessing costs. Each node has m children; for $1 \le i \le m$, we store Voronoi diagram of sites of the union of the subtrees of its first (and last) i children, $O(m)$ diagrams at each node. A query takes $O(\log_m n \log m) = O(\log n)$ time to find those $O(\log_m n)$ nodes that cover P_ℓ or P_r (at most one of each at each tree level), and we need to do point location in $O(\log_m n)$ nodes and each takes $O(\log n)$. So the total query time is $O(\log_m n \log n)$. Each site is stored at $O(\log_m n)$ nodes, and at each node there are $O(m)$ copies of it. So the size of the data structure is $O(mn \log_m n)$, and it takes $O(mn \log_m n \log n)$ expected time to construct it. If we set $m = n^{1/\delta}$, for any $1 \le \delta \le \log n$, we obtain a data structure of size $O(\delta n^{1+1/\delta})$ in $O(\delta n^{1+1/\delta} \log n)$ expected preprocessing time and each query takes $O(\delta \log n)$ time.

4 Approximate Nearest Neighbor for L_2

In the Euclidean metric, finding the exact answer to nearest-neighbor problem under uncertainty is more complicated than in rectilinear metric. In order to obtain fast queries, we consider approximating the answer. In the case where the query is uncertain (aggregate queries), but each site is a point, Li et al. provided a $\sqrt{2}$-approximation answer [9]. We generalize this result for the version of the problem where both queries and sites are uncertain.

In the following theorem, we show that an uncertain query Q can be represented by a single point $\odot$, the center of its minimum enclosing circle, so that the distance of Q from the site nearest to $\odot$ is an approximation of the distance to the true nearest neighbor.

Theorem 5 *When querying a set of uncertain sites with an uncertain query Q, if C is the minimum enclosing circle of Q with radius r and with center at $\odot$, P the nearest site to $\odot$, and P^* the nearest site to Q, then $l = d(Q, P)$ will be a $\sqrt{2}$-approximation of $l^* = d(Q, P^*)$.*

Proof. Let d and d^* be the distance of $\odot$ from P and P^*, respectively. Let z be the farthest point in P^* from $\odot$ (so that $d(z, \odot) = d^*$) and AB be the diameter of C orthogonal to $\odot z$. Connect z to $\odot$ and extend it so that it hits the circle at E (see Figure 5). By the triangle inequality,

$$l = d(Q, P) \le d + r.$$

Since C is a minimum enclosing circle, there should be a query point q on $\frown AEB$, and $\angle z \odot q \ge \pi/2$. Therefore,

$$d(q, z) \ge \sqrt{d^{*2} + r^2}.$$

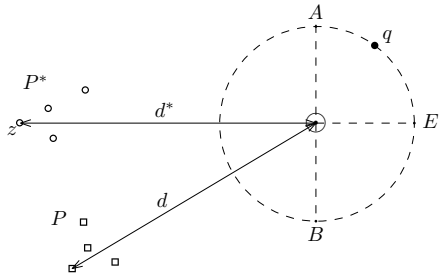


Figure 5: The nearest site to the center of an aggregate query is an approximate nearest neighbor.

By definition

$$l^* = d(Q, P^*) \geq d(q, z),$$

and since P is the nearest site to $\odot$

$$d^* \geq d.$$

As a result,

$$l^* \geq \sqrt{d^2 + r^2}.$$

If both d and r are equal to zero, since $l \leq d + r$, l is also zero, so $l \leq \sqrt{2}l^*$ holds. Otherwise, $\sqrt{d^2 + r^2} \neq 0$ and

$$l \leq \frac{d + r}{\sqrt{d^2 + r^2}} l^* \leq \sqrt{2}l^*.$$

□

The challenge here is finding the nearest uncertain site to the point $\odot$. Since this distance is equivalent to Hausdorff distance we can find the nearest site by performing a point location query in the Hausdorff Voronoi Diagram which takes poly-logarithmic time. The preprocessing time and the size of the diagram depends on how separated the sites are [5, 14]. In the case where convex hulls of uncertain points are disjoint, the diagram can be created in $O(s \log^3 s)$ time using linear space [7].

5 Conclusion

We studied NN searching with uncertain sites and queries. Under L_∞ metric, we provided two algorithms to find the exact NN. There is a trade-off between the preprocessing and query time as shown in Table 1. One obvious open problem is to establish a lower bound on the query time when using, say, linear space.

For L_2 version of the problem, we presented a $\sqrt{2}$-approximation algorithm. Is there is an algorithm with sublinear query time to find the exact nearest neighbor? Moreover, there is no lower bound on query time for the Euclidean metric.

Preproc. time	DS Size	Query time
$O(n^2 \log n + s)$	$O(n^2)$	$O(\log n + k)$
$O(n \log^2 n + s)$	$O(n \log n)$	$O(\log^2 n + k)$
$O(mn \log_m n \log n + s)$	$O(mn \log_m n)$	$O(\log_m n \log n + k)$
$O(\delta n^{1+1/\delta} \log n + s)$	$O(\delta n^{1+1/\delta})$	$O(\delta \log n + k)$

Table 1: The trade-off between preprocessing and query time.

References

[1] P. K. Agarwal, B. Aronov, S. Har-Peled, J. M. Phillips, K. Yi, and W. Zhang. Nearest-neighbor searching under uncertainty II. *ACM Trans. Algorithms*, 13(1):3:1–3:25, 2016.

[2] P. K. Agarwal, A. Efrat, S. Sankararaman, and W. Zhang. Nearest-neighbor searching under uncertainty. In *PODS*, pages 225–236, 2012.

[3] J. L. Bentley and J. B. Saxe. Decomposable searching problems I. Static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.

[4] R. Cheng, X. Xie, M. L. Yiu, J. Chen, and L. Sun. UV-diagram: A Voronoi diagram for uncertain data. In *26th International Conference on Data Engineering*, pages 796–807. IEEE, 2010.

[5] O. Cheong, H. Everett, M. Glisse, J. Gudmundsson, S. Hornus, S. Lazard, M. Lee, and H.-S. Na. Farthest-polygon Voronoi diagrams. *Computational Geometry*, 44(4):234–247, 2011.

[6] W. Evans and J. Sember. Guaranteed Voronoi diagrams of uncertain sites. In *20th Canadian Conference on Computational Geometry*, pages 207–210, 2008.

[7] J. Iacono, E. Khramtcova, and S. Langerman. Searching edges in the overlap of two plane graphs. *CoRR*, abs/1701.02229, 2017.

[8] R. Klein. Abstract Voronoi diagrams and their applications. *Computational Geometry and its Applications*, 333:148–157, 1988.

[9] F. Li, B. Yao, and P. Kumar. Group enclosing queries. *IEEE Transactions on Knowledge and Data Engineering*, 23(10):1526–1540, 2011.

[10] X. Lian and L. Chen. Probabilistic group nearest neighbor queries in uncertain databases. *IEEE Transactions on Knowledge and Data Engineering*, 20(6):809–824, 2008.

[11] J. Pach and M. Sharir. The upper envelope of piecewise linear functions and the boundary of a region enclosed by convex plates: Combinatorial analysis. *Discrete & Computational Geometry*, 4(1):291–309, 1989.

[12] D. Papadias, Q. Shen, Y. Tao, and K. Mouratidis. Group nearest neighbor queries. In *20th International Conference on Data Engineering*, pages 301–312, 2004.

[13] D. Papadias, Y. Tao, K. Mouratidis, and C. K. Hui. Aggregate nearest neighbor queries in spatial databases. *ACM Transactions on Database Systems (TODS)*, 30(2):529–576, 2005.

[14] E. Papadopoulou. The Hausdorff Voronoi diagram of point clusters in the plane. *Algorithmica*, 40(2):63–82, 2004.

[15] G. Voronoï. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. Deuxième mémoire. Recherches sur les parallélloèdres primitifs. *Journal für die reine und angewandte Mathematik*, 134:198–287, 1908.

[16] H. Wang. Aggregate-max nearest neighbor searching in the plane. In *Canadian Conference on Computational Geometry*, pages 71–76, 2013.

[17] P. Zhang, R. Cheng, N. Mamoulis, M. Renz, A. Züfle, Y. Tang, and T. Emrich. Voronoi-based nearest neighbor search for multi-dimensional uncertain databases. In *29th International Conference on Data Engineering*, pages 158–169, 2013.

[18] W. Zhang. Nearest-neighbor searching under uncertainty. Master's thesis, Department of Computer Science, Duke University, 2012.

A Seed Placement Strategy for Conforming Voronoi Meshing

Ahmed Abdelkader* Chandrajit L. Bajaj† Mohamed S. Ebeida‡ Scott A. Mitchell§

Abstract

We show how to place a set of seed points such that a given piecewise linear complex is the union of some faces in the resulting Voronoi diagram. The seeds are placed on sufficiently small spheres centered at input vertices and are arranged into little circles around each half-edge where every seed is mirrored across the associated triangle. The Voronoi faces common to the seeds of such arrangements yield a mesh conforming to the input complex. If the input contains sharp angles, then additional seeds are needed, analogous to nonobtuse refinement. Finally, we propose local optimizations to reduce the number of seeds and output facets.

1 Introduction

In many applications, it is required to capture the geometry of some domain of interest, e.g., for the purposes of engineering design and simulations. When the input is a sufficiently dense sample of points from the boundary, surface reconstruction algorithms can produce a good approximation of the surface [1]. On the other hand, volume decompositions with theoretical guarantees can be obtained using tetrahedral cells. However, there has been a growing interest in polyhedral cells, which are known to be more efficient at filling the space with fewer cells and can offer certain advantages in terms of numerical stability. Utilizing the Voronoi cells of some interior sample of points has been considered, but ensuring that cells conform naturally to the surface, i.e., without clipping, remains challenging [6]. Similar to the study of conforming tetrahedral meshing [7, 8, 5], we study the analogous question in the polyhedral case. For background and applications of representing and approximating geometries by Voronoi cells, we refer the reader to [3, 9] and the references therein.

Given a piecewise linear complex (PLC) $\mathcal{C}$, we seek a reconstruction of $\mathcal{C}$, or rather a refinement of it, by Voronoi faces such that each input face is the union of a number of output faces. Depending on the geometry of $\mathcal{C}$, the number of Voronoi cells may be large, so those results are mostly of theoretical interest.

*University of Maryland, College Park, akader@cs.umd.edu
†University of Texas, Austin, bajaj@ices.utexas.edu
‡Sandia National Laboratories, msebeid@sandia.gov
§Sandia National Laboratories, samitch@sandia.gov

In Section 2, we show how to obtain a Voronoi mesh conforming to an input PLC. We rely on certain spherical neighborhoods being empty and assume we can place seeds arbitrarily close to input vertices. In Section 3, we show that seeds can be placed at a non-zero distance from vertices and that allowing overlapping sphere neighborhoods can help reduce the number of seeds needed. Finally, in Section 4, we describe refinement procedures to enforce the required empty neighborhood condition.

2 Basic Seed Placement Strategy

Allowing the seeds $\mathcal{S}$ to be placed arbitrarily close to features of $\mathcal{C}$, we develop the basis of the proposed strategy in 2.1. Then, sufficient conditions for such a strategy to work are derived in 2.2. We prove in 2.3 that a subset of $\mathrm{Vor}(\mathcal{S})$ yields a mesh that conforms to $\mathcal{C}$.

2.1 Overview

We place seeds near input faces, cospherical around vertices, cocircular around edges, and mirrored across triangles. We examine face types in sequence and anticipate sufficient conditions for correctness.

2.1.1 Placement for Vertices

A vertex v in $\mathrm{Vor}(\mathcal{S})$ is equidistant to at least four seeds in $\mathcal{S}$ which are closest to it, and are not cocircular. To ensure every vertex $v_i \in \mathcal{C}$ is a vertex in $\mathrm{Vor}(\mathcal{S})$, we define a sphere S_i of radius ϵ_v centered at v_i, and place at least four seeds on it. Using a sufficiently small ϵ_v, no other seeds lie inside S_i.

2.1.2 Placement for Edges

All points in the interior of an edge e in $\mathrm{Vor}(\mathcal{S})$ are equidistant to at least three seeds in $\mathcal{S}$, forming a circle perpendicular to and centered at the line supporting e. Each $e_{ij} = (v_i, v_j) \in \mathcal{C}$ can be reconstructed as two edges in $Vor(\mathcal{S})$. Define two circles C_{ij} and C_{ji} of radius ϵ_e on spheres S_i and S_j, perpendicular to and centered at e_{ij}. Three or more seeds on each circle are used to reconstruct the edge. For sufficiently small ϵ_v and ϵ_e, no other seeds on S_i or S_j are closer. Additional conditions, e.g., angle bounds, are required to ensure that no other seed is closer to any point on e_{ij}.

2.1.3 Placement for Triangles

All points in the interior of a Voronoi facet f are equidistant to the two seeds closest to f, such that the plane supporting the facet is the bisector between these seeds. For each triangle $\triangle_{ijk} \in \mathcal{C}$, each edge circle provides one mirrored pair of seeds at height h above and below $\triangle_{ijk}$. The value of h depends on the dihedral angle at the edge and is chosen to ensure that the seed pairs from the two adjacent triangles do not overlap. Additional conditions, e.g., obtuse dihedral angles, help ensure that no other seed is closer to any point on $\triangle_{ijk}$.

2.1.4 Recovering the PLC

For a vertex v_i with fewer than two edges, add extra seeds on S_i so there are at least four. All seeds on S_i get a *vertex label* ℓ_i^v. Circle C_{ij} contributes one pair of seeds for each triangle incident on e_{ij}; for edges with fewer than two triangles, add extra cocircular seeds so there are at least three. All seeds on C_{ij} get an *edge label* ℓ_{ij}^e. Finally, all seeds mirrored across facet $\triangle_{ijk}$ get a *facet label* ℓ_{ijk}^f. A labeled seed *witnesses* the associated input face. Denote the witnesses of face f by $\mathcal{S}_f$.

The refinement of $\mathcal{C}$ is the *witnessed* faces of $\mathrm{Vor}(\mathcal{S})$ shared between the appropriate number of seeds with matching labels. Denote these faces as $\mathrm{VoRef}(\mathcal{C}) \subset \mathrm{Vor}(\mathcal{S})$, where "subset" is as a complex. By construction, $\mathcal{S}_f$ refines f, $\forall f \in \mathcal{C}$. In Section 4, we show that non-manifold PLCs can be recovered using a more aggressive strategy requiring *extra* seeds.

2.2 Definitions and Preliminaries

Denote the input complex by $\mathcal{C} = (\mathcal{V}, \mathcal{E}, \mathcal{F})$, where $\mathcal{V}$ is a set of vertices in $\mathbb{R}^3$, $\mathcal{E}$ a set of edges and $\mathcal{F}$ a set of triangles. Note that $\mathcal{C}$ may contain *isolated vertices* not incident on any edges and *isolated edges* not incident on any triangles.

For $x \in \mathbb{R}^3$, let $N_0(x)$ be the closest points in $\mathcal{V}$ to x. Define $\kappa_0(x) := \|x - v\|$, with $v \in N_0(x)$, and let S_x be the sphere centered at x with radius $\kappa_0(x)$. Similarly, define $N(.)$ using $\mathcal{S}$, and $N_f(.)$ and $\kappa_f(.)$ using $\mathcal{S}_f$.

For edge e_{ij}, let m_{ij} be the midpoint and S_{ij} the *diametric-sphere*, i.e., the sphere with e_{ij} as a diameter. For $\triangle_{ijk}$, let S_{ijk} be the smallest enclosing sphere and c_{ijk} its center. Let (a, b) denote $\overline{ab} \setminus \{a, b\}$.

The basic strategy described in 2.1 requires the input to satisfy a condition like the following:

Definition 1 (Closeness) *$\forall x \in e_{ij}, N_0(x) \subseteq \{v_i, v_j\}$ if e_{ij} is isolated, and $\forall x \in \triangle_{ijk}$, $N_0(x) \subseteq \{v_i, v_j, v_k\}$.*

In 2.3, we prove that the basic strategy described in 2.1 can refine PLCs satisfying closeness. We use the following definition to characterize vertices close enough to spoil the closeness condition.

Definition 2 (ball neighborhood) *For an edge e_{ij} it is the diametric-sphere S_{ij}, and for a triangle $\triangle_{ijk}$ the union of the smallest enclosing S_{ijk} with $\{S_{ij}, S_{jk}, S_{ik}\}$.*

Lemma 3 *$\triangle_{ijk}$ has an empty ball neighborhood iff $\triangle_{ijk}$ satisfies the closeness condition.*

Proof. ($\Rightarrow$) WLOG take $x \in \triangle_{ijk}$ such that v_i is the nearest vertex in $\triangle_{ijk}$ to x. Letting S_p^i be the sphere centered at p with radius $\|v_i - p\|$, it is clear that if S_x^i is empty, $N_0(x) = v_i$ and closeness holds. Take $y = \overrightarrow{v_i x} \cap \overline{c_{ijk} m_{ij}}$ and observe that $S_x^i \subset S_y^i$. Let C_{ij} be the circle $S_{ijk} \cap S_{ij}$ centered at m_{ij}. For any $z \in C_{ij}$ we may write $\|m_{ij} - z\| = \|m_{ij} - v_i\|$. As $C_{ij} \perp \triangle_{ijk}$, $\overline{y m_{ij}} \perp \overline{m_{ij} z}$ and we get $\|y - z\|^2 = \|m_{ij} - z\|^2 + \|y - m_{ij}\|^2 = \|v_i - m_{ij}\|^2 + \|y - m_{ij}\|^2 = \|v_i - y\|^2$. Hence, $z \subset S_y^i$ implying $C_{ij} \subset S_y^i$. Recalling that $y \in \overline{c_{ijk} m_{ij}}$ we get $S_x^i \subset S_y^i \subset S_{ijk} \cup S_{ij}$, which is empty by assumption.

($\Leftarrow$) If $\exists v_a \in \mathcal{C}$ such that $v_a \in S_{ijk} \cup S_{ij}$ and $a \notin \{i, j, k\}$ then, either $v_a \in N_0(m_{ij})$ or $v_a \in N_0(c_{ijk})$. □

At first glance, empty ball neighborhoods appear rather restrictive. However, for planar triangulations, nonobtuseness is sufficient to guarantee it, which can be enforced by nonobtuse refinement [4].

For non-planar triangulations, nonobtuseness is not sufficient, and we must also consider the distance to non-incident vertices. We start in Section 4.1 by showing that for many common triangulations, empty ball neighborhoods can be guaranteed without much refinement. Then, in Section 4.2 we proceed to outline a more aggressive variant of the strategy, reminiscent of nonobtuse refinement [4], that ensures correct output regardless of input angles and distances. Hence, any PLC can be refined.

2.3 Placement under Closeness with $\epsilon_v \to 0$

We analyze the basic strategy in 2.1 for refining an input PLC $\mathcal{C}$ when the closeness condition (Definition 1) is satisfied. Throughout this analysis, we take $\epsilon_v, \epsilon_e \to 0$. Figure 1, illustrates the different ways in which the seeds in $\mathcal{S}$ refine a nonobtuse triangle. In 3.1, we show it is feasible for non-zero radii $\epsilon_v, \epsilon_e > 0$ within a constant factor of the smallest geometric distances and angle sines.

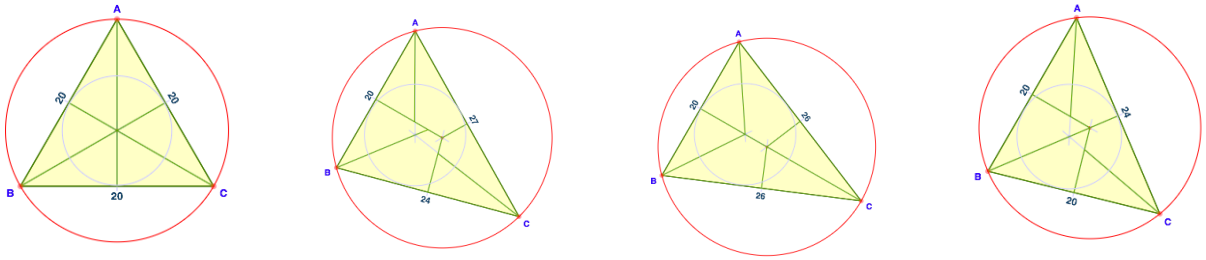
Claim 4 *$\forall v_i \in \mathcal{C}, v_i \subset Vor(\mathcal{S})$.*

Proof. As $\epsilon_v \to 0$, $N(v_i) \subset S_i$. □

Claim 5 *$\forall e_{ij} \in \mathcal{C}, \{m_{ij}\} \cup \{\overline{v_i m_{ij}}, \overline{v_j m_{ij}}\} \subset Vor(\mathcal{S})$.*

Proof. WLOG take $x \in (v_i, v_j)$ such that $v_i \in N_0(x)$. As $\epsilon_v \to 0$, $N(x) \subset C_{ij}$, so x lies on a Voronoi edge. As $\epsilon_v \to 0$, $N(m_{ij}) \subset C_{ij} \cup C_{ji}$ so m_{ij} is a vertex. □

For $\triangle_{ijk}$, let β_i be the first intersection of the ray starting at v_i and bisecting its angle with any of $\{\overline{c_{ijk} m_{ij}}, \overline{c_{ijk} m_{ik}}, \overline{c_{ijk} m_{jk}}\}$.



(a) $|AB| = |BC| = |AC|$. (b) $|AB| < |BC| < |AC|$. (c) $|AB| < |BC| = |AC|$. (d) $|AB| = |BC| < |AC|$.

Figure 1: Case analysis for refining a triangle into six facets, triangles and quads, based on the relative length of $\overline{AB}$. Voronoi edges (thick) lie on angle and edge-perpendicular bisectors (thin). Many edges are collinear.

Claim 6 $\forall \triangle_{ijk} \in \mathcal{C}$, *we have* $\{c_{ijk}, \beta_i, \beta_j, \beta_k\} \cup \{\overline{c_{ijk}m_{ij}}, \overline{c_{ijk}m_{ik}}, \overline{c_{ijk}m_{jk}}, \overline{v_i\beta_i}, \overline{v_j\beta_j}, \overline{v_k\beta_k}\} \subset \mathit{Vor}(\mathcal{S})$.

Proof. As $\epsilon_v \to 0$, $N(c_{ijk}) \subset S_i \cup S_j \cup S_k$. WLOG taking $x \in (c_{ijk}, m_{ij})$, $N(x) \subset C_{ij} \cup C_{ji} \cup C_{ik} \cup C_{jk}$ as $\epsilon_v \to 0$ so $\overline{c_{ijk}m_{ij}}$ is covered by a sequence of collinear Voronoi edges. Similarly, as $\epsilon_v \to 0$, $N(\beta_i) \subset C_{ij} \cup C_{ji} \cup C_{ik} \cup C_{jk}$ and taking $x \in (v_i, \beta_i)$, $N(x) \in C_{ij} \cup C_{ik}$ so $\overline{v_i\beta_i}$ appears exactly in $\mathrm{Vor}(\mathcal{S})$.

Let β_i' and β_j' be the intersections between the line supporting $\overline{c_{ijk}m_{ij}}$ and the rays bisecting the angles at v_i and v_j, respectively. We define an ordering on $\overrightarrow{m_{ij}c_{ijk}}$ such that $x_1 < x_2$ if $\|m_{ij} - x\| < \|m_{ij} - x_2\|$. WLOG, let $\beta_i' \leq \beta_j'$. We have the following cases; see Figure 1.

case	$\lvert\{e\}\rvert$	case	$\lvert\{e\}\rvert$
$\beta_i' = \beta_j' = c_{ijk}$	1	$\beta_i' < c_{ijk} < \beta_j'$	2
$\beta_i' < \beta_j' = c_{ijk}$	2	$c_{ijk} = \beta_i' < \beta_j'$	1
$\beta_i' = \beta_j' < c_{ijk}$	2	$c_{ijk} < \beta_i' = \beta_j'$	1
$\beta_i' < \beta_j' < c_{ijk}$	3	$c_{ijk} < \beta_i' < \beta_j'$	1

□

Claim 7 *Each* $\triangle_{ijk} \in \mathcal{C}$ *appears as 6 facets in* $\mathit{Vor}(\mathcal{S})$.

Proof. $\forall x \in \triangle_{ijk}$, $N(x) \subset C_{ij} \cup C_{ji} \cup C_{jk} \cup C_{kj} \cup C_{ik} \cup C_{ki}$. The mirrored pair of seeds on each of the six circles contributes a Voronoi facet aligned with $\triangle_{ijk}$. □

Corollary 8 *Letting* v, e *and* f *be the number of vertices, edges and facets in* $\mathcal{C}$*, the basic placement strategy in 2.1 generates at most* $v + 3e + 4f$ *vertices, at most* $2e + 9f$ *edges and exactly* $6f$ *facets in* $\mathit{VoRef}(\mathcal{C})$.

Theorem 9 *Given a PLC* $\mathcal{C}$ *satisfying the closeness condition,* $\mathcal{C} = \mathit{VoRef}(\mathcal{C})$.

Proof. ($\mathcal{C} \subset \mathrm{VoRef}(\mathcal{C})$) Claims 4, 5 and 7 establish that all vertices, edges and facets of $\mathcal{C}$ belong to $\mathrm{Vor}(\mathcal{S})$. By examining the arguments made above, it is clear that $\forall x \in \mathcal{C}$, x lies on some face in $\mathrm{Vor}(\mathcal{S})$ common to the appropriate number of correctly labeled seeds in $\mathcal{S}$.

($\mathrm{VoRef}(\mathcal{C}) \subset \mathcal{C}$) Assume for contradiction that $\exists x \in \mathrm{VoRef}(\mathcal{C})$ such that $x \notin \mathcal{C}$. By definition of $\mathrm{VoRef}(\mathcal{C})$, x lies on some face in $\mathrm{Vor}(\mathcal{S})$ common to at least two seeds with matching labels. But, as $x \notin \mathcal{C}$, no seeds in $\mathcal{S}$ would be labeled to retain it. □

3 Placement under Closeness with Non-zero Radii

Recall that per 2.1, all seeds in $\mathcal{S}$ were labeled with the associated face to serve as witnesses upon recovering the PLC from $\mathrm{Vor}(\mathcal{S})$. When $\epsilon_v, \epsilon_e \to 0$, ball neighborhoods free of input vertices were sufficient. Using non-zero radii, the natural analog is to require witnessed neighborhoods free of bad seeds. In this section, we also assume $\mathcal{C}$ satisfies the closeness condition.

One way to think of the *witnessed neighborhood* for $\triangle_{ijk}$ is to take a clone of its ball neighborhood endowed with the vertex spheres $\{S_i, S_j, S_k\}$ with ϵ_v set initially to 0. Then, as ϵ_v and ϵ_e increase to a non-zero value, the vertex spheres grow while the cloned ball neighborhood starts to shrink as the spheres centered at any $x \in \triangle_{ijk}$ need only touch the nearest witness in $\mathcal{S}_{ijk}$ rather than the original vertices $\{v_i, v_j, v_k\}$. As $\mathcal{S}_f$ refines f, the witnessed neighborhood is the union of spheres centered at the vertices $v \subset \mathrm{VoRef}(f)$ with radius equal to $\kappa_f(v)$.

If a seed s was not given an appropriate label for some x, we say s is a *non-witness* for x. If a non-witness seed $s \in \mathcal{S}$ is closer to x than its witnesses, then $\mathrm{Vor}(\mathcal{S})$ fails to conform to $\mathcal{C}$. The following definition characterizes problematic placements for non-zero radii.

Definition 10 (Encroached Faces) *If a non-witness seed* $s \in \mathcal{S} \setminus \mathcal{S}_f$ *lies in the witnessed neighborhood of* f*, we say that* s encroaches *on* f.

3.1 Non-overlapping Radii

The basic strategy in 2.1 was described for sphere radii ϵ_v and circle radii ϵ_e approaching zero. Here we show these radii can be non-zero.

If $\mathcal{C}$ is a planar triangulation, the basic strategy does not create encroachments. The radius of S_i can be in the range $[0, \lambda_i)$ where $\lambda_i = min_{(i,j)\in\mathcal{C}}\|v_iv_j\|/2$. This ensures vertex balls do not overlap, and only the faces and edges incident on v_i intersect S_i. For finite radii edge circles, let α_i be the minimum angle at v_i, then C_{ij} can have a radius in the range $[0, r_i \sin\frac{\alpha_i}{2})$ for all edges $(i,j) \in \mathcal{C}$. Again, this ensures that edge circles do not intersect, and only triangles incident on an edge intersect its circles. Although for non-zero radii each face is not partitioned as nicely as in 2.3, Corollary 8 and Theorem 9 still hold.

If $\mathcal{C}$ is non-planar, it may contain non-incident elements that come arbitrarily close together. Denote the minimum distance between any two non-incident features by δ_v. Set all sphere radii to $\epsilon_v = \delta_v/3 > 0$ [7]. Recalling that seeds lie on spheres of radius ϵ_v around input vertices, define the *clearance* at a point x on some face $f \in \mathcal{C}$ as $cl(x) = \kappa_0(x) - \kappa_f(x)$. A sufficient condition for encroachment-free witnessed neighborhoods can be stated as $cl(x) \geq \epsilon_v \ \forall x \in f$. The basic strategy achieves this when $\epsilon_e \to 0$. For $\epsilon_e > 0$, we amend the strategy and allow a slightly smaller upper bound. Consider a shrunken ball neighborhood that only extends to the seeds generated on the spheres around the vertices of the face; call this the (ϵ_v, ϵ_e)-neighborhood of f. We add extra seeds to provide a safe lower bound on clearance.

Recall that all points e_{ij} are protected by seeds on $C_{ij} \cup C_{ji}$. We ensure a similar protection for all points on a triangles $\triangle_{ijk}$. Fixing S_i, consider the two great circles going through the pairs of seeds generated on C_{ij} and C_{ik} for e_{ij} and e_{ik}, respectively, and perpendicular to $\triangle_{ijk}$. Let h_i be the smaller of the heights of those seed pairs. We add extra seed pairs on S_i with uniform spacing between the two great circles at height h_i. The spacing is chosen to ensure $cl(x) \geq \min\{cl(m_{ij}), cl(m_{ik}), cl(m_{jk})\} \forall x$ in the interior of $\triangle_{ijk}$, and the number of extra seeds is finite as $\epsilon_e > 0$.

Fixing a face $f \in \mathcal{C}$ and a non-incident vertex v_i, let p be the closest point on S_i to the $(\epsilon_v, 0)$-neighborhood of f and note that $\forall x \in f$, $\|x - p\| \geq \kappa_0(x) - \epsilon_v$. For $\epsilon_e > 0$, the (ϵ_v, ϵ_e)-neighborhood might contain p and intersect S_i in a circle C_{if}. We show that any seeds on S_i lie outside C_{if} and do not encroach. Consider edge e_{ij} incident on v_i with witnesses on C_{ij}. We have two cases: (1) $p \in e_{ij}$. As the radius of C_{if} is at most ϵ_e, C_{ij} lies outside the (ϵ_v, ϵ_e)-neighborhood. (2) $p \notin e_{ij}$. Let p' be the closest point of f's (ϵ_v, ϵ_e)-neighborhood to p, and let $q = S_i \cap e_{ij}$. Define δ_e as the minimum $\|p' - q\|$ for any such points p, q. Then, we require $\epsilon_e \leq \delta_e/3$. To account for edges incident on the same vertex, let α_{min} be the minimum angle between two incident features of $\mathcal{C}$ and define $\alpha^* = \min\{\alpha_{min}, \pi/10\}$. We set $\epsilon_e = \min\{\delta_e/3, \epsilon_v \cdot \sin(\alpha^*/3)\} > 0$.

The preceding discussion establishes the following statement.

Theorem 11 *Any PLC satisfying the closeness condition admits a finite refinement for some $\epsilon_v, \epsilon_e > 0$. If the PLC is planar, the refinement has linear size.*

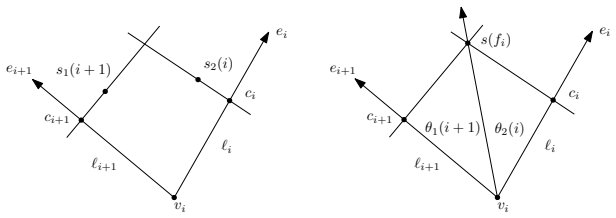
Note that while δ_v is an intrinsic feature of the input PLC, δ_e apparently arises due to this specific approach. To yield larger δ_e, one may attempt refinement to further reduce ball neighborhoods, e.g., by regular subdivision. It would be interesting to enable larger non-zero radii that only depend on intrinsic input properties and derive a bound on the output size.

3.2 Fewer Steiner Points in 2D by Overlapping Radii

The primitives in 2.1 created non-overlapping vertex spheres and edge circles, and introduced many seeds on them. In particular, it generates twelve seeds for each triangle, with one pair for each half-edge. For adjacent edges around a vertex, if we can make edge circles larger so that they overlap, then we may use their two intersection points as the two necessary seeds for both edges, and generate only six seeds per triangle. Observe that the segment between such intersection points is perpendicular to the facet.

For vertices sharing an edge, if we make vertex spheres large enough to overlap, we may use the same seed pair for both endpoints of the edge, and again generate only six seeds per triangle. In the extreme, if we can perform both, this results in just one seed pair for all three edges of a triangle. We leave the study of these two additional variants as future work and only consider sharing seeds between edge circles around vertices.

Figure 2a shows schematically the basic setup for this scenario. For a given vertex v, we order the n edges in counter-clockwise order, and identify faces with the right edge. Let c_k denote the center of the circle for edge e_k and $\ell_k = \|v - c_k\|$. Each edge e_k gets two seed pairs that we denote by $s_1(k)$ (right) and $s_2(k)$ (left). If R is the radius of the sphere around vertex v, then the only restriction we have here is that $\ell_k < R \ \forall k$.



(a) Basic strategy: two seed pairs per vertex of a face.

(b) Overlapping circles: one seed pair shared by two edges.

Figure 2: Reducing the number of seeds per face.

We show how to use just three seed pairs per face by allowing circles of consecutive edges to share a pair. Figure 2b shows schematically the new situation. For a given vertex v, we now generate just one seed pair for each face f_i, denoted by $s(f_i)$ in the figure. By projecting that seed pair onto the face, $\overrightarrow{v\, s(f_i)}$ partitions the angle between e_i and e_{i+1} as $\theta_2(i)+\theta_1(i+1)$, where $\theta_1(k)$ and $\theta_2(k)$ denote the right and left angles around edge e_k. Observe that now $\{\ell_k\}$ are no longer independent.

$$\ell_{i+1} = \frac{\cos\theta_1(i+1)}{\cos\theta_2(i)} \cdot \ell_i. \tag{1}$$

WLOG, fixing ℓ_1 we find that:

$$\ell_1 = \left(\frac{\cos\theta_1(1)}{\cos\theta_2(n)} \times \cdots \times \frac{\cos\theta_1(2)}{\cos\theta_2(1)}\right)\ell_1. \tag{2}$$

Rearranging, we get the additional requirement that $\prod_{i=1}^{n} \frac{\cos\theta_1(i)}{\cos\theta_2(i)} = 1$.

One easy way to satisfy Equation 2 is to enforce that seed pairs are placed above and below angle bisectors. This immediately sets all ratios in the product to 1. However, letting γ_i denote $\|v - s(f_i)\|$, we need to ensure no prefix product results in some $\gamma_i > R$. Note that γ_i are related by a similar product of cosine ratios and that such products telescope. In particular, γ_{min} and γ_{max} correspond to θ_{min} and θ_{max}. Moreover, they are related by the following relation:

$$\gamma_{min} \cos\frac{\theta_{min}}{2} = \gamma_{max} \cos\frac{\theta_{max}}{2}. \tag{3}$$

If the triangulation is nonobtuse, we know that $\theta_{min} \leq \theta_{max} \leq \frac{\pi}{2}$ and the cosine is monotonically increasing. As we require $\gamma_{max} < R$, we can bound $\theta_{min} \geq 2\cos^{-1}\frac{R}{\sqrt{2}\gamma_{min}}$. An explicit bound is readily available if γ_{min} is expressed as a constant fraction of R. For example, requiring $\gamma_{min} \geq \frac{R}{2}$ yields $\theta_{min} \geq 17.87°$.

4 Refinement for Closeness

In the previous section, the closeness condition was essential to the the refinement strategies and analyses we presented. In this section, we show how to enforce such condition for an arbitrary input PLC.

4.1 Flat Complexes

We show that any PLC with *flat* dihedral angles can be refined, by showing that it can be refined into one satisfying the closeness condition. For clarity we will call the standard dihedral angle between two triangles sharing an edge the *edge-dihedral*. We define the *vertex-dihedral* for a triangle $\triangle_{123}$ and a vertex v_4 as the minimum edge-dihedral between $\triangle_{123}$ and one of the three triangles $\triangle_{124}, \triangle_{143}$, and $\triangle_{423}$. Obtuse edge- and vertex-dihedrals imply that v_4 lies outside S_{123}, the smallest enclosing sphere of $\triangle_{123}$.

Definition 12 (Flat Complex) *A PLC is* flat *if all edge- and vertex-dihedrals between* adjacent *faces are obtuse, where two faces are* adjacent *if they have a non-empty intersection.*

Lemma 13 *Any flat PLC can be refined into one with empty ball neighborhoods.*

Proof. First, refine to obtain nonobtuse triangles [4]. Second, iteratively refine every edge into two and every triangle into four through regular subdivision, stopping when all ball neighborhoods are empty. This will terminate because ball-neighborhoods shrink through subdivision, so eventually the only vertices v close enough to intersect a ball neighborhood of face f, are such that the original triangles containing v and f are the same or adjacent. Any such v on an adjacent face f' must lie outside the edge-diameter sphere of the common edge with f by the nonobtuseness of f'. Then, flatness ensures v lies outside the smallest enclosing sphere of f. □

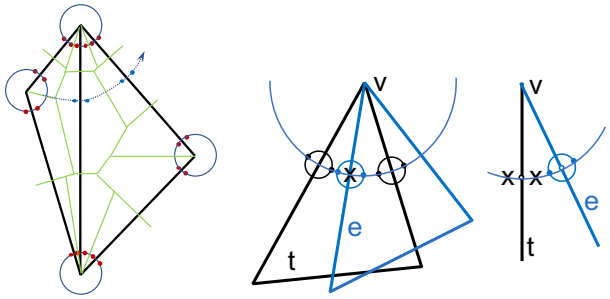
Recall [1] that an ϵ-lfs sampling of a surface $\mathcal{M}$ is a set of points $\mathcal{P}$ on $\mathcal{M}$ such that $\forall x \in \mathcal{M}$, $\exists p \in \mathcal{P}$ such that $\|xp\| \leq \epsilon \cdot \text{lfs}(x)$, where lfs denotes the *local feature size* defined as the distance from x to the *medial axis* of $\mathcal{M}$. It is well-known that a triangulation of an ϵ-lfs sampling, for a small enough ϵ, provides flat angles [1, 2]. Further, triangle edge lengths are small compared to the local feature size, so no regular subdivision in the proof of Lemma 13 is needed.

Theorem 14 *Any flat PLC can be refined. A nonobtuse triangulation with vertices from an ϵ-lfs sampling can be refined into a linear number of faces.*

4.2 Witness Refinement

We show that any PLC can be refined, even if dihedrals are not flat and the complex is non-manifold. The reason is that we can place *extra witness seeds* for each face, so they are the closest seeds for any face point. As before, seeds are mirrored pairs for triangles, and cocircular for edges. The method is analogous to nonobtuse triangle refinement [4], however we do not need to explicitly maintain a triangular cell complex.

Extra witness seeds. We split an encroached face with extra seeds, which shrinks its witnessed neighborhood. In general, if a seed s encroaches on a face f, then we split f near the point closest to s but outside any vertex sphere or original edge circle. For example, for a triangle with an adjacent edge making an acute vertex-dihedral, the seeds on the edge circle may encroach on the triangle. We split the triangle at the shared vertex's ball radius, mirrored through the triangle point closest to the edge; see Figure 3b.



(a) Extra seeds (blue) splitting an edge opposite an obtuse edge-edge angle, and propagating.

(b) Top view (left) and side view (right), of triangle t and edge e with a sharp vertex-dihedral at v. Edge circle seeds encroach and induce a seed pair x on t.

Figure 3: Seed refinement on encroached faces. Edge-refinement would produce more Voronoi faces.

Note that the seeds introduced for a split may then encroach on a different face, which then requires further splits; see Figure 3a. We may have propagating paths, as in nonobtuse refinement [4], leading to splitting a face multiple times. Fortunately, there is a range of locations where a split will remove the encroachment. Limiting nonobtuse propagation paths is a lengthy analysis [4], and we leave a similar analysis for polynomial-size seed splitting of non-manifold complexes for future work.

However, we show that a finite refinement is achievable, by spacing seeds based on the geometry, with no propagation paths needed. Let all vertex balls have the same radius ϵ_v and all edge circles the same radius ϵ_e. Thus radii are non-zero but vertex spheres are non-overlapping, and edge circles are non-overlapping; see Section 3.1 for the details. For ease of exposition, let all subsequent extra seeds be infinitely close to the face they witness. Split all edges outside vertex balls into segments of length at most $\epsilon_e/2$. Form maximal packings of $\epsilon_t = \epsilon_e \sin(\alpha/2)/2$ radius spheres inside triangles, but outside the vertex spheres and ϵ_e-radius spheres around each split edge seed. Thus the closest seed to any face point is a witness seed. See Ebeida and Mitchell [6] for a similar construction.

Theorem 15 *Any PLC admits a finite refinement, including non-manifold triangulations.*

5 Conclusion

We showed how to generate a set of seed points such that the faces of the resulting Voronoi diagram conform to an arbitrary piecewise linear complex. The proposed seed placement strategies require certain neighborhoods to be empty of input vertices and can be ensured by a process similar to nonobtuse refinement of triangulations. The number of output faces depends on the complex's geometric and topological properties.

It would be interesting to generate an all-quadrilateral refinement. Without the optimizations, for scalene nonobtuse triangles the Voronoi faces are quadrilateral. The Voronoi cells that refine the input have "degenerate" position, being co-spherical and co-circular. It is unclear if this degeneracy allows further optimizations to reduce the number of Steiner points.

We leave open the problem of producing a polynomial bound on the number of seeds needed for non-planar triangulations. An analysis similar to Bishop [4] should suffice, but a path may cross an edge or triangle more times than in the planar triangulation case, and any polynomial bounds will likely be larger.

Acknowledgement

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR), Applied Mathematics Program. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

References

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

[2] N. Amenta and T. K. Dey. Normal variation for adaptive feature size. *CoRR*, abs/1408.0314, 2014.

[3] T. Biedl, M. Held, and S. Huber. Recognizing straight skeletons and Voronoi diagrams and reconstructing their input. In *10th International Symposium on Voronoi Diagrams in Science and Engineering*, 2013.

[4] C. J. Bishop. Nonobtuse triangulations of PSLGs. *Discrete & Computational Geometry*, 56(1):43–92, 2016.

[5] D. Engwirda. Conforming restricted delaunay mesh generation for piecewise smooth complexes. *Procedia Engineering*, 2016. 25th International Meshing Roundtable.

[6] M. S. Mohamed S. Ebeida and S. A. Mitchell. Uniform random Voronoi meshes. In *20th International Meshing Roundtable*, pages 258–275, 2011.

[7] M. Murphy, D. M. Mount, and C. W. Gable. A point-placement strategy for conforming Delaunay tetrahedralization. *International Journal of Computational Geometry & Applications*, 11(06), 2001.

[8] A. Rand and N. Walkington. Collars and intestines: Practical conforming Delaunay refinement. In *Proceedings of the 18th International Meshing Roundtable*, 2009.

[9] A. Spettl, T. Brereton, Q. Duan, T. Werz, C. E. Krill III, D. P. Kroese, and V. Schmidt. Fitting Laguerre tessellation approximations to tomographic image data. *Philosophical Magazine*, 96(2), 2016.

On Compatible Triangulations with a Minimum Number of Steiner Points*

Anna Lubiw† Debajyoti Mondal†

Abstract

Two vertex-labelled polygons are *compatible* if they have the same clockwise cyclic ordering of vertices. The definition extends to polygonal regions (polygons with holes) and to triangulations—for every face, the clockwise cyclic order of vertices on the boundary must be the same. It is known that every pair of compatible n-vertex polygonal regions can be extended to compatible triangulations by adding $O(n^2)$ Steiner points. Furthermore, $\Omega(n^2)$ Steiner points are sometimes necessary, even for a pair of polygons. Compatible triangulations provide piecewise linear homeomorphisms and are also a crucial first step in morphing planar graph drawings, aka "2D shape animation." An intriguing open question, first posed by Aronov, Seidel, and Souvaine in 1993, is to decide if two compatible polygons have compatible triangulations with at most k Steiner points. In this paper we prove the problem to be NP-hard for polygons with holes. The question remains open for simple polygons.

1 Introduction

For many computational geometry problems involving a polygon or polygonal region, the standard first step is to triangulate the region. However, for some problems, such as morphing of polygons, or finding a homeomorphism between polygons, the input consists of two polygons with a correspondence between them, and the desirable first step is to triangulate them in a consistent way. Unlike for a single polygon, it may be necessary to add new vertices, called "Steiner points." Our paper is about this harder problem, which was called "joint triangulation" by Saalfeld [13] and "compatible triangulation" by Aronov, Seidel, and Souvaine [3].

Research on finding compatible triangulations is motivated by applications in morphing [2] and 2D shape animation [5, 16], and in computing piecewise linear homeomorphisms of polygons.

Throughout, we deal with vertex-labelled straight-line planar drawings. The most general input we consider is a polygon with holes (a polygonal region), where we allow a hole to degenerate to a single point. Two polygons are *compatible* if they have the same clockwise cyclic ordering of vertices. Two polygonal regions P_1 and P_2 are *compatible* if their outer polygons are compatible, and their holes are compatible, i.e. each hole (considered as a polygon) in P_1 corresponds to a compatible hole in P_2. Note that the labelling provides the correspondence.

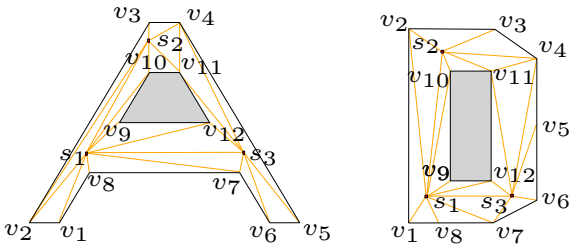


Figure 1: Two compatible polygons, each with one hole (shaded gray), and compatible triangulations of them with 3 Steiner points.

A *triangulation* $T(P)$ *of a polygonal region* P is a subdivision of its interior region into triangular faces. The vertices of $T(P) \setminus P$ are called *Steiner points* of $T(P)$. A pair of triangulations $T(P_1)$ and $T(P_2)$ of compatible polygonal regions P_1 and P_2, respectively, are *compatible* if their faces are compatible, i.e. every face of $T(P_1)$ (considered as a polygon) corresponds to a compatible face of $T(P_2)$. Again, the labelling provides the correspondence. Figure 1 illustrates a pair of compatible polygonal regions and their compatible triangulations.

Two special cases of compatible triangulations were studied independently. Saalfeld in 1987 [13] considered the case of two rectangles each with n points inside them (where the correspondence between the points is given) and showed that compatible triangulations always exist. Saalfeld's construction may require an exponential number of Steiner points [15]. Aronov et al., in 1993 [3] considered the case of simple compatible polygons. They showed that there exist compatible triangulations with $O(n^2)$ Steiner points. (A similar construction was given by Thomassen in 1983 [17, Theorem 4.1].) Furthermore, Aronov et al. gave an $O(n^2)$-time algorithm to compute such compatible triangulations, and they gave examples where $\Omega(n^2)$ Steiner points are necessary. They posed as an open problem to decide if two polygons have a compatible triangulation with k Steiner points, and observed that the case $k = 0$ can be decided in polynomial time via dynamic programming.

*Work is supported by NSERC.

†Cheriton School of Computer Science, University of Waterloo, {alubiw,dmondal}@uwaterloo.ca

Our Result. We show that it is NP-hard to decide if two compatible polygonal regions have compatible triangulations with at most k Steiner points, where $k \in \mathbb{N}$ is given as part of the input.

Further Background. There are a number of further results for the case of two simple polygons. Kranakis and Urrutia [9] gave an $O(n + r^2)$-time algorithm to find compatible triangulations of simple compatible polygons with $O(n + r^2)$ Steiner points, where r is the number of reflex vertices. Gupta and Wenger [8] gave a polynomial-time algorithm that provides an $O(\log n)$ approximation to the minimum number of Steiner points. A number of heuristic algorithms have been proposed—see e.g., [5, 16].

There is also a line of research on the case of polygons with point holes (Saalfeld's problem). Souvaine and Wenger [15] gave an $O(n^2)$-time algorithm to compute compatible triangulations with $O(n^2)$ Steiner points, and asked if there is a polynomial-time algorithm to construct compatible triangulations with the minimum number of Steiner points. Pach et al. [11] proved that $\Omega(n^2)$ Steiner points are sometimes necessary.

For the case of general polygonal regions—which encompasses both the above special cases—Babikov et al. [4] gave an $O(n^2)$-time algorithm to compute compatible triangulations with $O(n^2)$ Steiner points.

One approach to computing compatible triangulations is to first compute a triangulation for one of the polygonal regions, and then draw its underlying graph into the other polygonal region using polylines for drawing edges. The edge bends give rise to the Steiner points. This idea relates to the problem of drawing a planar graph on a given set of points, where the correspondence between vertices and the points is given. Pach and Wenger [12] gave an $O(n^2)$-time algorithm to compute such an embedding with $O(n^2)$ bends in total, and this was extended to deal with a bounding polygonal region in [6].

The version of the compatible triangulation problem where the correspondence between the two polygonal regions is *not* given is also well-studied and very relevant in practice, e.g. see [5]. In this setting, Aichholzer et al. [1] made the fascinating conjecture that for any two point sets each with n points, of which h lie on the convex hull, there is a mapping between them that permits compatible triangulations with no Steiner points.

2 Preliminaries

Let P be a polygon, possibly with holes. Two points a, b in P are *visible* if the line segment between them lies strictly inside P; they are *1-bend visible* if there is a point c inside P that is visible to both a and b.

A *dent* on the boundary of P consists of three consecutive vertices u, d, v of P such that d is convex and u, v are reflex vertices, e.g., see the polygon P_1 in Figure 2. We refer to d as the *peak* of the dent. The *visibility region* of d consists of all the points inside P that are visible to P. An *inward dent* on the boundary of P consists of three consecutive vertices u, d, v of P such that d is reflex and u, v are convex vertices. The following simple lemma about dents in compatible triangulations of polygons will be a key ingredient of our NP-hardness proof.

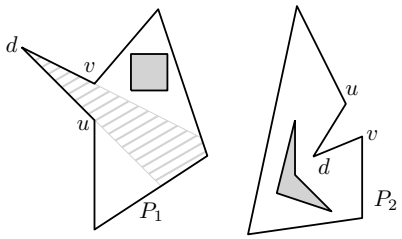


Figure 2: Illustration for Lemma 1. The visibility region of d is shown in gray stripes.

Lemma 1 *Let P_1 and P_2 be a pair of compatible polygons. Assume that P_1 contains a dent u, d, v, and let Ψ be the visibility region of d in P_1. If u, v are not visible in P_2, then in any compatible triangulations d must be adjacent either to a Steiner point or a vertex (except u and v) inside Ψ.*

Proof. Any triangulation of P_1 (even with Steiner points) must use the edge (u, v) or an edge incident to d. In compatible triangulations of P_1 and P_2 the edge (u, v) is ruled out, and therefore d must be adjacent to a Steiner point or a vertex in $\Psi \setminus \{u, v\}$. □

3 NP-Hardness

In this section we prove that given a pair of compatible polygonal regions P_1, P_2, and $k \in \mathbb{N}$, it is NP-hard to decide if there are compatible triangulations of P_1 and P_2 with at most k Steiner points.

We reduce from the monotone rectilinear planar 3-SAT problem (MRP-3SAT), which is NP-complete [7]. The input of an MRP-3SAT instance I is a collection C of clauses over a set U of Boolean variables such that each clause contains at most three literals, and is either *positive* (consists of only positive literals), or *negative* (consists of only negative literals). Moreover, the corresponding *SAT-graph* G_I (the bipartite graph with vertex set $C \cup U$ and edge set $\{(c, x) \in C \times U : x \text{ appears in } c\}$) admits a planar drawing Γ satisfying the following properties:

- Each vertex in G_I is drawn as an axis-aligned rectangle in Γ.
- All the rectangles representing variables lie along

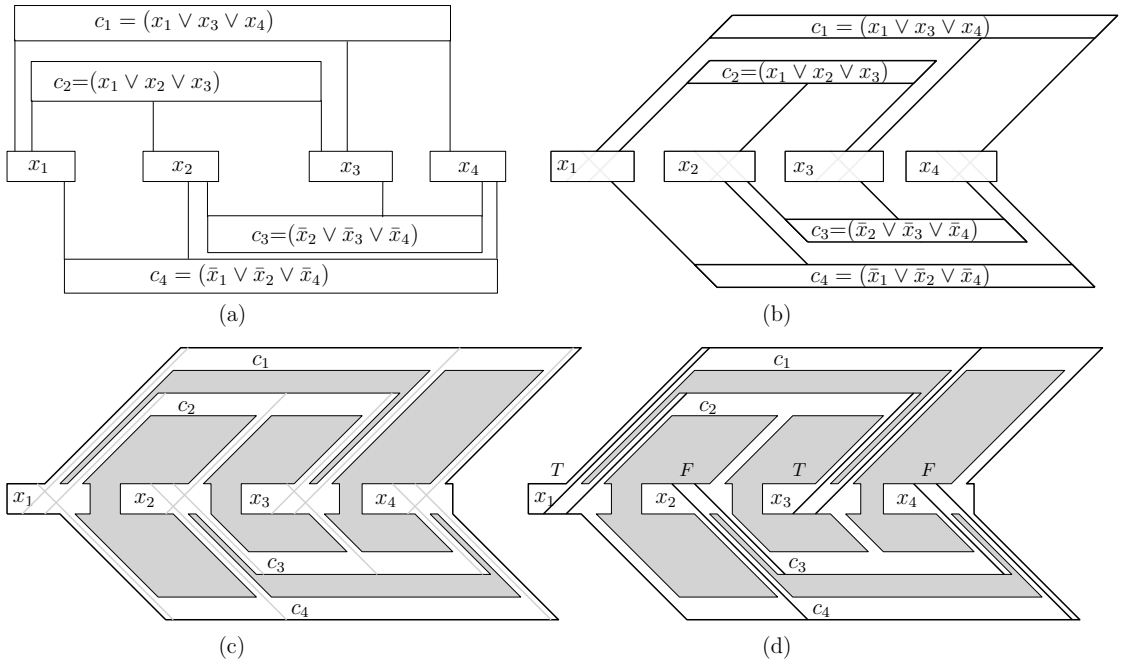


Figure 3: (a) An instance I of MRP-3SAT, and the corresponding drawing Γ. (b) Γ'. (c)–(d) Illustration for the hardness reduction.

a horizontal line ℓ.
- The rectangles representing positive (respectively, negative) clauses lie above (respectively, below) ℓ.
- Each edge (c, x) of G_I is drawn as a vertical line segment that connects the rectangles corresponding to c and x, e.g., see Figure 3(a).

The MRP-3SAT problem asks whether there is a truth assignment for U satisfying all clauses in C.

Given an instance $I = (U, C)$ of MRP-3SAT, we construct a pair of compatible polygonal regions P_1 and P_2 such that they admit compatible triangulations with at most $5|C|$ Steiner points, if and only if I is satisfiable.

Idea of the reduction: We first ensure that every clause in I has exactly three literals, by duplicating literals if necessary. Let the resulting instance be I'. It is straightforward to observe that I' is also an instance of MRP-3SAT, and I' is satisfiable if and only if I is satisfiable. Let Γ be the drawing corresponding to $G_{I'}$.

We modify the drawing Γ such that the edges and vertices corresponding to the positive (resp., negative) clauses become parallelograms, slanted 45° (resp., −45°) to the right, e.g., see Figure 3(b). For each clause $c \in C$, let $R(c)$ denote the parallelogram corresponding to c. We call $R(c)$ the "clause region". For each variable $u \in U$, let $B(u)$ denote the rectangle corresponding to u. We call $B(u)$ the "variable region". We call the edges of $G_{I'}$ *connectors* and we call the connectors that are incident to the top (resp., bottom) side of $B(u)$ *top* (resp., *bottom*) *connectors* of $B(u)$. We ensure that the extension of every top connector intersects the extensions of all the bottom connectors inside $B(u)$. Let the resulting drawing be Γ'. We construct P_1 and P_2 by modifying two distinct copies of Γ'.

We prove that in any compatible triangulations with $5|C|$ Steiner points, for each clause c, there is a triangulation edge e_c that lies along one of the connectors incident to the clause region. If c is positive (resp., negative) then we can set the variable corresponding to e_c to true (resp., false) and this will satisfy the clause. We get a valid truth-value assignment because a variable region cannot contain extensions of both top and bottom connectors. Figures 3(c)–(d) illustrate a satisfying truth assignment for I. On the other hand, given a satisfying truth assignment, we show how to find compatible triangulations for P_1 and P_2 using $5|C|$ Steiner points.

3.1 Construction of Polygonal Region P_1

We modify a copy Γ'_1 of Γ' to construct P_1. First we create a *channel* of small non-zero width around each connector so that we have a polygon with holes. We denote the copies of $R(c)$ and $B(u)$ in P_1 by $R_1(c)$ and $B_1(u)$. We create nine dents with peaks $u, v, w, q, q', r, r', s, s'$ in the boundary of $R_1(c)$, as shown in Figures 4(a)–(b). The visibility region of each dent is illustrated using gray straight lines.

As illustrated in Figure 4(a), we place a hole h in the

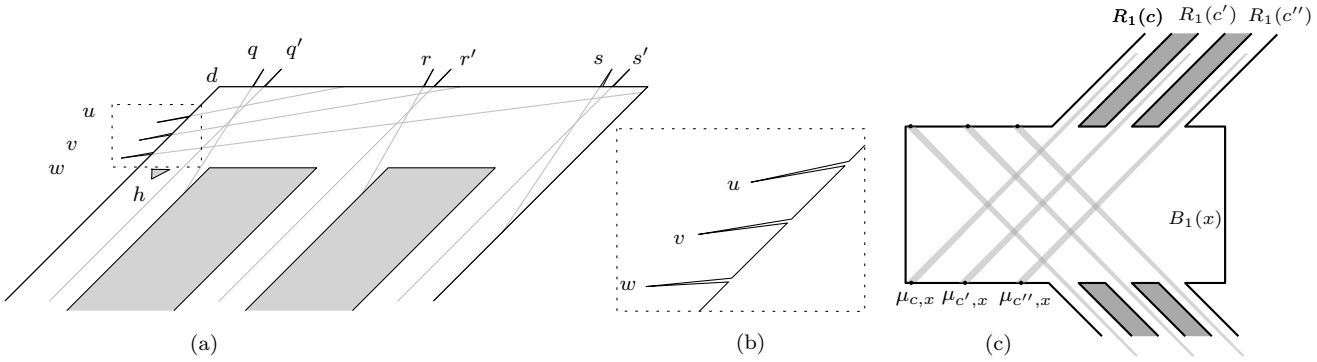


Figure 4: (a) A clause region in P_1. (b) A close-up of the dents corresponding to u, v, w. (c) Illustration for $B_1(x)$.

leftmost channel of $R_1(c)$, not intersecting the visibility regions of the peaks $u, v, w, q, q', r, r', s, s'$. We refer the reader to the full version [10] for the formal details of the construction and the precise placement of h.

We now modify the rectangles that correspond to the variables. Let x be a literal and let $B_1(x)$ be the corresponding rectangle in Γ'_1. See Figure 4(c). For every positive (resp., negative) clause c containing x, one or more[1] visibility regions corresponding to the peaks of $R_1(c)$ enter $B_1(x)$. We ensure that the visibility regions entering from the top (resp., bottom) of $B_1(x)$ are disjoint and only intersect the bottom (resp., top) side of $B_1(x)$. For each clause c containing x, we construct a vertex $\mu_{c,x}$ on the side of $B_1(x)$ such that $\mu_{c,x}$ is visible to the corresponding peak of $R_1(c)$. We refer to these newly constructed points as the *μ-points* of $B_1(x)$.

3.2 Construction of Polygonal Region P_2

We modify a copy Γ'_2 of Γ' to construct P_2. As in the construction of P_1, we create a channel of small non-zero width around each connector so that we have a polygon with holes. We denote the copies of $R(c)$ and $B(u)$ in P_2 by $R_2(c)$ and $B_2(u)$. We create four inward dents on the boundary of $R_2(c)$, and place the points $u, v, w, d, q, q', r, r', s, s'$, as shown in Figure 5(a). Finally, we place the hole h ensuring that no peak in $\{u, v, w\}$ is 1-bend visible to $\{q', r', s'\}$, e.g., see Figure 5(b). We refer the reader to the full version [10] for the formal details of the construction.

We now modify the rectangles that correspond to the literals. Let x be a literal and let $B_2(x)$ be the corresponding rectangle in Γ'_2. The modification for $B_2(x)$ is analogous to that of $B_1(x)$. Specifically, for every visibility region (of some peak $p \in \{q', r', s'\}$) that intersects the box $B_1(x)$ in Γ'_1, we construct a point μ on the boundary of box $B_2(x)$ such that μ and p are visible in P_2. Figure 5(b) illustrates such visibilities with dashed lines.

[1]Recall that c may contain duplicates of a literal.

3.3 Properties of Compatible Drawings

In this section we prove some key properties of compatible triangulations $T(P_1)$ and $T(P_2)$ of P_1 and P_2, respectively. For clause c, let $\overline{R}_1(c)$ be the clause region $R_1(c)$ plus its three attached channels.

Lemma 2 *If c is a clause such that no peak q', r', s' is adjacent in $T(P_1)$ to a point outside $\overline{R}_1(c)$, then there are at least 6 Steiner points in $\overline{R}_1(c)$.*

Proof. Consider the 9 points $\{u, v, w, q, q', r, r', s, s'\}$. In P_1 each point in this set is the peak of a dent, so by Lemma 1, each of these 9 points must be adjacent in $T(P_1)$ to a vertex or a Steiner point. The only vertices visible to any of the 9 peaks are the μ-points visible to q', r', s', but they lie outside $\overline{R}_1(c)$. We assumed there is no edge from q', r', s' to a point outside $\overline{R}_1(c)$. The other 6 peaks are not visible to any point outside $\overline{R}_1(c)$. Thus each of the 9 peaks must be adjacent to a Steiner point in $\overline{R}_1(c)$. No point in $\overline{R}_1(c)$ is visible to more than two peaks. Thus we need at least $\lceil \frac{9}{2} \rceil = 5$ Steiner points. The only way that 5 Steiner points suffice is to use 4 Steiner points that are each adjacent to two peaks. Pairs of peaks that are visible to a common point in both P_1 and P_2 are indicated by edges in the graph H shown in Figure 6(a). We require a matching of size 4 in H. Observe that H is bipartite so the maximum size of a matching is equal to the minimum size of a vertex cover. The set $\{q, r, s\}$ is a vertex cover of size 3. Thus there is no matching of size 4, and the Lemma follows. □

Lemma 3 *For any clause c, there are at least 5 Steiner points in $\overline{R}_1(c)$.*

Proof. Consider the triangulation of P_1. The case where no peak q', r', s' has an incident edge to a point outside $\overline{R}_1(c)$ is covered by Lemma 2. It remains to consider the cases when there is such an edge.

Our argument will be partly about the graph H (in Figure 6(a)) of pairs of peaks that are visible to a common point in both P_1 and P_2, and partly about the

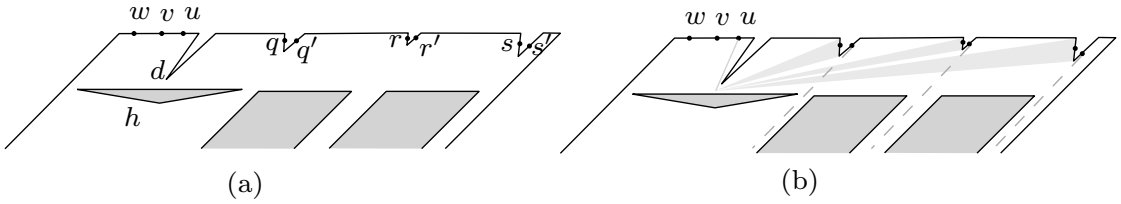


Figure 5: (a) A clause region in P_2. (b) The vertex u is not 1-bend visible to q', r', s'.

geometry of P_1. First we note that the argument used above in the proof of Lemma 2 can be strengthened to show that if we use just one edge from a peak to a point outside the clause region then we still need 5 Steiner points inside the region. In graph H, observe that if one of q', r', s' is removed, then we have 8 vertices, and a maximum matching of size 3, which means that we can use 3 edges (Steiner points) to cover 6 vertices, leaving 2 vertices that need one Steiner point each, for a total of 5 Steiner points. It remains to consider the cases where at least two of the points q', r', s' have an incident edge to a point outside the clause region. We deal with the case where q' has such an edge and the case where r' has such an edge but q' does not.

Suppose there is an edge e from q' to a point outside $\overline{R}_1(c)$. Observe that edge e cuts off the visibility regions of v and w. The effect on graph H is to remove the edges of H incident to v and w, e.g., see Figure 6(b). Thus we need one Steiner point for each of v and w, one Steiner point for r (irrespective of how r' is connected), one Steiner point for s and one more for u, a total of at least 5.

Next suppose there is no edge from q' to a point outside of $\overline{R}_1(c)$, but there is an edge e' from r' to a point outside of $\overline{R}_1(c)$. The edge e' cuts off the visibility region of w. The effect on graph H is to remove the edges (w, r) and (w, s), e.g., see Figure 6(c). We then need a Steiner point for s (irrespective of how s' is connected), and for the remaining 6 vertices $\{u, v, w, q, q', r\}$, we have a subgraph with a minimum vertex cover $\{q, r\}$ of size 2, thus a maximum matching of 2 edges (Steiner points) to cover 4 vertices, leaving 2 vertices that need one Steiner point each, for a total of 5 Steiner points. □

Lemma 4 *If $T(P_1)$ and $T(P_2)$ use $5|C|$ Steiner points each, then for any clause c, there is an edge in $T(P_1)$ from at least one of q', r', s' to a μ-point.*

Proof. By Lemma 3 every region $\overline{R}_1(c)$ has at least 5 Steiner points. Thus every such region must have exactly 5 Steiner points and there are no Steiner points in the variable regions. Suppose there is a clause c such that $T(P_1)$ has no edge from q', r' or s' to a μ-point. Then there is no edge from q', r' or s' to a point outside $\overline{R}_1(c)$. But then by Lemma 2 the clause region must have at least 6 Steiner points, a contradiction. □

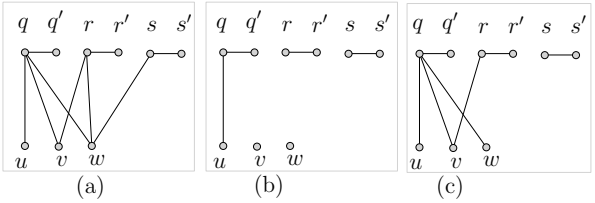


Figure 6: (a) Graph H of pairs of peaks that that are visible to a common Steiner point in both P_1 and P_2. (b)–(c) Illustration for Lemma 3.

3.4 Reduction

Theorem 5 *The following problem is NP-hard: Given a pair of compatible polygonal regions P_1, P_2, and $k \in \mathbb{N}$, decide if P_1 and P_2 have compatible triangulations with at most k Steiner points.*

Proof. Let $I = (U, C)$ be an instance of MRP-3SAT, and let P_1 and P_2 be the corresponding compatible polygons, as described in Sections 3.1–3.2. The full version [10] presents further details on how to construct P_1 and P_2 using a polynomial number of bits, so this is a polynomial-time reduction. We now prove that P_1 and P_2 admit a pair of compatible triangulations, each with at most $5|C|$ Steiner points, if and only if I admits a satisfying truth assignment.

We first assume that P_1 and P_2 admit compatible triangulations with at most $5|C|$ Steiner points. By Lemma 4, for any clause c there is an edge in the triangulation of P_1 from at least one peak $z \in \{q', r', s'\}$ to a μ-point, say $\mu_{c,x}$. We use the edge $(z, \mu_{c,x})$ to assign a truth value to variable x. If c is a positive (resp., negative) clause, then we set x to true (resp., false). Clearly we have satisfied each clause. If there is a variable u whose truth value is not assigned yet, then setting the truth value of u arbitrarily would still keep the clauses satisfied. It remains to show that the truth-value assignment is consistent. Suppose there is a variable u such that some clause c forces u to be true, and some other clause c' forces u to be false. Without loss of generality we may assume that c is positive and c' is negative. Consequently, in each of $R_1(c)$ and $R_1(c')$, there exists a peak that is incident to some μ-point in $B_1(x)$. By construction of the μ-points in $B_1(x)$, the

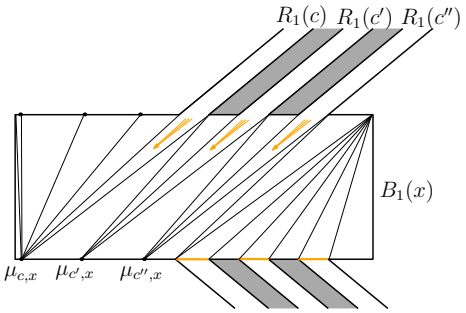


Figure 7: A triangulation for $B_1(x)$, where $x = true$.

two corresponding edges cross, a contradiction.

Assume now that I admits a satisfying truth assignment. We will find corresponding compatible triangulations of P_1 and P_2. For each variable x, if x is set to true, then we close the channels of the negative clauses and construct the compatible triangulations of the rectangles $B_1(x)$ and $B_2(x)$ using the μ-points on the bottom side of these rectangles, e.g., see Figure 7. The construction when x is set to false is symmetric.

Since every clause contains at least one true literal, for every clause c, there exist one or more peaks in P_1 that are visible to their corresponding μ-points. We show that in each scenario, the corresponding clause gadgets can be triangulated in a compatible fashion. We include the details in the full version [10]. □

4 Conclusion

We have proved that computing compatible triangulations with at most k Steiner points is NP-hard for polygons with holes. The following questions are open:

1. Is the problem in NP? Is it complete for existential theory of the reals [14]?
2. What is the complexity of the problem for a pair of simple polygons? For a pair of rectangles with points inside?
3. How hard is it to decide if two polygonal regions, or two rectangles with points inside, have compatible triangulations with no Steiner points? For simple polygons, this can be decided in polynomial-time [3].

References

[1] O. Aichholzer, F. Aurenhammer, F. Hurtado, and H. Krasser. Towards compatible triangulations. *Theoretical Computer Science*, 296(1):3–13, 2003.

[2] S. Alamdari, P. Angelini, F. Barrera-Cruz, T. M. Chan, G. Da Lozzo, G. Di Battista, F. Frati, P. Haxell, A. Lubiw, M. Patrignani, V. Roselli, S. Singla, and B. T. Wilkinson. How to morph planar graph drawings. *to appear in SIAM Journal on Computing*, 2017.

[3] B. Aronov, R. Seidel, and D. L. Souvaine. On compatible triangulations of simple polygons. *Computational Geometry*, 3:27–35, 1993.

[4] M. Babikov, D. L. Souvaine, and R. Wenger. Constructing piecewise linear homeomorphisms of polygons with holes. In *Proceedings of the 9th Canadian Conference on Computational Geometry (CCCG)*, 1997.

[5] W. V. Baxter III, P. Barla, and K.-i. Anjyo. Compatible embedding for 2D shape animation. *IEEE Transactions on Visualization and Computer Graphics*, 15(5):867–879, 2009.

[6] T. M. Chan, F. Frati, C. Gutwenger, A. Lubiw, P. Mutzel, and M. Schaefer. Drawing partially embedded and simultaneously planar graphs. *Journal of Graph Algorithms and Applications*, 19(2):681–706, 2015.

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

[8] H. Gupta and R. Wenger. Constructing pairwise disjoint paths with few links. *ACM Transactions on Algorithms*, 3(3):26, 2007.

[9] E. Kranakis and J. Urrutia. Isomorphic triangulations with small number of Steiner points. *International Journal of Computational Geometry & Applications*, 9(2):171–180, 1999.

[10] A. Lubiw and D. Mondal. On compatible triangulations with a minimum number of Steiner points. *CoRR*, abs/1706.09086, 2017. http://arxiv.org/abs/1706.09086.

[11] J. Pach, F. Shahrokhi, and M. Szegedy. Applications of the crossing number. *Algorithmica*, 16(1):111–117, 1996.

[12] J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. *Graphs and Combinatorics*, 17(4):717–728, 2001.

[13] A. Saalfeld. Joint triangulations and triangulation maps. In *Proceedings of the Third Annual Symposium on Computational Geometry (SoCG)*, pages 195–204. ACM, 1987.

[14] M. Schaefer. Complexity of some geometric and topological problems. In *Proceedings of the 17th International Symposium on Graph Drawing (GD)*, volume 5849 of *LNCS*, pages 334–344. Springer, 2010.

[15] D. L. Souvaine and R. Wenger. Constructing piecewise linear homeomorphisms. Technical report, DIMACS, New Brunswick, New Jersey, 1994.

[16] V. Surazhsky and C. Gotsman. High quality compatible triangulations. *Engineering with Computers*, 20(2):147–156, 2004.

[17] C. Thomassen. Deformations of plane graphs. *Journal of Combinatorial Theory, Series B*, 34(3):244–257, 1983.

Planarity Preserving Augmentation of Topological and Geometric Plane Graphs to Meet Parity Constraints

I. Aldana-Galván* J.L. Álvarez-Rebollar† J.C. Catana-Salazar* E. Solís-Villarreal* J. Urrutia‡
C. Velarde§

Abstract

We introduce the augmentation problem to meet parity constraints in topological and plane geometric graphs. We show a family of plane topological graphs such that any augmentation leaves at least $\frac{2n}{5}$ vertices without meeting their parity constraints, and a family of plane geometric trees such that any augmentation leaves at least $\lfloor \frac{n}{10} \rfloor$ vertices without meeting their parity constraints. We prove that the problem of adding a minimum number of edges to plane topological graphs is $\mathcal{NP}$-Hard. When the input graph is a topological tree finding a minimum set of edges that needed to be added to meet a parity constraint is solvable in $\mathcal{O}(n)$ time and $\mathcal{O}(1)$ space. We also establish a lower bound of $\lceil \frac{11n}{15} \rceil$ on the number of necessary edges to augment a topological graph when the graph is augmentable, and a lower bound of $\lceil \frac{6n}{11} \rceil$ on the number of necessary edges to augment a geometric tree when the tree is also augmentable to meet the parity constraints.

1 Introduction

A *topological graph* is a graph together with an embedding on the plane, such that the vertices are represented by distinct points and the edges are represented by Jordan arcs connecting pairs of vertices.

A *geometric graph* is a graph in which its vertices are represented by points on the plane, and its edges by straight line segments joining pairs of vertices. A *planar* graph is a graph that can be embedded in the plane in such a way that its edges may intersect only at their endpoints. Such an embedding is called a *planar embedding* of the graph.

*Posgrado en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México, Ciudad de México, México, ialdana@ciencias.unam.mx, {j.catanas, solis_e}@uxmcc2.iimas.unam.mx

†Posgrado en Ciencias Matemáticas, Universidad Nacional Autónoma de México, Ciudad de México, México, chepomich1306@gmail.com

‡Instituto de Matemáticas, Universidad Nacional Autónoma de México, Ciudad de México, México, urrutia@matem.unam.mx

§Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México, Ciudad de México, México, velarde@unam.mx

A *plane graph* is a planar emmbeding of a planar graph, and we refer to its points as vertices and lines as edges. Two or more geometric graphs are *compatible* if their union is a plane geometric graph.

Given a plane topological (resp. geometric) graph $G = (V, E)$ and a set of parity constraints $C = \{c_1, c_2, ..., c_n\}$ where each $v_i \in V$ has assigned the constraint c_i (to be of degree odd or to be of degree even), the *augmentation problem to meet parity constraints* is that of finding a set of edges E', where $E' \cap E = \emptyset$, such that:

1. $G' = (V, E \cup E')$ is a plane topological (resp. geometric) simple graph.
2. The degree of each vertex $v_i \in G'$ meets its parity constraint c_i.

Observe that if a vertex of G does not meet its parity constraint, then its degree must increase by an odd integer. In what follows we will denote by P the set of vertices of G that do not satisfy its degree constraints in $C = \{c_1, c_2, ..., c_n\}$. Let H be the graph with vertex set V, and edge set E'. The degree of each vertex in H is odd, and thus H has an even number of vertices.

We say that the neighborhood of a vertex is *saturated* if there is no edge that can be added to G, incident to v, and avoiding edge crossings. For example, if G is a planar graph and the subgraph induced by v_i and its neighbors is a wheel with no other vertices inside it, then the neighborhood of v_i is saturated, or for short v_i is saturated. Thus from now on we will assume that the degree of any vertex in P is smaller than $n - 1$ and its neighborhood is not saturated.

It is easy to see that there are many planar graphs that cannot be extended to meet a set of parity constraints. For example take a planar graph that is a triangulation Δ minus two edges $e = (u, v)$ and $e' = (x, y)$ such that u, v, x, and y are different vertices. Then we cannot change the parities of u and x without breaking the planarity of Δ.

Then, the graphs we study in this paper must have the following properties:

1. The graphs are simple.

2. If a vertex is in P, its degree is smaller than $n-1$ and its neighborhood is not saturated.

Let $G = (V, E)$ be a topological (resp. geometric) simple plane graph, and P the set of vertices not meeting their parity constraints in G. The complementary graph $\overline{G} = (V, \overline{E})$ of G, is composed by the set of vertices V and all the edges $e \notin E$ that can be added to G in such a way that $G \cup e$ is plane.

In all the figures throughout this paper, the vertices in P are represented as empty discs, and the dashed edges represent edges added by the augmentation process.

2 Planarity Preserving Augmentation of Topological Graphs

First we consider the case when the input graph is a plane topological tree.

Theorem 1 *Let $T = (V, E)$ be a plane topological tree that is not a star, and that we want to augment to a plane graph with a set of parity constraints $C = \{c_1, c_2, ..., c_n\}$. Then T can always be augmented to meet its parity constraints in $\mathcal{O}(n)$ time, with the addition of at most $\frac{k}{2} + 1$ edges, where $|P| = k$.*

Proof. Let $T' = (V', E')$ be the minimal topological connected subgraph of T containing all the elements in P. If T' is a star, our problem can be solved easily by using a vertex v in T not adjacent to the center of the star, see Figure 1. Suppose then that T' is not a star.

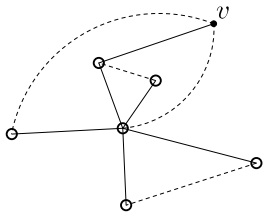


Figure 1: Augmentation of T' when it is a star.

Let $F_{T'}$ be the unique face in T'. It is easy to see that the nodes in T not in T' can be discarded now, and reinserted again once we finish our augmentation process in T'. Note first that any pair of vertices in P can be joined by a Jordan curve contained in $F_{T'}$.

Let $v_1, v_2 \in P$ be two vertices such that their edge distance in T' is at least three, and such that when we add the edge (v_1, v_2) to T' we form a cycle $\mathcal{C}$ containing no vertex of T' in its interior. Such vertices exist, for otherwise, T' would be a star.

Let F_a be the face bounded by $\mathcal{C}$. See Figure 2.

The addition of e_a changes the parities of v_1 and v_2, therefore these two vertices no longer belong to P.

We proceed by iteratively looking for a pair of vertices $v_i, v_{i+1} \in P$, such that their edge distance in T' is at least two and such that the cycle obtained by adding the edge $e_j = (v_i, v_{i+1})$ contains no vertices of P in its interior. These vertices will exist as long as we have at least four vertices in P whose parities have not changed.

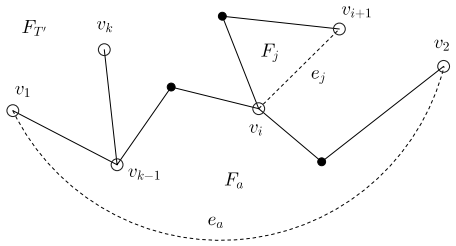


Figure 2: A topological tree.

Let v_{k-1}, v_k be the last two vertices in P. Note that these two vertices form the only one pair that could not be at distance two. Then, we use the first pair of joined vertices v_1, v_2 as follows. Since v_1 is at least at distance three from v_2 then we have the following cases:

- *Case 1:* v_1 is at least at distance two from v_{k-1} and v_2 is at least at distance two from v_k. Then, we can exchange e_a by (v_1,v_{k-1}) and (v_2,v_k), or by (v_1,v_k) and (v_2,v_{k-1}).
- *Case 2:* Suppose w.l.o.g. that v_1 is at distance one from v_{k-1} and v_2 is at least at distance two from v_k. Then, we can exchange e_a by (v_1,v_k) and (v_2,v_{k-1}).
- *Case 3:* Suppose w.l.o.g. that v_1 is at distance one from v_{k-1} and v_2 is at distance one from v_k. Then, we can exchange e_a by (v_1,v_k) and (v_2,v_{k-1}).

It is not hard to see that the above process can be carried out in linear time.

It follows that by adding a compatible matching of the vertices in P, or a compatible matching that covers all the vertices of P but two (which can be joined by a path with two edges) we have a topological graph obtained by the addition of at most $\frac{k}{2}+1$ edges. Moreover the bound is tight since there is no way to augment a topological tree with k vertices in P with less than $\frac{k}{2}$ edges. □

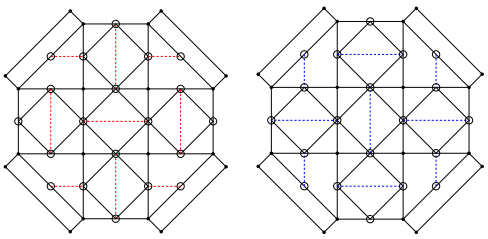
Next, we present a result about the hardness of the augmentation problem to meet parity constraints.

Theorem 2 *Let $G = (V, E)$ be a plane topological graph and C a set of parity constraints assigned to V. The problem of deciding if there exists a set E' with the minimum number of edges such that, $G' = (V, E \cup E')$ is a plane topological graph meeting all the parity constraints in C while preserving its embedding is $\mathcal{NP}$-Hard.*

Proof. We do the proof by reducing the Planar 3SAT Problem to the augmentation problem to meet parity

constraints. Given a planar 3SAT formula Φ, we build a plane topological graph G_Φ that we want to augment with the minimum number of edges such that all of its vertices meet their parity constraints.

We define three subgraphs, or gadgets, as building blocks for our reduction: The basic gadget, the literal gadget and the clause gadget. The basic gadget consists of a graph which has only two possible augmentations: Positive (in red) and Negative (in blue), as illustrated in Figure 5.

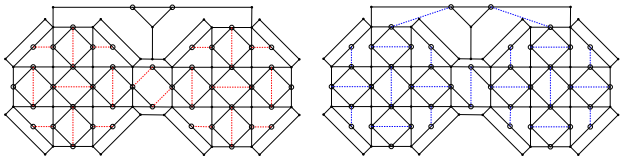


(a) (b)

Figure 3: Possible augmentations of the basic gadget: (a) Positive. (b) Negative.

A literal gadget is a subgraph that has butterfly shape, see Figure 4. The wings are two basic gadgets, the body has two white vertices, and the antennae has other two white vertices. Both wings must have the same augmentation, either positive or negative. Accordingly, we call the former a positive augmentation and the latter a negative augmentation of the literal gadget.

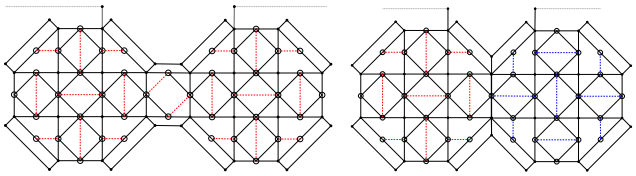
Note that when the literal is assigned a positive value, the antennae of the butterfly remain without changing their parity.



(a) (b)

Figure 4: Possible augmentations of a literal gadget. (a) A positive augmentation. (b) A negative augmentation

For each occurrence of a variable x in Φ we will have a literal gadget. All the literal gadgets of the same variable will be joined to form a chain of butterflies, where the right wing of a butterfly will be joined to the left wing of the next butterfly. Two consecutive butterflies will be joined as follows: If in both occurrences x is negated or non-negated then they will be joined as illustrated in Figure 5a. Otherwise, they will be joined as illustrated in Figure 5b. Finally, we join the leftmost



(a) (b)

Figure 5: The union of two literals of the same variable: (a) When both occurrences are negated or non-negated (with a positive augmentation). (b) When one occurrence is negated and the other is not (with a positive augmentation, left wing and negative augmentation, right wing).

wing with the rightmost wing with a band. If the obtained chain of butterflies has an odd number of white vertices we add an extra white vertex inside the band as illustrated in Figure 6.

A clause gadget F_c is joined with three literal gadgets ℓ_i, ℓ_j, and ℓ_k, an example of a clause is illustrated in Figure 6. We say that F_c has true value if the white vertices of F_c are matched with two vertices of the literal gadgets having positive value, otherwise F_c has false value. For example, if ℓ_i is the only one having positive value, then you can join the two vertices of the antennae of ℓ_i with the two white vertices of F_c changing their parity and F_c has true value. If all the literals have negative value, then the two white vertices of F_c cannot be augmented with only one edge and F_c has false value. It is straightforward to see that all the other cases can be solved leaving F_c with true value.

The proof follows since if it can be found a set with the minimum number of edges to augment G_Φ to meet its parity constraints, then, an assignation of values that satisfies Φ is obtained. □

Next, we present a family of plane topological graphs to establish a lower bound on the number of edges needed to be added to a plane topological graph to meet a set of parity constraints, while preserving their planarity.

Theorem 3 *There exists a family of plane topological graphs G with n vertices such that any augmentation of them in which all of its vertices have even degree requires the addition of at least $\lceil \frac{11n}{15} \rceil$ edges.*

Proof. Consider the graph shown in Figure 7. Such graph has 15 vertices, 12 of which are odd degree vertices. We claim that this graph cannot be augmented to meet its parity constraints with less than 12 edges.

Each leaf of the graph is enclosed in a triangular face. Note that there is no way to join a leaf with any other odd degree vertex using only one edge avoiding crossings. Note also that there is no way to join two inner

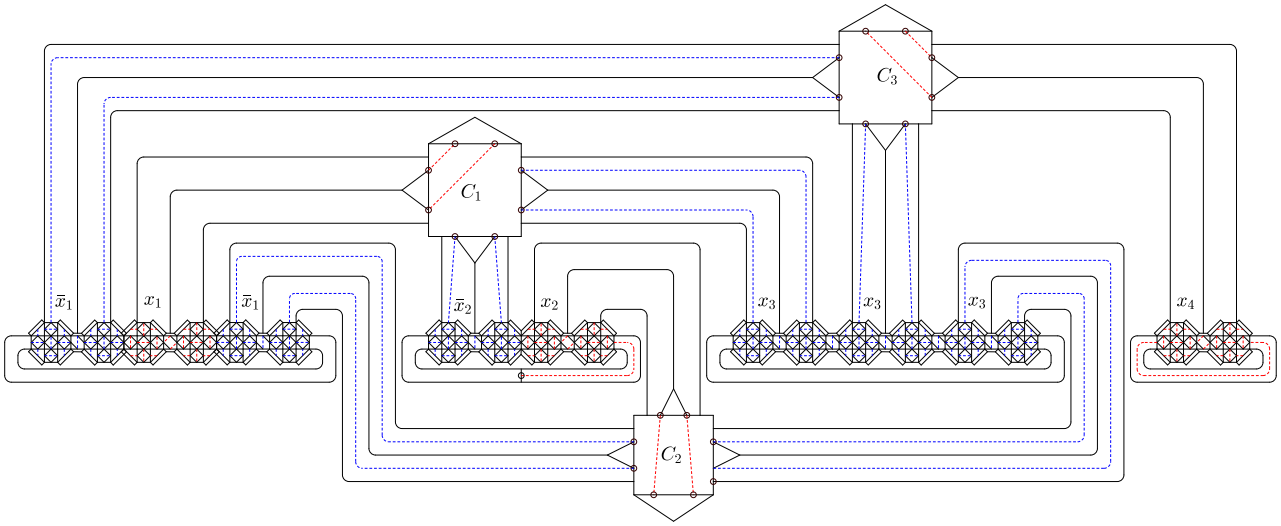


Figure 6: Graph G_Φ for $(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee x_3 \vee x_4)$, augmented in accordance to the assignment $(x_1, x_2, x_3, x_4) = (T, T, F, T)$. The distribution of the clause and literal gadgets is consistent with the topology of the given 3SAT plane graph for Φ.

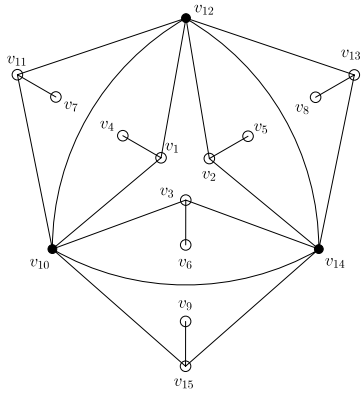


Figure 7: A plane topological graph G that requires at least 12 edges to augment such that all of its vertices end with even degree.

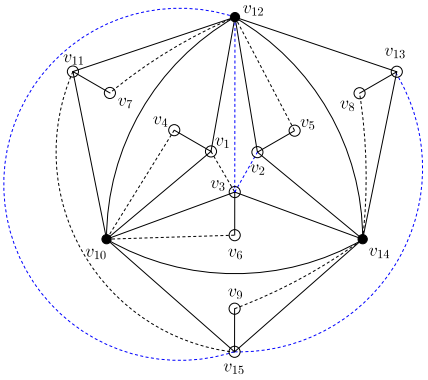


Figure 8: The dashed lines change the parities of all the odd vertices of G.

odd degree vertices with two outer odd degree vertices with disjoint paths. Therefore, two inner odd degree vertices have to be joined with an edge. The same happens for the three outer odd degree vertices. Figure 8 shows an example of the previous described augmentation. Finally, there are two odd degree vertices, one in the inner face and one in the exterior face of the graph, such that the only way to change their parity is to join them with a path with 4 edges, depicted in Figure 8 with dashed blue lines. Therefore we require a total of 12 edges to augment the graph in order to meet its parity constraints.

Now consider an even triangulation with n vertices (a triangulation is said to be even if all its vertices have even degree). Assign to each vertex of the triangulation a copy of the graph shown in Figure 7. Embed each copy in one of the adjacent faces of its assigned vertex, in such a way that there are no two copies inside of the same face, we can do this because the triangulation has $2n-4$ faces. Replace each vertex of the triangulation by the vertex v_{10}, v_{12} or v_{14} of its assigned copy. Figure 9 shows an example of this construction.

Note that the augmentation of the leaves and the most inner odd degree vertices of each copy (save the vertices that correspond to v_2 and v_{13}) requires the same number of edges as in Figure 8. We can join a copy of the vertex v_2 to a copy of the vertex v_{13} with a path of length 3 instead of a path of length 4, as shown in Figure 9, thus saving one edge per pair. It follows that it is required $\lceil \frac{11n}{15} \rceil$ edges to augment such graph to meet its parity constraints. □

Theorem 4 *There is a family of topological graphs such that it is not possible augment it to change the parity of all of its n vertices. In these graphs, we can change the parity of at most* $\frac{2n}{5}$ *of its vertices.*

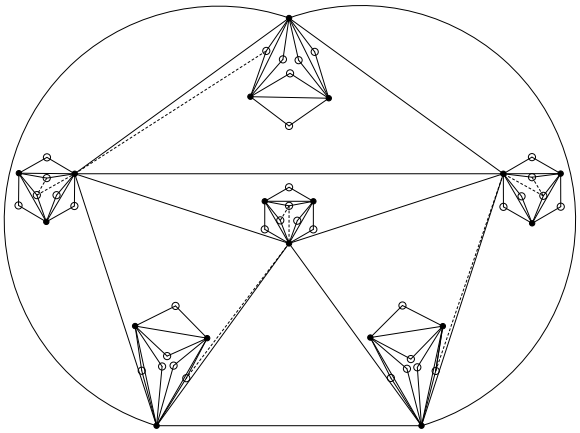


Figure 9: A plane topological graph requiring $\lceil \frac{11n}{15} \rceil$ edges to augment it to an Eulerian plane topological graph.

Proof. Our family of graphs is constructed as follows: Take k disjoint pentagons, and construct a graph whose vertices are the vertices of our pentagons. Add enough edges until the exterior of the union of the pentagons is triangulated, see Figure 10.

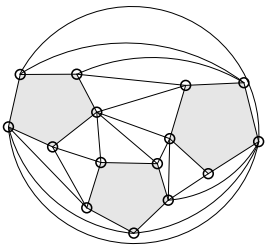


Figure 10: A plane topological graph in which any augmentation leaves $\frac{2n}{5}$ vertices without meeting its parity constraints.

It is easy to see that we can change the parity of only two of the vertices of each pentagon, that is, only $\frac{2n}{5}$ of the vertices of graph. □

3 Planarity Preserving Augmentation of Geometric Graphs

First we consider the special case when the input graph is a plane geometric tree. We show that the family of geometric trees proposed by C. Toth in [1], and refined later by A. García and J. Tejel in [2], establishes a lower bound on the number of edges needed to augment a plane geometric tree to meet a set of parity constraints while preserving its planarity.

Theorem 5 *There exists a family of plane geometric trees such that to augment them to meet a set of parity constraints requires the addition of* $\lceil \frac{6n}{11} \rceil$ *edges.*

Proof. In the family of trees that we will generate, we want to change the parity of all of its vertices of odd degree.

The basis of our construction is a tree similar to that introduced in [2]. It consists of a tree with 7 leaves and 8 vertices of odd degree shown in Figure 11.

Since the four internal leaves of the tree are placed in such a way that they cannot see each other, the only two edges joining two odd degree most inner vertices are (h_2, v_1) and (h_3, v_1). In the exterior, only two of the three external leaves, h_5, h_6, and h_7, of the tree can be joined by an edge. Thus, we can add one edge in the interior of the tree, and one edge between two external leaves joining odd degree vertices. Suppose w.l.o.g. that we joined h_3 to v_1 and h_6 to h_7.

Since the 4 odd degree vertices remaining do not have direct visibility with each other, then at least two edges to join each pair of them are needed. Therefore, to augment the basic construction, 6 edges are necessary.

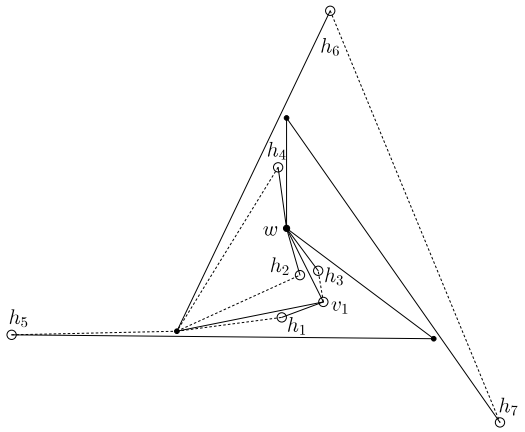


Figure 11: A plane geometric tree that requires six additional edges to augment it to a graph in which all of its vertices have even degree.

To generalize the construction to a family of plane geometric trees, we take a copy scaled down of the basic construction and we embed it attached to vertex w as shown in Figure 11). Note that vertex w becomes an odd degree vertex.

If we iterate this process, at each iteration we add 11 vertices, and create 8 vertices with odd degree. It follows that to augment any member of the obtained family of trees in which all of their vertices have even degree we need at least $\lceil \frac{6n}{11} \rceil$ edges. □

We show now a family of geometric trees in which not all of their vertices can change parity when we add edges to them. Such a family was initially proposed by García, Huemer, Hurtado and Tejel in [3].

Theorem 6 *There exists a family of plane geometric trees, such that no matter how we augment them, at*

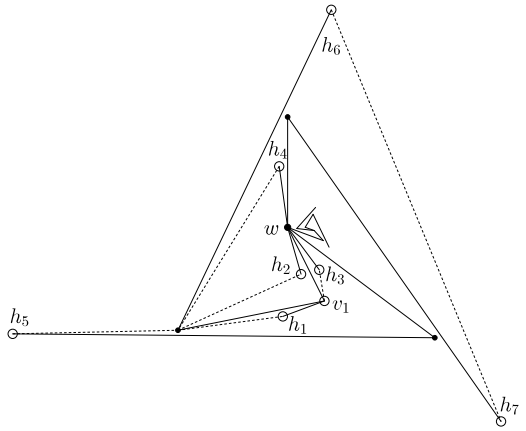


Figure 12: Constructing a family of plane geometric trees that require $\lceil \frac{6n}{11} \rceil$ additional edges in order to meet their parity constraints.

least $\lfloor \frac{n-1}{10} \rfloor$ *of its vertices are left with their parity unchanged.*

Proof. Consider the geometric tree T shown in Figure 13. We want to show that no matter how we augment it, at least $\lfloor \frac{n}{10} \rfloor$ of its vertices remain with odd degree. Note that in the complementary graph $\overline{G}$ of T, v_1 has degree 1 as it only sees vertex z (apart from its neighbors in T). A symmetric situation happens with v_2. If (v_1,z) is selected to augment T, then (v_2,w) cannot be part of such augmentation since both edges cross each other. Thus in any augmentation of T one of v_1 or v_2 remains unchanged.

We can replicate the previous situation to build an arbitrarily large tree as shown in Figure 14. In this manner, any augmentation leaves at least one odd vertex per pair of degree-3 vertices.

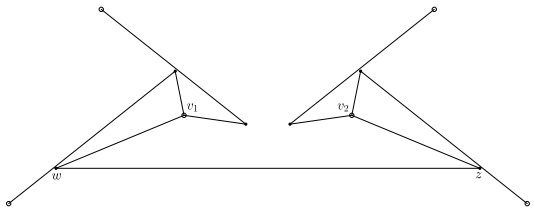


Figure 13: In any augmentation of this graph, at least one of v_1 and v_2 cannot change its parity.

It follows that any augmentation of T leaves at least $\lfloor \frac{n-1}{10} \rfloor$ odd degree vertices without meeting their parity constraints. □

4 Conclusions

In this paper, we studied the augmentation problem to meet parity constraints in topological and plane geometric graphs. We obtained a family of plane topological graphs with a set of parity constraints such that any augmentation of them leaves at least $\frac{2n}{5}$ vertices without meeting their parity constraints. We also obtained a family of plane geometric trees such that any augmentation leaves at least $\lfloor \frac{n-1}{10} \rfloor$ vertices without meeting their parity constraints. We also proved that the complexity of finding the smallest number of edges needed to augment a plane topological graph to meet a set of parity constrains is $\mathcal{NP}$-Hard.

The case in which the input graph is a topological tree, the problem is always solvable with the minimum number of additional edges in $\mathcal{O}(n)$ time and $\mathcal{O}(1)$ space. We also established a lower bound of $\lceil \frac{11n}{15} \rceil$ on the number of necessary edges to augment a topological graph when the graph is augmentable, and a lower bound of $\lceil \frac{6n}{11} \rceil$ on the number of necessary edges to augment a geometric tree when the tree is also augmentable. Finding upper bounds in the two previous problems is still open.

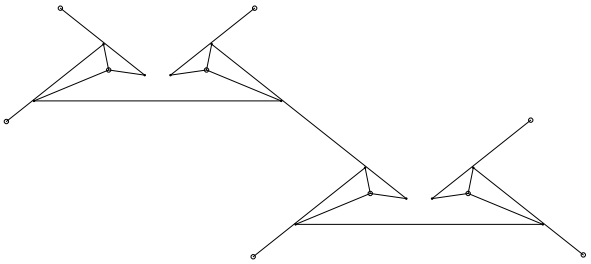


Figure 14: Family of plane geometric trees such that any augmentation leaves at least $\lfloor \frac{n-1}{10} \rfloor$ vertices without meeting their parity constraints.

5 Acknowledgment

We are grateful to Alfredo García and Javier Tejel for their contribution with useful ideas to the elaboration of this paper.

References

[1] Tóth, Csaba D. Connectivity augmentation in planar straight line graphs. *European Journal of Combinatorics*, 33(3):408–425. Elsevier, 2012.

[2] Hurtado, Ferran and Tóth, Csaba D. Plane geometric graph augmentation: a generic perspective. *Thirty Essays on Geometric Graph Theory*, pages 327–354. Springer, 2013.

[3] García, Alfredo and Huemer, Clemens and Hurtado, Ferran and Tejel, Javier Compatible spanning trees. *Computational Geometry*, pages 563–584. Elsevier, 2014.

Interference Minimization in k-Connected Wireless Networks*

Stephane Durocher† Sahar Mehrpour‡

Abstract

Given a set of positions for wireless nodes, the *k-connected interference minimization* problem seeks to assign a transmission radius to each node such that the resulting network is k-connected and the maximum interference is minimized. We show there exist sets of n points on the line for which any k-connected network has maximum interference $\Omega(\sqrt{kn})$. We present polynomial-time algorithms that assign transmission radii to any given set of n nodes to produce a k-connected network with maximum interference $O(\sqrt{kn})$ in one dimension and $O(\min\{k\sqrt{n}, k\log\lambda\})$ in two dimensions, where λ denotes the ratio of the longest to shortest distances between any pair of nodes.

1 Introduction

1.1 Interference Minimization and k-Connectivity

A network must be connected if a multi-hop communication channel is required between every pair of nodes. Various secondary objectives can be considered in addition to the connectivity requirement, often resulting in an optimization problem to construct a network that meets both criteria. Common additional objectives include minimizing the maximum or average power consumption, sender-receiver route length, node degree, ratio of route length to Euclidean distance, and, of particular relevance to wireless networks, interference [11]. By increasing or decreasing its transmission power, a wireless node increases or decreases its transmission range. If wireless signal strength is assumed to fade uniformly in all directions, then the range within which transmission exceeds a given minimum threshold corresponds to a disk centred at the point of transmission; we refer to the disk's radius as the transmitting node's *transmission radius*. Under the *receiver-based interference model* [16], two nodes p_1 and p_2 can communicate if they lie mutually in each other's transmission ranges, and any node q_1 that lies in the transmission range of a node q_2 receives interference from q_2, regardless of whether q_1 can communicate with q_2. Given a set of node positions as input, the objective of the *interference minimization problem* is to assign a transmission radius to each node to produce a connected network that minimizes the maximum interference among all nodes. The interference minimization problem has been examined extensively under the receiver-based interference model over the past decade (e.g., [2, 3, 5, 7, 11, 13, 16–18]).

Maintaining network connectivity is critical to preserving multi-hop communication channels between all pairs of nodes. Connectivity alone is insufficient to preserve communication in case of node failure: a connected network can become disconnected when even a single node fails. Guaranteeing network connectivity in the presence of node failure requires multiple disjoint routes joining every pair of nodes, i.e., redundancy in the network's connectivity. A network is *k-connected* if it remains connected whenever fewer than k nodes are removed. The factor k parameterizes the network's degree of connectivity. In this work, we examine interference minimization on k-connected networks. Given a set of node positions, the *k-connected interference minimization problem* is to assign a transmission radius to each node to produce a k-connected network while minimizing the maximum interference at any node. To the authors' knowledge, this is the first work to examine interference minimization in k-connected networks.

1.2 Definitions

We represent the position of a wireless node by a point $p_i \in \mathbb{R}^d$. The set $P = \{p_1, \ldots, p_n\} \subseteq \mathbb{R}^d$ represents positions for a set of n nodes, along with a corresponding function, $r : P \to \mathbb{R}^+$, that associates a positive real transmission radius with each node. Communication in a wireless network is often modelled by a symmetric disk graph (SDG); the *symmetric disk graph* of P with respect to r is an undirected graph with vertex set P and edge set $\{(p,q) \mid \{p,q\} \subseteq P \wedge r(p) \geq \text{dist}(p,q) \wedge r(q) \geq \text{dist}(p,q)\}$, where $\text{dist}(u,v)$ denotes the Euclidean distance between the points u and v in $\mathbb{R}^d$ [1]. In this paper we focus on point sets in one or two dimensions ($d \in \{1,2\}$).

von Rickenbach et al. [16] introduced the receiver-centric interference model. In this model, the *interference* at the node $p \in P$, denoted $I(p)$, is the number of nodes in P whose transmission range covers p. That is, $I(p) = |\{q \mid q \in P \wedge \text{dist}(p,q) \leq r(q)\}|$. The *maximum interference* for the set of points P with transmission radii given by r is the maximum $I(p)$ over all

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

†University of Manitoba, Canada, durocher@cs.umanitoba.ca
‡University of Utah, USA, mehrpour@cs.utah.edu

$p \in P$. For a given graph G on the point set P, let $I(G) = \max_{p \in P} I(p)$. The *interference minimization problem* is to assign transmission radii (i.e., to define the function r) for a given set of points $P \subseteq \mathbb{R}^d$ such that the corresponding symmetric disk graph G is connected and $I(G)$ is minimized.

A graph G is *connected* if there is a path (a sequence of adjacent vertices) joining every pair of vertices in G. A graph G is *k-connected* if there are k disjoint paths between every pair of vertices in G or, equivalently, if the removal of any j vertices does not disconnect G, for all $j < k$. The *k-connected interference minimization problem* is to assign transmission radii (i.e., define the function r) for a given set of points $P \subseteq \mathbb{R}^d$ such that the corresponding symmetric disk graph G is k-connected and $I(G)$ is minimized. Let $\mathrm{OPT}_k(P)$ denote the minimum maximum interference among all k-connected networks on P.

Given a set $P \subseteq \mathbb{R}^d$, let $\mathrm{MST}(P)$ denote its Euclidean minimum spanning tree, $\mathrm{DT}(P)$ its generalized Delaunay triangulation, and $\lambda = d_{\max}/d_{\min}$ the ratio of the maximum and minimum distances between any two points in P, i.e., $d_{\max} = \max_{\{p,q\} \subseteq P} \mathrm{dist}(p,q)$ and $d_{\min} = \min_{\{p,q\} \subseteq P} \mathrm{dist}(p,q)$. A set $P = \{p_1, \ldots, p_n\}$ of n points in $\mathbb{R}$ ordered such that $p_i < p_j$ for all $i < j$ contains an *exponential chain* of size m if there exist m integers $1 \leq a_1 < a_2 < \cdots < a_m \leq n$ (or $n \geq a_1 > a_2 > \cdots > a_m \geq 1$) such that $\mathrm{dist}(p_{a_i}, p_{a_i+1}) \geq \mathrm{dist}(p_{a_1}, p_{a_i})$ for all $i \in \{1, \ldots, m\}$. That is, the transmission range of p_{a_i} in $\mathrm{MST}(P)$ covers $\{p_{a_{i+1}}, \ldots, p_{a_m}\}$. For example, the set $\{2^i \mid i \in \{0, \ldots, m\}\}$ forms an exponential chain of size m. See Figure 1. Given a set $P \subseteq \mathbb{R}$ of n node positions and an assignment of transmission radii corresponding to the symmetric disk graph G on P, von Rickenbach et al. [16] define a *hub* node as any vertex of G that has at least one neighbour to its right; a non-hub node in P has all of its neighbours to its left. For networks in $\mathbb{R}^2$, a subset $H \subseteq P$ may be identified as a set of hubs, where these hub nodes provide a connected or k-connected backbone to which non-hub nodes connect.

Recall the definition of an ϵ-net [8]. Given a set P of points in $\mathbb{R}^2$ and a family $\mathcal{R}$ of regions (*ranges*) in $\mathbb{R}^2$, the pair $(P, \mathcal{R})$ is a *range space*. For any given $\epsilon \in (0,1)$, an *ϵ-net* of the range space $(P, \mathcal{R})$ is a subset $S \subseteq P$ such that for any region $R \in \mathcal{R}$, if $|R \cap P| \geq \epsilon n$, then $R \cap S \neq \varnothing$. As do Halldórsson and Tokuyama [7], our algorithm uses the set $\mathcal{R}$ of ranges consisting of all equilateral triangles with one edge parallel to the x-axis.

1.3 Overview of Results

We begin with a discussion of related work in Section 2. In Section 3 we establish a lower bound of $\Omega(\sqrt{kn})$ on the worst-case maximum interference among all k-connected networks on a given set of n points in $\mathbb{R}$. This bound applies to point sets in $\mathbb{R}^d$ for any $d \geq 1$ and any $1 \leq k < n$, and improves on the lower bounds of $\Omega(k)$ due to k-connectivity and $\Omega(\sqrt{n})$ for maximum interference in a connected network [16]. In Section 4 we generalize a technique introduced by von Rickenbach et al. [16] and apply it to give an $O(n \log(n/k))$-time algorithm that assigns transmission radii to any set of n nodes in $\mathbb{R}$ to give a k-connected network with maximum interference $O(\sqrt{kn})$ for any $1 \leq k < n$, asymptotically matching our lower bound; interestingly, the dependence on k is $O(\sqrt{k})$, as opposed to being linear in k. In Section 5 we generalize techniques introduced by Halldórsson and Tokuyama [7] and apply them to develop two algorithms that assign transmission radii to any set P of n nodes in $\mathbb{R}^2$ to give k-connected networks with maximum interference $O(k \log \lambda)$ and $O(k\sqrt{n})$, respectively, in $O(n \log \lambda)$ and $O(nk + n \log n + k^3 \sqrt{n} \log n)$ time, respectively. We conclude with a discussion and directions for future research in Section 6.

2 Related Work

Buchin [3] showed that finding an optimal solution to the interference minimization problem is NP-complete in two dimensions. At present, the problem's complexity remains open in one dimension.

Several studies examine the interference minimization problem in one dimension, also known as the *highway model*. von Rickenbach et al. [16] gave an $O(n^2)$-time $O(n^{1/4})$-approximation algorithm and showed a tight asymptotic bound of $\Theta(\sqrt{n})$ on the worst-case minimum maximum interference of any set P of n points in $\mathbb{R}$. Their approximation algorithm applies one of two strategies, $\mathrm{MST}(P)$ or a hub backbone, whichever has lower interference. $\mathrm{MST}(P)$ provides low interference when P is near to being uniformly distributed. If P contains an exponential chain of size m, then $I(\mathrm{MST}(P)) \in \Omega(m)$ [16]. The hub strategy of von Rickenbach et al. [16] selects every $\sqrt{n}$th node as a hub according to their ordering on the line, forms a connected backbone network on the hubs (e.g., their MST), and connects each non-hub node to its nearest hub, giving a network with maximum interference $O(\sqrt{n})$ for any set of n points in $\mathbb{R}$. Tan et al. [18] gave an algorithm that finds an optimal solution for any set P of n points in $\mathbb{R}$ in $O(n^{3+\mathrm{OPT}_1(P)})$ time.

The interference problem has also been examined extensively in two dimensions. Halldórsson and Tokuyama [7] used ϵ-nets to define a backbone of $O(\sqrt{n})$ hub nodes, resulting in a network with maximum interference $O(\sqrt{n})$ for any set of n points in $\mathbb{R}^2$. See Section 2.1 for a detailed description. Halldórsson and Tokuyama [7] present a second algorithm using a quadtree decomposition that guarantees maximum interference $O(\log \lambda)$ for any set of points P in $\mathbb{R}^2$. As

the quadtree is constructed, each non-empty square B_i of width w_i contains some set $P_i \subseteq P$. A representative point $p \in P_i$ is selected arbitrarily and its transmission radius is set to $\max\{\sqrt{2}w_i, \text{dist}(p,q)\}$, where q is the representative of the parent square to B_i. The square B_i is divided into four squares of width $w_i/2$ and $P_i \setminus \{p\}$ is partitioned accordingly. The recursion terminates when $P_i = \varnothing$. Still in $\mathbb{R}^2$, Holec [9] used linear programming to give an algorithm with maximum interference $O(\text{OPT}_1(P)^2 \log n)$. Aslanyan and Rolim [2] also proposed an algorithm that finds a connected network by applying an approximation algorithm for a variant of the minimum membership set cover problem.

In addition to the worst-case results described above, the interference minimization problem has been examined in the randomized setting. Kranakis et al. [13] proved that MST(P) has maximum interference $\Theta((\log n)^{1/2})$ with high probability for any set P of n points selected uniformly at random in $[0,1]$. Khabbazian et al. [11] showed that MST(P) has maximum interference $O(\log n)$ with high probability for any set P of n points selected uniformly at random in $[0,1]^d$; Devroye and Morin [5] improved these results to show that MST(P) has maximum interference $\Theta((\log n)^{1/2})$ with high probability and, furthermore, that $\text{OPT}_1(P) \in O((\log n)^{1/3})$ and $\text{OPT}_1(P) \in \Omega((\log n)^{1/4})$ with high probability, showing that for nearly all point sets P, MST(P) does not minimize interference.

2.1 $O(\sqrt{n})$ Interference in $\mathbb{R}^2$

We include a detailed overview of the algorithm of Halldórsson and Tokuyama [7] using ϵ-nets, which will be important to describe our algorithm presented in Section 5.2. Given a set P of n points in $\mathbb{R}^2$, the algorithm selects an ϵ-net $H \subseteq P$ of size $O(\epsilon^{-1})$ to serve as a set of hubs. Hubs are connected by MST(H), and each non-hub node (the set $P \setminus H$) connects to its nearest hub in H. Each node receives interference from at most $|H| \in O(\epsilon^{-1})$ hubs and $O(n\epsilon)$ non-hub nodes. Consequently, the resulting network has maximum interference $O(\epsilon n + \epsilon^{-1})$, which corresponds to maximum interference $O(\sqrt{n})$ when $\epsilon = n^{-1/2}$.

Halldórsson and Tokuyama [7] describe the following algorithm to find an ϵ-net $H \subseteq P$ of size $O(\epsilon^{-1})$. The algorithm begins by greedily constructing a maximal family of disjoint subsets $\{P_1, \ldots, P_l\}$ such that for each i, $P_i \subseteq P$, $|P_i| = \epsilon n/5$, and there exists a range $R \in \mathcal{R}$ such that $R \cap P = P_i$. Select any range $R_0 \in \mathcal{R}$ such that $P \subseteq R_0$, and let $V(R_0)$ denote the set of three vertices on its boundary. Let $\widetilde{P} = V(R_0) \cup \bigcup_{i=1}^{l} P_i$. Two nodes $\{p,q\} \subseteq \widetilde{P}$ form a *generalized Delaunay pair* with respect to $\mathcal{R}$ if there exists a range $R \in \mathcal{R}$ such that p and q are on the boundary of R and $R \cap \widetilde{P} = \{p,q\}$. Construct DT($\widetilde{P}$) by adding an edge between all generalized Delaunay pairs in $\widetilde{P}$. Consider a set of colours $\{c_1, \ldots, c_{l+3}\}$. For each i, assign each $p \in P_i$ the colour c_i, and colour the points in $V(R_0)$ distinctly using the three remaining colours. A *corridor* refers to a maximal chain of 2-coloured triangles in DT($\widetilde{P}$). Each corridor is greedily partitioned into *subcorridors* such that the union of the Delaunay triangles in each subcorridor contains $\epsilon n/5$ nodes of P. The set of endpoints of subcorridors corresponds to the set H of hubs. Since each corridor contains $O(\epsilon n)$ points of P, the number of subcorridors and, therefore, $|H|$ are $O(\epsilon^{-1})$.

3 Lower Bounds

We show the following lower bound:

Theorem 1 *For every n and every k, $1 \le k \le n$, there exists a set of n points $P \subseteq \mathbb{R}$ such that every k-connected network on P has maximum interference $\Omega(\sqrt{kn})$.*

Proof. Consider the set $P = \{p \mid p = 2^i,\ i \in \{0, \ldots, n-1\}\}$ that forms an exponential chain of size n on the line. Consider any k-connected network on P. Let H denote the set of hub vertices and let S denote the set of non-hub vertices, where $|H| + |S| = n$. Since the network is k-connected, all vertices have between k and Δ neighbours, where Δ denotes the maximum vertex degree. Consequently, the first k vertices on the left of the chain are hubs and, furthermore, these k vertices form a clique. Every hub interferes with the leftmost node in the exponential chain. Therefore, the interference at the first node (and, therefore, the maximum interference) is at least $|H|-1$. Similarly, the maximum interference is at least Δ. That is,

$$I(G) \ge \max\{|H| - 1, \Delta\}. \tag{1}$$

Let $E_{S \to H}$ denote the set of edges that join a non-hub vertex to a hub vertex. Similarly, let $E_{H \to H}$ denote the set of edges joining pairs of hubs. This gives,

$$k|S| \le |E_{S \to H}|. \tag{2}$$

Since the first k hubs form a clique, there are $\binom{k}{2}$ edges among these. So we have,

$$\binom{k}{2} \le |E_{H \to H}|. \tag{3}$$

The number of edge endpoints at a hub is bounded by

$$\begin{aligned}
&|E_{S \to H}| + 2|E_{H \to H}| \le |H|\Delta \\
\Rightarrow\ & k|S| + 2\binom{k}{2} \le |H| \cdot I(G) \text{ (by (1), (2) and (3))} \\
\Rightarrow\ & k(n - |H|) + k(k-1) \le |H| \cdot I(G) \\
\Rightarrow\ & k(n + k - 1) \le |H|(I(G) + k)
\end{aligned}$$

$$\leq (I(G)+1)(I(G)+k) \text{ (by (1))}$$
$$\Rightarrow \quad I(G) \geq \frac{\sqrt{(4n-6)k+5k^2+1}-(k+1)}{2}. \quad (4)$$

Next we show that $I(G) \in \Omega(\sqrt{nk})$ for all $n \geq 5$. The result holds trivially for $n \in O(1)$ and, specifically, for $n < 5$. Assume

$$n \geq 5 \quad (5)$$
$$\Rightarrow \quad 3n+k \geq 14 \text{ (since } k \geq 1)$$
$$\Rightarrow \quad 3nk+k^2 \geq 14k$$
$$\Rightarrow \quad 4nk+k^2 \geq 14k+3 \text{ (by (5) since } k \geq 1)$$
$$\Rightarrow \quad 4nk-6k+5k^2+1 \geq 4k^2+8k+4$$
$$\Rightarrow \sqrt{(4n-6)k+5k^2+1} \geq 2(k+1)$$
$$\Rightarrow \quad -(k+1) \geq -\frac{\sqrt{(4n-6)k+5k^2+1}}{2}$$
$$\Rightarrow \quad I(G) \geq \frac{\sqrt{(4n-6)k+5k^2+1}}{4} \text{ (by (4))}$$
$$\geq \frac{\sqrt{2nk+5k^2}}{4} \text{ (by (5))}$$
$$\in \Omega(\sqrt{nk}). \square$$

As we show in Theorem 2, the lower bound of Theorem 1 is asymptotically tight.

4 k-Connected Networks in One Dimension

In this section, we present an algorithm that constructs a k-connected network on any set P of n points in $\mathbb{R}$. Our algorithm generalizes the hub technique applied in the algorithm of von Rickenbach et al. [16] to construct a connected network with maximum interference $O(\sqrt{n})$, as discussed in Section 2.

Instead of every $\sqrt{n}$th node as in [16], we select every $\sqrt{n/(2k+1)}$th node as a hub, resulting in $\lceil\sqrt{n(2k+1)}\rceil$ hubs. Specifically, select the ith node as a hub if $i = \lfloor j\sqrt{n/(2k+1)}\rfloor$ for some $j \in \mathbb{Z}$ (where nodes are numbered $i = 0, \ldots, n-1$). Set each hub node's transmission radius to its furthest point in P (forming a clique on the hubs). Finally, set each non-hub node's transmission radius to the further of the kth hub to its left and the kth hub to its right.

Theorem 2 *Given any set P of n points in $\mathbb{R}$ and any $k < n$, transmission radii corresponding to a k-connected network on P with maximum interference $O(\sqrt{kn})$ can be found in $O(n\log(n/k))$ time.*

Proof. First we show that the network produced is k-connected.

$$n > k$$
$$\Rightarrow \quad n > \frac{k}{2+1/k}$$
$$\Rightarrow \quad \sqrt{n(2k+1)} > k.$$
$$\Rightarrow \quad \left\lceil\sqrt{n(2k+1)}\right\rceil > k.$$

Therefore, there are at least k hubs. Since the hubs form a clique and each non-hub node is connected to k hubs, the network is k-connected.

Next we bound the maximum interference. Choose any point $p \in P$. The interference at p, denoted $I(p)$, is the sum of the interference it receives from hub and non-hub nodes. Hub nodes define a partition of non-hub nodes into $\lceil\sqrt{n(2k+1)}\rceil$ intervals. Suppose the hub at the left end of each interval belongs to that interval. Let I_i denote the interval that contains p, where intervals are numbered in order from the left. Let h_l and h_r denote the respective hubs at the left and right extremities of I_i. Three types of non-hub nodes interfere with p: nodes in I_i, nodes in I_j for $j < i$ that are connected to h_r, and nodes in I_j for $j > i$ that are connected to h_l. Since each non-hub node connects to its k nearest hubs, p may receive interference from non-hub nodes in k intervals on each side, or $2k$ total intervals, corresponding to at most $\lceil 2k\sqrt{n/(2k+1)}\rceil$ non-hub nodes in other intervals. In addition, p may receive interference from non-hub nodes within its own interval. Finally, p receives interference from at most $\lceil\sqrt{n(2k+1)}\rceil$ hubs. Summing these gives

$$\begin{aligned} I(p) &\leq \left\lceil\sqrt{n(2k+1)}\right\rceil + \left\lceil 2k\sqrt{\frac{n}{2k+1}}\right\rceil + \left\lceil\sqrt{\frac{n}{2k+1}}\right\rceil \\ &< \sqrt{n(2k+1)} + (2k+1)\sqrt{\frac{n}{2k+1}} + 3 \\ &= 2\sqrt{n(2k+1)} + 3 \\ &\in O(\sqrt{kn}). \end{aligned}$$

The hubs can be identified in $O(n\log(n/k))$ time by near-sorting P, e.g., by a partial execution of deterministic quicksort to partition P into blocks of size $\sqrt{n/(2k+1)}$ that returns the partition pivots in sorted order. The list of hubs is traversed in $O(\sqrt{n/k})$ time to assign a transmission radius to each hub, corresponding to the further of the leftmost or rightmost points in P. Non-hub nodes are examined in block sequence, in arbitrary order within a given block. Each non-hub's transmission radius is set to the maximum distance of its kth hub to the left and its kth hub to the right in $O(n)$ total time, achieved by simultaneously traversing the list of hubs and referring to the $(i-k)$th and $(i+k)$th hubs, where i denotes the block index. The total time is dominated by near-sorting, resulting in $O(n\log(n/k))$ time in the worst case. $\square$

This guaranteed $O(\sqrt{kn})$ maximum interference matches the lower bound of $\Omega(\sqrt{kn})$ established in Theorem 1, showing that our algorithm is asymptotically optimal in the worst case. Previously, we knew $I(G) \in \Omega(\sqrt{n})$ in the worst case, implied by $k = 1$ [16], and $I(G) \in \Omega(k)$, since every node in a k-connected graph has at least k neighbours. Furthermore, $I(G) \to n-1$ as $k \to n-1$. The interesting implication of Theorem 2, however, is for values of k between these two extrema: that the worst-case maximum interference's dependence on k is sublinear for all values of k.

5 k-Connected Networks in Two Dimensions

In this section we present two algorithms that generalize techniques applied in algorithms of Halldórsson and Tokuyama [7] described in Section 2. Given a set P of n points in $\mathbb{R}^2$, our algorithms construct respective k-connected networks on P with maximum interference $O(k \log \lambda)$ and $O(k\sqrt{n})$, for any k.

5.1 Quadtree Decomposition

Theorem 3 *Given any set P of n points in $\mathbb{R}^2$ and any $k < n$, transmission radii corresponding to a k-connected network on P with maximum interference $O(k \log \lambda)$ can be found in $O(n \log \lambda)$ time, where $\lambda = d_{\max}/d_{\min}$ is the ratio of the maximum and minimum distances between any two points in P.*

Proof. Let B_0 be an axis-parallel square of minimum width $w_0 \le d_{\max}$ that contains P. Select any set of k points $R_0 \subseteq P$ as representatives for B_0 and set their transmission radii to $\sqrt{2}w_0$. Divide B_0 into four subsquares of width $w_0/2$ and partition $P \setminus R_0$ accordingly. This procedure is applied recursively as follows. Each non-empty square B_i of width w_i contains some set $P_i \subseteq P$. Select a representative set $R_i \subseteq P_i$ arbitrarily, where $|R_i| = \min\{k, |P_i|\}$. Set the transmission radius of each $p \in R_i$ to $\max_{q \in B_j} \mathrm{dist}(p, q)$, where B_j is the parent square to B_i (i.e., q is one of the corners of B_j). The square B_i is divided into four squares of width $w_i/2$ and $P_i \setminus R_i$ is partitioned accordingly. The recursion terminates when $|P_i| \le k$.

The first k representatives form a k-clique. Each remaining node is connected to the k representatives of its parent square. Consequently, any node forms a k-connected graph with its ancestors in the quadtree. Therefore, the entire network is k-connected.

The width of the root square is at most $d_{\max}$. The width of the lowest leaf square in the quadtree is at least $d_{\min}/(2\sqrt{2})$. Therefore, the height of the quadtree is at most $\lceil \log(2\sqrt{2}\lambda) \rceil = \lceil 3/2 + \log \lambda \rceil$. Each representative interferes with at most 32 cells at its level in the quadtree; see Figure 2. Therefore, each node $p \in P$ receives interference from at most $32k$ nodes at each level of the tree, for a total interference of at most $32k\lceil 3/2 + \log \lambda \rceil \in O(k \log \lambda)$.

At each node of the quadtree, k representatives are selected and have their transmission radii assigned, and the set P_i is partitioned into four subsets in $O(|P_i|)$ time. Since the quadtree's height is $O(\log \lambda)$, the total time is $O(n \log \lambda)$. □

5.2 $O(k\sqrt{n})$ Interference

In this section we describe an algorithm that constructs a k-connected network with maximum interference $O(k\sqrt{n})$ for any given set P of n points in $\mathbb{R}^2$. We assume a non-degeneracy condition on points, specifically, that no two points lie on the same line forming an angle of 0, $\pi/3$, or $2\pi/3$ with the x-axis.

This algorithm first selects a set H of $O(k\sqrt{n})$ hubs by finding an $((k\sqrt{n})^{-1})$-net of size $O(k\sqrt{n})$ on P as in the algorithm of Halldórsson and Tokuyama [7] described in Section 2.1. Consequently, any range containing at least $O(\sqrt{n}/k)$ points of P must contain a hub. Next, a k-connected backbone is built on the hubs. Finally, each non-hub node is connected to its k nearest hubs.

It suffices to k-connect the hubs by forming a clique on the hubs. Although the hubs could be k-connected by applying the algorithm recursively, this does not lead to any asymptotic reduction in the maximum interference. Connecting hubs by a tree, such as the MST or the local neighbourhood graph, does not guarantee k-connectivity after non-hubs connect to their k nearest hubs. For small k (e.g., $k \le 3$) the Delaunay triangulation provides a good strategy for k-connecting hubs, but a more general strategy is required for larger k.

We analyze the maximum interference of the resulting network. Consider an arbitrary point $p \in P$. Divide the plane around p into six cones $R_1(P), \ldots, R_6(p)$ such that for each i, $R_i(p)$ is the cone consisting of all rays with apex p and angle in $[(i-1)\pi/3, i\pi/3]$. Without loss of generality, we consider the cone $R_1(p)$; analogous results apply to the remaining cones. Let $h_1, \ldots, h_k$ denote the k hubs nearest to p in $R_1(p)$ ordered by increasing distance to p. Let $l_\alpha(p)$ denote the line through p with angle α.

Lemma 4 *No point in $R_1(p) \cap (P \setminus H)$ lies on the right of $l_{2\pi/3}(h_k)$ and interferes with p.*

Proof. For the sake of contradiction, assume such a point q exists. Consequently, the transmission radius of q is at least $\mathrm{dist}(p, q)$, and so, q is connected to some hub $h \in H$ where $\mathrm{dist}(p, q) < \mathrm{dist}(q, h)$. However, $\mathrm{dist}(q, h_i) < \mathrm{dist}(p, q) < \mathrm{dist}(q, h)$ for all $i \in \{1, \ldots, k\}$, contradicting the fact that q is connected to its k nearest hubs. □

Lemma 5 *There are $O(k\sqrt{n})$ nodes in the area enclosed by $l_0(p)$, $l_{\pi/3}(p)$, and $l_{2\pi/3}(h_k)$.*

Proof. We decompose the range enclosed by $l_0(p)$, $l_{\pi/3}(p)$, and $l_{2\pi/3}(h_k)$ into smaller regions and count the vertices in each region. The first region is the range enclosed by $l_0(p)$, $l_{\pi/3}(p)$, and $l_{2\pi/3}(h_1)$. As this range contains no hub, it contains at most $c\sqrt{n}/k$ nodes of P, for some fixed $c \in \mathbb{R}^+$.

For each $i \in \{1, \ldots, k-1\}$, let Q_i denote the isosceles trapezoidal region enclosed by $l_0(p)$, $l_{\pi/3}(p)$, $l_{2\pi/3}(h_i)$, and $l_{2\pi/3}(h_{i+1})$. We identify ranges in $\mathcal{R}$ that contain no hub whose union covers Q_i. Let H'_1 be a list of the i nearest hubs to p in descending order according to their distance to $l_{\pi/3}(p)$. For each j, let h'_j denote the first hub in the list H'_j. Let A_1 be the range enclosed by $l_0(p)$, $l_{\pi/3}(h'_1)$, and $l_{2\pi/3}(h_k)$. For $j \geq 2$, let $H'_j = H'_{j-1} \setminus \{h'_{j-1}$ and all hubs below $l_0(h'_{j-1})\}$. If $H'_j \neq \varnothing$, let A_j be the range enclosed by $l_0(h'_{j-1})$, $l_{\pi/3}(h'_j)$, and $l_{2\pi/3}(h_i)$. Otherwise, A_{j-1} is the final range necessary to cover Q_i, and we let A_{j-1} be the range enclosed by $l_0(h'_{j-1})$, $l_{\pi/3}(p)$, and $l_{2\pi/3}(h_k)$. This procedure selects at most $i+1$ ranges whose union covers Q_i, each of which contains no hub in its interior. See Figure 3.

Along with the first range, the region $\bigcup_{i=1}^{k-1} Q_i$ is exactly the entire region enclosed by $l_0(p)$, $l_{\pi/3}(p)$, and $l_{2\pi/3}(h_k)$. Since each Q_i can be covered by $i+1$ ranges, each of which contains no hub in its interior, the entire region can be covered by $3k/2+k^2/2$ ranges. Since each empty range contains at most $c\sqrt{n}/k$ nodes of P, the region enclosed by $l_0(p)$, $l_{\pi/3}(p)$, and $l_{2\pi/3}(h_k)$ contains at most $ck\sqrt{n} \in O(k\sqrt{n})$ nodes of P. □

Theorem 6 *Given any set P of n points in $\mathbb{R}^2$ and any $k < n$, transmission radii corresponding to a k-connected network on P with maximum interference $O(k\sqrt{n})$ can be found in $O(nk + n\log n + k^3\sqrt{n}\log n)$ time.*

Proof. We first argue that the resulting network is k-connected. The clique of hubs is k-connected. Each non-hub node is connected to k hubs. Therefore, the entire network is k-connected.

Next we bound the maximum interference. By Lemmas 4 and 5, for any node $p \in P$, $O(k\sqrt{n})$ non-hub nodes interfere with p in each of the six cones around p. There are $O(k\sqrt{n})$ hubs, each of which may interfere with p. Therefore, $I(p) \in O(k\sqrt{n})$.

Finally we analyze the algorithm's running time. Since this algorithm require running part of the algorithm of Halldórsson and Tokuyama [7] described in Section 2.1, we begin by analyzing the time it takes to build the ϵ-net.

Greedily constructing the maximal family of disjoint subsets can be achieved in $O(n\log n)$ time. Similarly, the generalized Delaunay triangulation can be constructed in $O(n\log n)$ time [6] after constructing the Θ-graph (e.g., see [4, 10, 15]). Finding corridors, subcorridors, and their endpoints can done greedily in $O(n)$ time.

In our algorithm we form a clique on the set H of hubs, which can be done in $O(|H|\log|H|)$ time by finding the convex hull of the hubs and setting the transmission radius of each hub to the distance to its furthest hub in $O(\log|H|)$ time per hub using binary search on the boundary of the convex hull, or $O(|H|\log|H|)$ total time. In the final step, we set the transmission radius of each non-hub node to the distance to its kth nearest hub. To do so we can compute a k-nearest neighbour Voronoi diagram of the set H of hubs in $O(k^2|H|\log|H|)$ time [14], upon which a point location data structure (e.g., [12]) is constructed in $O(k|H|(\log k+\log|H|))$ time and applied in $O(k+\log|H|)$ time per non-hub node, or $O(nk+n\log|H|)$ total time. Thus, the running time is dominated by the larger of $O(nk)$, $O(n\log n)$, and $O(k^2|H|\log|H|)$. Since $|H| \in O(k\sqrt{n})$, this gives a total running time of $O(nk + n\log n + k^3\sqrt{n}\log n)$. □

6 Discussion and Directions for Future Research

We showed asymptotically tight upper and lower bounds of $\Theta(\sqrt{kn})$ on the worst-case maximum interference for k-connected networks in one dimension. The lower bound $\Omega(\sqrt{kn})$ applies in two dimensions, where we showed an upper bound of $O(k\sqrt{n})$, leaving open the question of whether a k-connected network with lower maximum interference can be found. In particular, is maximum interference $O(\sqrt{kn})$ always achievable in two dimensions?

von Rickenbach et al. [16] gave a polynomial-time algorithm that builds a connected network with interference at most $O(n^{1/4} \cdot \mathrm{OPT}_1(P))$ for any set P of n points on the line. Their algorithm constructs a network either by applying the hub strategy or returning $\mathrm{MST}(P)$, whichever has lower maximum interference. To bound the approximation factor they rely on a pair of lemmas showing that $\mathrm{OPT}_1(P) \in O(\sqrt{n})$ and $\mathrm{OPT}_1(P) \in \Omega(\sqrt{I(\mathrm{MST}(P))})$. A natural direction for future research is to determine whether this approximation algorithm can be generalized to build a k-connected network in one dimension. Instead of connecting to the nearest neighbours to the left and right as in a one-dimensional MST, we can consider the graph $\mathrm{MST}_k(P)$, in which each point connects to its k nearest neighbours to the left and k nearest neighbours to the right. In Theorem 2 we showed the generalization of the first lemma, i.e., that $\mathrm{OPT}_k(P) \in O(\sqrt{nk})$. It remains open whether the second lemma generalizes. I.e., is $\mathrm{OPT}_k(P) \in \Omega(\sqrt{I(\mathrm{MST}_k(P))})$ for any set $P \subseteq \mathbb{R}$?

Finally, Buchin [3] showed that the problem of finding a connected network that minimizes maximum interference for a given set of n points in two dimensions is NP-complete. The complexity of the interference minimization problem in one dimension remains an important open question.

References

[1] A. K. Abu-Affash, R. Aschner, P. Carmi, and M. J. Katz. The MST of symmetric disk graphs is light. *Comp. Geom.: Theory & Appl.*, 5(1–2):54–61, 2012.

[2] H. Aslanyan and J. D. P. Rolim. Interference minimization in wireless networks. In *Proc. EUC*, pages 444–449, 2010.

[3] K. Buchin. Minimizing the maximum interference is hard. *CoRR*, abs/0802.2134, 2008.

[4] K. Clarkson. Approximation algorithms for shortest path motion planning. In *Proc. ACM STOC*, pages 56–65, 1987.

[5] L. Devroye and P. Morin. A note on interference in random point sets. In *Proc. CCCG*, volume 24, pages 201–206, 2012.

[6] S. Fortune. Voronoi diagrams and Delaunay triangulations. *Comp. in Euclidean Geom.*, 1:193–233, 1992.

[7] M. M. Halldórsson and T. Tokuyama. Minimizing interference of a wireless ad-hoc network in a plane. *Theor. Comp. Sci.*, 402(1):29–42, 2008. Algorithmic Aspects of Wireless Sensor Networks.

[8] D. Haussler and E. Welzl. ϵ-nets and simplex range queries. *Disc. & Comp. Geom.*, 2:127–151, 1987.

[9] E. Holec. Using linear programming to minimize interference in wireless networks. Master's thesis, Univ. Minnesota, 2013.

[10] J. M. Keil. Approximating the complete Euclidean graph. In *Proc. SWAT*, pages 208–213. Springer, 1988.

[11] M. Khabbazian, S. Durocher, A. Haghnegahdar, and F. Kuhn. Bounding interference in wireless ad hoc networks with nodes in random position. *IEEE/ACM Trans. Net.*, 23(4):1078–1091, 2015.

[12] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM J. Comp.*, 12(1):28–35, 1983.

[13] E. Kranakis, D. Krizanc, P. Morin, L. Narayanan, and L. Stacho. A tight bound on the maximum interference of random sensors in the highway model. *CoRR*, abs/1007.2120, 2010.

[14] D.-T. Lee. On k-nearest neighbor Voronoi diagrams in the plane. *IEEE Trans. Comp.*, 31(6):478–487, 1982.

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

[16] P. v. Rickenbach, S. Schmid, R. Wattenhofer, and A. Zollinger. A robust interference model for wireless ad-hoc networks. In *Proc. IPDPS*, pages 239.1–8, 2005.

[17] H. Tan. *Minimizing interference in wireless sensor networks*. PhD thesis, Univ. Hong Kong, 2011.

[18] H. Tan, T. Lou, F. C. M. Lau, Y. Wang, and S. Chen. Minimizing interference for the highway model in wireless ad-hoc and sensor networks. In *Proc. SOFSEM*, pages 520–532, 2011.

A Appendix: Figures

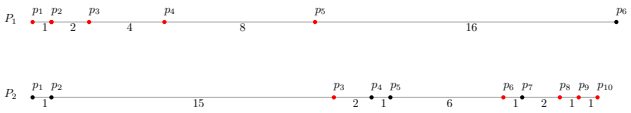


Figure 1: The first five points of P_1 form an exponential chain of size 5, where $a_1 = 5, a_2 = 4, \ldots, a_5 = 1$. An exponential chain need not be a perfect geometric sequence, nor need its points be consecutive. For example, $a_1 = 3$, $a_2 = 6$, $a_3 = 8$, $a_4 = 9$, $a_5 = 10$ (red points) is an exponential chain of size 5 in P_2. The exponential chain property holds for $i = 1$ in P_2 since $\text{dist}(p_{a_1}, p_{a_1-1}) = \text{dist}(p_3, p_2) = 15 \geq \max_{j=2}^{5} \text{dist}(p_{a_1}, p_{a_j}) = \text{dist}(p_3, p_{10}) = 14$; it also holds for all $i \in \{2, 3, 4\}$.

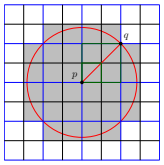


Figure 2: A point p is selected as a representative for a quadtree cell, denoted by the smaller bold green square of width w_i. In the worst case, the furthest representative, q, of the parent square of p, denoted by the larger bold green square, is a distance $2\sqrt{2}w_i$ from p. Consequently, p's transmission range interferes with at most 32 cells of the quadtree at its level.

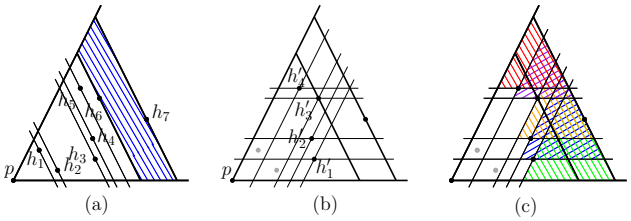


Figure 3: (a) The shaded region is the trapezoid Q_6. (b) Four hubs that determine the ranges used to cover Q_6. (c) The five empty ranges whose union covers Q_6.

Optimal Orientation of Symmetric Directional Antennas on a Line

AmirMahdi Ahmadinejad* Fatemeh Baharifard† Khadijeh Sheikhan‡ Hamid Zarrabi-Zadeh§

Abstract

In this paper, we study the problem of optimal orientation of directional antennas on a line in the symmetric model of communication. We propose an optimal algorithm to find the minimum radius and the orientation of antennas, when antennas are placed on a point set P on a line, and each antenna has angle less than π. We show that the connected graph induced by this optimal orientation is a 7-hop spanner with respect to the unit disk graph of P. Moreover, we present a deterministic local routing algorithm that is guaranteed to find a path between any pair of antennas in the communication graph whose number of edges is at most 7 times the number of edges between that pair in the unit disk graph.

1 Introduction

Wireless networks have received cosiderable attention in recent years due to their vast applications in various areas [11, 12]. Most of the time, wireless networks are modelled as a set P of n wireless nodes, where each node is equipped with an omni-directional antenna whose coverage area is a disk. Assuming identical transmission range for antennas, one can properly scale distances to make this transmission range equal to unit, and hence, the communication graph of antennas becomes equal to the *unit disk graph* of P, in which two antennas are connected if and only if the distance between them is at most unit.

Recent attention in the area of wireless networks has shifted from omni-directional antennas to *directional antennas*, due to their desirable properties such as improving security and reducing overlap [3]. A directional antenna can focus its transmission energy in a specific direction by narrowing coverage area, which is modelled by a sector of a fixed angle α and a radius r (see Figure 1(a) for an example). Antennas at different nodes can be oriented in different directions. There are two main models of communication in networks with directional antennas. In the *asymmetric* model, each antenna has a directed link to any other node that lies in its coverage area. In the *symmetric* model, there exists a link between two antennas u and v, if and only if u lies in the coverage area of v, and v lies in the coverage area of u. The symmetric model of communication is more practical, especially in networks where *handshaking* is required before transmitting data [7]. An example is illustrated in Figure 1(b).

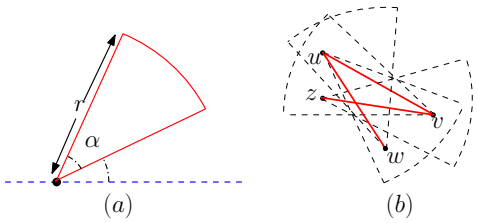


Figure 1: (a) A directional antenna. (b) A symmetric communication graph.

A network is called a *spanner*, if there is a path between any pairs of nodes, within a guaranteed ratio to the shortest paths between those nodes in an underlying base graph. This ratio is also called the *stretch factor* [14]. While the finite stretch factor is sufficient for existence of such a path between nodes through the network, the problem of efficiently finding the shortest path is central to many fields such as robotics and communication networks. In many cases, a node is not aware of the whole structure of the graph, and must learn this information through exploration. Algorithms for routing in these types of environments are called *local routing algorithms*. In local routing, for routing from a source point s to a destination point t, the current point u only knows about its neighbors and the location of t and should decide the next movement only using this information. A routing algorithm is *c-competitive* if the total distance traveled by the algorithm from any point s to any destination t, is not more than c times the length of the shortest path between those nodes in the graph. Parameter c is called the *competitive ratio* of the algorithm [4].

In this paper, we focus on the 1-dimensional version, where directional antennas are located on a set of points along a line. We assume the symmetric model for communication between the antennas. First we study the

*Management Science and Engineering Department, Stanford University, Stanford, CA. ahmadi@stanford.edu.

†School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran. f.baharifard@ipm.ir.

‡Computer Science and Engineering Department, NYU Tandon School of Engineering, Brooklyn, NY. khadijeh@nyu.edu.

§Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. zarrabi@sharif.edu.

optimal orientation that, while it results in connectivity of the network, requires a minimum radius for the antennas. Then we prove that the resulting communication graph is a spanner with a constant stretch factor and also present a competitive local routing algorithm for this communication graph.

Related Work. The connectivity of communication graphs in the symmetric model was first studied by Ben-Moshe *et al.* [2]. They considered a limited setting (i.e., quadrant antennas and half-strip antennas) in which the orientation of antennas were chosen from a fixed set of directions. They showed how to orient antennas so that the communication graph becomes connected. Subsequent studies considered a more general setting, where each antenna can have an arbitrary orientation. Carmi *et al.* [7] proved that for $\alpha \geq \pi/3$, it is always possible to orient antennas so that the induced graph is connected. However, in their construction, the radius of the antennas are related to the diameter of the set of nodes, and hence the communication graphs can have a very large stretch factor, e.g., $O(n)$, compared to the original unit disk graph (i.e., the omni-directional graph of radius 1). Therefore, subsequent work considered the radius and stretch factor of the communication graph and proposed some approximation algorithms to minimize these factors. Aschner *et al.* [1] presented an algorithm to orient the antennas with angle $\pi/2$ and radius $14\sqrt{2}$ to obtain a 8-hop spanner, assuming that the unit disk graph of the nodes is connected. In a *t-hop spanner*, the number of hops (i.e., links) in a shortest link path between any pair of nodes is at most t times the number of hops in the shortest link path between those two nodes in the base graph, which happens to be a unit disk graph in this case. Tran *et al.* [15] improved the radius for the case $\alpha = \pi/2$ to 9. Dobrev *et al.* [10] showed that the connectivity problem is NP-hard for $\alpha < \pi/3$, where the radius is 1, and showed how to construct hop spanners for various values of $\alpha \geq \pi/2$.

Moreover, the problem of assigning transmission ranges to the omni-directional antennas placed arbitrarily on a line in order to achieve a strongly connected communication network with minimum total power consumption, was studied in the literature. Kirousis *et al.* [13] proposed an $O(n^4)$ time algorithm to obtain an optimal solution for this problem. Then, Das *et al.* [9] and Carmi *et al.* [6] improved the running time to $O(n^3)$ and $O(n^2)$, respectively. Also, Clementi *et al.* [8] considered the range assignment and stretch factor for noted problem. They presented a 2-approximation algorithm for the range assignment with running time $O(hn^3)$, where any pair of stations can communicate in at most h hops, to have a spanner with respect to the number of links. Furthermore, Carmi *et al.* [6] proposed a polynomial time algorithm to find the minimum radius whose induced communication graph becomes a t-spanner, for any $t \geq 1$. This problem was also studied for the asymmetric model of communication and Caragiannis *et al.* [5] proved that for a set of n points on a line, $0 \leq \alpha < \pi$ and $r > 0$, there exists an orientation of sectors of angle α and radius r at the points so that the communication graph is strongly connected if and only if the distance between points i and $i + 2$ is at most r, for any $i = 1, 2, \ldots, n - 2$.

Our Results. In this paper, we study the problem of orienting a set of directional antennas on a line, to make the resulting communication graph connected, while the transmission range (radius) is minimized. We present an efficient algorithm that finds an orientation with optimal radius in linear time. This is indeed the first algorithm for the problem that achieves an optimal radius.

We prove that the communication graph obtained from this orientation is a 7-hop spanner, meaning that the shortest link distance between any pair of nodes in the resulting communication graph is at most 7 times the shortest link distance between those nodes in the unit disk graph of the points. In other words, we compare the stretch factor of our connected directional network to that of a connected omni-directional network. We also present an algorithm to route locally in this communication graph with a competitive ratio of 7. To the best of our knowledge, there is no previous result for routing locally and competitively in the communication graph of directional antennas, and hence, we are presenting the first such result in this paper.

2 Preliminaries

Let P be a set of points in the plane, and G be a graph on the vertex set P. For two points $p, q \in P$, we denote by $\delta_G(p, q)$ the *shortest link distance* between p and q in G, i.e. the minimum number of edges needed to connect p and q in G. If the graph G is clear from the context, we simply write $\delta(p, q)$ instead of $\delta_G(p, q)$. A path that realizes $\delta(p, q)$ is called a *shortest path*. For two points p and q in the plane, the Euclidean distance between p and q is denoted by $\|pq\|$. Throughout the paper, the farthest and nearest neighbors are in terms of the Euclidean distance.

For a point set P, we denote by $UDG(P)$ the *unit disk graph* of P, i.e. a graph on the vertex set P in which two vertices are connected if and only if they are within distance unit of each other. Throughout this paper, we assume that $UDG(P)$ is connected, which is necessary for the omni-directional network on P to be connected. We also assume that the largest edge in $UDG(P)$ has *unit* length. This assumption can be easily realized by a proper scaling of the point set.

Given two graphs H and G on the vertex set P, we call H a *t-hop spanner* with respect to G, if for any two vertices u and v in G, we have $\delta_H(u, v) \leq t \cdot \delta_G(u, v)$.

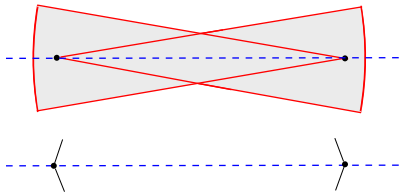


Figure 2: The right and left orientations.

Given a routing algorithm A on G, we say that A is *c-competitive*, if for any pair of vertices s and t, the number edges on the path found by A from s to t in G is at most c times the shortest link distance between s and t in $UDG(P)$.

3 An Algorithm for the Optimal Orientation

In this section, we propose a linear-time algorithm for the optimal orientation in one dimension. More precisely, we direct antennas located on a point set P placed on a line, to obtain a connected communication graph, while minimizing the radius. The challenging part of the problem is when $\alpha < \pi$. (The case $\alpha \geq \pi$ is pretty straight-forward.) In this case, each antenna covers at most a half-plane. Since antennas are located on a line, their orientation can be viewed as either left or right, ignoring the value of α. We denote antennas facing left and right using symbols $\rangle$ and $\langle$, respectively (see Figure 2).

We first present a lemma, describing a useful property of the optimal orientations.

Lemma 1 *There is always an optimal orientation for the antenna set P, in which no three consecutive antennas are in the same direction.*

Proof. Given an optimal orientation, let a *bad triple* be a set of three consecutive antennas with the same orientation. We observe that for any bad triple, the middle antenna is not needed for the connectivity of its left and right neighbors, so we can change its direction without harming the connectivity of the two neighboring antennas. Since the left and right antennas remain connected regardless of the direction of the middle antenna, the middle antenna also remains connected in either direction. Now, consider the optimal solution with the minimum number of bad triples. If this number is not zero, then we can decrease it by the above observation (finding a bad triple and changing the direction of the middle one). Therefore, there is always an optimal solution with no bad triples. □

Using Lemma 1, we can devise a dynamic programming approach to find an optimal orientation in linear time. In an orientation of antennas, let a *block* be a maximal sequence of consecutive antennas starting with one or more antennas facing to the right, and followed with one or more antennas facing to the left. For example, the orientation $\langle\rangle\langle\langle\rangle\langle\langle\rangle\rangle$ consists of three blocks of lengths 2, 3, and 5, respectively. In an optimal orientation, the leftmost (resp., rightmost) antenna is directed to the right (resp., left), and hence, an optimal orientation can be viewed as a series of blocks. By Lemma 1, there is an optimal orientation in which all blocks are either $\langle\rangle$, $\langle\langle\rangle\rangle$, $\langle\langle\rangle$, or $\langle\rangle\rangle$. We try to find such an optimal orientation using dynamic programming.

We observe that the following two conditions are necessary and sufficient for an orientation to have a connected communication graph:

(I) The subgraph of each block is connected.

(II) Each block has edges to its neighboring blocks.

These conditions guarantee that the graph is composed of a set of connected components, each of which connected to its two neighboring components, and hence, the whole graph is connected. By the first condition, the nodes in a block must be able to communicate without getting help from other blocks. Since the minimum radius for block $\langle\langle\rangle\rangle$ is equal to the maximum of the radii for two blocks $\langle\langle\rangle$ and $\langle\rangle\rangle$, we only need to consider these three types: $\langle\rangle$, $\langle\langle\rangle$, and $\langle\rangle\rangle$. This condition holds if and only if the leftmost and rightmost antennas in the block are connected to at least one other node. It means that the leftmost $\rangle$ and $\langle$ must be connected, and analogously the rightmost ones should cover each other. This suggests the lower bound on the radius in this orientation.

By the second condition, two neighboring blocks should be able to directly communicate. Two consecutive blocks $\mathcal{B}_1$ and $\mathcal{B}_2$ ($\mathcal{B}_1$ is to the left of $\mathcal{B}_2$) are connected to each other, if and only if the rightmost $\langle$ in $\mathcal{B}_1$ is connected to the leftmost $\rangle$ in $\mathcal{B}_2$. So the distance between these two nodes is another lower bound on the radius.

By the structure of the blocks, there is always an optimal orientation, which ends with the patterns illustrated in Figure 3 (since the rightmost part of the configuration is considered, the block $\langle\rangle\rangle$ is always as good as the block $\langle\langle\rangle$ as illustrated in case 1). As we can see in Figure 3, the $\langle\rangle$ setting appears in every case. Now let $x_1 < x_2 < \cdots < x_n$ be the position of the antennas P on the real line, we define r_i to be the optimal radius for the subproblem restricted to the first i antennas with an extra restriction that the last block has the $\langle\rangle$ setting (like cases 2 and 3). Thus, we have the following recursive formula for r_i, when $i > 4$:

$$r_i = \min\{\max\{r_{i-2}, x_i - x_{i-3}\}, \max\{r_{i-3}, x_i - x_{i-4}\}\}$$

Actually, in the subproblems like cases 2 and 3, we need radius at least $x_i - x_{i-3}$ and $x_i - x_{i-4}$, respectively for the connectivity condition (II) to hold for the

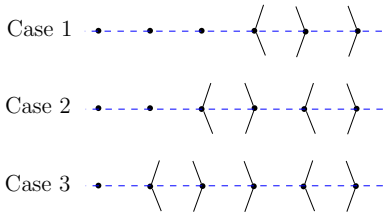


Figure 3: Optimal substructures for orienting antennas.

last two blocks (these radii surely guarantee that condition (I) holds for these blocks). Moreover, by the observation in Figure 3, to have connectivity condition (I) for the last block in case 1, the radius must be at least $x_n - x_{n-2}$. So, the optimal radius is equal to $\min\{r_n, \max\{r_{n-1}, x_n - x_{n-2}\}\}$. The following pseudocode shows the dynamic programming algorithm based on the above recursive formula.

Algorithm 1 OPTIMAL ORIENTATION

input: $x_1, x_2, \cdots, x_n$ the position of antenna set P
output: Optimal radius r

1: **for** $i \leftarrow 1$ to 4 **do**
2: $\quad r_i \leftarrow x_i - x_1$
3: **for** $i \leftarrow 5$ to n **do**
4: $\quad r_i \leftarrow \min\{\max\{r_{i-2}, x_i - x_{i-3}\}, \max\{r_{i-3}, x_i - x_{i-4}\}\}$
5: $r \leftarrow \min\{r_n, \max\{r_{n-1}, x_n - x_{n-2}\}\}$

Algorithm 1 only computes the optimal radius. However, it can be easily modified to output the optimal orientation as well, by storing in a second table the directions minimizing the radii in lines 4 and 5 of the algorithm. All together, we get the following result.

Theorem 2 *Let P be a set of points on a line. There exists a linear-time algorithm that finds an optimal radius r and an optimal orientation of antennas with angle $\alpha < \pi$ and radius r located on P, such that the resulting communication graph $\mathcal{G}(P)$ is connected.*

Remark. Given that a linear-time algorithm exists for the optimal orientation in one dimension, one may be tempted to find a simpler greedy strategy for the problem. For example, for the decision version of the problem which asks for a fixed radius r, if an orientation exists that makes the resulting communication graph connected, the following greedy strategy seems promising: starting from the leftmost antenna p, find the rightmost antenna q which is within distance r of p. We then orient p to the right and q to the left. All other antennas between p and q can be safely oriented to the right. We then repeat this process, with the antenna to the left of q as p. It is not hard to see that this greedy strategy may not work properly (see Figure 4).

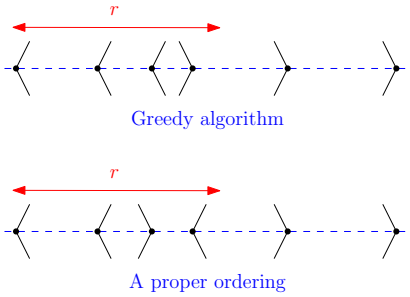


Figure 4: The greedy algorithm fails to build a connected communication graph using radius r.

4 Stretch Factor of the Optimal Orientation

In this section, we prove that the communication graph obtained by Algorithm 1 is a t-hop spanner with respect to the unit disk graph of P.

Theorem 3 *Let P be a point set on a line such that $UDG(P)$ is connected. The communication graph $\mathcal{G}(P)$ obtained by the optimal orientation in Algorithm 1 is a 7-hop spanner of $UDG(P)$.*

Proof. Consider an arbitrary edge $(u, v) \in UDG(P)$. We show that $\delta(u, v)$ in $\mathcal{G}(P)$ is at most 7, while all possible orientations of the antennas located on u and v are considered. Assume w.l.o.g. that u is to the left of v. There are three possible cases.

- Antennas at u and v have right and left directions, respectively ($\langle\rangle$): r is greater than or equal to the unit to guarantee the connectivity of the communication graph. So, there is a direct edge between u and v in $\mathcal{G}(P)$.
- Antennas at u and v have the same directions (either $\langle\langle$ or $\rangle\rangle$): We assume w.l.o.g. that these antennas have $\langle\langle$ setting. Let v' be the nearest neighbor of v in $\mathcal{G}(P)$ (v and v' are in the same block). If u and v' are in the same block, there is a direct edge between them and so $\delta(u, v) = 2$. Otherwise, consider the block $\mathcal{B}$ that is located to the left of the block of v. According to the connectivity condition (II), v' connects to a point in block $\mathcal{B}$, such that this point has a neighbor u' in this block with left orientation. Since u' lies between u and v, the distance between u and u' is less than or equal to unit and thus, by the previous case, u connects to point u'. Therefore, $\delta(u, v)$ is at most 4 in this case.
- Antennas at u and v have left and right directions, respectively (i.e., $\rangle\langle$): Consider the nearest neighbors of u and v, and call them u' and v', respectively. u and u' are in a block $\mathcal{B}_1$, and v and v' are in a block $\mathcal{B}_2$. If $\mathcal{B}_1$ and $\mathcal{B}_2$ are two consecutive blocks, due to the connectivity condition (II), u'

and v' connect to each other with a direct edge, and hence $\delta(u, v) = 3$. Otherwise, u' connects directly to a point u'' in its right block, and v' connects directly to a point v'' in its left block. If u'' and v'' are in a common block, there is an edge between them and $\delta(u, v) = 5$. But if they are in two different blocks, we need three edges to connect them to each other. Since at first u'' and v'' must connect to their nearest neighbors, who have right and left directions respectively, we can then use the first case of the proof to connect these neighbors with one more edge. So, $\delta(u, v)$ is at most 7 in this case.

Now, let p and q be two arbitrary points in P, and $p_0 = p, p_1, \ldots, p_t = q$ be the shortest link distance between p and q in $UDG(P)$. Since $\|p_i p_{i+1}\|$ is less than or equal to the unit, each link (p_i, p_{i+1}) either exist or is replaced by a path of link length at most 7 in $\mathcal{G}(P)$. Therefore, the communication graph $\mathcal{G}(P)$ is a 7-hop spanner. □

5 Local Routing for the Optimal Orientation

In the previous section, we proved that to transfer data between two points that communicate with each other directly in the unit disk graph, there is a path with at most 7 hops in the resulting communication graph of optimal orientation. Although we proved the existence of such path, we need to provide a routing algorithm to find it. Here, we propose a local routing algorithm for communication graph $\mathcal{G}(P)$ of the optimal orientation of the antenna set P.

According to the orientation of antennas (left or right) in the communication graph $\mathcal{G}(P)$, each point connects to some points located either to its left or its right. Therefore, the direction of transfer is predetermined and in each state we just need to choose the best neighbor of the current point for the next step. We assume that the neighbors of each point are sorted in their x-coordinates. We propose Algorithm 2 to route from s to t in graph $\mathcal{G}(P)$. During the algorithm, if the orientation of the antenna on the current point u is in the direction of the destination, we go to the farthest neighbor of u in order to close the gap to t as much as possible, and if the orientation of the antenna located on u is in opposite of the direction of the destination, we go to the nearest neighbor in order to increase the distance to t the least.

To prove the correctness of the algorithm, we assume w.l.o.g that s is to the left of t, and then show that we will certainly reach from s to t after visiting a finite number of points. We denote by $\pi(s, t)$ the path obtained by Algorithm 2. Moreover, we define the *head* of a block to be the rightmost antenna with right direction in that block.

Lemma 4 *In $\pi(s, t)$, each antenna with right direction, except s and t, is the head of a block, and these heads appear in the ascending order of their x-coordinates along $\pi(s, t)$.*

Algorithm 2 ROUTING($\mathcal{G}(P), s, t$)

input: Communication graph $\mathcal{G}(P)$, point s and t
output: Routing from s to t

```
1: while s is not directly connected to t do
2:     if the antenna on s is oriented toward t then
3:         u ← farthest neighbor of s
4:         ROUTING(G(P), u, t)
5:     else
6:         u ← nearest neighbor of s
7:         ROUTING(G(P), u, t)
```

Proof. Every antenna with left direction in $\pi(s, t)$, except t, can not see t. Therefore, we go to its nearest neighbor, which has right direction and is therefore the head of a block. In Algorithm 2, if the current point u is a head, we go toward t or to the farthest neighbor of it, say u'. Since the direction of a head is right, by the connectivity condition (II), u' is located in a block which lies to the right of u. Now, either u' directly connects to t, or we go to the head of its block, whose x-coordinate is greater than u. □

By Lemma 4, the points in $\pi(s, t)$ are alternating heads of blocks in ascending x-coordinates. Since the number of blocks is finite, the proposed routing algorithm reaches from s to t after a finite number of steps by the invariant property. (If it passes over t, after one backward movement it certainly gets to t.)

5.1 Competitive Ratio of the Routing Algorithm

Here, we compare the path $\pi(s, t)$, obtained by Algorithm 2 on $\mathcal{G}(P)$, with a shortest path between s and t in $UDG(P)$ and show that Algorithm 2 can route locally and competitively on graph $\mathcal{G}(P)$. So, we first prove a lemma.

Lemma 5 *If h_1, h_2, h_3, and h_4 are four consecutive heads in $\pi(s, t)$, then $\|h_1 h_4\| \geq r$.*

Proof. If $\|h_1 h_4\| < r$, there is an antenna p in the block to which h_3 belongs, such that the direction of p is left and its Euclidean distance to h_1 is less than r. Thus, there is a direct edge between h_1 and p. Since h_1, h_2 and h_3 are consecutive heads in $\pi(s, t)$, in the routing algorithm we go along the path from h_1 to an antenna q with left direction, which is located between h_2 and h_3, and then go from q to h_2 with a movement. We know that $\|h_1 q\| < \|h_1 p\|$, and that both p and q are neighbors of h_1. (The status of antennas can be illustrated as $\langle^{h_1} \ldots \langle^{h_2} \rangle^{q} \ldots \langle^{h_3} \rangle^{p} \ldots \langle^{h_4}$.) Therefore, in the routing algorithm, we go after h_1 to its farthest neighbor which

is not q. But, this contradicts the assumption that h_1 and h_2 are consecutive heads along the path, and this completes the proof. □

Corollary 1 *Since the antennas in $\pi(s,t)$ have alternating left and right directions, we use at most six edges to move from h_1 to h_4, and after these steps, h_4 becomes at least r times closer to t than h_1, i.e., $\|h_1t\| - \|h_4t\| \geq r$.*

If the distance between two arbitrary points s and t is in the range $[(k-1)r, kr]$ for a positive integer k, by Corollary 1, after $6(k-1)$ steps, the Euclidean distance between the current point u and t becomes less than or equal to r. On the other hand, we proved in Section 4 that for any two points u and v in $\mathcal{G}(P)$ with distance less than or equal to unit, $\delta(u,v) \leq 7$. We can easily generalize this result to the case when the distance between two points is at most r. Therefore, for the current point u and the destination point t, there is a path with at most 7 edges connecting them, which is exactly the path found by Algorithm 2. Therefore, for reaching from s to t, we pass at most $6(k-1)+7 = 6k+1$ edges, and hence, $|\pi(s,t)| \leq 6k+1$.

In $UDG(P)$, by passing each edge in a shortest path from s to t, we get closer to t by at most one unit. So, if the distance between two arbitrary points s and t is in the range $[(k-1)r, kr]$, we have $\delta_{UDG}(s,t) \geq kr$, and because r is greater than or equal to unit, we have $\frac{|\pi(s,t)|}{\delta_{UDG}(s,t)} \leq (6 + \frac{1}{k})$. The following theorem summarizes the result.

Theorem 6 *Let P be a set of points on a line such that $UDG(P)$ is connected. Algorithm 2 is a 7-competitive routing algorithm with respect to the $UDG(P)$, for the communication graph $\mathcal{G}(P)$ computed by Algorithm 1.*

6 Conclusion

In this paper, we studied the problem of orienting directional antennas in the symmetric model of communication, and presented an efficient linear-time dynamic programming algorithm for finding an optimal orientation with a minimum radius in one dimension. Moreover, we showed that the induced communication graph of the optimal orientation is a t-hop spanner, for a small stretch factor $t \leq 7$. We also presented a 7-competitive local routing algorithm on the resulting graph.

Several interesting problems remain open. The main question is how to extend the results of this paper to two and higher dimensions. In particular, there is a 2-approximation algorithm for the problem (in a limited setting) in two dimensions. However, it is not yet known whether the problem in the plane is NP-hard, or can be solved optimally in polynomial time. Moreover, finding routing algorithms for networks with directional antennas in two and higher dimensions remains open.

References

[1] R. Aschner, M. Katz, and G. Morgenstern. Symmetric connectivity with directional antennas. *Comput. Geom. Theory Appl.*, 46(9):1017–1026, 2013.

[2] B. Ben-Moshe, P. Carmi, L. Chaitman, M. Katz, G. Morgenstern, and Y. Stein. Direction assignment in wireless networks. In *Proc. 22nd Canad. Conf. Computat. Geom.*, pages 39–42, 2010.

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

[4] P. Bose and P. Morin. Online routing in triangulations. *SIAM J. Comput.*, 33(4):937–951, 2004.

[5] I. Caragiannis, C. Kaklamanis, E. Kranakis, D. Krizanc, and A. Wiese. Communication in wireless networks with directional antennas. In *Proc. 20th ACM Sympos. Parallel Algorithms Architect.*, pages 344–351, 2008.

[6] P. Carmi and L. Chaitman-Yerushalmi. On the minimum cost range assignment problem. In *Proc. 26th Annu. Internat. Sympos. Algorithms Comput.*, pages 95–105, 2015.

[7] P. Carmi, M. Katz, Z. Lotker, and A. Rosen. Connectivity guarantees for wireless networks with directional antennas. *Comput. Geom. Theory Appl.*, 44(9):477–485, 2011.

[8] A. Clementi, A. Ferreira, P. Penna, S. Perennes, and R. Silvestri. The minimum range assignment problem on linear radio networks. In *Proc. 8th Annu. European Sympos. Algorithms*, pages 143–154, 2000.

[9] G. K. Das, S. C. Ghosh, and S. C. Nandy. Improved algorithm for minimum cost range assignment problem for linear radio networks. *Int. J. Found. Comput. Sci.*, 18(3):619–635, 2007.

[10] S. Dobrev, M. Eftekhari, F. MacQuarrie, J. Manuch, O. M. Ponce, L. Narayanan, J. Opatrny, and L. Stacho. Connectivity with directional antennas in the symmetric communication model. *Comput. Geom. Theory Appl.*, 55:1–25, 2016.

[11] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proc. 5th Annu. Internat. Conf. Mobile Comput. Networking*, pages 263–270, 1999.

[12] J.-P. Hubaux, T. Gross, J.-Y. L. Boudec, and M. Vetterli. Towards self-organized mobile ad hoc networks: The terminodes project. *IEEE Commun. Mag.*, 39(1):118–124, 2001.

[13] L. Kirousis, E. Kranakis, D. Krizanc, and A. Pelc. Power consumption in packet radio networks. *Theoret. Comput. Sci.*, 243(1-2):289–305, 2000.

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

[15] T. Tran, M. K. An, and D. Huynh. Antenna orientation and range assignment in WSNs with directional antennas. In *Proc. 35th Annu. Joint Conf. IEEE Comput. Commun. Societies*, pages 1–9, 2016.

Range Assignment of Base-Stations Maximizing Coverage Area without Interference

Ankush Acharyya*† Minati De‡§ Subhas C. Nandy*¶

Abstract

This note is a study on the problem of maximizing the sum of area of non-overlapping disks centered at a set of given points in $\mathbb{R}^2$. If the points of P are placed on a straight-line, then the problem is solvable in polynomial time. Eppstein [CCCG, pages 260–265, 2016] proposed an $O(n^{\frac{3}{2}})$ time algorithm, for maximizing the sum of radii of non-overlapping balls or disks when the points are arbitrarily placed on a plane. We show that the solution to this problem gives a 2-approximation solution for the area maximization problem by non-overlapping disks or balls. We also present simulation results. Finally, we propose a PTAS for our problem.

Keywords: Quadratic programming, discrete packing, range assignment in wireless communication, approximation algorithm, PTAS.

1 Introduction

Geometric packing problem is an important area of research in computational geometry, and has wide applications in cartography, sensor network, wireless communication, to name a few. In the disk packing problem, the objective is to place maximum number of congruent disks (of a given radius) in a given region. Toth 1940 [3, 12] first gave a complete proof that hexagonal lattice packing produces the densest of all possible disk packings of both regular and irregular region. Several variations of this problem are possible depending on various applications [2, 12]. In this note, we will consider the following variation of the packing problem:

Maximum area discrete packing (MADP): Given a set of points $P = \{p_1, p_2, \ldots, p_n\}$ in $\mathbb{R}^2$, compute the radii of a set of non-overlapping disks $\mathcal{C} = \{C_1, C_2, \ldots, C_n\}$, where C_i is centered at $p_i \in P$, such that $\sum_{i=1}^{n} \text{area}(C_i)$ is maximum.

The problem can be formulated as a quadratic programming problem as follows. Let r_i be the radius of the disk C_i. Our objective is:

Maximize $\sum_{i=1}^{n} r_i^2$
Subject to $r_i + r_j \leq \text{dist}(p_i, p_j)$, $\forall\ p_i, p_j \in P$, $i \neq j$.

Here, $\text{dist}(p_i, p_j)$ denotes the Euclidean distance of p_i and p_j.

The motivation of the problem stems from the range assignment problem in wireless networks. Here the inputs are the base-stations. Each base-station is assigned with a range, and it covers a circular area centered at that base-station with radius equal to its assigned range. The objective is to maximize the area coverage by these base-stations without any interference. In other words, the area covered by two different base-stations should not overlap. Surprisingly, to the best of our knowledge, there is no literature for the MADP problem. A related problem, namely *maximum perimeter discrete packing* (**MPDP**) problem, is studied recently by Eppstein [4], where the objective is to compute the radii of the disks in $\mathcal{C}$ maximizing $\sum_{i=1}^{n} r_i$ subject to the same set of linear constraints. This is a linear programming problem for which polynomial time algorithm exists [10]. In particular, here each constraint consists of only two variables, and such a linear programming problem can be solved in $O(mn^3 \log m)$ time [9], where n and m are number of variables and number of constraints respectively. In [4], a graph-theoretic formulation of the MPDP problem is suggested. Let $G = (V, E)$ be a complete graph whose vertices V correspond to the points in P; the weight of edge $(i, j) \in E$ $(i \neq j)$ is $\text{dist}(p_i, p_j)$, which corresponds to the constraint $r_i + r_j \leq \text{dist}(p_i, p_j)$. They computed the minimum weight cycle cover of G in time $O(mn + n^2 \log n)$ time. Since $m = O(n^2)$ in our case, the time complexity of this algorithm is $O(n^3)$. They further considered the fact that a constraint $r_i + r_j \leq \text{dist}(p_i, p_j)$ is useful if $\delta(p_i) + \delta(p_j) \geq \text{dist}(p_i, p_j)$, where $\delta(p)$ is the distance of the point p and its nearest neighbor in P; otherwise that constraint is redundant. They also showed that the number of useful constraints is $O(n)$, and thus the overall time complexity becomes $O(n^2 \log n)$. They used further graph structure to reduce the time complexity. In $\mathbb{R}^d$, the time complexity of this problem is shown to be $O(n^{2-\frac{1}{d}})$.

It is well-known that if Q is a positive definite matrix, then the quadratic programming problem which minimizes $\tilde{X}'Q\tilde{X}$ subject to a set of linear constraints

*ACM Unit, Indian Statistical Institute.
†ankush_r@isical.ac.in
‡Department of CSA, Indian Institute of Science.
§minati@csa.iisc.ernet.in
¶nandysc@isical.ac.in

$A\tilde{X} \leq \tilde{b}$, $\tilde{X} \geq 0$ is solvable in polynomial time [8]. However, if we present our maximization problem as a minimization problem, the diagonal entries of the matrix Q are all -1 and the off-diagonal entries are all zero. Thus, all the eigen values of the matrix Q are -1. It is already proved that the quadratic programming problem is NP-hard when at least one of the eigen values of the matrix Q is negative [11]. Recently, MADP problem is also shown to be NP-hard [1]. For the minimization version of an NP-hard quadratic programming with n variables and m constraints, an $(1 - \frac{1-\epsilon}{(m(1+\epsilon))^2})$-approximation algorithm is proposed in [6], which works for all $\epsilon \in (0, 1 - \frac{1}{\sqrt{2}})$. The time complexity of this algorithm is $(n^3(m \log \frac{1}{\delta} + \log\log \frac{1}{\epsilon}))$, where δ is the radius of the largest ball inside the feasible region defined by the given set of constraints.

For our MADP problem, a *4-approximation algorithm* is easy to get.

> For each point $p_i \in P$, let $\mathcal{N}(p_i) \in P$ be its nearest neighbor. We assign $r_i = \frac{1}{2}\text{dist}(p_i, \mathcal{N}(p_i))$ for each $i = 1, 2, \ldots, n$. Thus, all the constraints are satisfied. The approximation factor follows from the fact that r_i can take maximum value of $\text{dist}(p_i, \mathcal{N}(p_i))$.

In this note, we first show that if the points in P are placed on a straight line, then the MADP problem can be optimally solved in $O(n^2)$ time. As a feasible solution of the MPDP problem is also a feasible solution of the MADP problem, it is very natural to ask whether an optimal solution of the MPDP problem is a good solution for the MADP problem. We answer this question in the affirmative. We show that the optimum solution for the MPDP problem proposed in [4] is a 2-approximation result for the MADP problem. Finally, we propose a PTAS for the MADP problem.

2 Preliminaries

In a solution of the MADP problem, each disk is centered at some point in P. A solution of the MADP problem is said to be *maximal* if each disk touches some other disk in the solution[1]. From now onwards, by a *solution* of a MADP problem, we will mean it to be a *maximal solution*.

The nearest neighbor of a point $p_i \in P$ is denoted by $\mathcal{N}(p_i) \in P$. Here, a point $p_i \in P$ is said to be a *defining point* of the said solution if it appears on the boundary of some disk in the solution; otherwise it is said to be a *non-defining point*. A *non-defining point* $p_i \in P$ will be covered with a disk C_i centered at point p_i, and its radius r_i is either *equal to* or *less than* $\text{dist}(p_i, q_i)$, where $q_i = \mathcal{N}(p_i)$ is a defining point. In the former case, C_i is said to have *full-radius*, and in the later case, C_i is said to have *part-radius* since the boundary of C_i does not have any point in P. Let us consider a neighbor p_j of the point p_i which has a disk $\mathcal{C}_j$ of radius r_j. We will use the term *residue-distance* to indicate a feasible radius for the disk C_i of length $|\text{dist}(p_i, p_j) - r_j|$, $i \neq j$, if $|\text{dist}(p_i, p_j) - r_j| \leq |\text{dist}(p_i, \mathcal{N}(p_i))|$ (see Figure 1). Thus, the residue-distance of a disk C_i (centered at p_i) is zero if $\mathcal{N}(p_i)$ is a *defining point*. For each full-radius (resp. part-radius) of a disk C_i corresponding to p_i, we define a *full-radius interval (resp. part-radius interval)* of length 2·full-radius (resp. 2·part-radius) whose center lies on p_i.

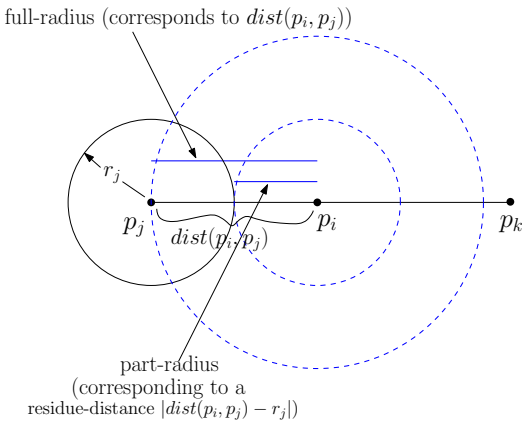


Figure 1: full-radius, part-radius and residue-distance of C_i with respect to p_j

3 MADP problem on a line

In this section we are going to consider a constrained version of the MADP problem, where the point set $P = \{p_1, p_2, \ldots, p_n\}$ lies on a given line L, which is assumed to be the *x-axis*. We also assume $\{p_1, p_2, \ldots, p_n\}$ is sorted in left to right order. Our objective is to place non-overlapping disks centered at each point $p_i \in P$ such that the sum of the area formed by those disks is maximized.

Lemma 1 *In the optimum solution of the MADP problem on a line, at least one of the leftmost or rightmost point in P must be either a* defining point *or its corresponding disk has* full radius.

Proof. Let us denote $d(p_i, p_{i+1}) = d_i$ for all $i = 1, 2, \ldots, n-1$. For the contradiction, let the leftmost point p_1 in P has radius r_1 satisfying $0 < r_1 < \text{dist}(p_1, \mathcal{N}(p_1))$ (see Figure 2). If $r_2 = d_2 < d_1 - r_1$,

[1] If a zero-radius disk does not touch any other disk in the solution, it or its neighboring disk can be enlarged to increase the total area in the solution.

then we can increase r_1, indicating the non-optimality of the solution. If $r_2 = d_1 - r_1$, then $r_3 = \min(d_3, (d_2 - (d_1 - r_1)))$. Assuming $r_3 = d_2 - (d_1 - r_1)$ and proceeding similarly, we may reach one of the following two situations:

1. $r_k = d_{k-1} - (d_{k-2} - (\ldots(d_1 - r_1)))\ldots)$, and the values of $r_{k+1}, \ldots, r_n$ are independent of r_1.
2. $r_{n-1} = d_{n-2} - (d_{n-3} - (\ldots(d_1 - r_1)))\ldots)$ and $r_n = d_{n-1} - r_{n-1}$.

In Case 1, we show that $S_k = r_1^2 + r_2^2 + \ldots + r_k^2$ can be increased while keeping the values of $r_{k+1}, \ldots, r_n$ unchanged.

$$\begin{aligned} S_k &= \pi \cdot (r_1^2 + (d_1 - r_1)^2 + (d_2 - (d_1 - r_1))^2 + \ldots \\ &\quad +(d_k - (d_{k-1} - (\ldots(d_1 - r_1))))^2) \\ &= \pi \cdot (k \cdot r_1^2 - 2r_1 \cdot c_2 + c_1), \end{aligned}$$

where $c_1 = d_1^2 + (d_2 - d_1)^2 + \ldots$
$+(d_k - (d_{k-1} - (\ldots + (-1)^k \cdot d_1)))^2$, and
$c_2 = (d_1 - (d_2 - d_1) + \ldots$
$+(-1)^{k-1}(d_k - (d_{k-1} - (\ldots + (-1)^k \cdot d_1)))))$.

Thus, S_k is a parabolic function whose minimum is attained at $r_1 = \frac{c_2}{k}$, and it attains maximum at the boundary values of the feasible region of r_1, i.e either at $r_1 = 0$ or d_1.

In Case 2, if $r_n > r_{n-1}$, we can increase the sum S_n by setting $r_n = d_{n-1}$, $r_{n-1} = 0$ and keeping $r_1, r_2, \ldots, r_{n-2}$ unchanged. Now, $r_1^2 + r_2^2 + \ldots + r_{n-2}^2$ can further be increased as in Case 1. Similarly, if $r_1 > r_2$ then also S_n can be increased by setting $r_1 = d_1$ and $r_2 = 0$, and then maximizing $r_3^2 + r_4^2 + \ldots + r_n^2$ as in Case 1. If $r_n \leq r_{n-1}$ and $r_1 \leq r_2$, then also S_n is a parabolic function of r_1, and it is maximized at either $r_1 = 0$ or $r_1 = \min(d_1, \alpha)$ where α = value of r_1 for which $r_{n-1} = d_{n-1}$ [2]. □

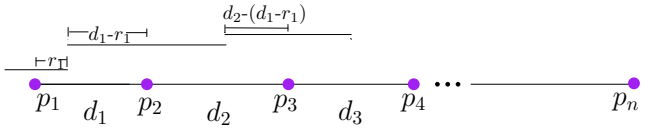


Figure 2: An instance in which $k = 3$

In an optimum solution all the disks have either full-radius or zero radius or has radius equal to the residue distance with respect to the radius of its neighboring points.

Full-radius disks (intervals) are easy to get. For each point p_i, find its nearest neighbor $\mathcal{N}(p_i) = p_{i-1}$ or p_{i+1}, and define an interval of length $2 \cdot \text{dist}(p_i, \mathcal{N}(p_i))$, centered at p_i. We now describe the generation of all possible part-radius intervals for each point $p_i \in P$ considering them in left to right order.

- For both the points p_1 and p_2, there is no part-radius interval.
- If $\mathcal{N}(p_2) = p_1$, then for point p_3, there is a part-radius interval of length $2(d_2 - d_1)$, centered at p_3; otherwise there is no part-radius interval for the point p_3.
- In general, for an arbitrary point p_k if there are m number of part-radius intervals $I_1, I_2, \ldots, I_m$ of lengths $2\delta_1, 2\delta_2, \ldots, 2\delta_m$ respectively, then each of these intervals I_j gives birth to a part-radius interval for the point p_{k+1} with center at p_{k+1} of length $2 \cdot (d_k - \delta_j)$.
  In addition, if $\mathcal{N}(p_k) = p_{k-1}$, then for point p_{k+1}, there is another part-radius interval centered at p_{k+1} and of length $2(d_k - d_{k-1})$.

Finally, we have $\mathcal{I} = \cup_{i=1}^{n} \mathcal{I}_i$. A similar process is performed to generate part-radius intervals $\mathcal{J}$ by considering the points in P in right to left order.

Lemma 2 *For a set P of n points lying on a line L, the maximum number of intervals generated by the above procedure is $\Theta(n^2)$.*

Proof. Let us first consider the forward pass as explained above. Here, for each point p_i (in order) a full-radius interval is generated, and the full-radius interval for point p_i may generate a part-radius interval for each point $p_j, j = i+1, \ldots, n$. Thus, for all the points in P, we may get $O(n^2)$ intervals. To justify the number of intervals is $\Omega(n^2)$, see the demonstration in Figure 3. Here the points $p_i = (x_i, 0)$, $i = 1, 2, \ldots, n$ are placed on the x-axis, where $x_1 = 0, x_2 = 1$ and $x_i = (x_{i-1} - x_{i-2}) + 0.5$, $i = 3, 4, \ldots, n$. Here for each generated interval at p_i, a part-radius interval for the points $p_j, j = i+1, \ldots, n$ will be generated. The same argument follows for the reverse pass also. □

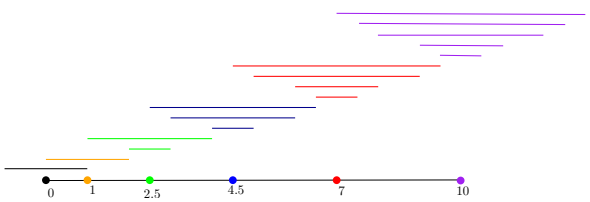


Figure 3: An $\Omega(n^2)$ instance of full and part radius intervals

For each of these intervals we assign weight equal to the square of their half-length. We sort the right end

[2] Here right-end of the feasible region of r_1 is obtained by placing a disk of radius d_{n-1} at p_n, and placing disks at points $p_{n-1}, \ldots, p_2$ touching those of $p_n, \ldots, p_3$, and then placing the disk of radius α at p_1 that touches the disk at p_2. Here surely $\alpha \leq d_1$.

points of these intervals. For this sorted set of weighted intervals, we find the maximum weight independent set. This leads us to the following theorem.

Theorem 3 *Given a set P of n points on a line L, one can place non-overlapping disks maximizing sum of their area in $O(n^2)$ time.*

Proof. We can generate the intervals in $O(n^2)$ time as follows. Given a set of intervals $\mathcal{I}_i$ (of full- and part-radius) generated for a point p_i which are sorted by their right end-points, we can generate the set of part-radius intervals $\mathcal{I}_{i+1}$ for the point p_{i+1} in $O(i)$ time. Thus, total time for interval generation is $O(n^2)$ in the worst case. Since intervals for each point p_i are generated in sorted manner, ordering them with respect to their end-points also takes $O(n^2)$ time. Finally, computing the maximum weight independent set of the sorted set of intervals $\cup_{i+1}^{n}\mathcal{I}_i$ using dynamic programming needs $O(n^2)$ time [7].

The correctness of the algorithm follows from the fact that, if there is a interval θ corresponding to point p_i in the optimum solution that does not belong to $\mathcal{I} \cup \mathcal{J}$, then it is not generated by any interval in $\mathcal{I}_{i-1}$ and $\mathcal{J}_{i+1}$. As a result it does not touch any interval of $\mathcal{I}_{i+1}$ and also $\mathcal{J}_{i-1}$. Thus, interval θ can be elongated to increase the total covering area. □

4 Approximation algorithm

In this section, we first show that the optimum solution for the MPDP problem proposed in [4] gives a 2-approximation result for the MADP problem. We also propose a PTAS for the problem.

4.1 2-factor approximation algorithm

Given a set of points P in the plane, let $R = \{r_i, i = 1, 2, \ldots, n\}$ be the set of radii of the points in P obtained by the optimum solution for MPDP problem [4]. It is clear that any feasible solution of the MPDP is a feasible solution of the MADP problem. We show that an optimal solution of the MPDP problem is at most $2 \times OPT$, where OPT is the optimum solution of the corresponding MADP problem.

Lemma 4 *[4] The maximum sum of radii of non-overlapping disks, centered at points $p_i \in P$, equals half of the minimum total edge length of a collection of vertex-disjoint cycles (allowing 2-cycles) spanning the complete geometric graph on the points $p_i \in P$ with each edge having length equal to the distance between the end-points of that edge.*

Lemma 5 *[4] In the minimum total edge length of a collection of vertex-disjoint cycles, each cycle is either of odd length or a 2-cycle (i.e., a single edge).*

The implication of Lemma 4 and 5 is that *in the optimum solution of the MPDP problem, each disk touches its neighboring disk(s) in the cycle in which it appears.*

In [4], an $O(n^{1.5})$ time algorithm is proposed to compute the minimum length cycle cover $\mathcal{C}$ of the complete geometric graph G with a set P of n points on the plane. From the geometric property of the Euclidean distances, they show that if a subgraph G' of G is formed by removing all the edges (p_i, p_j) satisfying $\text{dist}(p_i, \mathcal{N}(p_i)) + \text{dist}(p_j, \mathcal{N}(p_j)) < \text{dist}(p_i, p_j)$, then the minimum weight cycle cover of G' remains same as that in G. We now prove the main result in this section.

Lemma 6 *For a given set of points P arbitrarily placed in the plane, the radii $\{r_i, i = 1, 2, \ldots, n\}$ in the optimum solution of the MPDP problem is a 2-approximation result for the MADP problem for the point set P.*

Proof. As mentioned, MPDP algorithm generates the cycles $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$. We need to show that $\sum_{\alpha=1}^{n} r_\alpha^2 \geq \frac{1}{2}\sum_{\alpha=1}^{n} \rho_\alpha^2$, where ρ_α is the radius in the optimum solution of the MADP problem for the point p_α. We show that $\sum_{p_\alpha \in C_i}^{n} r_\alpha^2 \geq \frac{1}{2}\sum_{p_\alpha \in C_i}^{n} \rho_\alpha^2$ for each cycle $C_i \in \mathcal{C}$. As each disk participates in exactly one of the cycles, aggregating these relations for all the cycles $C_i, i = 1, 2, \ldots, k$, we will have the desired result. Let us consider the following two cases separately.

C_i **is a 2-cycle** (p_α, p_β)**:** Let $r = \text{dist}(p_\alpha, p_\beta)$. As the disks centered at p_α and p_β in R are touching to each other, let $r_\alpha = \frac{r}{2} - \delta$ and $r_\beta = \frac{r}{2} + \delta$. Thus, $r_\alpha^2 + r_\beta^2 \geq \frac{r^2}{2}$.

Note that in the optimum solution of the MADP problem, the disks for p_α, p_β may not be touching, but $\rho_\alpha + \rho_\beta \leq \text{dist}(p_\alpha, p_\beta)$. So, the upper bound of the sum of squares of the radii in the optimum solution is: $\rho_\alpha^2 + \rho_\beta^2 \leq (\rho_\alpha + \rho_\beta)^2 \leq (\text{dist}(p_\alpha, p_\beta))^2 = r^2$.

Thus, for the two-cycle $\mathcal{C}_i = (p_\alpha, p_\beta)$, we have $r_\alpha^2 + r_\beta^2 \geq \frac{1}{2}(\rho_\alpha^2 + \rho_\beta^2)$.

C_i **is an odd cycle:** Let the length of the cycle be m. Without loss of generality, assume that the vertices be $p_1, p_2, \ldots, p_m$. For each edge $(p_\alpha, p_{\alpha+1})$ of this cycle (where the indices are numbered modulo m), we have $r_\alpha^2 + r_{\alpha+1}^2 \geq \frac{1}{2}(\rho_\alpha^2 + \rho_{\alpha+1}^2)$ (as explained in the earlier case). Adding these inequalities for $\alpha = 1, 2, \ldots, m$, we have $2\sum_{\alpha=1}^{m} r_\alpha^2 \geq \frac{1}{2}[2\sum_{\alpha=1}^{m} \rho_\alpha^2]$. Ignoring 2 in both sides, we have the result. □

Combining Lemma 6 with the time complexity result in [4], we have the following result.

Theorem 7 *For a given set of points P arbitrarily placed in the plane, one can compute a 2-approximation result of the MADP problem in $O(n^{\frac{3}{2}})$ time.*

4.2 Experimental results

We performed a thorough experimental study on this problem by considering random instances. We considered point sets of different size n, and generated 50 samples[3], where each sample consists of n points. For each sample, we formulated the quadratic programming problem, and run LINDO software to generate the optimum solution MADP_{opt}. We also run the MPDP algorithm [4]. Let $\text{MADP}_{sol} = \sum_{i=1}^{n} r_i^2$, where $\{r_1, r_2, \ldots, r_n\}$ is the optimum solution of the MPDP problem. For each sample, we computed the ratio $\frac{\text{MADP}_{opt}}{\text{MADP}_{sol}}$, and compute the *average* and *maximum* of these ratios. Finally, we report MADP_{avg} and MADP_{max} for each n. Though, we could only show that the result of the MADP problem using the radii obtained by the MPDP algorithm is a 2-approximation result, it shows much better performance in our experiment on random instances.

Table 1: Experimental result

n	Sum of square of radii obtained by MPDP algorithm	
	average	maximum
10	1.18383	1.5467
20	1.16704	1.38786
30	1.1568	1.39329
40	1.15132	1.18855
50	1.1728	1.23154

4.3 PTAS

In this section, we propose a PTAS for the MADP problem. In [5], Erlebach et al. proposed a $(1+\frac{1}{k})$-approximation algorithm for the maximum weight independent set for the intersection graph of a set of weighted disks of arbitrary size. We will use this algorithm in designing our PTAS.

For each $p_i \in P$, let the maximum possible radius be $\ell_i = \text{dist}(p_i, \mathcal{N}(p_i))$. Thus, the maximum possible area be $\alpha_i = \pi\ell_i^2$. Given an integer k, we compute $h_i = \frac{\alpha_i}{k}$, and define $k+1$ circles $\mathcal{C}_i = \{C_0^i, C_1^i, \ldots, C_k^i\}$ centered at p_i with area $\{0, h_i, 2h_i, \ldots, kh_i\}$ (see Figure 4). Each disk is assigned weight equal to its area. Now we consider all the disks $\cup_{i=1}^{n}\mathcal{C}_i$, and use the algorithm of [5] to compute the maximum weight independent set (MWIS) $\mathcal{A}$. Note that the number of disks centered at any point p_i present in both the optimum solution and in our algorithm for the MWIS problem of $\cup_{i=1}^{n}\mathcal{C}_i$ is exactly one.

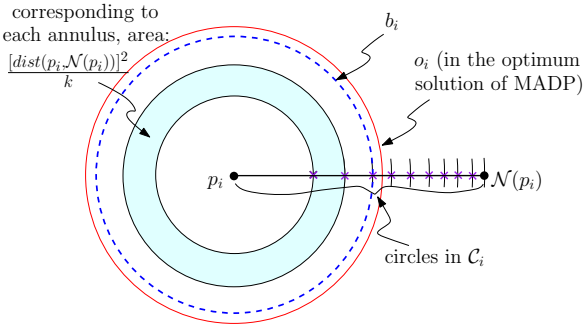


Figure 4: Demonstration of PTAS

Let o_i and a_i be the disks centered at p_i in the optimum solution and in our solution ($\mathcal{A}$) respectively, and O_i, A_i be their respective area. Let $\Theta = \sum_{i=1}^{n} A_i$ be the solution obtained by our algorithm, and $OPT = \sum_{i=1}^{n} O_i$ be the value of the optimum solution. We need to analyze the bound on $\frac{OPT}{\Theta}$.

Let $\widetilde{OPT}$ be the optimum solution of the MWIS problem among the set of disks $\cup_{i=1}^{n}\mathcal{C}_i$. Thus, $\frac{OPT}{\Theta} = \frac{OPT}{\widetilde{OPT}} \times \frac{\widetilde{OPT}}{\Theta}$. Following [5], $\frac{\widetilde{OPT}}{\Theta} \leq 1 + \frac{1}{k}$. It remains to analyze $\frac{OPT}{\widetilde{OPT}}$.

Now, let us consider the disks in OPT. For each point p_i, let b_i be the largest disk in $\mathcal{C}_i$ among those which are smaller than equal to o_i (see the blue and red disks in Figure 4). Thus, $\{b_1, b_2 \ldots, b_n\}$ is a feasible solution. Let $LB(OPT) = \sum_{i=1}^{n} B_i$, where B_i = area of the disk b_i. $LB(OPT)$ is the lower bound of OPT.

$\frac{OPT}{\widetilde{OPT}} = \frac{OPT}{LB(OPT)} \times \frac{LB(OPT)}{\widetilde{OPT}}$. Since $\widetilde{OPT}$ is the optimum solution among the disks $\cup_{i=1}^{n}\mathcal{C}_i$, and $LB(OPT)$ is a feasible solution of the MWIS problem among the disks $\cup_{i=1}^{n}\mathcal{C}_i$, we have $\widetilde{OPT} \geq LB(OPT)$.

Now, consider $OPT - \widetilde{OPT} \leq OPT - LB(OPT) = \sum_{i=1}^{n}(O_i - B_i) \leq \frac{1}{k}\sum_{i=1}^{n}\ell_i^2$, since $O_i - B_i \leq \frac{1}{k}\ell_i^2$ by our construction (see Figure 4). We also have $OPT \geq \frac{1}{4}\sum_{i=1}^{n}\ell_i^2$ from the method of getting the 4-approximation result, mentioned in Section 1.

Thus, $\frac{OPT-\widetilde{OPT}}{OPT} \leq \frac{4}{k}$, implying $\frac{\widetilde{OPT}}{OPT} \geq 1 - \frac{4}{k}$.

In other words, $\frac{OPT}{\widetilde{OPT}} \leq 1 + \frac{1}{k'}$, where $k' = \frac{k-4}{4}$. Thus, $\frac{OPT}{\Theta} \leq (1+\frac{1}{k})(1+\frac{1}{k'}) \leq (1+\frac{1}{k''})$, where $k'' = \frac{k-4}{5}$. Thus, we have the following result.

[3]Since generating the optimum result is time consuming, the table entries for $n = 40$ and 50, the average and maximum is computed only for 5 samples.

Theorem 8 *Given a set of points P in $\mathbb{R}^2$ and a positive integer k, we can get a $(1+\frac{1}{k})$-approximation algorithm with time complexity $(nk)^{O(k^2)}$.*

5 Summary

Following Eppstein's work [4] on placing non-overlapping disks for a set given points on the plane to maximize perimeter, we tried to study the area maximization problem under the same setup. We observe that the solution of the perimeter maximization problem gives a 2-approximation result of the area maximization problem. Though the perimeter maximization problem is polynomially solvable, the area maximization problem is NP-hard [1]. However, the said problem has a PTAS. Needs to mention that, if the points are placed on a straight line, then the area maximization problem is solvable in polynomial time.

References

[1] Ankush Acharyya, Minati De, Subhas C Nandy, and Bodhayan Roy. Range Assignment of Base-Stations Maximizing Coverage Area without Interference. *arXiv preprint arXiv:1705.09346*, 2017.

[2] Cédric Bentz, Denis Cornaz, and Bernard Ries. Packing and covering with linear programming: A survey. *European Journal of Operational Research*, 227(3):409–422, 2013.

[3] Hai-Chau Chang and Lih-Chung Wang. A Simple Proof of Thue's Theorem on Circle Packing.

[4] David Eppstein. Maximizing the Sum of Radii of Disjoint Balls or Disks. In *Proceedings of the 28th Canadian Conference on Computational Geometry, CCCG 2016, August 3-5, 2016, Simon Fraser University, Vancouver, British Columbia, Canada*, pages 260–265, 2016.

[5] Thomas Erlebach, Klaus Jansen, and Eike Seidel. Polynomial-time approximation schemes for geometric intersection graphs. *SIAM Journal on Computing*, 34(6):1302–1323, 2005.

[6] Minyue Fu, Zhi-Quan Luo, and Yinyu Ye. Approximation Algorithms for Quadratic Programming. *J. Comb. Optim.*, 2(1):29–50, 1998.

[7] Jon Kleinberg and Eva Tardos. *Algorithm design*. Pearson Education India, 2006.

[8] S. P. Tarasov; L. G. Khachiyan M. K. Kozlov. Polynomial solvability of convex quadratic programming. *Doklady Akademii Nauk SSSR*, 248:1049–1051, 1979.

[9] Nimrod Megiddo. Towards a Genuinely Polynomial Algorithm for Linear Programming. *SIAM J. Comput.*, 12(2):347–353, 1983.

[10] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1982.

[11] Panos M. Pardalos and Stephen A. Vavasis. Quadratic programming with one negative eigenvalue is NP-hard. *J. Global Optimization*, 1(1):15–22, 1991.

[12] Gábor Fejes Tóth. Packing and Covering. In *Handbook of Discrete and Computational Geometry, Second Edition.*, pages 25–52. 2004.

Nonoverlapping Grid-aligned Rectangle Placement for High Value Areas

Stephen Rowe* Christopher G. Valicka† Scott A. Mitchell‡ Simon X. Zou‡

Abstract

We consider heuristic and optimal solutions to a discrete geometric bin packing problem that arises in a resource allocation problem. An imaging sensor is assigned to collect data over a large area, but some subregions are more valuable than others. To capture these high-value regions with higher fidelity, we can assign some number of non-overlapping rectangular subsets, called "subfootprints." The sensor image is partitioned into squares called "chips", and each chip is further partitioned into pixels. Pixels may have different values. Subfootprints are restricted to rectangular collections of chips, but we are free to choose different rectangle heights, widths, and areas. We seek the optimal arrangement over the family of possible rectangle shapes and sizes.

We provide a mixed-integer linear program optimization formulation, as well as a greedy heuristic, to solve this problem. For the meta-problem, we have some freedom to align the chip boundaries to different pixels. However, it is too expensive to solve the optimization formulation for each alignment. However, we show that the greedy heuristic can inform which pixel alignments are worth solving the optimization over. We use a variant of k-means clustering to group greedy solutions by their transport shape-similarity. For each cluster, we run the optimization problem over the greedy layout with the highest value. In practice this efficiently explores the geometric configuration space, and produces solutions close to the global optimum. We show a contrived example using surveillance of the Mississippi River. Our software is available as open-source in the Github repository "GeoPlace."

1 Introduction

We study a geometric bin packing problem that arises in bandwidth-constrained sensing, e.g. satellite-based sensors. A camera, or other sensor, observes a region of interest. The image, called a *footprint*, is transmitted to some earth-based analyst for downstream processing. The capacity to capture new images exceeds the transmission capacity, so we must make some choices about what data to transmit. Further, some pixels of the image are more interesting than others to the analyst, so it makes sense to devote extra bandwidth to those. Sensor systems are often designed with the capability to devote extra bandwidth (or resolution) to selected subareas, called *sub-footprints*.

However, we are not free to select an arbitrary set of pixels, but are limited by the hardware and software design of the system. A common design for satellite-borne sensors is to partition the pixels of the footprint into a regular rectangular grid of *chips*, each comprised of the same number of pixels. Further, they have the capability to select sub-footprints that are rectangular sets of chips. Each sub-footprint is allocated the same bandwidth, rather than each chip, so typically sub-footprints of about the same area are desired, but this is not a constraint. Sub-footprints are constrained not to overlap, but we may select the width and height of each one independently. That is, the sub-footprints are not required to be translations or rotations of a single shape. The system supports a maximum number of subfootprints. Our objective is to optimally choose rectangular subfootprints that cover the pixels of particular interest, while making efficient use of of the available bandwidth.

Existing techniques for solving this problem usually involve some manual placement of the sub-footprints by an experienced human, someone who would rather be considering higher-level questions. Also, real-time situations arise in which human intervention is impractical. In practice, data collection scenarios are planned hours or more in advance and planners may choose near-optimal sub-footprint placements. At collection time, however, target and spacecraft conditions usually create a difference between planned and actual sensor pointing, external information changes pixel priorities, or both. Accordingly, optimal sub-footprint placements are often different from planned placements and the time required for operators to update solutions is longer than the time available to upload solutions to the spacecraft for execution. Quantifying satellite sensor time can be ambiguous as satellite missions differ, as do the costs of different satellite platforms. In scientific applications, timely information regarding weather or sea-state can have implications regarding the safety of critical infrastructure and human lives. In commercial applications, missed or poorly chosen footprint placements can result in reduced revenue.

Algorithmic solutions to related problems in the satellite sensor planning and tasking literature are limited. Song et al. [6] present algorithms to solve the Satel-

*Systems Mission Engineering, Sandia National Laboratories.
†Information Systems Analysis Center,
‡Center for Computing Research, samitch@sandia.gov,

lite Frame Selection (SFS) problem: for each satellite imaging opportunity, a rectangular satellite footprint is selected to maximally balance coverage of a set of client imaging requests with the resolution requested. Often, sensor placement problems assume circular or elliptical sensor footprints that are allowed to overlap when determining coverage [7], and many placement problems focus on collection scheduling problems [5, 3, 9] with less emphasis on the geometric aspects of the problem.

However, the computational geometry community has considered similar problems arising from other contexts. Packing rectangles into integer grids for efficient device layouts on semiconductor wafers [1] appears related to selecting our sub-footprints. Shifting an integer grid so that many cells contain valuable points [2] appears related to shifting our footprint by a few pixels for improved sub-footprint solutions. By demonstrating a geometric solution to the "sub-footprint" problem, we hope to inspire the computational geometry community to generate novel solutions for satellite planning.

2 "Sub-footprint" Problem Statement

The image footprint is defined as a Cartesian grid of $M \times N$ pixels, which is partitioned into a coarser grid of $M_C \times N_C$ chips, where each chip is comprised of $\Delta x \times \Delta y$ pixels. We must select sub-footprints that are rectangular arrays of chips. The chips are in fixed position relative to the footprint, but it is possible to shift the entire footprint grid by several pixels for better pixel-to-chip alignment. The number of sub-footprints is K, where $K \leq K_{\max}$. The available bandwidth is B. The bandwidth allocated to each sub-footprint is equal, regardless of the number of chips A^k it contains. Consequently, a chip in sub-footprint k receives B/KA^k bandwidth.

Each pixel has a corresponding *priority* value. We define the priority of a chip to be the sum of the priorities of its corresponding pixels. The value of a sub-footprint is the sum of the value of its chips, and the value of the solution is the sum of sub-footprint values. Our goal is to choose the highest value sub-footprints, constrained by the bandwidth. (We also consider the problem of minimizing necessary bandwidth subject to the constraint that all valuable pixels are captured.) One could choose a single large sub-footprint to cover many chips, but the disadvantages are that the bandwidth per chip is low, and because the sub-footprint is rectangular it is likely to cover many low-priority chips. On the other hand, one could choose many small sub-footprints to cover the chips of interest, but there is an upper bound on the number of sub-footprints, and they must not overlap, so it may not be feasible to capture all the high value chips or use the bandwidth efficiently.

Solutions based on optimization formulations are often preferred in this context because they may be seamlessly coupled to a larger optimization-based scheduling framework. However, objectives are often not crisply defined, and finding an exactly optimal solution is often overkill. Solvers tend to be orders of magnitude slower than geometric algorithms. Our solution includes an optimization formulation (Section 3), but we use geometric algorithms to propose a tractable set of scenarios for the optimizer (Section 4).

3 Mixed-Integer Problem

We seek a mixed-integer linear constrained optimization problem (MIP) description that can be efficiently solved by current solvers. The model should try to simultaneously cover high priority chips, give high priority chips the most bandwidth possible, and guarantee that the available bandwidth is not exceeded.

As a secondary objective, we prefer to give higher priority chips more bandwidth; consequently, if two solutions both include a chip with a high priority, we prefer the solution where it is in a smaller sub-footprint. We model this with an objective-function penalty-term proportional to sub-footprint area.

3.1 Footprint Position Constraints, Chip Definitions

We let S^k denote the k-th sub-footprint, and $C^{i,j}$ refer to the chip in the ith row and jth column of the image. We begin by enforcing that the coordinates of a sub-footprint's lower left corner S_0 are not more than the coordinates of its upper right corner S_1. By "coordinate," we mean its chip index: the lower left chip $C^{0,0}$ of the image lies at $(0, 0)$, and the upper right at $(M_C - 1, N_C - 1)$. A sub-footprint covering a single chip has $S_0 = S_1$. Let S^k_{0x} denote the variable for the x coordinate, etc. We constrain

$$S_{0x} \leq S_{1x}, \quad S_{0y} \leq S_{1y}. \tag{1}$$

3.2 Non-Overlapping Rectangles

We label sub-footprints based on the lexicographic order of their lower corner. This reduces the optimization solve time significantly by eliminating symmetric solutions, since sub-footprints are equivalent under relabelling. By "lexicographic order" we mean $(x_0, y_0) \leq (x_1, y_1)$ if and only if $y_0 < y_1$ or $y_0 = y_1$ and $x_0 \leq x_1$. We enforce this by the following constraint $\forall k$:

$$S^k_{0x} + M_C \cdot S^k_{0y} \leq S^{k+1}_{0x} + M_C \cdot S^{k+1}_{0y} \tag{2}$$

We start by ensuring rectangles do not overlap by disjunctive constraints: S^k must be completely to the right of S^l, or S^k must be above S^l, or S^k must be below S^l. The lexicographic ordering already ensures

S^k is not left of S^l. For all $l > k$,

$$S^l_{0x} + D(1 - Y^{k,\ell}_{\text{right}}) \geq S^k_{1x} + 1 \tag{3}$$

$$S^k_{0y} + D(1 - Y^{k,\ell}_{\text{above}}) \geq S^l_{1y} + 1 \tag{4}$$

$$S^l_{0y} + D(1 - Y^{k,\ell}_{\text{below}}) \geq S^k_{1y} + 1 \tag{5}$$

$$Y^{k,\ell}_{\text{right}} + Y^{k,\ell}_{\text{above}} + Y^{k,\ell}_{\text{below}} \geq 1, \tag{6}$$

where constant $D \geq \max(M_C + 1, N_C + 1)$ is a large constant, and 1's are added to avoid strict inequalities.

However, we still have the problem of computing sub-footprint area, which is naturally a product of two variables. Next, in Section 3.3, we introduce indicator variables for whether each chip is covered. These provide sub-footprint areas, and may render the disjunctive constraints obsolete for some objective functions.

3.3 Chip Indicator Variables

We denote the priority of chip $C^{i,j}$ as $p_{i,j}$. We introduce chip-in-sub-footprint indicator variables $\chi^k_{i,j}$ where

$$\begin{cases} \chi^k_{i,j} = & 1 \text{ if } C^{i,j} \in S^k, \\ & 0 \text{ otherwise.} \end{cases} \tag{7}$$

The *area* of a sub-footprint is the number of chips contained within it: $A^k = \sum_{i,j} \chi^k_{i,j}$.

Sub-footprints having empty intersection is equivalent to a chip being in only one sub-footprint, modeled by the constraint $\sum_k \chi^k_{i,j} \leq 1 \, \forall \, i, j$.

3.3.1 Chip in Footprint, χ

To properly set each χ we define four indicator variables V_ℓ, with $\chi = 1$ iff all V_ℓ are 1. We let $V_{\text{right}} = 1$ iff C is to the right of S_0, etc.

To ensure chips outside the sub-footprint have $V = 0$,

$$S_{0x} - (1 - V_{\text{right}}) \cdot D \leq C_x, \tag{8}$$

$$S_{0y} - (1 - V_{\text{above}}) \cdot D \leq C_y, \tag{9}$$

$$S_{1x} + (1 - V_{\text{left}}) \cdot D \geq C_x, \tag{10}$$

$$S_{1y} + (1 - V_{\text{below}}) \cdot D \geq C_y. \tag{11}$$

For the converse, chips inside have $V = 1$, by

$$S_{0x} + V_{\text{right}} \cdot D \geq C_x + 1, \tag{12}$$

$$S_{0y} + V_{\text{above}} \cdot D \geq C_y + 1, \tag{13}$$

$$S_{1x} - V_{\text{left}} \cdot D \leq C_x - 1, \tag{14}$$

$$S_{1y} - V_{\text{below}} \cdot D \leq C_y - 1. \tag{15}$$

For some variations, these four constraints are not required because the objective will ensure that optimal solutions satisfy them.

To force $\chi = 1$ if all V are 1, we add the constraint

$$\chi \geq V_{\text{right}} + V_{\text{above}} + V_{\text{left}} + V_{\text{below}} - 3. \tag{16}$$

And to ensure $\chi = 0$ if any V is 0, we constrain

$$\chi \leq V_\ell \, \forall \ell. \tag{17}$$

3.4 Area and Bandwidth Constraints

The area A^k of sub-footprint S^k is simply the number of chips within the sub-footprint, so the optimization problem defines variables A^k with constraints $A^k = \sum_{i,j} \chi^k_{i,j}$. The area is important for bandwidth considerations. The system is designed so that, given bandwidth B, each sub-footprint gets bandwidth B/K, and the bandwidth per chip is B/KA^k. Here B and K are constants. Consequently, larger area sub-footprints get smaller bandwidths per chip; We will revisit this issue in Section 3.5.

Since each chip requires a minimum bandwidth for high fidelity, the bandwidth constrains the total number of chips that can be downlinked, and the maximum area of a single sub-footprint. We constrain

$$K \cdot A^k \leq B. \tag{18}$$

3.5 Objective Function

We describe the principles behind each of the objective function terms. The primary criterion is to maximize the priority of chips covered by sub-footprints. A *reward term* captures the benefit of covering a chip with a sub-footprint:

$$R = \sum_{k \in K} \sum_{i,j} \chi^k_{i,j} \cdot p_{i,j}. \tag{19}$$

We explicitly penalize covering zero-priority chips:

$$P_0 = \gamma \sum_{k,i,j} \chi^k_{i,j}, \tag{20}$$

where γ is a constant, less than the smallest positive priority. This is necessary to ensure sub-footprints are as small as possible while still covering the same positive-priority chips, because we are constrained by the bandwidth, rather than optimizing bandwidth.

A secondary criterion is to place high-priority chips in smaller sub-footprints, because they get more bandwidth devoted to them from the fixed bandwidth per sub-footprint. This inspires a *penalty term* for a chip that increases with its value and the containing sub-footprint's area:

$$P_A = \epsilon \sum_k \sum_{i,j} A^k \cdot \chi^k_{i,j} \cdot p_{i,j}, \tag{21}$$

where ϵ is another such constant.

Given how bandwidth is assigned to chips, It appears natural to *divide* the reward by the area, but this leads to a non-linear objective function, which is more expensive to solve, so we subtract the penalty instead. Our objective function F is then

$$F = R - P_A - P_0 \tag{22}$$

The problem with this formulation is that it is still non-linear: (21) contains the product of variables $\chi^k_{i,j}$ and A^k. To keep the constraints linear, we introduce new real valued variables Z to replace $\chi \cdot A$. Consequently, the new reward term replacing (19) and (21) is

$$\tilde{R} = \sum_k \sum_{i,j} z^k_{i,j} \cdot p_{i,j}. \tag{23}$$

We design constraints so that Z is nearly equal to χ, but exactly zero in the case that $\chi = 0$, and penalized by sub-footprint area A. We let Z be real-valued between 0 and 1, and enforce $z^k_{i,j} \leq \chi^k_{i,j}$, so $\chi = 0 \implies Z = 0$.

We introduce slack variables s such that $\chi = 1 \implies s = 0$. We let s be real-valued in $[0, 1]$, with $s_{i,j} \leq 1 - \chi^k_{i,j}$. The sub-footprint area penalty is built into Z via the constraint that

$$z^k_{i,j} - s_{i,j} \leq \chi^k_{i,j} - \epsilon A^k. \tag{24}$$

And our objective function is

$$\tilde{F} = \tilde{R} - P_0. \tag{25}$$

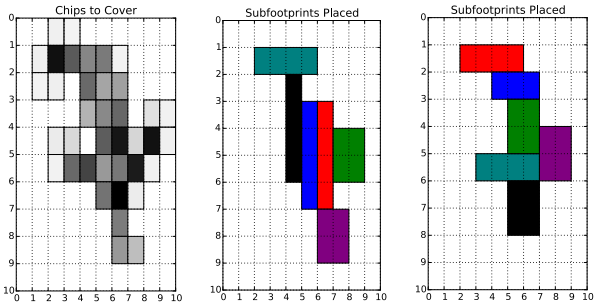
3.5.1 Optimized Bandwidth, Constrained Coverage

We also consider the problem of minimizing the required bandwidth, subject to the constraint that all chips above some threshold value must be covered. For this variant, we use the non-overlapping rectangle constraints from Section 3.2. We may either compute area using the chip indicator variables from Section 3.3, or approximate area by footprint perimeter.

4 Greedy Solution

While the optimization problem provides high quality solutions, solving it is time consuming, mostly due to the large number of binary or integer variables. E.g., order minutes for the example we demonstrate in this paper. In contrast, geometric solutions for problems of this size typically take fractions of a second. Hence, we explore a greedy heuristic for choosing sub-footprints, in order to inform the optimization problem. We iteratively choose the next sub-footprint that covers the highest priority chips, but does not overlap with any prior sub-footprint. The sub-footprint must have area $A^k \leq B/K$, but may otherwise be any shape of rectangle. We precompute the allowable sub-footprint shapes. The allowable shape with the largest area is not always chosen, e.g., 2×4 may be preferable to 3×3.

By definition, the greedy solution is not better than the optimal solution. Figure 1 displays the results of a greedy solution in Figure 1(b), and an optimization solution Figure 1(c), for the same set of chips from Figure 1(a). In this case the solutions cover different chips, and use sub-footprints of different shapes. The greedy solution appears less compact, with some small gaps of uncovered chips, as one might expect.



(a) Chips to cover. **(b)** Greedy solution. **(c)** Optimal MIP.

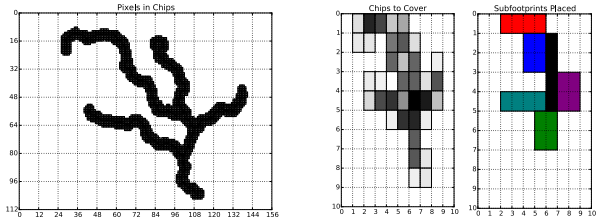
Figure 1: A greedy solution vs. an optimized solution for the same chip layout. In the left column, darker chips have higher value. This example was chosen to illustrate the potential for a large difference between the greedy and optimal solutions; in many examples they are more similar, or even identical.

5 Shifting Pixels to Find Optimal Sub-footprints

In Section 3, we considered the case that chips were in fixed position relative to the pixels of interest. Here, we explore the freedom to shift the entire footprint by a few pixels, in order to align chip boundaries with the pixels of interest, to enable higher value sub-footprint placements. Recall the priority of a chip is the sum of priorities of its pixels. Another possible benefit of shifting is to concentrate high priority pixels into fewer chips. We may shift by up to $\Delta x - 1 \times \Delta y - 1$ pixels, after which the the problem statement repeats itself through periodic symmetry.

Figure 2(a) shows an example corresponding to the Mississippi River. The image has $M_C \times N_C = 10 \times 10$ chips, $M \times N = 160 \times 120$ pixels. The darker pixels have higher priorities. We consider a maximum of $K = 6$ sub-footprints and bandwidth $B = 24$.

We exhaustively consider all possible shifts, and for each find the greedy solution. We also find the *perfect* solution of the B highest priority chips, independent of the sub-footprint geometric constraints. The perfect solution value is an upper bound on the optimal solution, just as the greedy solution is a lower bound. Evaluating all of these configurations is faster than solving a single optimization instance. The goal of this study was to confirm that shifting can provide improved solutions in practice, and further to provide guidelines for selecting which shifts are worth running the optimization algorithm over. Figure 2(b) displays outputs from two different shifts. The corresponding objective function values and generated sub-footprints are significantly different.



(a) The Mississippi River, courtesy Wikipedia. Darker pixels are more valuable.

(b) An optimal solution for the same set of pixels, but shifted to lie in different chips.

Figure 2: Shifting the pixels to align with chip boundaries results in a qualitatively different sub-footprint layout, and the value of the solution varies by about 3%, compared to Figure 1.

5.1 Selecting Layouts for MIP

The MIP runtime is too expensive to run over all shifts. Instead we seek a subset of promising shifts, and run the MIP only on those. We use the following variant of k-means clustering, where k is the budget of the number of MIPs we can run. The first cluster center is the layout with highest greedy value. Each subsequent cluster, up to k, is the layout with maximum *distance* from all prior clusters. We then form clusters by assigning all layouts to their closest center. We choose the layout with the highest greedy value in a cluster as its *exemplar*. We then run the MIP the exemplars. Figure 4 shows our example with $k = 10$. Figure 3 provides evidence that running the MIP on the exemplar is sufficient within a cluster, but that it is worthwhile running the MIP on multiple clusters.

We define the *distance* D between two layouts as a kind of transport or Wasserstein distance, the minimum *work* needed to transform one layout A to another B [8]. *Work* is defined as follows. The work for translating the entirety of A by an integral number of chips is zero. Each sub-footprint in A is matched to a sub-footprint in B; the work of re-labeling sub-footprints to provide a different matching is zero. The work of translating a chip from a sub-footprint in A to a sub-footprint in B is the L_1 distance between them.

This definition captures both the differences in the covered chips, but also the differences in how those chips are covered. (We also experimented with selecting layouts based on the *perfect* solution, but this was not as predictive. Specifically, sometimes the optimal solution was low. See Figure 3.)

Computing D is expensive, and getting it exact is unimportant, so we approximate it by D'. (We must compute Kn distances, where n is the number of layouts, and solving D is an optimization problem with many discrete decisions.) The D' distance between two sub-footprints is the sum of the L_1 distances between a greedy min L_1 pairing of their chips. The D' distance between two layouts is the sum of the D' distances between a greedy pairing of their sub-footprints. For the D' distance between layouts, we consider four translations of A, by up to one chip, taking advantage of the knowledge of how pixels were shifted.

Figure 3: Intra- and inter-cluster variability. Note the strong correlation between the greedy and optimal solution within a cluster; it suffices to run the MIP on the layout with the best greedy value within a cluster. The correlation is less strong between clusters, because of the qualitatively different geometric differences between the greedy and optimal solutions from one cluster to another. For example, for the 4th cluster, the greedy solution is optimal, while for other layouts there are significant gaps. Hence it is worthwhile running the MIP on the exemplar from each cluster.

6 GeoPlace Open Source Software

We provide optimization and heuristic methods for sub-footprint placement in the open-source software GeoPlace [1]. We provide several test models. The repository also holds optimization formulations for the related problem of placing a mosaic of footprints to cover a large area, and for scheduling the placement of footprints.

Optimization models were expressed in the Pyomo[2] [4] modeling language. The development environment was Linux and the source code was primarily developed in Python 2.7. Pyomo uses TPL's for the actual solvers, we primarily used CPLEX[3] (commercial) and Gurobi[4] (free academic use license). The repository includes additional Python and C++ routines for visualizing solutions.

7 Conclusions

We have shown optimization based-solutions to a rectangle placement problem. We have used greedy heuristics and geometric clustering to select promising and

[1] GeoPlace `https://github.com/cgvalic/GeoPlace`
[2] Pyomo `http://www.pyomo.org`
[3] CPLEX `https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer`
[4] Gurobi `http://www.gurobi.com`

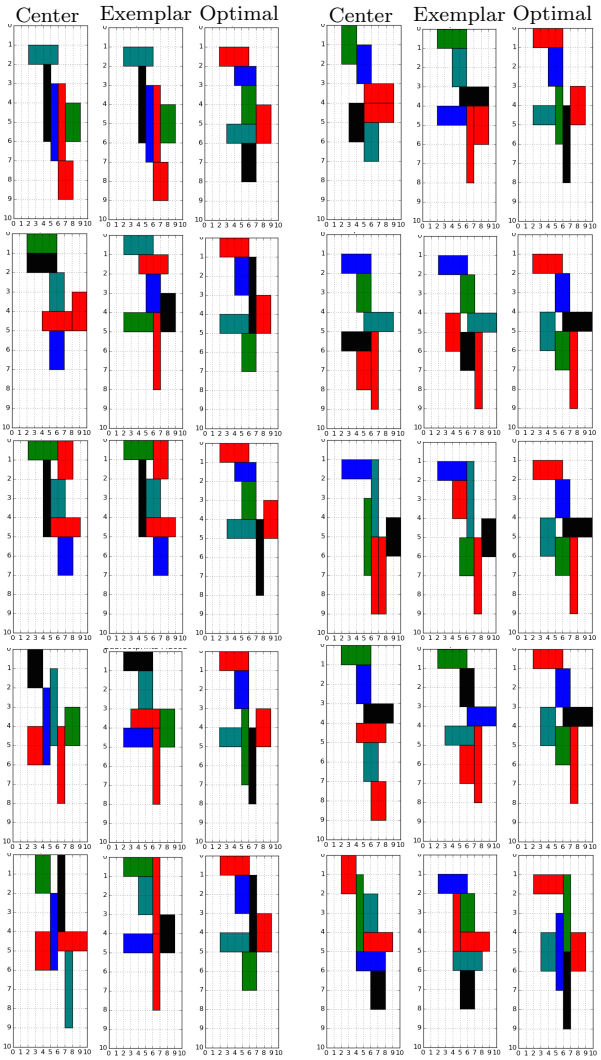


Figure 4: Greedy solutions for cluster centers and exemplars, and MIP optimal solution for exemplars. We consider 181 patterns of shifting up to 12 × 16 pixels. Here cluster centers (exemplars) = { 165(165), 38(114), 86(115), 176(173), 108(108), 188(189), 48(81), 85(142), 45(99) 56(23)}.

geometrically-distinct problem instances on which to invest the time to run the optimization solution. The problems are small enough that the complexity of the geometric algorithms is not an issue. Moreover, they are orders of magnitude faster than the optimization-based approaches. While the optimization problem is expensive to solve, it offers some advantages in terms of flexibility and utility. The approach is extensible to new objectives and constraints. The optimization problem may be incorporated into larger ones, such as scheduling. In the broader project, we developed scheduling software, and footprint and subfootprint placement algorithms.

Here we have sought to optimize the covered chips subject to a bandwidth constraint. In future work, it would be worth considering Pareto optimal solutions to the multi-objective optimization problem of maximizing chip coverage while minimizing bandwidth. We hope the community is inspired to develop geometric algorithms for more satellite planning and tasking problems.

Acknowledgements

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research (ASCR), Applied Mathematics Program. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA-0003525.

References

[1] M. Andersson, J. Gudmundsson, and C. Levcopoulos. Chips on wafers, or packing rectangles into grids. *Computational Geometry*, 30(2):95 – 111, 2005.

[2] P. Bose, M. van Kreveld, A. Maheshwari, P. Morin, and J. Morrison. Translating a regular grid over a point set. *Computational Geometry*, 25(1):21 – 34, 2003.

[3] J. Frank, A. Jónsson, R. Morris, and D. E. Smith. Planning and scheduling for fleets of Earth observing satellites. In *Proceedings of Sixth Int. Symp. on Artificial Intelligence, Robotics, Automation & Space*, 2001.

[4] W. E. Hart, C. Laird, J.-P. Watson, and D. L. Woodruff. *Pyomo–optimization modeling in Python*, volume 67. Springer Science & Business Media, 2012.

[5] G. Peng, L. Wen, Y. Feng, B. Baocun, and Y. Jing. Simulated annealing algorithm for EOS scheduling problem with task merging. In *Modelling, Identification and Control (ICMIC)*, pages 547–552, June 2011.

[6] D. Song, A. F. van der Stappen, and K. Goldberg. An exact algorithm optimizing coverage-resolution for automated satellite frame selection. In *Int. Conf. on Robotics and Automation*, volume 1, pages 63–70, April 2004.

[7] Y. Ulybyshev. Satellite constellation design for complex coverage. *Journal of Spacecraft and Rockets*, 45(4):843–849, 2008.

[8] R. C. Veltkamp. Shape matching: Similarity measures and algorithms. In *Proceedings Int. Conf. on Shape Modeling and Applications*, pages 188–197, May 2001.

[9] F. Xhafa, J. Sun, A. Barolli, A. Biberaj, and L. Barolli. Genetic algorithms for satellite scheduling problems. *Mobile Information Systems*, 8(4):351–377, Oct 2012.

Packing Boundary-Anchored Rectangles

Therese Biedl* Ahmad Biniaz† Anil Maheshwari† Saeed Mehrabi*

Abstract

In this paper, we study the *boundary-anchored rectangle packing* problem in which we are given a set P of points on the boundary of an axis-aligned square Q. The goal is to find a set of disjoint axis-aligned rectangles in Q such that each rectangle is anchored at some point in P, each point in P is used to anchor at most one rectangle, and the total area of the rectangles is maximized. We show how to solve this problem in linear-time in the number of points of P, provided that the points of P are given in sorted order along the boundary of Q. The solvability of the general version of this problem, in which the points of P can also lie in the interior of Q, in polynomial time, is still open.

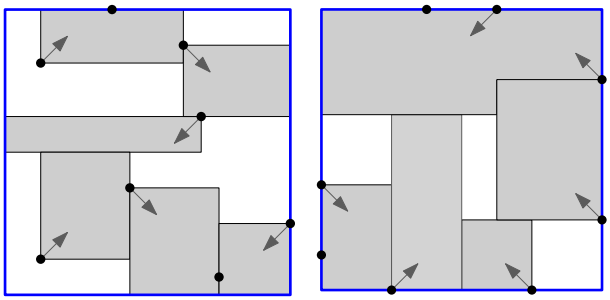
1 Introduction

Let Q be an axis-aligned square in the plane and P be a set of points in Q. Call a rectangle r *anchored* at a point $p \in P$ if p is a corner of r. The *anchored rectangle packing* (ARP) problem is to find a set S of disjoint axis-aligned rectangles in Q such that each rectangle in S is anchored at some point in P, each point in P is a corner of at most one rectangle in S, and the total area of the rectangles in S is maximized; see Figure 1(a). It is not known whether or not this problem is NP-hard. The best known approximation algorithm for this problem, which achieves ratio $7/12 - \varepsilon$, is due to Balas et al. [1]. They also studied several variants of this problem.

In this paper, we study a simpler variant of the anchored rectangle packing problem in which all the points of P lie on the boundary of Q. We refer to this variant as the *boundary-anchored rectangle packing* (BARP) problem; see Figure 1(b). We present a simple algorithm that solves the BARP problem in linear time, provided that the points of P are given in sorted order along the boundary of Q. Despite the simplicity of our algorithm, its correctness proof is non-trivial. We present our algorithm in Section 3, and prove its correctness in Section 4.

*Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada. biedl@uwaterloo.ca, smehrabi@uwaterloo.ca. Research of TB supported by NSERC. Part of this work was done while SM was visiting Carleton University.

†School of Computer Science, Carleton University, Ottawa, Canada. ahmad.biniaz@gmail.com, anil@scs.carleton.ca



(a) (b)

Figure 1: Instances of (a) the ARP problem, and (b) the BARP problem.

Related results. The rectangle packing problem is related to strip packing and bin packing problems, which are well-known optimization problems in computational geometry. Rectangle packing problems have applications in map labelling [4, 7]. Balas et al. [1] studied several variants of the anchored rectangle packing problem, namely, the *lower-left anchored rectangle packing* problem in which points of P are required to be on the lower-left corners of the rectangles in R, the *anchored square packing* problem in which every anchored rectangles is required to be a square, and the *lower-left anchored square packing* problem which is a combination the two previous problems. For the lower-left rectangle packing problem, Freedman [6] conjectured that there is a solution that covers 50% of the area of Q. The best known lower bound of 9.1% of the area of Q is due to Dumitrescu and Tóth [3]. Balas et al. [1] presented approximation algorithms with ratios $(7/12 - \varepsilon)$ and 5/32 for anchored rectangles and anchored square, respectively. They also presented a 1/3-approximation algorithm for the lower-left anchored square packing problem, and proved that this lower bound is tight. Balas and Tóth [2] studied the combinatorial structure of maximal anchored rectangle packings and showed that the number of such distinct packings with the maximum area can be exponential in the number n of points of P; they give an exponential upper bound of $2^n C_n$, where C_n denotes the nth Catalan number.

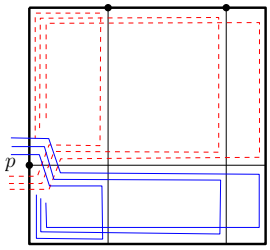


Figure 2: BARP can be solved via maximum-weight independent set in an outer-string graph.

2 An outline

We first briefly argue that BARP is sovable in polynomial time. It is easy to see [1] that in any rectangle packing the boundaries of rectangles must lie on the *grid* Λ obtained by extending rays inwards from all points until they hit the opposite boundary. For each point $p \in P$, there are $O(n^2)$ potential rectangles of Λ anchored at p and so we have $O(n^3)$ candidate rectangles, of which we must pick an independent set (among their intersection graph) such that the sum of the weights (defined to be the area of each rectangle) is maximized. If all points are on the boundary, then it is easy to represent each rectangle as a *string* (i.e., a Jordan curve) such that all strings have a point on the infinite face and two strings intersect if and only if not both rectangles should be taken, see Figure 2. It is known that maximum-weighted independent set is solvable in $O(N^3)$ time on an outer-string graph with a geometric representation of $O(N)$ [5]. As such, BARP is solvable in $O(n^9)$ time, but this is rather slow.

In this section, we give key insights that lead to faster algorithms. Define a *cell* to be a maximal rectangle not intersected by lines of grid Λ. Given an optimum solution S, define a *hole* of S to be a maximal connected region of Q that is not covered by S, see Figure 3(b). We show the following in Section 4:

Insight 1 *An optimal solution S either covers all of Q, or it has exactly one hole which is a single cell.*

It is quite easy to test whether all of Q can be covered (see Lemma 10). If this is not possible, then we want to minimize the hole. However, there are a quadratic number of cells, and more crucially, not all cells are feasible (i.e., can be holes in a packing). The second key result is therefore the following (by Theorem 2):

Lemma 1 *For any cell ψ, we can test in $O(1)$ time whether some packing covers $Q - \psi$.*

This immediately gives an $O(n^2 \log n)$ algorithm to find the best solution of type $Q - \psi$: sort the cells by increasing area, and test for each of them whether it is feasible until we succeed. However, it is not necessary to test each cell individually. We can characterize exactly when a cell ψ is feasible, based solely on where the supporting lines of ψ (which are either the boundary of Q or rays emanating from some points) have their endpoints. Hence we need not look at individual cells, only at the list of points on the four sides, to find the minimum area hole.

3 A Linear-Time Algorithm

Before stating this characterization, we need a few definitions. We write $P_B/P_L/P_T/P_R$ for the points of P on the bottom/left/top/right side. For a point p in the plane, we denote by $x(p)$ and $y(p)$ the x- and y-coordinates of p, respectively. The following theorem proved in Section 4 characterizes possible optimal solutions; Figure 7 on page 5 illustrates these configurations.

Theorem 2 *Any BARP instance has an optimal solution S with $i \leq 4$ rectangles. Moreover (up to rotating the instance by a multiple of 90° and/or reflecting horizontally) the anchor-points $p_1, \ldots, p_i$ used by S satisfy one of the following:*

1. *$i = 1$, and p_1 is the leftmost point of $P_L \cup P_B$.*
2. *$i = 2$, and one of the following holds:*
   (a) *p_1 is the bottommost point of P_L and p_2 is the leftmost point of $P_T \cup P_B$, or*
   (b) *p_1 and p_2 are the two points of $P_T \cup P_B$ with the closest x-coordinates.*
3. *$i = 3$, $p_1 \in P_B$ and $p_2 \in P_T \cup P_B$ have closest x-coordinates with $x(p_1) < x(p_2)$, and p_3 is the lowest point in P_L.*
4. *$i = 4$, $p_1 \in P_L$ and $p_3 \in P_R$ have closest y-coordinates with $y(p_1) > y(p_3)$, and $p_2 \in P_T$ and $p_4 \in P_B$ have the closest x-coordinates with $x(p_4) < x(p_2)$.*

Algorithm. Our algorithm proceeds as follows. For each of the four rotations, for each of the two reflections and for each rule 1, 2(a), 2(b), 3, and 4 in Theorem 2, compute the corresponding point set. Each of these up to 40 point sets defines a cell H, and a packing that covers $Q - H$ (see also Lemma 8). The algorithm returns the one that has the smallest hole H.

Having P_L, P_T, P_R, and P_B sorted along the boundary of Q, we can also compute sorted lists of $P_L \cup P_R$ and $P_T \cup P_B$ in linear time. The closest pair within each or between two of them can be computed in linear time. This implies our claimed running time.

The correctness will be proved in Section 4, and does not use that Q is a square, only that it is an axis-aligned rectangle. We hence have:

Theorem 3 *The boundary anchored rectangle packing problem for n points, given in sorted order on the boundary of a rectangle, can be solved in $O(n)$ time.*

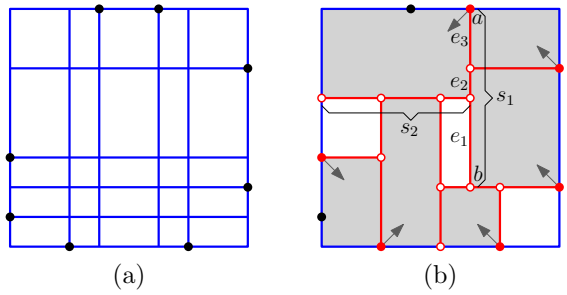


Figure 3: (a) The grid Λ. (b) White regions are holes. Graph $G(S)$ is in red (thick); filled vertices are points of P. The max-segment s_1 is introduced while s_2 is not.

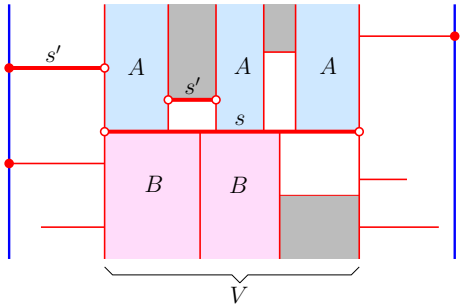


Figure 4: Illustration of the proof of Lemma 4.

4 Correctness of the Algorithm

We first eliminate some simple cases.

Observation 1 *Assume one of the following holds.*

(*i*) *there exists a point $p_1 \in P$ on a corner of Q, or*

(*ii*) *there exist two points in $p_1, p_2 \in P_L \cup P_R$ that have the same y-coordinates, or*

(*iii*) *there exist two points in $p_1, p_2 \in P_T \cup P_B$ that have the same x-coordinates.*

Then we can cover all of Q with anchored rectangles.

Proof. In case (i), one rectangle anchored at p_1 can cover all of Q. In case (ii) and (iii), two rectangles anchored at p_1, p_2 can cover all of Q. □

Since these conditions are easily tested, we assume for most of the remaining section that none of (i-iii) holds. (We will see that this implies that there must be a hole.)

We need some notation. Throughout this section, let S be a solution for the BARP problem. The term "rectangle" now means one of the rectangles used by S. Define $G(S)$ to be the graph whose vertices are the rectangle-corners that are not corners of Q, and whose edges are coincident with the rectangle-sides not on the boundary of Q; see Figure 3(b).

We define a *max-segment* of $G(S)$ to be a maximal chain s of collinear edges of $G(S)$. We say that s is *introduced* if at least one endpoint of s belongs to P and is used as anchor-point for some rectangle of S. Every edge e belongs to exactly one max-segment s_e; we say that e is *introduced* if s_e is. See Figure 3(b) We already know [1] that all boundaries of rectangles can be assumed to lie on the grid Λ, but we need to strengthen this a bit and prove the following:

Lemma 4 *There exists an optimal solution S such that all max-segments of S are introduced.*

Proof. Let S be an optimal solution that, among all optimal solutions, minimizes the number of max-segments. Assume for contradiction that there exists a max-segment s that is not introduced. After rotation we may assume that s is horizontal. Let V be the vertical slab defined by the two vertical lines through the endpoints of s; see Figure 4.

Consider moving s upward in parallel, i.e., shortening the rectangles A with their bottom sides on s and lengthening the rectangles B with their top sides on s. Observe first that these rectangles indeed can be shortened/lengthened, because none of them can be anchord at a point on s: the only points of s that are possibly in P are its ends, but neither of them anchors a rectangle since s is not introduced. If this move of s increases the coverage, then S was not optimal, a contradiction. If this decreases the coverage, then moving downward in parallel would increase the coverage, a contradiction. So the covered area must remain the same during the move. Shift s up until it hits either the boundary of Q or intersects some other horizontal max-segment s' of $G(S)$. If s hits the boundary of Q, then s disappears and will be deleted from $G(S)$. If s intersects s' of $G(S)$ (which may be inside V or only share an endpoint with the translated s) then the two max-segments merge into one. Either way we decrease the number of max-segments, which contradicts the choice of S and proves the lemma. □

From now on, without further mentioning, we assume that S is an optimum solution where all max-segments are introduced. We also assume that, among all such optimal solutions, S minimizes the number of rectangles.

Lemma 5 *Every internal vertex of $G(S)$ has degree three or four.*

Proof. Every internal vertex b of $G(S)$ resides on the corner(s) of axis-aligned rectangle(s), and so has degree at least 2 and at most 4. Assume for contradiction that b has degree exactly 2, and let a and c be its neighbours. After possible rotation, we may assume that a lies to the left of b, and c lies above b, as depicted in Figure 5. Thus, b is the bottom-right corner of some rectangle r_1, and no other rectangle has b on its boundary. This

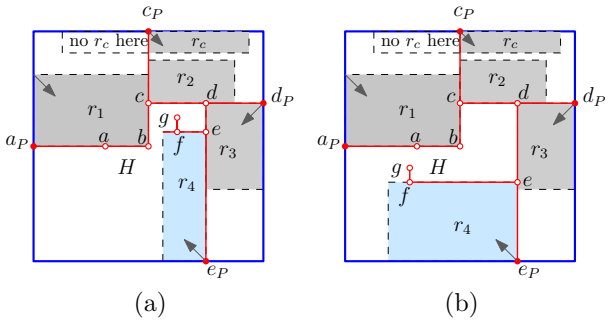


Figure 5: Illustration of the proof of Lemma 5.

implies that the region to the right of bc and below ab is a hole H. So rectangle r_1 is anchored either on the left or the top side of Q; after a possible diagonal flip we assume that it is anchored on the left.

Define a_P and c_P be the points of P that introduced ab and cb, respectively; we know that these must be on P_L respectively P_T since b has degree 2. By definition of "introduced" some rectangle r_c is anchored at c_P. We claim that r_c cannot have c_P as its top-right corner. Assume for contradiction that it did. Then we can expand r_c (if needed) to cover the entire rectangle spanned by a_P and c_P; this can only increase the coverage. In particular, the expanded r_c covers all of r_1. We know that $r_1 \neq r_c$ since r_1 was anchored on the left side of Q. This contradicts that S has the minimum number of rectangles, so r_c has c_P as its top-left corner.

If the right side $\mathrm{rs}(r_1)$ of r_1 is a sub-segment of bc, then we can stretch r_1 to the right to increase the coverage of S, contradicting optimality. So $\mathrm{rs}(r_1)$ must be a strict super-segment of bc, which in particular implies that c is interior and has no leftward edge. Since c is a vertex, it must have a rightward edge; let d be the vertex of H to the right of c. Let r_2 be the rectangle whose bottom-left corner is c; this exists since edge cd is the boundary of some rectangle(s), but the area below cd belongs to hole H. Rectangle r_2 cannot be anchored on the right, because otherwise we could expand r_c to cover all of r_2 and reduce the number of rectangles, a contradiction. So r_2 is anchored on the top, which implies that $r_2 = r_c$, else they would overlap.

If the bottom side $\mathrm{bs}(r_2)$ of r_2 is a sub-segment of cd, then we can stretch r_2 down to increase the coverage of S. So $\mathrm{bs}(r_2)$ is a strict super-segment of cd, which implies that d is interior. We iterate this process three times as follows. (i) Let e be the vertex of H that is below d, and let r_3 be the rectangle whose top-left corner is d. Argue as before that r_3 is anchored at the right endpoint d_P of the max-segment through cd, therefore the left side $\mathrm{ls}(r_3)$ is a strict super-segment of de and e is interior. (ii) Let f be the vertex of H that is to the left of e, and let r_4 be the rectangle whose top-right corner is e. Argue as before that r_4 is anchored at the bottom endpoint e_P of the max-segment through de, therefore the top side $\mathrm{ts}(r_4)$ is a strict super-segment of ef and f is interior. (iii) Finally, let g be the vertex of H that is above f (possibly $g = a$). Now observe that the max-segment through fg cannot reach the boundary of Q without intersecting r_4, r_1 or r_2. Therefore, fg is not introduced, a contradiction. □

We assumed that neither (ii) nor (iii) of Observation 1 holds, which means that any grid-line of grid Λ has exactly one end in P. So, we can direct the edges of the grid (and with it the edges of $G(S)$) from the end in P to the end not in P. See also Figure 7. Define a *guillotine cut* to be a max-segment of $G(S)$ for which both endpoints are on the boundary Q.

Lemma 6 *If there is no guillotine cut, then S has a hole H. Furthermore, H is a rectangle, H is not incident to the boundary of Q, and the boundary of H is a directed cycle of $G(S)$.*

Proof. We claim that no vertex w of $G(S)$ on the boundary of Q is a sink. For if the unique edge incident to w were directed $v \to w$, then by Lemma 4 and the way we directed the edges of $G(S)$, the point p that introduced vw would be on the opposite side and hence the max-segment pw would be a guillotine cut. Likewise no interior vertex w can be a sink, because $\deg(w) \geq 3$ by the previous lemma, which implies that two incident edge of w have the same orientation (horizontal or vertical). One of them then becomes outgoing at w since we direct edges along grid-lines. So $G(S)$ has no sink, which implies that it has a directed cycle C. The region enclosed by C has no point on the boundary, so no rectangle anchored on the boundary can cover parts of it without intersecting C. So the interior region of C is a hole H not incident to the boundary. We know that H is a rectangle since it has no vertex of degree 2 by the previous lemma, hence in particular no reflex vertex. □

This lemma serves as base-case for a stronger claim.

Lemma 7 *If S has holes, then it has a hole H that is a rectangle. Furthermore, every interior corner of H has an incoming edge that lies on H.*

Proof. If there is no guillotine cut, then Lemma 6 gives a rectangular hole that is interior and whose boundary is a directed cycle; this satisfies all claims. So, assume that there is a guillotine-cut aa', say it is horizontal. Since (ii) does not hold, not both a and a' can belong to P, say $a' \notin P$. Segment aa' divides Q into two rectangles Q_1 and Q_2 with Q_1 above Q_2; see Figure 6(a). There is a rectangle r_1 that is anchored at a; up to a vertical flip we may assume that r_1 is inside Q_1. Observe that r_1 must cover all of Q_1, else we could find a solution with

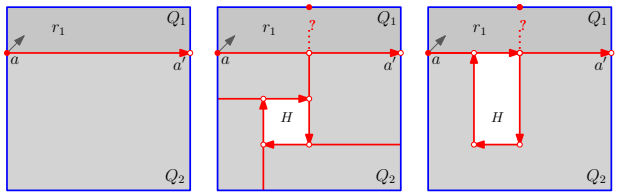


Figure 6: With a guillotine cut, a hole can be found in Q_2 recursively.

more coverage or fewer rectangles. Thus $S' := S \setminus \{r_1\}$ is an anchored-rectangle packing for Q_2 with anchor-points in $P \setminus \{a\}$. S' must be optimal for Q_2, else we could get a better packing for Q by adding r_1 to it. It cannot cover all of Q_2 since S had holes. So, induction applies to S', and it has a hole H.

Assume first that some vertical edge e of H is in the interior and directed downward, see Figure 6(b) and (c). Since e is introduced, the max-segment s_e containing it must then extend to the top of Q. This is impossible since s_e would intersect r_1. So all interior vertical edges of H are directed upwards.

This immediately shows that H cannot be in the interior of Q_2, because then its edges form a directed cycle and one of the vertical ones is directed downward. Likewise it is impossible that both vertical sides and the bottom side of H are interior to Q_2, since the tail-end of the bottom side has an incoming edge from H, which hence must be a downward vertical edge. Therefore H shares at least one side with the boundary of Q.

It remains to argue that any interior corner c of H has an incoming edge on H. If c was interior to Q_2 as well then this holds by induction. If c is interior to Q, but not to Q_2, then c lies on aa' but $c \neq a, a'$. Then the vertical edge of H incident to c is interior to Q_2, so it is directed upward as argued above and hence incoming to c as desired. □

Hence, hole H must satisfy this *hole-condition* on the edge-directions (at least for some optimal solution S); that is, every interior corner of H has an incoming edge that lies on H. It turns out that this condition is also sufficient.

Lemma 8 *Let H be a rectangle whose sides lie on $Q \cup \Lambda$. If every interior corner of H has an incoming edge that lies on H, then there exists a packing that covers $Q \setminus H$.*

Proof. Let $p_1, \ldots, p_i$ (for some $i \leq 4$) be the points of P that defined the grid-lines on which the sides of H reside. We distinguish cases (1-4) depending on how many sides of H are interior, where (2) splits further into (2a) and (2b) depending on whether the sides are

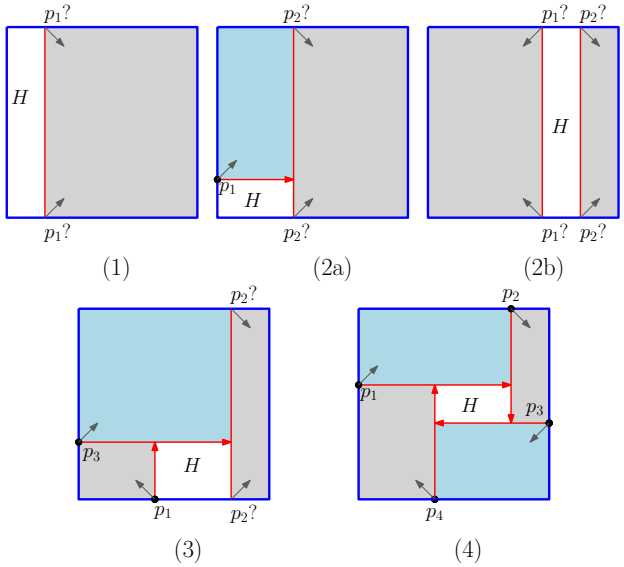


Figure 7: Any rectangle whose boundary is directed suitably can be realized as hole.

adjacent or parallel. After possible rotation, the hole is situated as shown in Figure 7. Every interior corner of H has an incoming edge that is on H, which (up to reflection) forces the location of some of $p_1, \ldots, p_i$ as indicated in the figure. In all cases, one verifies that i rectangles anchored at $p_1, \ldots, p_i$ suffice to cover $Q \setminus H$. □

We are now ready to prove Insight 1. To this end, we first show the following:

Lemma 9 *If S has holes, then it has exactly one hole H, and H is a cell of Λ.*

Proof. Lemma 7 shows we may assume H to be a rectangle where all interior corners have incoming edges on H. By Lemma 8, we can cover $Q \setminus H$ with anchored rectangles, which by maximality of S means that H is unique.

If H is not a cell, then it is bisected by some grid-line ℓ into two pieces H_1 and H_2. If some $H' \in \{H_1, H_2\}$ satisfies the hole-condition (i.e., all interior corners have incoming edges on H'), then we can create a packing that covers $Q \setminus H' \supset Q \setminus H$, which contradicts minimality of S. In fact, by inspecting the possible configurations of H in cases 1, 2a, 2b, 3, and 4, as well as possible placements of the "undecided" anchor-points and the orientation/direction of ℓ (see Figure 8, which shows all but one case), we observe that H_1 satisfies this condition as we can cover $Q \setminus H_1$ in each of these cases. So, there is a contradiction in all cases, and H must be one cell. □

By Lemma 9, we have characterized solutions that have holes. It remains to characterize solutions that do

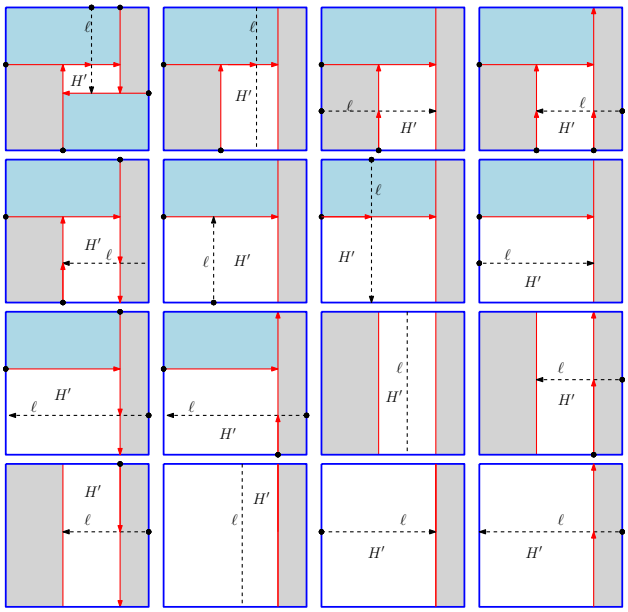


Figure 8: Any hole bisected by a grid-line ℓ gives rise to another hole H'.

not have holes; i.e., to show that the conditions (i-iii) of Observation 1 are necessary.

Lemma 10 *If Q can be covered with anchored rectangles, then one of (i-iii) holds.*

Proof. Let S be a packing that covers all of Q. If $G(S)$ has no edge, then all of Q must be covered by one rectangle, which hence must be anchored at a corner of Q and (i) holds. So assume that $G(S)$ has edges. By Lemma 6, since S has no hole there must be a guillotine-cut aa', say it is horizontal. If both a and a' are in P then (ii) holds and we are done, so assume $a \in P$ and $a' \notin P$.

Define Q_1, Q_2 and r_1 as in Lemma 7 and observe that $S' := S \setminus \{r_1\}$ covers all of Q_2 using anchor-points in $P' := P \setminus \{a\}$. Apply induction to S', P', Q_2. If (i) holds for them, then P' has a point on a corner of Q_2, which by $a, a' \notin P'$ is also a corner of Q and we are done. If (ii) holds for them, then two points in $P' \subset P$ have the same y-coordinate and we are done. Finally (iii) cannot hold for S', P', Q_2 because the top side of Q_2 has no point of P' on it since $a' \notin P$. □

We are finally ready to prove Theorem 2. Let S be the optimum solution with the minimum number of rectangles. If S covers all of Q, then by Lemma 10 one of (i-iii) holds. If (i) holds, then the corner in P will be chosen under rule (1). (In these and all other cases, "chosen" means "after a suitable rotation and/or reflection".) If (ii) or (iii) holds then the two points with the coinciding coordinate will be chosen under rule (2b).

If S has holes, then by Lemma 7 its unique hole H is a cell such that all interior corners of H have incoming edges on H. Let $p_1, \ldots, p_i$ be the points that introduce interior sides of H. We know that H has one of the types shown in Figure 7, and $p_1, \ldots, p_i$ hence will be considered under the corresponding rule. Moreover, all point sets that fit the type can be realized by Lemma 8. So H must be the one that minimizes the area, which corresponds to the points minimizing the x-distance resp. y-distance. So one of rules 1, 2a, 2b, 3 or 4 applies to the points $p_1, \ldots, p_i$ and Theorem 2 holds.

5 Conclusion

In this paper, we considered a variant of the anchored rectangle packing in which all points are on the boundary of the square Q. By exploiting the properties of an optimal solution, we gave an optimal linear-time exact algorithm for this problem. Observe that our algorithm covers nearly everything for large n (contrasting with the fraction of $7/12-\varepsilon$ achieved in the non-boundary case [1]). For there are (up to rotation) at least $n/2$ points in $R_B \cup P_T$, which define $n/2+1$ vertical slabs. Rule (1) or (2b) will consider the narrowest of them as hole, which has area at most $1/(n/2+1)$ if Q has area 1. So we cover a fraction of $1 - O(\frac{1}{n})$ of Q.

The most interesting open question is the status of arbitrary (non-boundary) anchored-rectangle packing. Is this polynomial-time solvable? As a first step, it would be interesting to characterize which polygonal curves on $Q \cup \Lambda$ could be boundaries of a hole in a solution.

References

[1] K. Balas, A. Dumitrescu, and C. D. Tóth. Anchored rectangle and square packings. In *SoCG 2016, Boston, MA, USA*, pages 13:1–13:16, 2016.

[2] K. Balas and C. D. Tóth. On the number of anchored rectangle packings for a planar point set. *Theor. Comput. Sci.*, 654:143–154, 2016.

[3] A. Dumitrescu and C. D. Tóth. Packing anchored rectangles. *Combinatorica*, 35(1):39–61, 2015.

[4] K. G. Kakoulis and I. G. Tollis. Labeling algorithms. In R. Tamassia, editor, *Handbook on Graph Drawing and Visualization.*, pages 489–515. Chapman and Hall/CRC, 2013.

[5] J. M. Keil, J. S. B. Mitchell, D. Pradhan, and M. Vatshelle. An algorithm for the maximum weight independent set problem on outerstring graphs. *Comput. Geom.*, 60:19–25, 2017.

[6] W. Tutte. Recent progress in combinatorics. In *proc. of the 3rd Waterloo Conference on Combinatorics*, 1968.

[7] M. J. van Kreveld, T. Strijk, and A. Wolff. Point labeling with sliding labels. *Comput. Geom.*, 13(1):21–47, 1999.

Dominating Set of Rectangles Intersecting a Straight Line

Supantha Pandit*

Abstract

We study the dominating set problem using axis-parallel rectangles and unit squares on the plane. These geometric objects are constrained to be intersected by a straight line which makes an angle with the x-axis. For axis-parallel rectangles, we prove that this problem is NP-complete. When the objects are axis-parallel unit square, we give a polynomial time algorithm. For unit squares which touch the straight line at a single point from either side of the straight line, we give an $O(n \log n)$ time algorithm.

Keywords: Dominating set, Straight line, Rectangles, Squares, NP-complete, Inclined line, Diagonal line, Touching a line, Intersecting a line.

1 Introduction

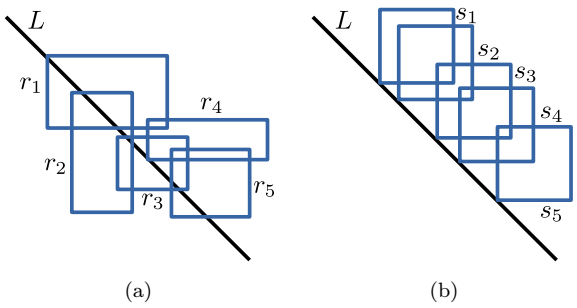
Dominating Set (DS) problem is a fundamental problem and has applications in diverse setting. This problem is defined as follows. Given a set $\mathcal{O}$ of objects, the objective is to find a subset $\mathcal{O}' \subseteq \mathcal{O}$ of objects such that every object in $\mathcal{O}$ is either in $\mathcal{O}'$ or has a non-empty intersection with an object in $\mathcal{O}'$. This problem is known to be NP-complete even with simple geometric objects like squares, disks, etc. There are many applications where minimum dominating set plays a crucial role, one of them being network routing [17]. In this work, we are interested in a special case of the *DS* problem where the given input objects are forced to intersect a given line which makes an angle with the x-axis. We define this problem formally as follows.

Dominating Set Problem with Objects Intersecting a Straight Line: Given a set of objects $\mathcal{O}$ and a straight line L such that the objects are intersecting the line L. The objective is to find a minimum cardinality subset $\mathcal{O}' \subseteq \mathcal{O}$ of objects such that any object in $\mathcal{O}$ is either belongs to $\mathcal{O}'$ or it has a non-empty intersection with an object in $\mathcal{O}'$.

*Indian Statistical Institute, Kolkata, India.
This work is partially supported by Grant No. PDF/2016/002490 for the National Post-doctoral Fellowship of the Science & Engineering Research Board, Department of Science and Technology, Government of India.
pantha.pandit@gmail.com

We are looking at this problem when the objects are axis-parallel rectangles and unit squares in the plane. Further, we assume that the rectangles or squares are either ***intersecting*** or ***touching*** L. Here, we assume that L makes an angle 135° with the x-axis. A set of rectangles is intersecting L if all the rectangles have a non-empty intersection with L (see Figure 1(a)). A set of rectangles is touching L if all the rectangles intersect L only at a corner point and this rectangles lie on the same side of L (see Figure 1(b)). Similarly, we define this two types of intersections for unit squares. In this paper, we consider the following three problems.

- **DS-Rec-IL**: Dominating set problem with rectangles intersecting a straight line.
- **DS-Sq-IL**: Dominating set problem with unit squares intersecting a straight line.
- **DS-Sq-TL**: Dominating set problem with unit squares touching a straight line.



(a) (b)

Figure 1: (a) A set of rectangles intersecting a straight line. (b) A set of unit squares touching a straight line.

1.1 Previous Work

The minimum dominating set problem is NP-complete for general graphs [6]. Further, it is $(1-\epsilon)\log n$ hard to approximate this problem for any $\epsilon > 0$ under standard complexity theoretic assumptions [19, 5, 2, 11]. There exists a greedy algorithm which produces an $O(\log n)$ approximation [20] for this problem. Dominating set problem with different classes of graphs like unit disk graphs, growth bounded graphs [10, 17] are

also studied in the literature. For graphs with polynomially bounded expansion, Har-Peled and Quanrud [9] designed a PTAS using local search algorithm. Gibson and Pirwani [8] designed a PTAS for arbitrary disks. For the intersection graphs of axis-parallel rectangles, ellipses, α-fat objects of constant description complexity, and of convex polygons with r-corners ($r \geq 4$), Erlebach and van Leeuwen [4] proved that the dominating set problem is APX-hard. This implies that, there is no PTAS for these problems unless P=NP. In [14], Marx proved that the problem is W[1]-hard for unit squares, which implies that no efficient-polynomial-time-approximation-scheme (EPTAS) is possible unless FPT = W[1] [15]. Erlebach and van Leeuwen [4] gave a $O(k)$, where $k > 0$, factor approximation factor for homothetic $2k$-regular polygons. They also provided an $O(k^2)$ factor approximation result for homothetic $(2k+1)$-regular polygons. For the homothetic convex polygons where each polygons has k-corners, the best known result is $O(k^4)$-approximation.

Chepoi and Felsner [1] considered the independent set and piercing set problems with rectangles where the rectangles are intersecting an axis-monotone curve. Recently, Correa et al. [3] studied the same problem, however instead of axes-monotone curve they considered a diagonal line. In [16] Their results were extended. Further, in [16], the authors considered the set cover and hitting set problems with other geometric objects as well.

1.2 Our Contributions

We list our contributions as follows.

- DS-Rec-IL problem is NP-complete. (Section 2)
- DS-Sq-IL problem can be solved in polynomial time. (Section 3)
- DS-Sq-TL problem can be solved in $O(n \log n)$ time. (Section 4)

1.3 Prerequisites

In this section, we provide some definitions and prerequisites that are used in the subsequent sections. We define **3-SAT** problem as follows. Given a 3-CNF formula F with n variables $x_1, x_2, \ldots, x_n$ and m clauses $C_1, C_2, \ldots, C_m$, where each clause contains exactly 3 literals, the goal is to find a truth assignment to the variables such that F is satisfied. This problem is known to be NP-complete [7]. We now embed the 3-CNF formula F in the plane as follows. For each variable or clause take a vertex in the plane. A literal is present in a clause iff their is an edge from the corresponding variable to that clause. The goal is now to find a satisfying assignment of F. This is **planar 3-SAT** (**P-3-SAT**) problem and Lichtenstein [13] proved that this problem is NP-complete. A further variation of P-3-SAT problem is the **rectilinear planar 3-SAT** (**R-P-3-SAT**) problem which is defined as follows. For each variable or clause we take a horizontal line segment. The variable segments are placed on a horizontal line and clause segments are connected to these variable segments either from above or below by vertical line segments called ***connections*** such that none of these line segments and connections intersect. The goal is to find a satisfying assignment of F. See Figure 2 for an instance of R-P-3-SAT problem. Knuth and Raghunathan [12] proved that R-P-3-SAT problem is NP-complete. Observe that the variable segments are ordered in the increasing x direction. Let $C_t = (x_i \vee \overline{x}_j \vee x_k)$ be a clause where x_i, x_j, x_k are in increasing order. Then we say that, x_i is the ***left*** variable, x_j is the ***middle*** variable, x_i is the ***right*** variable.

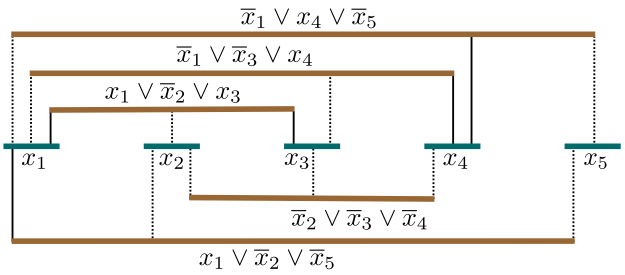


Figure 2: An instance of R-P-3-SAT problem. Solid (resp. dotted) clause vertical segments represent that the variable is positively (resp. negatively) present in the corresponding clauses.

Let us now consider the graph G given in Figure 3. The following claim can be easily proved.

Claim 1 *There are exactly two optimal dominating sets, $D_0 = \{v_4, v_8, \ldots, v_{8\tau}\}$ and $D_1 = \{v_2, v_6, \ldots, v_{8\tau-2}\}$ of vertices each with cost exactly 2τ for graph G.*

Proof. We already know that D_0 and D_1 are dominating sets. Thus the size of a minimum dominating set is atmost 2τ. In a triangle, e.g., vertices v_2, v_3, v_4, to dominate v_3 we should choose one of the vertices v_2, v_3, and v_4. Since there are 2τ such triangles and they are separated by a degree 2 vertex, the size of the minimum dominating set is at least 2τ and thus D_0 and D_1 are minimum dominating sets. Around a degree 2 vertex with non-adjacent neighbours, e.g., vertices v_4, v_5, v_6, we should choose one of the vertices v_4, v_5, and v_6. This means that we cannot choose any degree 2 vertex in a

minimum dominating set, and D_0 and D_1 are the only minimum dominating sets. □

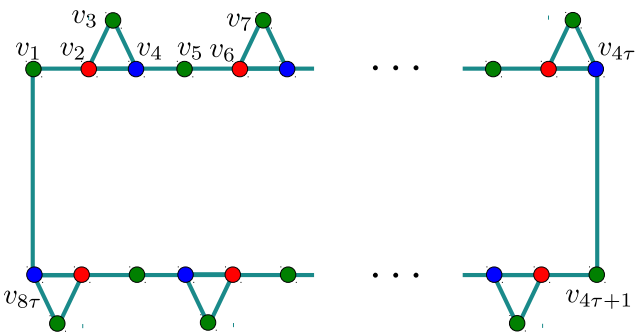


Figure 3: The graph G.

2 Intersecting Rectangles

In this section, we prove that DS-Rec-IL problem is NP-complete by giving a reduction from the R-P-3-SAT problem. We first modify R-P-3-SAT problem as follows. Instead of placing the variables on a horizontal line, place them on a diagonal line. Modify the clause vertical connections as follows (see Figure 4). For clauses which connect to the variables from above, remove the clause vertical connection for its left variable and directly connect the clause horizontal segments to the corresponding variables. To distinguish between the negative and positive connections we assume that at the meeting point of this clause horizontal segment and variable segment there is a spare vertical connections. Similar construction can be done for clauses which connect to the variables from below. Now we describe the reduction as follows.

Reduction: Given an R-P-3-SAT instance, we denote α to be the maximum number of clause vertical segments that connect to a single variable segment via connections either from above or below. For each variable x_i, we take 8α rectangles (4 rectangles are considered for each clause vertical connection) $R_i = \{r_1^i, r_2^i, \ldots, r_{8\alpha}^i\}$ as shown in Figure 5. The 4α rectangles $\{r_1^i, r_2^i, \ldots, r_{4\alpha}^i\}$ are above and the 4α rectangles $\{r_{4\alpha+1}^i, r_{4\alpha+2}^i, \ldots, r_{8\alpha}^i\}$ are below the line L. Note that, here we encode the graph in Figure 3 as a variable gadget of DS-Rec-IL with $\tau = \alpha$ where vertices represent the rectangles and there is an edge between two vertices if the two rectangles corresponding to these two vertices intersect. Therefore, by Claim 1 we conclude that for each variable gadget there are exactly two optimal dominating set of rectangles $R_i^1 = \{r_2^i, r_6^i, \ldots, r_{8\alpha-2}^i\}$ and $R_i^0 = \{r_4^i, r_8^i, \ldots, r_{8\alpha}^i\}$ each with cost 2α. The rest of

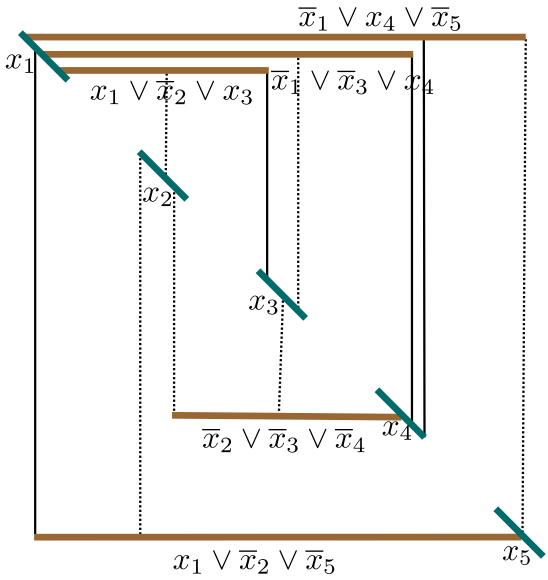


Figure 4: Modified R-P-3-SAT problem instance of the R-P-3-SAT problem instance in Figure 2.

the construction for the clauses connecting to the variables from above is similar for clauses connecting to the variables from below. Therefore, here we only describe the construction for clauses connecting to the variables from above.

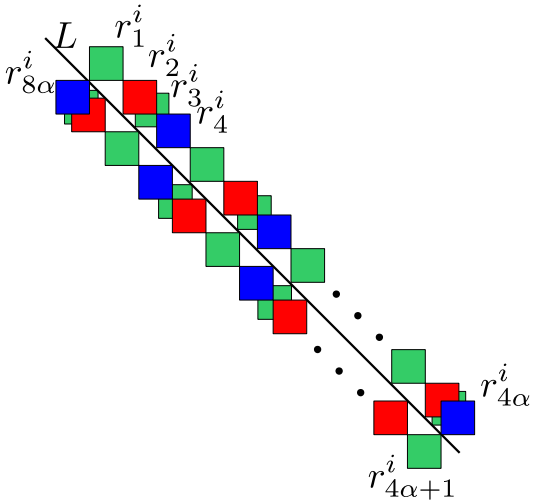


Figure 5: Structure of a variable gadget.

For each clause C_t, we take a thin rectangles r^t (see Figure 6). The bottom boundary of r^t are on the horizontal segment of C_t. We now describe how the rectangle r^t interact with the variable rectangles.

For each variable x_i, $1 \leq i \leq n$, sort the vertical connections from left to right which connect to x_i from clauses

connecting from above. Let clause C_t connects to x_i through the l_j-th connection, then we say that C_t is the l_j-th clause for variable x_i.

Let us assume that, C_t contains three variables x_i, x_j, and x_k in this order.

- Here x_i is a left variable in the clause C_t and C_t is the l_1-th clause for x_i. If x_i occurs as a positive literal in C_t, then r^t will intersect with the rectangle $r^i_{4l_1+4}$ only. Otherwise, r^t will intersect with the rectangle $r^i_{4l_1+2}$ only.
- Here x_j is a middle variable in the clause C_t and C_t is the l_2-th clause for x_j. If x_j occurs as a positive literal in C_t, then we extend the rectangle $r^j_{4l_2+4}$ upward such that it will intersect with the rectangle r^t. Otherwise, extend the rectangle $r^j_{4l_2+2}$ upward.
- Here x_k is a right variable in the clause C_t and C_t is the l_3-th clause for x_k. If x_k occurs as a positive literal in C_t, then we extend the rectangle $r^k_{4l_3+4}$ upward such that it will intersect with the rectangle r^t. Otherwise, extend the rectangle $r^k_{4l_3+2}$ upward.

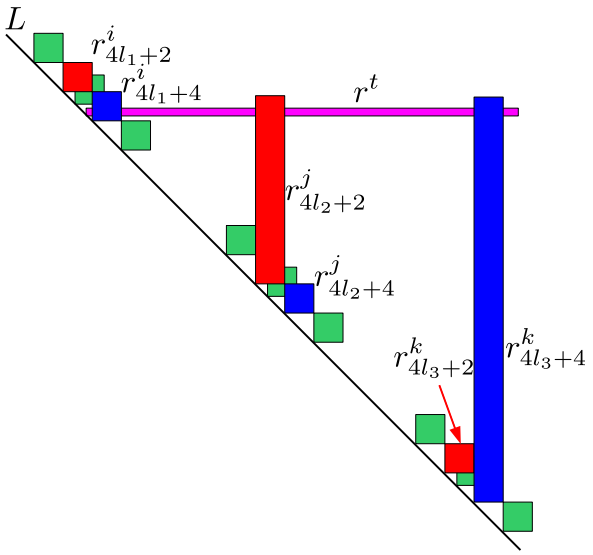


Figure 6: Clause gadget for the clause $C_t = (x_i \vee \overline{x}_j \vee x_k)$ and connection with the variable gadgets of x_i, x_j, x_k.

See Figure 6 for the above construction. Thus, from an instance F of the R-P-3-SAT problem, we created an instance $\mathscr{D}$ of the DS-Rec-IL problem. It is observe that the number of rectangles in $\mathscr{D}$ are $8\alpha n + m$ which is polynomial with respect to the number of variables n and clauses m of the formula F. Hence, this construction can be performed in polynomial time. The correctness of the above construction is shown in following lemma.

Lemma 1 *Formula F is satisfiable iff $\mathscr{D}$ has a solution with cost at most $2\alpha n$.*

Proof. Assume that F is satisfiable and let $A : \{x_1, x_2, \ldots, x_n\} \to \{true, false\}$ be a satisfying assignment. For the i-th variable gadget, take the solution R^0_i if $A(x_i) = true$ and R^1_i if $A(x_i) = false$. We choose a total of $2\alpha n$ rectangles and these rectangles dominates all the variable and clause rectangles.

On the other hand, suppose that there is a solution to $\mathscr{D}$ with cost at most $2\alpha n$. To dominate all the rectangles in a variable gadget requires at least 2α rectangles (see Claim 1). Note that all the variable gadgets are disjoint. Therefore, from each variable gadget we must choose exactly 2α rectangles (either set R^0_i or set R^1_i). We now show that $\mathscr{D}$ contains no rectangle r^t corresponding to the clause C_t. Assume that $r^t \in \mathscr{D}$. Let C_t contains the variable x_i. From the construction described above we say that r^t dominates a single vertex from the variable gadget of x_i and to dominate the remaining vertices from this gadget at least 2α rectangles are required. We now set the variable x_i to *true* if R^0_i is chosen from its variable gadget, otherwise set it to *false*. Since the solution dominates all the clause rectangles, hence by the construction we say that each clause is satisfied by this assignment. Therefore, the above assignment is a satisfying assignment. □

Clearly, DS-Rec-IL problem is in NP. Further, from Lemma 1, we conclude the following theorem.

Theorem 2 DS-Rec-IL *problem is* NP*-complete.*

Remark 1 *We prove that* DS-Rec-IL *problem is* NP*-complete even when each of the rectangles touches the straight line L at a single point from both sides of L.*

3 Intersecting Unit Squares

In this section, we show that DS-Sq-IL problem can be solved in polynomial time using dynamic programming. Let $S = \{s_1, s_2, \cdots, s_n\}$ be a set of n axis-parallel unit squares on the plane. The squares are intersecting a straight line L. We first rotate the given input configuration to make the straight line L parallel to the x-axis. Consider a horizontal strip T of height $\sqrt{2}$ such that the line L horizontally divides T into two equal parts above and below the line L. Since the squares are intersecting the line L, the center of all the squares in S are inside the strip T. The strip T is further partitioned into ***rectangles*** of width $\sqrt{2}$ and height $\sqrt{2}$. We remove all the rectangles that do not have any intersection with the given input squares. Clearly, there are at most $2n$ such rectangles that remain after the removal, since each square can intersect at most 2 rectangles.

Let $T_1, T_2, \ldots, T_k$ be these rectangles which are ordered from left to right. Add two additional rectangles T_0 and T_{k+1} such that, (i) T_0 is to the left of T_1, (ii) T_{k+1} is to the right of T_k, and (iii) no square in S intersects either T_0 or T_{k+1}.

Let $S_i \subseteq S$ be the set of squares which intersect the rectangle T_i. Further, let $S_i^c \subseteq S_i$ be the set of squares whose centers are inside T_i and $S_i^{nc} \subseteq S_i$ be the set of squares whose centers are outside T_i (see Figure 6). Clearly, $S_i^{nc} = S_i \setminus S_i^c$. More precisely, $S_i^{nc} \subseteq S_{i-1}^c \cup S_{i+1}^c$. We now prove the following result.

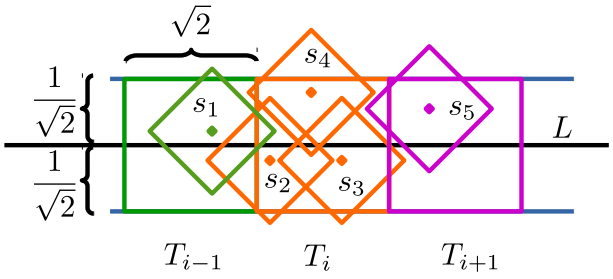


Figure 7: $S_i^c = \{s_2, s_3, s_4\}$ and $S_i^{nc} = \{s_1, s_5\}$.

Lemma 3 *The size of the optimal dominating set of squares for S_i is at most* 12.

Proof. We first prove that, at most 4 unit squares are sufficient to dominate all the squares in S_i^c. Observe that, both the width and height of the rectangle T_i are $\sqrt{2}$. Take 4 congruent squares $T_i^1, T_i^2, T_i^3, T_i^4$ such that each T_i^j, for $1 \leq j \leq 4$, is of length $\frac{1}{\sqrt{2}}$. If we arrange these 4 squares such that exactly 2 squares are in a column and exactly 2 squares are in a row, then their union fully cover the rectangle T_i. The center c_i^j of T_i^j is at most $\frac{1}{2}$ unit far from any other point inside T_i^j and hence at most one square with center inside T_i^j will dominate all the squares whose centers are inside T_i^j. Thus, any dominating set for the squares in S_i^c has size at most 4.

Observe that, the centers of the squares in S_i whose centers are outside T_i must belongs to T_{i-1} and T_{i+1}. This implies that, $S_i^{nc} \subseteq S_{i-1}^c \cup S_{i+1}^c$. Therefore, by the above argument we say that, the squares in S_i can be dominated by at most 12 squares, 4 squares each from rectangles T_{i-1}, T_i, and T_{i+1}. Since the squares whose center are inside T_i can only dominate a subset of squares in S_i, if an optimal solution OPT contains more than 12 square whose center are inside the rectangle T_i, we can replace them by 12 squares whose centers are in rectangles T_{i-1}, T_i, T_{i+1} without leaving any square to be dominated. This contradicts the assumption that OPT was an optimal dominating set. □

For $0 \leq i \leq k+1$, let $D(S_i', S_{i-1}')$ where $S_i' \subseteq S_i$ and $S_{i-1}' \subseteq S_{i-1}$ denote the size of an optimal dominating set δ for the squares which lie completely inside $\cup_{j=0}^{i} T_j$ such that $\delta \cap S_i = S_i'$ and $\delta \cap S_{i-1} = S_{i-1}'$. Note that by Lemma 3, we can assume that both S_i' and S_{i-1}' have at most 24 squares. $D(S_i', S_{i-1}')$ satisfies the following recurrence:

- If $S_i' \cup S_{i-1}'$ does not dominate all squares which lie completely inside $T_i \cup T_{i-1}$, then $D(S_i', S_{i-1}') = \infty$.
- Otherwise,

$$D(S_i', S_{i-1}') = \min_{\substack{S_{i-2}' \subseteq S_{i-2}, \\ |S_{i-2}'| \leq 12}} D(S_{i-1}', S_{i-2}') + |S_i'|$$

We calculate the minimum dominating set by evaluating the function $D(S_{k+1}', S_k')$.

Running Time: We now calculate the time required to compute the optimal dominating set. There are at most $O(n^{24})$ subproblems and each subproblem depends on $O(n^{12})$ smaller subproblems. Hence, the total time required is $n^{O(1)}$.

Therefore, we have the following theorem.

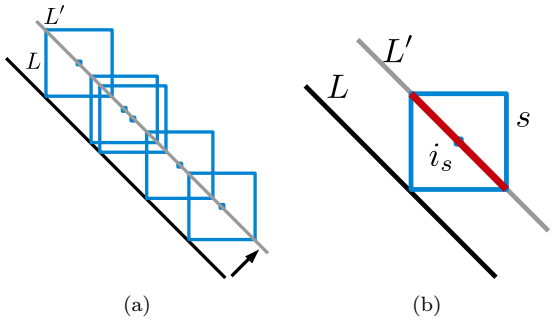
Theorem 4 DS-Sq-IL *Problem can be solved in polynomial time.*

4 Touching Unit Squares

In this section, we prove that DS-Sq-TL problem can be solved in $O(n \log n)$ time. We reduce this problem to the minimum dominating set problem with uniform intervals (all intervals have same length) on real line. Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of axis-parallel unit squares. The squares touches a straight line L from above (see Figure 1(b)). Observe that, all the centers of the squares are on a straight line parallel to the line L. We move the line L to a position L' in the orthogonal direction of L until it passes through all the centers of all the squares in S (see Figure 8(a)).

We create an instance I of the minimum dominating set problem with uniform intervals on real line from an instance of DS-Sq-TL problem as follows. Let $s \in S$ be a square touching the line L from above. We take an interval $i_s \in I$ as the intersection of the square s and the line L' (see Figure 8(b)). It is easy to observe that, two square s_1 and s_2 intersect if and only if the corresponding two intervals i_{s_1} and i_{s_2} of s_1 and s_2 respectively intersect.

We now solve the minimum dominating set problem on I. Let $\{i_{s_1}, i_{s_2}, \ldots, i_{s_k}\}$ be the set of intervals returned by the algorithm. We return the squares $\{s_1, s_2, \ldots, s_k\}$



(a) (b)

Figure 8: (a) Moving straight line L to L'. (b) A square s and its corresponding interval i_s.

as a solution of the DS-Sq-TL problem. The time required to solve the minimum dominating set problem is $O(n \log n)$ (greedy algorithm is enough, however one can look at [18]). Hence, we have the following theorem.

Theorem 5 *The* DS-Sq-TL *problem can be solved in* $O(n \log n)$ *time.*

References

[1] V. Chepoi and S. Felsner. Approximating hitting sets of axis-parallel rectangles intersecting a monotone curve. *Computational Geometry*, 46(9):1036 – 1041, 2013.

[2] M. Chlebík and J. Chlebíková. Approximation hardness of dominating set problems in bounded degree graphs. *Information and Computation*, 206(11):1264 – 1275, 2008.

[3] J. Correa, L. Feuilloley, P. Pérez-Lantero, and J. A. Soto. Independent and hitting sets of rectangles intersecting a diagonal line: Algorithms and complexity. *Discrete & Computational Geometry*, 53(2):344–365, 2015.

[4] T. Erlebach and E. J. Van Leeuwen. Domination in geometric intersection graphs. In *LATIN*, pages 747–758, 2008.

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

[6] M. R. Garey and D. S. Johnson. The rectilinear steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32(4):826–834, 1977.

[7] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1990.

[8] M. Gibson and I. A. Pirwani. Algorithms for dominating set in disk graphs: Breaking the $\log n$ barrier - (extended abstract). In *ESA*, pages 243–254, 2010.

[9] S. Har-Peled and K. Quanrud. Approximation algorithms for polynomial-expansion and low-density graphs. In *ESA*, pages 717–728, 2015.

[10] H. B. Hunt, M. V. Marathe, V. Radhakrishnan, S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. *Journal of Algorithms*, 26(2):238 – 274, 1998.

[11] V. Kann. *On the approximability of NP-complete optimization problems*. PhD thesis, 1992.

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

[13] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

[14] D. Marx. Parameterized complexity of independence and domination on geometric graphs. In *IWPEC*, pages 154–165, 2006.

[15] D. Marx. Parameterized complexity and approximation algorithms. *Comput. J.*, 51(1):60–78, 2008.

[16] A. Mudgal and S. Pandit. Covering, hitting, piercing and packing rectangles intersecting an inclined line. In *COCOA*, pages 126–137, 2015.

[17] T. Nieberg, J. Hurink, and W. Kern. Approximation schemes for wireless networks. *ACM Trans. Algorithms*, 4(4):49:1–49:17, 2008.

[18] G. Ramalingam and C. Rangan. A unified approach to domination problems on interval graphs. *Information Processing Letters*, 27(5):271 – 274, 1988.

[19] E. J. van Leeuwen. *Optimization and Approximation on Systems of Geometric Objects*. PhD thesis, 2009.

[20] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, Inc., 2001.

On Guarding Orthogonal Polygons with Bounded Treewidth

Therese Biedl and Saeed Mehrabi*

Abstract

There exist many variants of guarding an orthogonal polygon in an orthogonal fashion: sometimes a guard can see an entire rectangle, or along a staircase, or along an orthogonal path with at most k bends. In this paper, we study all these guarding models in the special case of orthogonal polygons that have bounded treewidth in some sense. Exploiting algorithms for graphs of bounded treewidth, we show that the problem of finding the minimum number of guards in these models becomes linear-time solvable in polygons of bounded treewidth.

1 Introduction

In this paper, we study orthogonal variants of the well-known art gallery problem. In the standard art gallery problem, we are given a polygon P and we want to guard P with the minimum number of point guards, where a guard g sees a point p if the line segment $\overline{gp}$ lies entirely inside P. This problem was introduced by Klee in 1973 [15] and has received much attention since. $\lfloor n/3 \rfloor$ guards are always sufficient and sometimes necessary [5], minimizing the number of guards is NP-hard on arbitrary polygons [13], orthogonal polygons [16], and even on simple monotone polygons [12]. The problem is APX-hard on simple polygons [9] and several approximation algorithms have been developed [11, 12].

Since the problem is hard, attention has focused on restricting the type of guards, their visibility or the shape of the polygon. In this paper, we consider several models of "orthogonal visibility", and study orthogonal polygons that have bounded treewidth in some sense. Treewidth (defined in Section 2.1) is normally a parameter of a graph, but we can define it for a polygon P as follows. Obtain the *standard pixelation* of P by extending a horizontal and a vertical ray inwards at every reflex vertex until it hits the boundary of P (see Figure 1). We can interpret this subdivision into rectangles as a planar straight-line graph by placing a vertex at any place incident to at least two segments, and define the treewidth of a polygon P to be the treewidth of the graph of the standard pixelation.

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

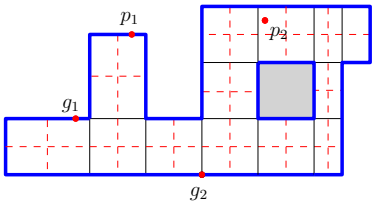


Figure 1: A polygon P with its standard pixelation (black, solid) and its 1-refinement (red, dashed). The gray area indicates a hole.

Motivation. One previously studied special case of the art gallery problem concerns *thin* polygons, defined to be orthogonal polygons for which every vertex of the standard pixelation lies on the boundary of the polygon. Thus a polygon is simple and thin if and only if the standard pixelation is an outer-planar graph. Tomás [17] showed that the (non-orthogonal) art gallery problem is NP-hard even for simple thin polygons if guards must be at vertices of the polygon. Naturally one wonders whether this NP-hardness can be transferred to orthogonal guarding models. This is not true, for example r-guarding (defined below) is polynomial on polygons whose standard pixelation is outer-planar, because it is polynomial on *any* simple polygon [18]. But what can be said about polygons that are "close" to being thin? Since outer-planar graphs have treewidth 2, this motivates the question of polygons where the standard pixelation has bounded treewidth.

The goal of this paper is to solve orthogonal guarding problems for polygons of bounded treewidth. There are many variants of what "orthogonal guarding" might mean; we list below the ones considered in this paper:

- Rectangle-guarding (r-guarding). A point guard g *r-guards* a point p if the minimum axis-aligned rectangle containing g and p is a subset of P.
- Staircase-guarding (s-guarding). A point guard g *s-guards* any point p that can be reached from g by a *staircase*, i.e., an orthogonal path inside P that is both x-monotone and y-monotone.
- Periscope-guarding. A *periscope guard* g can see all points p in which some orthogonal path inside P connects g to p and has at most one bend.

  A natural generalization of periscope-guards are *k-periscope guards* in which a point guard g can see

all points p that are connected via an orthogonal path inside P with at most k bends. (In contrast to s-guards, monotonicity of the path is not required.)

Another variant is to consider length rather than number of bends. Thus, an L_1-*distance guard* g (for some fixed distance-bound D) can see all points p for which some orthogonal path from g to p inside P has length at most D[1].

- Sliding cameras. Recently there has been much interest in *mobile guards*, where a guard can walk along a line segment inside polygon P, and can see all points that it can see from some point along the line segment. In an orthogonal setting, this type of guards becomes a *sliding camera*, i.e., an axis-aligned line segment s inside P that can see a point p if the perpendicular from p onto s lies inside P.

Related results. In the full version of the paper, we list (numerous) existing results about r-guarding, s-guarding, periscope guarding and sliding cameras. In a nutshell, most of these are NP-hard [4, 8, 10], and some of them can be solved in polynomial time if P has no holes [14, 18]. Of special relevance to the current paper is that r-guarding and guarding with sliding cameras can be solved in linear time for polygons with bounded treewidth [3, 4].

Our results. The main goal of this paper is to solve the s-guarding problem in polygons of bounded treewidth. The method used in [3, 4] does not work for this since s-guards can see along an infinite number of bends. Instead, we develop an entirely different approach. Note that the above guarding-models (except r-guarding) are defined as "there exists an orthogonal path from g to p that satisfies some property". One can argue (see Lemma 1) that we may assume the path to run along edges of the standard pixelation. The guarding problem then becomes the problem of reachability in a directed graph derived from the standard pixelation. This problem is polynomial in graphs of bounded treewidth, and we hence can solve the guarding problem for s-guards, k-periscope-guards, sliding cameras, and a special case of L_1-distance-guards, presuming the polygon has bounded treewidth.

One crucial ingredient (similarly used in [3, 4]) is that we can usually reduce the (infinite) set of possible guards to a finite set of "candidate guards", and the (infinite) set of points that need to be guarded to a finite set of "watch points" while maintaining an equivalent problem. This is not trivial (and in fact, false for some guarding-types), and may be of independent interest since it does not require the polygon to have bounded treewidth. We discuss this in Section 2.

To explain the construction for s-guarding, we first solve (in Section 3.1) a subproblem in which an s-guard can only see along a staircase in north-eastern direction. We then combine four of the obtained constructions to solve s-guarding (Section 3.2). In Section 4, we modify the construction to solve several other orthogonal guarding variants. We conclude in Section 5.

2 Preliminaries

Throughout the paper, let P denote an orthogonal polygon (possibly with holes) with n vertices. We already defined α-guards (for $\alpha = r, s$, periscope, etc.). The α-*guarding problem* consists of finding the minimum set of α-guards that can see all points in P. We solve a more general problem that allows to restrict the set of guards and points to be guarded. Thus, the (Γ, X)-α-*guarding problem*, for some (possibly infinite) sets $\Gamma \subseteq P$ and $X \subseteq P$, consists of finding a minimum subset S of Γ such that all points in X are α-guarded by some point in S, or reporting that no such set exists. Note that with this, we can for example restrict guards to be only at polygon-vertices or at the polygon-boundary, if so desired. The standard α-guarding problem is the same as the (P, P)-α-guarding problem.

Recall that the *standard pixelation* of P is obtained by extending a horizontal and a vertical ray inwards from any reflex vertex until they hit the boundary. For the rest of this paper, we refer to the standard pixelation simply as the pixelation of P. The *1-refinement* of the pixelation of P is the result of partitioning every pixel into four equal-sized rectangles. See Figure 1.

The pixelation of P can be seen as planar straight-line graph, with vertices at pixel-corners and edges along pixel-sides. For ease of notation we do not distinguish between the geometric construct (pixel/pixel-corner/pixel-side) and its equivalent in the graph (face/vertex/edge). To solve guarding problems, it usually suffices to study this graph due to the following lemma whose proof is given in the full version of the paper:

Lemma 1 *Let P be a polygon with the pixelation Ψ. Let π be an orthogonal path inside P that connects two vertices g, p of Ψ. Then there exists a path π' from g to p along edges of Ψ that satisfies*

- *π' is monotone if π was,*
- *π' has no more bends than π,*
- *π' is no longer than π.*

2.1 Tree decompositions

A *tree decomposition* of a graph G is a tree I and an assignment $\mathcal{X} : I \to 2^{V(G)}$ of *bags* to the nodes of I

[1] We use "L_1" to emphasize that this path *must* be orthogonal; the concept would make sense for non-orthogonal paths but we do not have any results for them.

such that (i) for any vertex v of G, the bags containing v form a connected subtree of I and (ii) for any edge (v, w) of G, some bag contains both v and w. The width of such a decomposition is $\max_{X \in \mathcal{X}} |X| - 1$, and the *treewidth* $tw(G)$ of G is the minimum width over all tree decompositions of G.

We aim to prove results for polygons where the pixelation has bounded treewidth. Because we sometimes use the 1-refinement of P instead, we need the following observation, which holds since every pixel-corner in a bag can be replaced by the 9 pixel-corners of the 1-refinement that it shares a pixel with.

Observation 2.1 *Let P be a polygon with the pixelation Ψ of treewidth t. Then the 1-refinement of Ψ has treewidth $O(t)$.*

The pixelation of an n-vertex polygon may well have $\Omega(n^2)$ vertices in general, but not for polygons of bounded treewidth. The following lemma is proved in the full version of the paper.

Lemma 2 *Let P be a polygon with n vertices and treewidth t. Then, the pixelation Ψ of P has $O(3^t n)$ vertices.*

The 1-refinement has asymptotically the same number of vertices as the pixelation, hence it also has $O(3^t n)$ vertices.

2.2 Reducing the problem size

In the standard guarding problem, guards can be at an infinite number of points inside P, and we must guard the infinite number of all points inside P. To reduce the guarding problem to a graph problem, we must argue that it suffices to consider a finite set of guards (we call them *candidate guards*) and to check that a finite set of points is guarded (we call them *watch points*). Such reductions are known for r-guarding [4] and sliding cameras [3]. Rather than re-proving it for each guarding type individually, we give here a general condition under which such a reduction is possible.

We need some notation. First, all our guarding models (with the exception of sliding cameras) use *point guards*, i.e., guards are points that belong to P. Also, all guarding models are *symmetric*, i.e., point g guards point p if and only if p guards g. We say that two guarding problems (Γ, X) and (Γ', X') are *equivalent* if given the solution of one of them, we can obtain the solution of the other one in linear time. We prove the following lemma in the full version due to space constraints.

Lemma 3 *Let P be an orthogonal polygon with the pixelation Ψ. Consider a guarding-model α that uses point guards, is symmetric, and satisfies the following:*

(a) For any pixel ψ and any point $g \in P$, if g α-guards one point p in the interior of ψ, then it α-guards all points in ψ.

(b) For any edge e of a pixel and any point $g \in P$, if g α-guards one point p in the interior of e, then it α-guards all points on e.

Then for any (possibly infinite) sets $X, \Gamma \subseteq P$ there exist (finite) sets X', Γ' such that (Γ, X)-α-guarding and (Γ', X')-α-guarding are equivalent. Moreover, X' and Γ' consist of vertices of the 1-refinement of Ψ.

It is easy to see that the conditions of Lemma 3 are satisfied for r-guarding, s-guarding and k-periscope guarding (for any k). We leave the details to the reader.

3 Algorithm for (Γ, X)-s-guarding

In this section, we give a linear-time algorithm for the (Γ, X)-s-guarding problem on any orthogonal polygon P with bounded treewidth. By Lemma 3, we may assume that Γ and X consist of vertices of the 1-refinement of the pixelation. As argued earlier, the 1-refinement also has bounded treewidth. Thus, it suffices to solve the (Γ, X)-s-guarding where Γ and X are vertices of the pixelation Ψ that has bounded treewidth.

3.1 (Γ, X)-NE-Guarding

For ease of explanation, we first solve a special case where guards can look in only two of the four directions and then show how to generalize it to s-guarding. We say that a point g *NE-guards* a point p if there exists an orthogonal path π inside P from g to p that goes alternately *north* and *east*; we call π a *NE-path*. Define NW-, SE- and SW-guarding analogously.

Note that NE-guarding does not satisfy the conditions of Lemma 3 because it is not symmetric; see e.g. Figure 2, where all crosses are needed to NE-guard all circles. So, we cannot solve the NE-guarding problem in general, but we can solve (X, Γ)-NE-guarding since we already know that X and Γ are vertices of the pixelation.

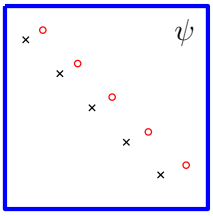


Figure 2: One pixel needs many guards.

Constructing an auxiliary graph H. Define graph H to be the graph of the pixelation of P and direct each edge of H toward north or east; see Figure 3 for an example. By assumption, $X \subseteq V(H)$ and $\Gamma \subseteq V(H)$.

By Lemma 1, there exists an NE-path from guard $g \in \Gamma$ to point $p \in X$ if and only if there exists one along the pixelation-edges. With our choice of edge-directions for H, hence there exists such a NE-path if and only if there exists a directed path from g to p in H.

Thus, (Γ, X)-NE-guarding reduces to the following problem which we call *reachability-cover*: Given a di-

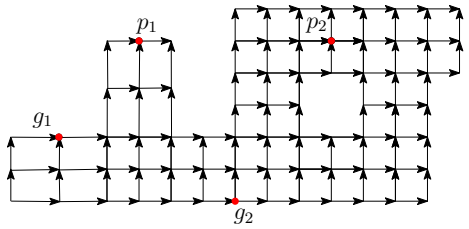


Figure 3: The graph H corresponding to NE-guarding the polygon of Figure 1. Guards and points have been shifted to pixelation-vertices.

rected graph G and vertex sets A and B, find a minimum set $S \subseteq A$ such that for any $t \in B$ there exists an $s \in S$ with a directed path from s to t. (X, Γ)-NE-guarding is equivalent to reachability-cover in graph H using $A := \Gamma$ and $B := X$.

Reachability-cover is NP-hard because set cover can easily be expressed in it. We now argue that reachability-cover can be solved in graphs of bounded treewidth, by appealing to monadic second order logic or MSOL (see [7] for an overview). Briefly, this means that the desired graph property can be expressed as a logical formula that may have quantifications, but only on variables and sets. Courcelle's theorem states that any problem expressible in MSOL can be solved in linear time on graphs of bounded treewidth [6]. (Courcelle's original result was only for decision problems, but it can easily be generalized to minimization problems.) Define $\texttt{Reachability}(u, v, G)$ to be the property that there exists a directed path from u to v in a directed graph G. This can be expressed in MSOL [7]. Consequently, the (Γ, X)-NE-guarding problem can be expressed in MSOL as follows:

$$\exists S \subseteq \Gamma : \forall p \in X : \exists g \in S : \texttt{Reachability}(g, p, H).$$

So we can solve the (Γ, X)-NE-guarding problem if Γ and X are vertices of a given pixelation that has bounded treewidth.

3.2 (Γ, X)-s-guarding

Solving the (Γ, X)-s-guarding problem now becomes very simple, by exploiting that a guard g s-guards a point p if only if g β-guards p for some $\beta \in$ {NE, NW, SE, SW}. We can solve the (Γ, X)-β-guarding problem for $\beta \neq$ NE similarly as in the previous section, by directing the auxiliary graph H according to the directions we wish to take. Let $H_{\texttt{NE}}, H_{\texttt{NW}}, H_{\texttt{SE}}, H_{\texttt{SW}}$ be the four copies of graph H (directed in four different ways) that we get. Define a new auxiliary graph H^* as follows (see also Figure 4): Initially, let $H^* := H_{\texttt{NE}} \cup H_{\texttt{NW}} \cup H_{\texttt{SE}} \cup H_{\texttt{SW}}$. For each $g \in \Gamma$, add to H^* a new vertex $v^\Gamma(g)$ and the directed edges $(v^\Gamma(g), v_\beta(g))$ where $v_\beta(g)$ (for $\beta \in$ {NE, NW, SE, SW}) is the vertex in H_β) corresponding to g. Similarly, for each $p \in X$, add to H^* a new vertex $v^X(p)$ and the directed edges $(v_\beta(p), v^X(p))$ for $\beta \in$ {NE, NW, SE, SW}).

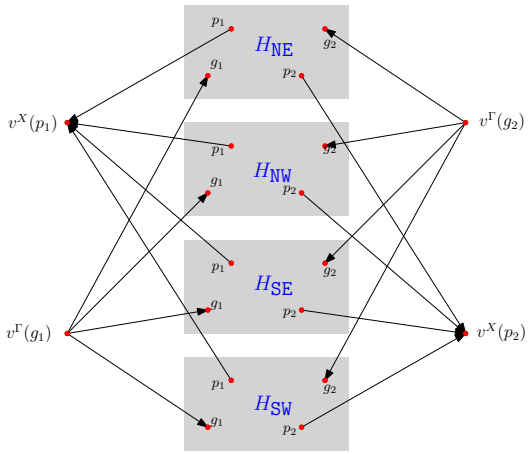


Figure 4: The construction of graph H^*.

If some guard g s-guards a point p, then there exists a β-path from g to p inside P for some $\beta \in$ {NE, NW, SE, SW}. We can turn this path into a β-path along pixelation-edges by Lemma 1, and therefore find a path from $v^\Gamma(g)$ to $v^X(p)$ by going to H_β and following the path within it. Vice versa, any directed path from $v^\Gamma(g)$ to $v^X(p)$ must stay inside H_β for some $\beta \in$ {NE, NW, SE, SW} since $v^\Gamma(g)$ is a source and $v^X(p)$ is a sink. Therefore (Γ, X)-s-guarding is the same as reachability-cover in H^* with respect to the sets $V(\Gamma) := \{v^\Gamma(g) : g \in \Gamma\}$ and $V(X) := \{v^X(p) : p \in X\}$.

It is easy to see that H^* has bounded treewidth if Ψ does, by replacing each vertex p of Ψ in a bag by its up to six copies $v_\beta(p)$, $v^X(p)$, $v^\Gamma(p)$. Now we put it all together. Assume P has bounded treewidth, hence its (standard) pixelation has bounded treewidth and $O(n)$ edges, and so does its 1-refinement. This is in fact the partition of P that we use to obtain H^*, therefore H^* also has bounded treewidth and $O(n)$ edges. We can apply Courcelle's theorem to solve reachability-cover in H^* and obtain:

Theorem 4 *Let P be an orthogonal polygon with bounded treewidth. Then, there exists a linear-time algorithm for the (Γ, X)-s-guarding problem on P.*

4 Other Guarding Types

In this section, we show how similar methods apply to other types of orthogonal guarding. The main difference is that we need edge-weights on the auxiliary graph. To solve the guarding problem, we hence use a version of reachability-cover defined as follows. The

(G, A, B, W)-*bounded-reachability-cover problem* has as input an edge-weighted directed graph G, two vertex sets A and B, and a length-bound W. The objective is to find a minimum-cardinality set $S \subseteq A$ such that for any $t \in B$ there exists an $s \in S$ with a directed path from s to t that has length at most W. We need to argue that this problem is solvable if G has bounded treewidth, at least if W is sufficiently small. Recall that reachability-cover can be expressed in monadic second-order logic. Arnborg et al. [1] introduced the class of *extended* monadic second-order problems which allow integer weights on the input. They showed that problems expressible in extended monadic second-order logic can be solved on graphs of bounded treewidth, with a run-time that is polynomial in the graph-size and the maximum weight.

4.1 L_1-distance guarding

We first study the L_1-distance guarding problem. We have not been able to solve the L_1-distance guarding problem for all polygons of bounded treewidth. The main problem is that the bounded-reachability-cover problem is solved in run-time that depends on the maximum weight. For this to be polynomial, we must assume that all edges of the input-polygon have integer length that is polynomial in n.

Let Γ and X be subsets of the vertices of the pixelation Ψ of P. Let H_{dist} be the auxiliary graph obtained from the pixelation graph by making all edges bi-directional. Set the weight of each edge to be its length. If a guard $g \in \Gamma$ sees a point $p \in X$ in the L_1-distance guarding model (with distance-bound D), then there exists a path π from g to p that has length at most D. By Lemma 1, we may assume that π runs along pixel-edges. Hence π gives rise to a directed path in H_{dist} of length less than D. Vice versa, any such path in H_{dist} means that g can L_1-distance-guard p. In consequence, the (Γ, X)-L_1-distance guarding problem is the same as the $(\Gamma, X, H_{\text{dist}}, D)$-bounded-reachability-cover problem. This can be solved in polynomial time, presuming the pixelation has bounded treewidth and $O(n)$ edges and the lengths of all edges of P are integers that are polynomial in n.

4.2 k-periscope guarding

For k-periscope guarding, we define an auxiliary graph H_{peri} based on the graph of the pixelation, but modify it near each vertex and add weights to encode the number of bends, rather than the length, of a path. If u is a vertex of the pixelation, then replace it with a K_4 as shown in Figure 5. We denote this copy of K_4 by K_4^u, and let its four vertices be u_N, u_S, u_W and u_E according to compass directions. For a vertex u on the boundary of P we omit those vertices in K_4^u that would fall outside P. We connect copies K_4^u and K_4^v of a pixel-edge (u, v) in the natural way, e.g. if (u, v) was vertical with u below v, then we connect u_N to v_S. All edges are bidirectional.

For any $g \in \Gamma$, define a new vertex $v^\Gamma(g)$ and add edges from it to all of g_N, g_S, g_E, g_W that exist in the graph. For any $p \in X$, define a new vertex $v^X(p)$ and add edges from all of p_N, p_S, p_E, p_W to $v^X(p)$. Set all edge weights to 0, except for the "diagonal" edges between consecutive vertices of a K_4, which have weight 1 as shown in Figure 5.

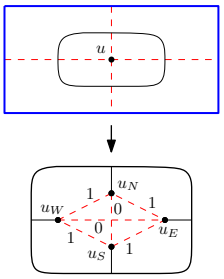


Figure 5: Adding K_4.

Clearly, $g \in \Gamma$ can see $p \in X$ (in the k-periscope guarding model) if and only if there is a directed path from $v^\Gamma(g)$ to $v^X(p)$ in the constructed graph that uses at most k diagonal edges, i.e., that has length at most k. Thus the k-periscope guarding model reduces to bounded-reachability-cover. Since k-periscope-guarding satisfies the conditions for Lemma 3, we can hence solve the k-periscope guarding problem in polynomial time in any polygon of bounded treewidth. Note that the run-time depends polynomially on k, so k need not be a constant.

4.3 Sliding cameras

It was already known that the sliding camera problem is polynomial in polygons of bounded treewidth [3]. However, using much the same auxiliary graph as in the previous subsection we can get a second (and in our opinion, easier) method of obtaining this result.

We solve the (Γ, X)-sliding camera guarding problem, for some set of sliding cameras Γ (which are segments inside P) and watch points X. It was argued in [3] that we may assume Γ to be a finite set of maximal segments that lie along the pixelation; in particular the endpoints of candidate guards are pixel-vertices. As for X, we cannot apply Lemma 3 directly, since sliding cameras are not point guards and hence not symmetric. But sliding cameras do satisfy conditions (a) and (b) of Lemma 3. As one can easily verify by following the proof, we may therefore assume X to consist of pixel-vertices of the 1-refinement. (A similar result was also argued in [3].)

We build an auxiliary graph H_{slide} almost exactly as in the previous subsection. Thus, start with the graph of the 1-refinement of the pixelation. Replace every vertex by a K_4, weighted as before. (All other edges receive weight 0.) For each $p \in X$, define a new vertex $v^X(p)$ and connect it as in the previous subsection, i.e., add edges from p_N, p_E, p_W, p_S to $v^X(p)$. For any sliding camera $\gamma \in \Gamma$, add a new vertex $v^\Gamma(\gamma)$. The only new thing is how these vertices get connected. If γ is hori-

zontal, then add an edge from $v^\Gamma(\gamma)$ to g_W, where g_W is the left endpoint of γ. If γ is vertical, then add an edge from $v^\Gamma(\gamma)$ to g_N, where g_N is the top endpoint of γ.

It is not hard to verify that a sliding camera γ can see a point p if and only if there exists a directed path from $v^\Gamma(\gamma)$ to $v^X(p)$ in H_{slide} that has length at most 1. Due to space constraints, we prove this formally in the full version of the paper. Therefore the sliding camera problem reduces to a bounded-reachability-cover problem where all weights are at most 1; this can be solved in polynomial (in fact, linear) time if the polygon has bounded treewidth.

5 Conclusion

In this paper, we gave algorithms for guarding orthogonal polygons of bounded treewidth. We considered various models of orthogonal guarding, and solved the guarding problem on such polygons for s-guards, k-periscope guards, and sliding cameras, and some other related guarding types.

As for open problems, the main question is whether these results could be used to obtain better approximation algorithms. Baker's method [2] yields a PTAS for many problems in planar graphs by splitting the graph into graphs of bounded treewidth and combining solutions suitably. However, this requires the problems to be "local" in some sense, and the guarding problems considered here are not local in that a guard may see a point whose distance in the graph of the pixelation is very far, which seems to make Baker's approach infeasible. Are these guarding problems APX-hard in polygons with holes?

References

[1] S. Arnborg, J. Lagergren, and D. Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991. 5

[2] B. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. ACM*, 41(1):153–180, 1994. 6

[3] T. Biedl, T. M. Chan, S. Lee, S. Mehrabi, F. Montecchiani, and H. Vosoughpour. On guarding orthogonal polygons with sliding cameras. In *WALCOM 2017*, volume 10167 of *LNCS*, pages 54–65. Springer, 2017. 2, 3, 5

[4] T. Biedl and S. Mehrabi. On r-guarding thin orthogonal polygons. In *ISAAC 2016*, volume 64 of *LIPIcs*, pages 17:1–17:13, 2016. 2, 3

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

[6] B. Courcelle. The monadic second-order logic of graphs. I. Recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990. 4

[7] B. Courcelle. On the expression of graph properties in some fragments of monadic second-order logic. In *DIAMCS Workshop on Descriptive Complexity and Finite Models*, pages 33–57. American Mathematical Society, 1997. 4

[8] S. Durocher and S. Mehrabi. Guarding orthogonal art galleries using sliding cameras: algorithmic and hardness results. In *MFCS 2013*, volume 8087 of *LNCS*, pages 314–324, 2013. 2

[9] S. Eidenbenz, C. Stamm, and P. Widmayer. Inapproximability results for guarding polygons and terrains. *Algorithmica*, 31(1):79–113, 2001. 1

[10] L. Gewali and S. C. Ntafos. Covering grids and orthogonal polygons with periscope guards. *Comput. Geom.*, 2:309–334, 1992. 2

[11] S. K. Ghosh. Approximation algorithms for art gallery problems in polygons. *Disc. App. Math.*, 158(6):718–722, 2010. 1

[12] E. Krohn and B. J. Nilsson. Approximate guarding of monotone and rectilinear polygons. *Algorithmica*, 66(3):564–594, 2013. 1

[13] D. T. Lee and A. K. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32(2):276–282, 1986. 1

[14] R. Motwani, A. Raghunathan, and H. Saran. Covering orthogonal polygons with star polygons: The perfect graph approach. In *SoCG 1988*, pages 211–223, 1988. 2

[15] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, New York, NY, 1987. 1

[16] D. Schuchardt and H.-D. Hecker. Two NP-hard art-gallery problems for ortho-polygons. *Mathematical Logic Quarterly*, 41(2):261–267, 1995. 1

[17] A. P. Tomás. Guarding thin orthogonal polygons is hard. In *Fundamentals of Computation Theory (FCT 2013)*, volume 8070 of *LNCS*, pages 305–316, 2013. 1

[18] C. Worman and J. M. Keil. Polygon decomposition and the orthogonal art gallery problem. *Int. J. Comput. Geometry Appl.*, 17(2):105–138, 2007. 1, 2

Beacon Coverage in Orthogonal Polyhedra

I. Aldana-Galván* J.L. Álvarez-Rebollar† J.C. Catana-Salazar* N. Marín-Nevárez*
E. Solís-Villarreal* J. Urrutia‡ C. Velarde§

Abstract

We consider a variant of the Art Gallery Problem in orthogonal polygons and orthogonal polyhedra using beacons as guards. A beacon is a device that attracts objects toward itself within a given domain. A beacon b covers a point if when a beacon attracts it, it reaches b. In this paper, we prove that there exist orthogonal polyhedra whose exterior cannot be covered even if we place a beacon at each of its vertices.

We also study the beacon coverage problem in orthogonal polyhedra, by extending the notion of vertex beacons to edge beacons. We prove that $\lfloor \frac{e}{12} \rfloor$ edge beacons are always sufficient while $\lfloor \frac{e}{21} \rfloor$ edge beacons are sometimes necessary to cover any orthogonal polyhedron. We also prove that $\lfloor \frac{e}{6} \rfloor$ edge beacons are always sufficient to cover simultaneously the interior and the exterior of any orthogonal polyhedron.

1 Introduction

A *beacon* is a fixed point in a polyhedron P that can induce a magnetic pull toward itself over all points in P. When a beacon b is activated, points in P move greedily to decrease their euclidean distance to b. A point p can move along any obstacles it hits on its way to a beacon b as long as its distance to b keeps on decreasing. Thus, the path from the initial position of p to a beacon b may alternate between moving in straight line segments contained in the interior of P and line segments on the faces of P.

The piecewise linear path created by the movement of p under the attraction of b is called the *attraction path* of p with respect to b. If the attraction path of p ends in b, we say that p is *covered* by b. If p is in a position where it is unable to move in such a way that its distance to b decreases, we say that it is 'stuck' and it has reached a local minimum, or dead end, see Figure 1.

Beacon attraction was introduced by Biro et al. [3, 4, 5]. This model extends the classical notion of visibility; if an object p is visible from a beacon q, then p moves towards q along the straight line segment joining p to q.

In this paper we consider two beacon *coverage* problems in orthogonal polygons and orthogonal polyhedra. The beacon coverage problem asks for a minimum set B of beacons placed in a domain P, in such a way that any point $p \in P$ is covered by at least one element of B. We then study the interior-exterior beacon coverage problem in which we ask for a minimum set B of beacons placed on the boundary of a domain P, in such a way that any point $p \in P$ and any point $p' \notin P$ are covered by at least one element of B.

2 Preliminaries

Let P be an orthogonal polygon on the plane. An edge e of P is a *right edge* if there is an $\varepsilon > 0$ such that any point at distance less than or equal to ε from any interior point of e and to the left of e belongs to the interior of P. *Left*, *top* and *bottom* edges are defined similarly.

A *polyhedron* in $\mathbb{R}^3$ is a compact connected set bounded by a piecewise linear 2-manifold. A *face* of a polyhedron is a maximal planar subset of its boundary whose interior is connected and non-empty. A polyhedron is *orthogonal* if all of its faces are parallel to the $\mathcal{XY}$, $\mathcal{XZ}$ or $\mathcal{YZ}$ planes. An *edge* is a minimal positive-length straight line segment shared by two faces and joining two vertices of the polyhedron. Each edge, with its two adjacent faces, determines a dihedral angle, internal to the polyhedron. In an orthogonal polyhedron each such angle is of either 90^o (at a *convex edge*) or 270^o (at a *reflex edge*).

An *$\mathcal{X}$-plane* is a plane that is perpendicular to the $\mathcal{X}$-axis; we define a *$\mathcal{Y}$-plane* and a *$\mathcal{Z}$-plane* in a similar way. An *$\mathcal{X}$-face* is a face of a polyhedron that is contained in an *$\mathcal{X}$-plane*; we define a *$\mathcal{Y}$-face* and a *$\mathcal{Z}$-face* in a similar way.

A $\mathcal{Y}$-face f of an orthogonal polyhedron P is a *left face* (*right face*), if for any interior point $q \in f$ there is an $\varepsilon > 0$ such that any point at distance less than or equal to ε from q and to the right (left) of f belongs to

*Posgrado en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México, Ciudad de México, México, ialdana@ciencias.unam.mx, {j.catanas, mnjn16, solis_e}@uxmcc2.iimas.unam.mx

†Posgrado en Ciencias Matemáticas, Universidad Nacional Autónoma de México, Ciudad de México, México, chepomich1306@gmail.com

‡Instituto de Matemáticas, Universidad Nacional Autónoma de México, Ciudad de México, México, urrutia@matem.unam.mx

§Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, Universidad Nacional Autónoma de México, Ciudad de México, México, velarde@unam.mx

the interior of P. In a similar way, $\mathcal{Z}$-faces are classified into *top* or *bottom* faces, and $\mathcal{X}$-faces, as *front* or *back* faces.

A *connected polyhedron* P is a *lifting polyhedron* if there exists a *$\mathcal{Z}$-plane* Q such that for all planes parallel to Q their intersection with P is either empty, or it is a vertical translation of the intersection of P with Q.

The next definition was given by Damian et al. in [7]. An *orthotree* P is an orthogonal polyhedron made of cuboids glued face to face, such that the dual graph of P is a tree. The intersection of two adjacent cuboids in P is a 2-dimensional face of both cuboids, namely, a non-degenerate rectangle.

In this paper we consider two models of beacon attraction, vertex beacons and edge beacons. In the *vertex beacon model*, we place point beacons on the vertices of P. In the *edge beacon* model, we place a point beacon at each point of a closed edge e of P. We call e an *edge beacon*. When an edge beacon b is activated in an orthogonal polyhedron, an object p always moves towards the point $q \in b$ closest to p. If p reaches q we say that b *covers* p. Observe that if q is not an endpoint of b, the attraction path of p to q is contained in the plane β orthogonal to b that contains q. Therefore, in this case the attraction path of p with respect to b is as in $\mathbb{R}^2$, considering q as the beacon and $P \cap \beta$ as the polygon.

Consider the connected component S of $\overline{pq} \cap P$ that contains p. If S contains other points different from p, then p continues moving to q along $\overrightarrow{pq}$. Otherwise, p hits ∂P and there are three cases: (i) If p hits a vertex v, then p gets stuck at v. (ii) If p hits a point x in the interior of an edge e, and the orthogonal projection q_e of q over the straight line that contains e is different from p, then p moves along $\overrightarrow{pq_e}$. Otherwise, p gets stuck at x. (iii) If p hits a point x in the interior of a face f, and the projection q_f of q over the plane that contains f is different from p, then p moves along $\overrightarrow{pq_f}$. Otherwise, p gets stuck at x.

Figure 1 shows two examples of points reaching local minima on their way to vertex and edge beacons.

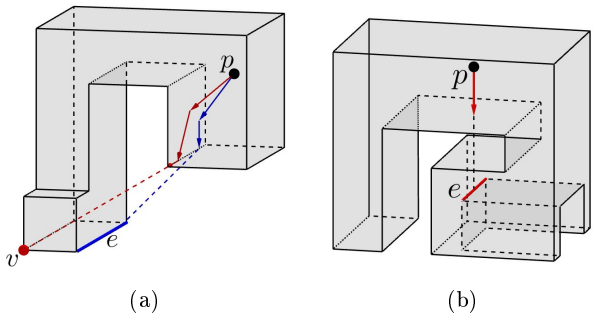


Figure 1: Two examples of points that reach a local minimum: (a) The attraction path of a point with respect to a vertex beacon and an edge beacon, both unreachable, and (b) the point gets stuck on its way to an edge beacon.

3 Covering orthogonal polygons

Bae et al. [2] proved that the interior of any orthogonal n-gon can be covered with $\lfloor \frac{n}{6} \rfloor$ vertex beacons. We consider now the problem of simultaneously covering the interior and exterior of orthogonal polygons.

Theorem 1 *Let P be an orthogonal polygon (possibly with holes) with n vertices. Then $\lfloor \frac{n}{4} \rfloor + 1$ vertex beacons are always sufficient to simultaneously cover the interior and the exterior of P.*

Proof. Suppose w.l.o.g. that there are at most $\lfloor \frac{n}{4} \rfloor$ right edges of P. Let B be the set of bottom vertices of the right edges of P. We place a beacon on each $b \in B$. We will prove that this set of beacons covers P. For each $b \in B$ consider the maximal vertical line segment s_b that contains b and is contained in P. Note that the set of line segments $S = \{s_b : b \in B\}$ divides P into histograms, such that in each histogram: There is only one right edge, and all top and bottom edges are contained in edges of P (since we only use vertical line segments to divide P), see Figure 2.

Let p be a point in a histogram H with $H \subset P$ and let b_h be one of the vertices of B that lie in the right edge of H. We will prove only the case when p is on or above b_h, the proof for the other case is symmetric. We claim that p is covered by the beacon placed in b_h. If p is on the right edge of H, we are done. Suppose p is above and to the left of b_h. Since H has only one right edge, the attraction path T of p with respect to b_h can only hit bottom edges of H. Since all the bottom edges of H are contained in edges of P, it follows that T is contained in H. If T does not finish at b_h, then it reaches a local minimum in a bottom edge of H. This local minimum has to be exactly above b_h, which is impossible because b_h is contained in the unique right edge of H. Therefore, these beacons cover the interior of P.

Now we prove that the beacons placed on the elements of B plus an extra beacon cover the exterior of P. Let R be a rectangle containing P in its interior. Let $P' = R \backslash int(P)$, where $int(P)$ denotes the interior of P. Note that the elements in B are bottom vertices of left edges of P'. As before, we can use the same technique to cover the interior of P' with beacons placed on the elements of B plus an extra beacon placed on the bottom vertex v_R of a leftmost edge of R. We will prove that this beacon can be replaced by a beacon placed on the bottom vertex v_l of the leftmost edge of P.

Let $H_R \subset P'$ be the histogram whose left edge is the left edge of R. Let p be a point contained in H_R. If p is to the left of the vertical line ℓ trough v_l then we are

done. We will prove only the case when p is to the right of ℓ, and above the horizontal line h trough v_l. The proof for the case when p is below h is symmetric.

Since H_R has only one left edge, the attraction path T of p with respect to v_l can only hit bottom edges of H_R. Since all the bottom edges of H_R are contained in edges of P except for the one that is contained in the bottom edge of R, T is contained in H_R. If T does not finish at v_l, then it reaches a local minimum in the interior of a bottom edge of H_R. This local minimum has to be exactly above v_l, which is impossible because v_l is contained in a leftmost edge of P. Therefore H_R is covered by the beacon placed at v_l. Hence the beacons in B together with v_l cover both the interior and the exterior of P. □

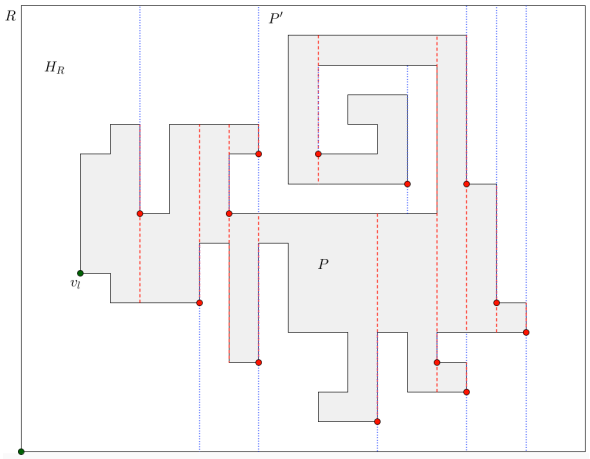


Figure 2: Regions obtained by the decomposition selecting the bottom vertices of the right edges of P.

4 Covering orthogonal polyhedra

It is known that not every polyhedron can be covered with vertex beacons [6], even if the polyhedron is orthogonal [1]. There exist well known families of orthogonal polyhedra whose interior can be covered with vertex beacons. Such is the case of the orthotrees [1]. However, there exists an orthotree polyhedron such that it is not possible to cover its exterior with vertex beacons. That is the case of the polyhedron shown in Figure 3a, that we describe next.

Our example is based on the octoplex polyhedron, proposed by T. S. Michael in [8]. The octoplex consists of a cuboid with six channels, each one of them going across a different face. It is known that the octoplex cannot be guarded with *vertex guards* (a vertex guard is a guard placed on a vertex). We take six notched beams arranged as the six channels of the octoplex. Then we join them by means of orthogonal pipes arranged properly to form an orthotree, see Figure 3a.

A point in the exterior of P that is not covered by any vertex of P is the "center" point p of the region enclosed by the beams of P. Consider the wedge (dihedral angle) whose axis is vertical and contains p. This wedge is delimited by the two edges at the end of the concavity of the notch of the red beam, see Figure 3b. Note that the beam divides the wedge into two connected regions: W_p which contains p and W'_p which does not. Following the notation of Figure 3c, the polyhedron is constructed with p at the origin so that q and v satisfy $\frac{q_y}{q_x} < \frac{v_y}{v_x}$, thus ensuring that the pq ray intersects the notch of the red beam. Therefore, any beacon placed in the interior of W'_p cannot cover p. Similarly, we construct a wedge for each beam in such a way that every vertex of P is in the interior of one of these wedges.

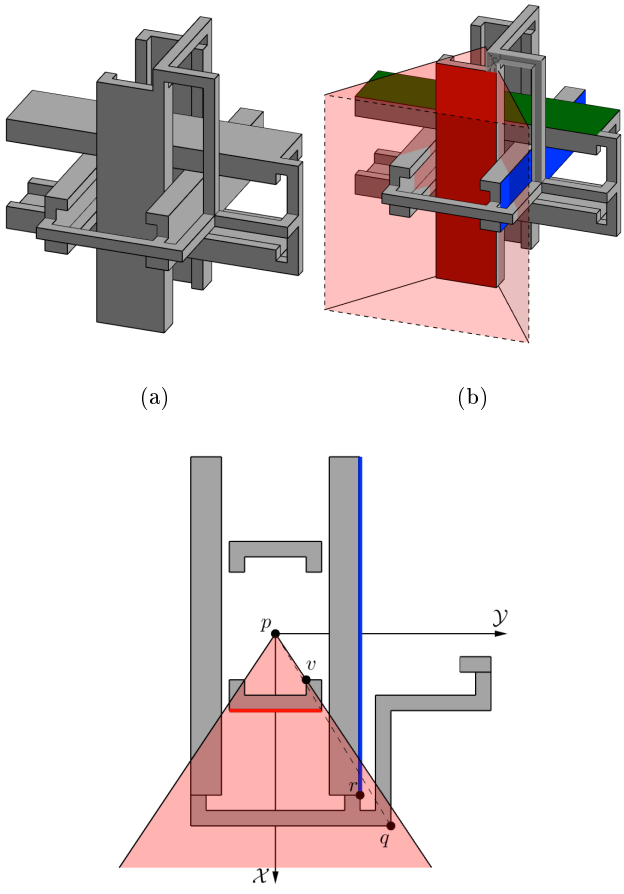


Figure 3: (a) An orthotree whose exterior cannot be covered with vertex beacons. (b) Wedge whose axis is vertical and contains the center point p. This wedge is delimited by the two edges at the end of the concavity of the notch of the red beam. (c) Orthogonal projection in the $\mathcal{XY}$ plane of some conveniently selected elements.

Since vertex beacons are not enough to cover orthogonal polyhedra, it is natural to study the edge beacon model. It is straightforward to see that if we place an edge beacon at each edge of a polyhedron P (orthogonal or not) these edge beacons always cover P.

Next, we prove that any orthogonal polyhedron with e edges can be covered with $\lfloor \frac{e}{12} \rfloor$ edge beacons and that sometimes $\lfloor \frac{e}{12} \rfloor$ edge beacons are necessary. We also prove that $\lfloor \frac{e}{6} \rfloor$ edge beacons are always sufficient to simultaneously cover the interior and exterior of any orthogonal polyhedron. In another paper we prove that any orthotree with n vertices can be guarded with at most $\lfloor \frac{n}{8} \rfloor$ vertex guards, and therefore covered using at most $\lfloor \frac{n}{8} \rfloor$ vertex beacons [1].

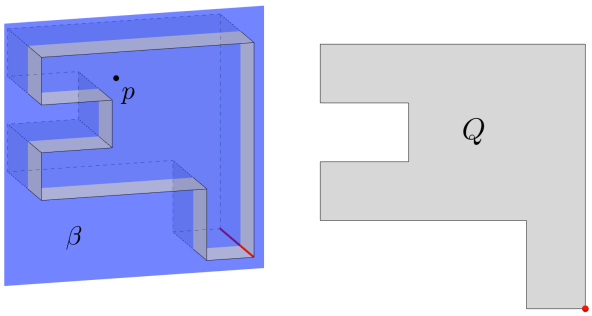
4.1 Covering orthogonal polyhedra with edge beacons

Now we define for each $F \in$ *{left, right, top, bottom, front, back}* and for each $E \in$ *{left, right, top, bottom, front, back}* the *F-E* rule. The *F-E* rule selects the E edges from the F faces of an orthogonal polyhedron P, seen from the outside. For example, the *right-bottom* rule selects the bottom edges of the right faces of P. Note that each face type contains only four different types of edges, namely, if $F =$ *front (or back)* then $E \in$ *{left, right, top, bottom}*, if $F =$ *top (or bottom)* then $E \in$ *{left, right, front, back}*, and if $F =$ *right (or left)* then $E \in$ *{front, back, top, bottom}*. Thus, a rule like the *top-bottom* rule selects no edges of P.

Lemma 2 *For every orthogonal polyhedron P there exists an* F-E *rule which selects at most $\lfloor \frac{e}{12} \rfloor$ edges from P, where e is the number of edges of P.*

Proof. Let A, B, C be the number of edges in the $\mathcal{Y}$, $\mathcal{X}$, and $\mathcal{Z}$ faces, respectively. Since $A+B+C = 2e$ one of A, B or C is at most $\lfloor \frac{2e}{3} \rfloor$. Then suppose w.l.o.g. that the set $F_{\mathcal{Y}}$ consisting of the $\mathcal{Y}$ faces has at most $\lfloor \frac{2e}{3} \rfloor$ edges of P. There are two kinds of faces in $F_{\mathcal{Y}}$: left and right. Let $R \subset F_{\mathcal{Y}}$ be the set of right faces of $F_{\mathcal{Y}}$ and let E_R be the set of edges that belong to faces of R. Suppose w.l.o.g. that $|E_R|$ is at most half of the number of edges belonging to the faces of $F_{\mathcal{Y}}$, i.e., $|E_R| \leq \lfloor \frac{e}{3} \rfloor$. There are four kinds of edges in E_R: top, bottom, front and back. Therefore one of these four types of edges has at most $\left\lfloor \frac{|E_R|}{4} \right\rfloor$ edges of P. Suppose w.l.o.g that the set of bottom edges of E_R has at most $\left\lfloor \frac{|E_R|}{4} \right\rfloor \leq \lfloor \frac{e}{12} \rfloor$ edges of P. Note that these are the edges selected by the *right-bottom* rule. In any other case the proof is analogous by selecting the appropriate *F-E* rule. □

Theorem 3 *Let P be an orthogonal polyhedron with e edges. Then $\lfloor \frac{e}{12} \rfloor$ edge beacons are always sufficient to cover P.*



(a) (b)

Figure 4: (a) The point p and the $\mathcal{X}-$plane β, (b) polygon Q.

Proof. By Lemma 2 we can suppose w.l.o.g. that the set B of edges selected by the *right-bottom* rule has at most $\lfloor \frac{e}{12} \rfloor$ edges.

We place an edge beacon on each $b \in B$. We will prove that this set of edge beacons covers P. Let p be a point in P. Let β be the $\mathcal{X}$-plane that contains p. Let Q be the connected component of $\beta \cap P$ that contains p, as shown in Figure 4. Note that Q is an orthogonal polygon, and that each bottom vertex of a right edge of Q is of the form $b \cap Q$ for some $b \in B$.

Since the attraction path of p with respect to an edge beacon remains in β, using the same reasoning as in the proof of Theorem 1, we can prove that p is covered by a beacon placed on an edge $b \in B$. □

Theorem 4 *There exists a family of orthogonal polyhedra with e edges, such that $\lfloor \frac{e}{21} \rfloor$ edge beacons are necessary to cover their interior.*

Proof. We construct a lifting polyhedron P, based on a rectangular spiral polygon consisting of a sequence of $r+1$ thin rectangles, Figure 5 shows a top view of P.

Let $e_0, e_1, \ldots, e_r$ be a set of consecutive convex edges of P that are parallel to the $\mathcal{Z}$-axis, whose orthogonal projections are shown in Figure 5. Let $e'_1, e'_2, \ldots, e'_{r-1}$ be the set of consecutive reflex edges of P that are parallel to the $\mathcal{Z}$-axis, and e'_0 and e'_r be the convex edges of P parallel to the $\mathcal{Z}$-axis that have an incident face in common with e'_1 and e'_{r-1}, respectively. From the top, they correspond to the reflex and convex vertices of the projection of P on the $\mathcal{XY}$ plane, see Figure 5.

Suppose for the sake of simplicity that $r = 7m$ for $m \in \mathbb{N}$. For each $0 \leq k < m$, we place a distinguished point p_k in the interior of P near enough the center of the face formed by the edges e'_{7k} and e'_{7k+1}, and a distinguished point p'_k in the center of the rectangle formed by the edges e'_{7k+4} and e_{7k+4}, as shown in Figure 5. Note that p'_k is in a region that is not covered by e_{7k+2}, e'_{7k+2}, e_{7k+6} neither e'_{7k+6}.

Note that there is no edge covering two distinguished points at the same time. Since there are $\frac{2r}{7}$ distinguished points and P has $e = 6(r+1)$ edges, we need at least $\lfloor \frac{e}{21} \rfloor$ edge beacons to cover P. □

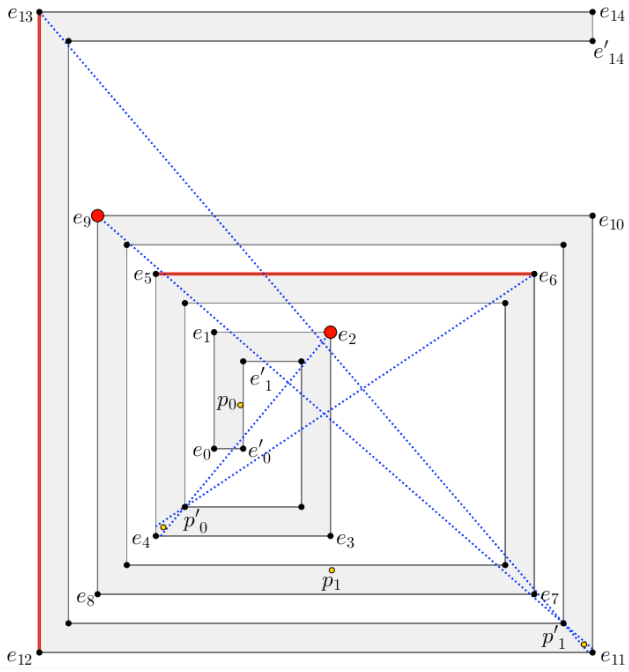


Figure 5: Orthogonal projection over the $\mathcal{XY}$ plane of a spiral polyhedron with e edges that requires $\lfloor \frac{e}{21} \rfloor$ edge beacons to cover its interior. Note that $\lfloor \frac{e}{21} \rfloor$ edge beacons, represented as the red edges and the red vertices, are sufficient to cover this polyhedron.

Theorem 5 *Let P be an orthogonal polyhedron with e edges. Then $\lfloor \frac{e}{6} \rfloor$ edge beacons are always sufficient to simultaneously cover the interior and exterior of P.*

Proof. Let X, Y, and Z be the number of edges incident to the $\mathcal{X}$-faces, the $\mathcal{Y}$-faces, and the $\mathcal{Z}$-faces respectively. Since $X + Y + Z = 2e$, one of X, Y, or Z is at most $\lfloor \frac{2e}{3} \rfloor$. Let F be the set of edges incident to the $\mathcal{Y}$-faces of P and suppose w.l.o.g. that $|F|$ is at most $\lfloor \frac{2e}{3} \rfloor$. There are two types of faces in F, left and right, and each of them contains four different types of edges: top, bottom, front, and back. Let E_{tb} be the set obtained by selecting first the top edges of the left faces of F and then the bottom edges of the right faces of F. In this manner, we can obtain only four different subsets of edges from F, and thus one of them contains at most $\left\lfloor \frac{|F|}{4} \right\rfloor$ edges of P.

Suppose w.l.o.g. that $|E_{tb}| = \left\lfloor \frac{|F|}{4} \right\rfloor \leq \lfloor \frac{e}{6} \rfloor$. We place a beacon in each $e \in E_{tb}$. Consider the bounding box B of P. Note that the top face of B contains the topmost faces of P, each of which contains at least one element of E_{tb}. Therefore, any point above the top face of B is covered. A similar reasoning can be used to prove that any point to the left of the left face of B, below the bottom face of B, or to the right of the right face of B is covered. A frontmost face of P contains at least the endpoints of two elements of E_{tb}, therefore, any point in front of the front face of B is covered. Analogously, a point to the back of the back face of B is also covered. We only have to prove that any point $p \notin P$ in the interior of the bounding box B can be covered by a beacon placed on an element of E_{tb}.

Let Q_p be the $\mathcal{X}$-plane containing p. Note that Q_p contains one or more polygons produced by the intersection of Q_p with P, and that each bottom vertex of a right edge of a polygon in Q_p and each top vertex of a left edge of a polygon in Q_p is of the form $b \cap Q_p$ for some $b \in E_{tb}$.

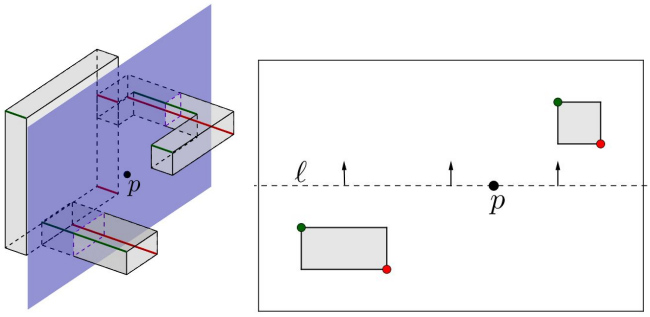
From $p \in Q_p$, shoot two vertical rays, one to the top and one to the bottom, and two horizontal rays, one to the left and one to the right. Two cases may arise, either a ray hits an edge of a polygon in Q_p, or it does not.

Suppose w.l.o.g that the ray ℓ shot up to the top hits a bottom edge e of a polygon in Q_p. If we slide ℓ to the right three cases may occur:

- We reach the endpoint v of e, and v is a convex vertex. Since v is the bottom vertex of a right edge of a polygon in Q_p, it corresponds to an element of E_{tb} in P.
- We reach the endpoint v of e, and v is a reflex vertex. Vertex v is the top vertex of a left edge of a polygon in Q_p, therefore it corresponds to an element of E_{tb} in P.
- We reach a vertical edge of a polygon in Q_p, which corresponds to a left face with an element of E_{tb} in P.

In any case, p is covered by a beacon. The proof for the other cases is similar. Now suppose that none of the rays shot up from p hits an edge of Q_p. Let ℓ be the line parallel to the $\mathcal{X}$-axis that contains p and suppose w.l.o.g that there exists a polygon above p in Q_p. Continuously move ℓ to the top maintaining it horizontal until it hits an edge a of a polygon in Q_p. Since a is a bottom edge, it has a right vertex corresponding to an element of E_{tb}. Figure 6 shows an example of this case. The proof for the case when there exists a polygon below p in Q_p is symmetric.

It follows that the exterior of P is covered. Notice that E_{tb} is composed by the edges selected by the *top-left* and the *bottom-right F-E* rules, it follows from the construction of the proof of Theorem 3 that the interior of P is also covered. □



(a) (b)

Figure 6: (a) An orthogonal polyhedron and a point p in its exterior, p is intersected by an $\mathcal{X}$-plane. (b) The intersection of the $\mathcal{X}$-plane with p. Red vertices represent bottom right edges. Green vertices represent top left edges.

5 Conclusions

In this paper we define a model of attraction by edge beacons. We shown an orthogonal polyhedron whose exterior cannot be covered with vertex beacons. We proved that $\lfloor \frac{e}{12} \rfloor$ edge beacons are always sufficient and $\lfloor \frac{e}{21} \rfloor$ edge beacons are sometimes necessary to cover the interior of an orthogonal polyhedron. We are also interested in covering both the interior and exterior of an orthogonal polyhedron at the same time. We proved that $\lfloor \frac{e}{6} \rfloor$ edge beacons are always sufficient to simultaneously cover the interior and exterior of an orthogonal polyhedron.

Some interesting open problems are: Closing the gap between the upper and lower bounds in both the interior and in the interior-exterior beacon coverage problems in orthogonal polyhedra. Perhaps more challenging is the study of the beacon coverage problem in general polyhedra.

References

[1] I. Aldana-Galván, J. Álvarez-Rebollar, J. Catana-Salazar, N. Marín-Nevárez, E. Solís-Villarreal, J. Urrutia, and C. Velarde. Covering orthotrees with guards and beacons. In *In proceedings of XVII Spanish Meeting on Computational Geometry. Alicante, Spain, July 26-28*, pages 56–68, 2017.

[2] S. W. Bae, C.-S. Shin, and A. Vigneron. Tight bounds for beacon-based coverage in simple rectilinear polygons. In *Latin American Symposium on Theoretical Informatics*, pages 110–122. Springer, 2016.

[3] M. Biro. *Beacon-based routing and guarding*. PhD thesis, State University of New York at Stony Brook, 2013.

[4] M. Biro, J. Gao, J. Iwerks, I. Kostitsyna, and J. Mitchell. Beacon-based routing and coverage. In *21st Fall Workshop on Computational Geometry*, 2011.

[5] M. Biro, J. Iwerks, I. Kostitsyna, and J. S. Mitchell. Beacon-based algorithms for geometric routing. In *Workshop on Algorithms and Data Structures*, pages 158–169. Springer, 2013.

[6] J. Cleve. Combinatorics of beacon-based routing and guarding in three dimensions. Master's thesis, Freie Universität Berlin, 2017.

[7] M. Damian, R. Flatland, H. Meijer, and J. O'Rourke. Unfolding well-separated orthotrees. In *15th Annu. Fall Workshop Comput. Geom*, pages 23–25. Citeseer, 2005.

[8] T. Michael. Guards, galleries, fortresses, and the octoplex. *The College Mathematics Journal*, 42(3):191–200, 2011.

Discrete Surveillance Tours in Polygonal Domains

Elmar Langetepe* Bengt J. Nilsson† Eli Packer‡

Abstract

The watchman route of a polygon is a closed tour that sees all points of the polygon. Computing the shortest such tour is a well-studied problem. Another reasonable optimization criterion is to require that the tour minimizes the hiding time of the points in the polygon, i.e., the maximum time during which any points is not seen by the agent following the tour at unit speed. We call such tours *surveillance routes*.

We show a linear time 3/2-approximation algorithm for the optimum surveillance tour problem in rectilinear polygons using the L_1-metric. We also present an $O(\text{polylog}\, w_{\max})$-approximation algorithm for the optimum weighted discrete surveillance route in a simple polygon with weight values in the range $[1, w_{\max}]$. Our algorithm can have superpolynomial complexity since the tour may have to see points of high weight many times.

1 Introduction

Visibility coverage of polygons with guards (mainly known as *Art Gallery* problems) have been central geometric problems for many years. Usually guards are defined as static points that see in any direction for any distance and visibility is defined by the clearance of straight lines between two features (in other words, two features see each other if the segment that connects them does not intersect (the interior of) any other feature of the input). Coverage is achieved if any point inside the polygon is visible by at least one guard.

Several art gallery variants have been proposed for different kind of settings. These include different classes of polygons, such as rectilinear and monotone polygons, and different types of guards, such as edge and segment guards; see [7, 8, 9, 11].

Allowing a guard to move inside the polygons defines a related problem but yet with very different properties. Here, a set of mobile guards walk on closed cycles (also called *tours* or *routes*) so that any point inside the polygon is seen by at least one guard during its walk along the tour. The number of guards is a parameter of the problem and the measure criteria relates to the length of the tours (e.g., minimize the longest tour). Several solutions have been proposed for the case of a single mobile guard, a *shortest watchman route* in a simple polygon. The currently fastest one combines algorithms by Tan [10] and Dror *et al.* [3], to achieve asymptotic running time $O(n^4 \log n)$.

We want to guard a given simple polygon $\mathbf{P}$, but rather than finding a shortest tour that covers the points of $\mathbf{P}$, we are interested in a tour that minimizes the maximum duration in which any of the points in $\mathbf{P}$ are not guarded. We call such a tour an *optimum surveillance route* for the polygon, abbreviated *OSR*. Kamphans and Langetepe [5] study a similar concept (*inspection paths*) but their optimization measure is the sum of the durations where features are not covered rather than the maximum duration.

We also consider a discrete version of the minimum surveillance tour problem where a given finite subset $\mathcal{S}$ of points in the polygon is to be guarded. We further generalize this version of the problem by associating weights to the points of $\mathcal{S}$.

We show a linear time 3/2-approximation algorithm for the optimum surveillance tour problem in rectilinear polygons using the L_1-metric. We also present a $O(\text{polylog}\, w_{\max})$-approximation algorithm for the optimum weighted discrete surveillance route in a simple polygon with weight values in the range $[1, w_{\max}]$.

2 Preliminaries

Let $\mathbf{V}(p)$ denote the *visibility polygon* of a point $p \in \mathbf{P}$, i.e., the set of all points q in $\mathbf{P}$ such that the segment pq fully lies inside $\mathbf{P}$. Obviously, the visibility polygon $\mathbf{V}(p)$ is a simple polygon itself. A watchman route is a closed tour within the polygon that sees all points of the polygon. Hence, a tour T is a watchman route if $\forall p \in \mathbf{P}; \mathbf{V}(p) \cap T \neq \emptyset$.

A reasonable extension of the concept of a watchman route is to require that the tour minimizes the hiding time of the points in the polygon, i.e., the maximum time during which any point in the polygon is not seen by an agent following the tour at unit speed. To formally define this, we introduce the concept of *hidden pieces* of a tour T.

Definition 1 Given a tour T and a point p, the *hidden pieces*, $\mathcal{H}_T(p)$, of T with respect to p is the set of

*Institute of Computer Science I, University of Bonn, 53117 Bonn, Germany. elmar.langetepe@cs.uni-bonn.de

†Department of Computer Science, Malmö University, SE-205 06 Malmö, Sweden. bengt.nilsson.TS@mah.se

‡Intel Corporation, Givataim, Israel. eli.packer@intel.com

maximal paths $\mathcal{H}_T(p) \stackrel{def}{=} \{T \setminus \mathbf{V}(p)\}$.

The visibility polygon $\mathbf{V}(p)$ of p subdivides the hidden pieces of T into a number of subpaths $X_1, X_2, \ldots, X_m$ that do not have any points seen from p. Hence, $\mathcal{H}_T(p) = \{X_1, X_2, \ldots, X_m\}$.

Definition 2 Given a tour T and a point p in $\mathbf{P}$, the *hiding cost* (Mitchell [4] calls it the *dark cost*), $hc_T(p)$, of T with respect to p is the length of the longest path X in $\mathcal{H}_T(p)$ if p is visible from T, i.e.,

$$hc_T(p) \stackrel{def}{=} \begin{cases} \infty, & \text{if } \mathbf{V}(p) \cap T = \emptyset, \\ \max_{X \in \mathcal{H}_T(p)}\{\|X\|\}, & \text{if } \mathbf{V}(p) \cap T \neq \emptyset, \end{cases}$$

where $\|X\|$ denotes the length of X in a given metric.

Given the definition of the hiding cost, we can define the surveillance cost or delay of a tour.

Definition 3 Given a tour T, the *surveillance cost* or *delay*, $d(T)$, of T is given by

$$d(T) \stackrel{def}{=} \max_{p \in \mathbf{P}}\{hc_T(p)\}. \tag{1}$$

We say that the tour T is a surveillance route for $\mathbf{P}$ if $d(T)$ is finite.

With this definition, it is clear that any surveillance route is also a watchman route, since all points of the polygon must be seen by the route for it to have finite surveillance cost.

Given a finite set of points $\mathcal{S}$ in $\mathbf{P}$ to be guarded, we define a discrete version of the surveillance cost or delay of a tour.

Definition 4 Given a tour T, the *discrete surveillance cost* or *discrete delay*, $d(T)$, of T with respect to a finite point set $\mathcal{S}$ to be guarded is given by

$$d_{\mathcal{S}}(T) \stackrel{def}{=} \max_{p \in \mathcal{S}}\{hc_T(p)\}. \tag{2}$$

We say that the tour T is a discrete surveillance route for $\mathcal{S}$ in $\mathbf{P}$ if $d_{\mathcal{S}}(T)$ is finite.

We make use of classical notation; see for example [2]; for the following definitions. To every reflex vertex in $\mathbf{P}$ we can associate two *extensions*, i.e., the two maximal line segments in $\mathbf{P}$ through the vertex and collinear to the two edges adjacent to the vertex; see Figure 1(a). We associate a direction to an extension e collinear to an edge e_v by giving e the same direction as e_v has when $\mathbf{P}$ is traversed in counterclockwise order. This allows us to refer to the regions to the left and right of an extension, meaning those point reached by a left turn or a right turn respectively from the directed segment e. Let $\mathbf{L}(e)$ denote the part of $\mathbf{P}$ to the left of e and $\mathbf{R}(e)$ the part to the right of e.

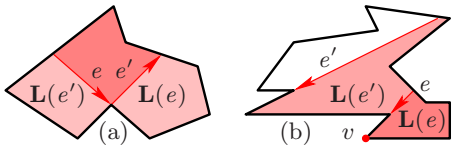


Figure 1: Illustrating definitions. (a) the two extensions issuing from a reflex vertex. (b) e dominates e', e is essential and v is an essential vertex.

We say that e is a *visibility extension* with respect to a surveillance route T, if T has some point in $\mathbf{R}(e)$.

The visibility extensions capture visibility information in the sense that a surveillance route must have points to the left of each of them.

We say that an extension e *dominates* another extension e', if $\mathbf{L}(e)$ is properly contained in $\mathbf{L}(e')$.

Definition 5 A visibility extension e is *essential*, if e is not dominated by any other visibility extension.

An essential extension e is collinear to an edge with one reflex and one convex vertex, since if both vertices are reflex, then there is another essential extension (issuing from the other reflex vertex) that dominates e, giving us a contradiction.

Definition 6 Let v be the convex vertex of the edge collinear to an essential extension e. We call the convex vertex v an *essential vertex*; see Figure 1(b).

The essential vertices play an important role for surveillance routes as we show in the next lemma.

For a polygon $\mathbf{P}$, we let *OSR* denote an *optimum surveillance route*, i.e., a tour T for which $d(T)$ is minimal.

Lemma 7 *If $\mathbf{P}$ is such such that $d(OSR) > 0$, then the delay of OSR is attained at some essential vertex of $\mathbf{P}$, i.e., there is an essential vertex v such that*

$$d(OSR) = hc_{OSR}(v). \tag{3}$$

Proof. Let p be a point in $\mathbf{P}$ such that $d(OSR) = hc_{OSR}(p) > 0$. Since $d(OSR) > 0$, the point p exists. Let X be a path in $\mathcal{H}_{OSR}(p)$ having the length of $d(OSR)$. The path X starts and finishes at an edge e of $\mathbf{V}(p)$ having a reflex vertex r of $\mathbf{P}$ as one endpoint. The segment e and the point p are collinear; see Figure 2. Thus, the segment e subdivides $\mathbf{P}$ into two parts, $\mathbf{P}_X$, containing the path X and $\bar{\mathbf{P}}_X$, not containing the path X. The part $\bar{\mathbf{P}}_X$ contains the point p and has e as a boundary edge, the point r is a reflex vertex of both $\mathbf{P}$ and $\bar{\mathbf{P}}_X$.

To prove that there is an *essential vertex* with hiding cost at least as high as that of p, follow the boundary of $\bar{\mathbf{P}}_X$ from r away from e until the first convex vertex u is reached and let u' be the last reflex vertex as we move along the boundary from r. We note that we could have $u' = r$. Let e' be the extension collinear to the edge

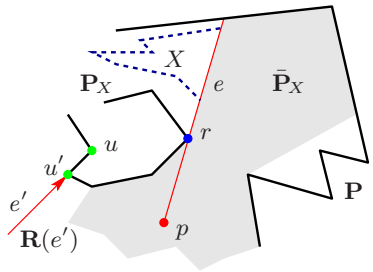


Figure 2: Illustrating the proof of Lemma 7.

$[u, u']$. Since the sequence of vertices along $\bar{\mathbf{P}}_X$ from r to u' is reflex, the extension e' is completely interior to $\bar{\mathbf{P}}_X$ and it is also a visibility extension since *OSR* has points in $\mathbf{R}(e')$, e.g., the path X in $\mathbf{P}_X$. Therefore, either e' is an essential extension or it is dominated by an essential extension. Let v denote the essential vertex for this essential extension, independently of whether the essential extension is e' or some other dominating extension. By construction, since e' is contained in $\bar{\mathbf{P}}_X$, the visibility polygon $\mathbf{V}(v)$ does not see any point in in $\mathbf{P}_X$, and hence $hc_{OSR}(v) \geq \|X\| = hc_{OSR}(p)$, proving the lemma. □

Lemma 7 shows that the optimum surveillance route in a simple polygon is the optimum discrete surveillance route of the essential vertices of the polygon, i.e., $d(OSR) = d_{\mathcal{V}}(OSR)$, where $\mathcal{V}$ is the set of essential vertices of the polygon.

3 L_1-Surveillance Routes in Rectilinear Polygons

In [6], the authors show that the shortest watchman route and the optimum surveillance route are not necessarily the same and that the shortest watchman route is a 2-approximation to the optimum surveillance route in a simple polygon. It is still an open question whether the optimum surveillance route in a simple polygon can be computed in polynomial time, assuming P ≠ NP.

Even considering rectilinear polygons in the L_1-metric, a *SWR* is not necessarily the optimum surveillance tour; see Figure 3(a) and (b). The rectilinear polygon has five essential extensions, dotted lines, four of which have unit length and the fifth (the top middle one) is arbitrarily short. The length of a *SWR* is just over 8 which is also the surveillance cost. However, by revisiting the short extension we obtain a slightly longer tour with surveillance cost of just over 7.

We define a particular L_1-shortest watchman route.

Definition 8 In a rectilinear polygon, we call an L_1-shortest watchman route that has maximal interior area a *maximum shortest watchman route* and denote it by *MSWR*.

A *MSWR* has the special property that between any two consecutive essential extensions e and e', the *MSWR* follows a rectilinear shortest path between e and e'. A *MSWR* can be computed in linear time by a straightforward modification of the algorithm of Chin and Ntafos [1].

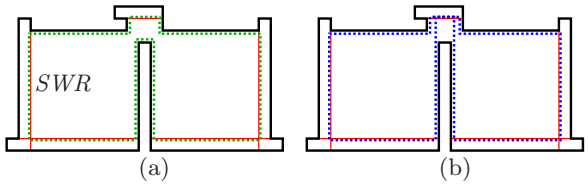


Figure 3: Counterexample showing that an L_1-optimal *SWR* is not an L_1-optimal *OSR* in rectilinear polygons.

Theorem 9 *A MSWR is a 3/2-approximation for the L_1-optimal OSR in a rectilinear polygon.*

Proof. According to Lemma 7, there is an essential vertex v for which the hiding cost attains the delay of *OSR*. Let X be the path in $\mathcal{H}_{OSR}(v)$ with $\|X\| = d(OSR)$. We claim that $\|X\| \geq 2\|MSWR\|/3$ thus giving us that

$$d(MSWR) \leq \|MSWR\| \leq \frac{3}{2}\|X\| = \frac{3}{2}d(OSR). \quad (4)$$

To prove that $\|X\| \geq 2\|MSWR\|/3$, assume for a contradiction that $\|X\| < 2\|MSWR\|/3$. Let e be the essential extension of v and let p and q be the two endpoints of X on e. Since $[p, q] \cup X$ is a watchman route we have that $\|[p, q]\| + \|X\| \geq \|MSWR\|$ and therefore $\|[p, q]\| > \|MSWR\|/3$. Without loss of generality, we can assume that e is vertical. We construct the two maximal horizontal line segments interior to the polygon that go through the points p and q. The two segments subdivide the polygon into three pieces, $\mathbf{P}_\mathrm{T}$ the top piece, $\mathbf{P}_\mathrm{M}$ the middle piece, and $\mathbf{P}_\mathrm{B}$ the bottom piece; see Figure 4.

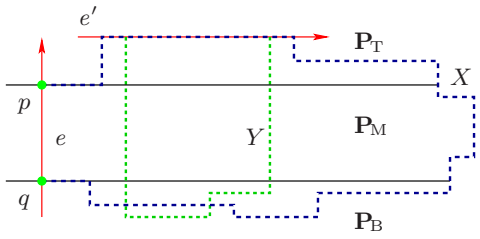


Figure 4: Illustrating the proof of Theorem 9.

Both $\mathbf{P}_\mathrm{T}$ and $\mathbf{P}_\mathrm{B}$ must contain essential extensions since otherwise, the path X is not part of the optimum surveillance route, giving us an immediate contradiction.

Therefore, let e' be an essential extension in $\mathbf{P}_\mathrm{T}$ with v' as the essential vertex and consider the set $\mathcal{H}_{OSR}(v')$. Some path Y in this set must visit essential extensions in $\mathbf{P}_\mathrm{B}$ and must therefore have length at least $2\|[p, q]\| > 2\|MSWR\|/3$; see Figure 4. Hence,

$$d(OSR) \geq hc_{OSR}(v') \geq \frac{2}{3}\|MSWR\| > \|X\| = d(OSR), \quad (5)$$

and we have a contradiction. □

Remark: Also in the case of the L_1-metric in rectilinear polygons, it is an open question whether the optimum surveillance route can be computed in polynomial time, assuming P ≠ NP.

4 Weighted Discrete Surveillance Routes

In this section, we consider the weighted discrete surveillance route problem in a simple polygon and define it as follows. Let $\mathbf{P}$ be a simple polygon with n edges and let $\mathcal{S}$ be a finite set of points inside $\mathbf{P}$. To each point $p \in \mathcal{S}$ is associated a weight $w(p)$. The idea is that points with higher weights have higher priority and need to be guarded more often than ones with lower weights. Given some tour T, we define the *weighted discrete delay* as

$$d^w_{\mathcal{S}}(T) \stackrel{def}{=} \max_{p\in\mathcal{S}}\{w(p) \cdot hc_T(p)\}. \tag{6}$$

We call a tour that achieves the minimum weighted delay on a finite set of points $\mathcal{S}$ in $\mathbf{P}$ with weights $w(\cdot)$ an *optimum weighted discrete surveillance route*, *OWDSR*.

For simplicity we assume that all weights are positive, that the smallest weight is equal to 1, and that the largest weight value is $w_{\max}$.

In [6], the authors show that the problem of computing an *OWDSR* is NP-hard already for the two weight values 1 and 2, that the shortest watchman route limited to see the points in $\mathcal{S}$ is a $2w_{\max}$-approximation of a *OWDSR*, and they present an $O(|\mathcal{S}|^3 n \log n)$ time constant-factor approximation algorithm for a *OWDSR* in the case of two arbitrary weight values.

4.1 A Simple Approximation Algorithm

We can immediately improve on the $2w_{\max}$-approximation in [6] as follows. Given the points in $\mathcal{S}$ and the weight values $1 = w(p_1) \le \cdots \le w(p_{|\mathcal{S}|}) = w_{\max}$, we scale all the weight values $w(p_i) \in [1, \sqrt{w_{\max}}[$ to 1, where $[x, y[$ denotes the right open ended interval from x to y, and all the weight values $w(p_i) \in [\sqrt{w_{\max}}, w_{\max}]$ to $\sqrt{w_{\max}}$. We next apply the c-approximation algorithm for two weight values on the scaled problem instance, giving us the following theorem.

Theorem 10 *The algorithm above computes a $c\sqrt{w_{\max}}$-approximation of an OWDSR guarding the points of $\mathcal{S}$ in $\mathbf{P}$ having arbitrary weight values in $O(|\mathcal{S}|^3 n \log n)$ time.*

4.2 An Improved Approximation Algorithm

In the following, we abuse language somewhat and say that a tour *visits* a point p, when we actually mean that the tour intersects $\mathbf{V}(p)$.

For an discrete weighted surveillance tour V, visiting the points in a finite weighted point set $\mathcal{S}$ in $\mathbf{P}$ (we assume that V is the shortest tour that visits the points in this order), we have the following inequality,

$$\forall p \in \mathcal{S} \quad hc_V(p) \le \|V\|. \tag{7}$$

If V is such that it visits some point p' only once, then

$$\|V\| \le 2 \cdot hc_V(p'). \tag{8}$$

Given the points of $\mathcal{S}$ in $\mathbf{P}$ with weights in the range $w(p) \in [1, w_{\max}]$, we partition the set $\mathcal{S}$ into disjoint subsets $\mathcal{S}_l$, $0 \le l \le M$, such that each point $p \in \mathcal{S}_l$ has $w(p) \in [w_{\max}^{l/M}, w_{\max}^{(l+1)/M}[$. We can scale the instance so that each point has weight $w_{\max}^{l/M}$. If we can find an x-approximate solution for the scaled instance, we immediately have an algorithm with approximation factor

$$x \cdot w_{\max}^{1/M} \tag{9}$$

for the original input instance.

We let $\mathcal{I}_i$, $0 \le i \le m$, be the nonempty sets of scaled points so that for each point $p \in \mathcal{I}_i$, the weight $w(p) = w_i = w_{\max}^{l/M}$, for some $l \ge i$. In fact, if $p \in \mathcal{I}_i$ and $p' \in \mathcal{I}_{i'}$ with $0 \le i < i' \le m$, then $w(p) = w_i = w_{\max}^{l/M}$ and $w(p') = w_{i'} = w_{\max}^{l'/M}$, with $l < l'$. Since the sets $\mathcal{I}_i$, $0 \le i \le m$, are nonempty, we have $m \le |\mathcal{S}|$.

For each $0 \le i \le m$, let W_i denote a shortest tour in $\mathbf{P}$ that visits all the points in $\mathcal{I}_i$. Each such tour can be computed in $O(|\mathcal{I}_i|^3 n \log n)$ time [3, 10], and hence, all these tours can be computed in $O(|\mathcal{S}|^3 n \log n)$ time. Similarly, let T_i denote a tour in $\mathbf{P}$ with minimum delay for the scaled points in $\mathcal{I}_i$. From [6], we know that $d^w_{\mathcal{I}_i}(W_i) = d_{\mathcal{I}_i}(W_i) \le 2d_{\mathcal{I}_i}(T_i) = 2d^w_{\mathcal{I}_i}(T_i)$ since the weights of the points in $\mathcal{I}_i$ are the same.

We furthermore define $\mathcal{I}_{i,j} = \bigcup_{i\le\iota\le j} \mathcal{I}_\iota$. Thus, the set $\mathcal{I}_{0,m}$ represents the scaled weight points of the original instance $\mathcal{S}$. Let $W_{i,j}$ denote a shortest tour in $\mathbf{P}$ that visits all the points in $\mathcal{I}_{i,j}$ and let $T_{i,j}$ denote a tour in $\mathbf{P}$ with minimum weighted delay for these points.

For each $0 \le i \le j \le m$, we define a tour $S_{i,j}$ that visits all the points in $\mathcal{I}_{i,j}$ at least once and has short weighted delay. We have $S_{i,i} = W_{i,i} = W_i$, when $i = j$. For $i < j$, with $l \stackrel{def}{=} \lfloor (i + j)/2 \rfloor$, let $N_{i,l}$ denote the number of times points from $\mathcal{I}_{i,l}$ are visited as we follow the tour $S_{i,l}$ around once. We note that since the same point can be visited several times, $N_{i,l}$ can be substantially larger than $|\mathcal{I}_{i,l}|$. The tour $S_{i,j}$ is the tour with smallest weighted delay out of a set of tours $\{U^k_{i,j} \mid 1 \le k \le N_{i,l}\}$, each tour $U^k_{i,j}$ defined recursively from $S_{i,l}$ and $S_{l+1,j}$.

The tour $U^k_{i,j}$ is constructed as follows: let $r_{i,j}$ be a point on $S_{l+1,j}$ so that $\max_{p\in\mathcal{I}_{i,l}}\{SP(r_{i,j}, \mathbf{V}(p))\}$ is minimized. We denote this length by $D_{i,j}$. Let $SP(S_{i,l}, S_{l+1,j})$ be the shortest path between $S_{i,l}$ and

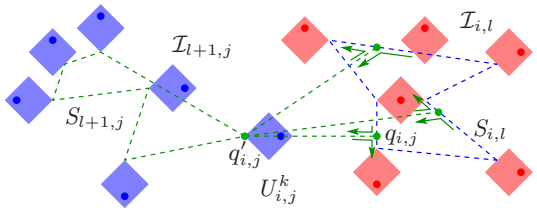


Figure 5: Schematic illustration of the construction of the tour $U^k_{i,j}$, $1 \le k \le N_{i,l}$. Red and blue regions are the visibility polygons of points in $\mathcal{I}_{i,l}$ and $\mathcal{I}_{l+1,j}$ respectively.

$S_{l+1,j}$ with endpoints $q_{i,j}$ on $S_{i,l}$ and $q'_{i,j}$ on $S_{l+1,j}$. Evidently, $\|SP(S_{i,l}, S_{l+1,j})\| \le D_{i,j}$. Let $\delta_{i,j} = \max\{D_{i,j}, \|S_{i,l}\|/2k\}$. We partition $S_{i,l}$ into at most k subpaths $Y_1, \ldots, Y_k$, each (except the last) of length $\delta_{i,j}$ and with Y_1 starting at $q_{i,j}$. $U^k_{i,j}$ is the tour obtained by first following $S_{l+1,j}$ around from $q'_{i,j}$ back to $q'_{i,j}$, then moving to $q_{i,j}$, following Y_1, moving back to $q'_{i,j}$, doing one more tour around $S_{l+1,j}$, moving to the first point of Y_2 and following Y_2, moving back to $q'_{i,j}$, make a tour around $S_{l+1,j}$, and continue alternating between following each subsequent subpath Y_κ and making a tour around $S_{l+1,j}$; see Figure 5. $U^k_{i,j}$ makes at most k rounds around $S_{l+1,j}$.

The tour among $U^1_{i,j}, \ldots, U^{N_{i,l}}_{i,j}$ with the smallest weighted delay becomes $S_{i,j}$. We show that $S_{i,j}$ has small weighted delay.

Lemma 11 *There exists a positive constant a such that*

$$d^w_{\mathcal{I}_{i,j}}(S_{i,j}) \le a^{1+\log(j-i+1)} \cdot d^w_{\mathcal{I}_{i,j}}(T_{i,j}),$$

for every $0 \le i \le j \le m$.

Proof. We make a proof by induction on $j - i$. We show the lemma to be true for $i = j$ and then proceed inductively for successively larger values of $j - i$.

From [6], we know that $d_{\mathcal{I}_i}(W_i) \le 2d_{\mathcal{I}_i}(T_i)$, for $0 \le i \le m$. Thus, all weights being equal,

$$d^w_{\mathcal{I}_i}(S_{i,i}) = d^w_{\mathcal{I}_i}(W_i) \le 2d^w_{\mathcal{I}_i}(T_i) \le\ a^1 \cdot d^w_{\mathcal{I}_i}(T_i), \qquad (10)$$

if $2 \le a$, proving the base case when $i = j$.

For the induction step, consider a tour $T_{i,j}$, an optimal solution for *OWDSR* in **P** that sees all the scaled points in $\mathcal{I}_{i,j}$ and has minimum weighted discrete delay.

We partition $T_{i,j}$ into subpaths as follows: let H_1 be the shortest subpath of $T_{i,j}$ that sees each point of $\mathcal{I}_{l+1,j}$ at least once, with $l \stackrel{def}{=} \lfloor (i+j)/2 \rfloor$ as usual, and the first visits of points in $\mathcal{I}_{i,j}$ before and after H_1 are points in $\mathcal{I}_{i,l}$. Follow $T_{i,j}$ from an endpoint of H_1 until a point of $\mathcal{I}_{l+1,j}$ is seen again. We let this subpath be L_1. Continue along $T_{i,j}$ until each point of $\mathcal{I}_{l+1,j}$ has been seen again and the next visit is to a point in $\mathcal{I}_{i,l}$, giving the subpath H_2, followed by the subpath L_2 of visits to points in $\mathcal{I}_{i,l}$, and so on. Continue subdividing $T_{i,j}$ into $2K$ subpaths, $H_1, L_1, \ldots, H_K, L_K$, for some value K, such that L_K connects back to H_1 and each H_k visits all the points of $\mathcal{I}_{l+1,j}$ and each L_k, except possibly L_K, only visits points in $\mathcal{I}_{i,l}$. The subpath L_K can visit some but not all points in $\mathcal{I}_{l+1,j}$.

For each path H_k, $1 \le k \le K$, we shortcut any detours that H_k makes to visit points in $\mathcal{I}_{i,l}$, then go back to the beginning of H_k giving us the tour H'_k. From there, we visit each (unvisited) point in $\mathcal{I}_{i,l}$ that was shortcut from H_k in the same order and continue following L_k, giving the path L'_k. Let Z be the tour $Z = \bigcup_{1 \le k \le K} H'_k \cup L'_k$. We have $\|H'_k\| \le 2\|H_k\|$ and $\|L'_k\| \le \|H_k\| + \|L_k\|$, for all $1 \le k \le K$. Hence,

$$d^w_{\mathcal{I}_{i,j}}(Z) \le 3d^w_{\mathcal{I}_{i,j}}(T_{i,j}) \text{ and } \|Z\| \le 3\|T_{i,j}\|. \qquad (11)$$

Also, for any $0 \le i \le j \le m$ and $l = \lfloor (i+j)/2 \rfloor$, we have by definition,

$$\forall k\ \forall p \in \mathcal{I}_{l+1,j} \quad hc_{T_{l+1,j}}(p) \le \|W_{l+1,j}\| \le \|H'_k\|, \qquad (12)$$

$$\forall p \in \mathcal{I}_{i,l} \quad hc_{T_{i,l}}(p) \le \|W_{i,l}\| \le \sum_{1 \le k \le K} \|L'_k\|. \qquad (13)$$

We compare the tour $U^K_{i,j}$, constructed from $S_{i,l}$ and $S_{l+1,j}$, with the tour Z constructed from an *OWDSR* $T_{i,j}$ for the point set $\mathcal{I}_{i,j}$ above. Note that we can assume that we know the value of K since we compute $U^k_{i,j}$, for all $1 \le k \le N_{i,l}$.

For a point $p \in \mathcal{I}_{l+1,j}$, the hiding cost of p is bounded by

$$\begin{aligned}
hc_{U^K_{i,j}}(p) &\le hc_{S_{l+1,j}}(p) + 2\delta_{i,j} + \|S_{i,l}\|/K \\
&\le hc_{S_{l+1,j}}(p) + 2\|S_{i,l}\|/K \\
(8) \quad &\le hc_{S_{l+1,j}}(p) + 4hc_{S_{i,l}}(p_i)/K \\
(\text{ind.}) \quad &\le a^{1+\log(j-l)} \cdot hc_{T_{l+1,j}}(p) \\
&\quad + 4a^{1+\log(l-i+1)} \cdot hc_{T_{i,l}}(p_i)/K \\
(13) \quad &\le a^{1+\log(j-l)} \cdot hc_{T_{l+1,j}}(p) \\
&\quad + 4a^{1+\log(l-i+1)} \cdot \sum_{1 \le k \le K} \|L'_k\|/K \\
&\le 4a^{1+\log(j-l)} \cdot \left(hc_{T_{l+1,j}}(p) + \max_{1 \le k \le K}\{\|L'_k\|\}\right) \\
&\le 8a^{1+\log(j-l)} \cdot \max\left\{hc_{T_{l+1,j}}(p), \max_{1 \le k \le K}\{\|L'_k\|\}\right\} \\
&\le 8a^{1+\log(j-l)} \cdot hc_Z(p) \\
(11) \quad &\le a^{1+\log(j-i+1)} \cdot hc_{T_{i,j}}(p), \qquad (14)
\end{aligned}$$

if $a \ge 24$, where $p_i \in \mathcal{I}_i$ is visited only once by $S_{i,l}$.

For a point $p \in \mathcal{I}_i$, the hiding cost of p is bounded by

$$\begin{aligned}
hc_{U^K_{i,j}}(p) &\le K \cdot \|S_{l+1,j}\| + 2K \cdot \delta_{i,j} + \|S_{i,l}\| \\
&\le K \cdot \|S_{l+1,j}\| + 2\|S_{i,l}\| \\
(8) \quad &\le 2K \cdot hc_{S_{l+1,j}}(p_{l+1}) + 4hc_{S_{i,l}}(p) \\
(\text{ind.}) \quad &\le 2Ka^{1+\log(j-l)} \cdot hc_{T_{l+1,j}}(p_{l+1}) \\
&\quad + 4a^{1+\log(l-i+1)} \cdot hc_{T_{i,l}}(p) \\
(12), (13) \quad &\le 2Ka^{1+\log(j-l)} \cdot \min_{1 \le k \le K}\{\|H'_k\|\} \\
&\quad + 4a^{1+\log(l-i+1)} \cdot \sum_{1 \le k \le K} \|L'_k\|
\end{aligned}$$

$$\leq 4a^{1+\log(j-l)} \cdot \Big(\sum_{1 \leq k \leq K} \|H'_k\| + \|L'_k\| \Big)$$

$$\leq 4a^{1+\log(j-l)} \cdot \|Z\|$$

$$\leq 8a^{1+\log(j-l)} \cdot hc_Z(p) \quad (8)$$

$$\leq a^{1+\log(j-i+1)} \cdot hc_{T_{i,j}}(p), \quad (11) \quad (15)$$

if $a \geq 24$, since $p \in \mathcal{I}_i$ is visited only once by $S_{i,l}$ and has maximal hiding cost among the points in $\mathcal{I}_{i.j}$, and $p_{l+1} \in \mathcal{I}_{l+1}$ is visited only once by $S_{l+1,j}$.

For a point $p \in \mathcal{I}_{i+1,l}$, the point p is in the upper half of some recursive division of the sets $\mathcal{I}_{i,l}, \mathcal{I}_{i,\lfloor (i+l)/2 \rfloor}, \mathcal{I}_{i,\lfloor (i+\lfloor (i+l)/2 \rfloor)/2 \rfloor}, \ldots, \mathcal{I}_{i,i+1}$, for which an inequality similar to (14) and (15) applies. We omit the details.

Thus, for any point $p \in \mathcal{I}_{i,j}$, we have that there is a constant $a > 1$ such that

$$hc_{S_{i,j}}(p) \leq a^{1+\log(j-i+1)} \cdot hc_{T_{i,j}}(p)$$

and the lemma is therefore proved. □

We compute $S_{0,m}$ by establishing $U^k_{0,m}$, for every $1 \leq k \leq N_{0,\lfloor m/2 \rfloor}$, and each of these are computed recursively from $S_{0,\lfloor m/2 \rfloor}$ and $S_{\lfloor m/2 \rfloor+1,m}$. Given these two tours, $U^k_{0,m}$ is constructed by copying $S_{\lfloor m/2 \rfloor+1,m}$ at most k times and connecting each tour to the at most k subpaths of $S_{0,\lfloor m/2 \rfloor}$ using shortest paths. This takes $O(k|\mathcal{S}| + kn)$ time for each $U^k_{0,m}$. Note that $N_{0,\lfloor m/2 \rfloor} \in |\mathcal{S}|^m$, thus this takes $O(|\mathcal{S}|^{m+1}n)$ time in total. At each level of the recursion we use this amount of time and we have $\log m+1$ levels, giving us $O(|\mathcal{S}|^{O(m)}n)$ time. The preprocessing step of computing all the discrete watchman routes W_i, for $0 \leq i \leq m$ takes $O(|\mathcal{S}|^3 n \log n)$ time, and hence, the total complexity is bounded by $O(|\mathcal{S}|^{O(m)} n \log n)$. We note that the complexity is superpolynomial in $|\mathcal{S}|$ and n.

By carefully considering the weight ratios in the construction, we could perhaps limit the computation to the relevant values of k, reducing the necessity to compute $U^k_{i,j}$ for all values of k up to $N_{i,l}$. We could thus potentially make the algorithm take polynomial time in the size of the output tour $S_{0,m}$.

Theorem 12 *There is an $O(|\mathcal{S}|^{O(\log w_{\max})} \cdot n \log n)$ time algorithm that computes a $O(\operatorname{polylog} w_{\max})$-approximate weighted discrete surveillance tour to the original unscaled weighted point set $\mathcal{S}$ in $\mathbf{P}$ having n edges.*

Proof. Apply the algorithm described above with

$$m = \max\{\, 3, \lceil \log w_{\max} / \log \log w_{\max} \rceil \}\,,$$

where a is the constant in Lemma 11 and $m+1$ is the number of weight values, with all the original weights scaled to the lowest value in their respective interval $[w^{i/m}_{\max}, w^{(i+1)/m}_{\max}[$, for $0 \leq i \leq m$.

From Lemma 11, we have that the scaled instance is approximated within an approximation factor of $a^{1+\log(m+1)} \leq a^2 m^{\log a}$ and by our choice of the value m, we have $a^2 m^{\log a} \geq w^{1/m}_{\max}$ and by (9), the approximation factor for the unscaled instance is bounded by $w^{1/m}_{\max} \cdot a^2 m^{\log a} \leq a^4 m^{2 \log a} \in O(\operatorname{polylog} w_{\max})$.

The running time follows from the discussion above. □

5 Conclusions

We present a linear time 3/2-approximation algorithm for the optimum surveillance tour problem in rectilinear polygons in the L_1-metric. It is still an open problem whether an optimum tour can be computed in polynomial time assuming P ≠ NP. We believe that the same approach should also give a 3/2-approximation for general simple polygons in the L_1-metric.

We also present an $O(\operatorname{polylog} w_{\max})$-approximation algorithm for the optimum weighted discrete surveillance route in a simple polygon with weight values in the range $[1, w_{\max}]$.

The deeper complexity relationships of the optimum weighted discrete surveillance tour problem in simple polygons remains to be investigated. For two weight values, the problem is NP-hard but constant factor approximable [6]. It is not evident that a polynomial time constant factor approximation algorithm exists for the general problem assuming P ≠ NP.

References

[1] W. Chin, S. Ntafos. Optimum Watchman Routes. *Information Processing Letters*, 28:39–44, 1988.

[2] W. Chin, S. Ntafos. Shortest Watchman Routes in Simple Polygons. *Disc. & Comp. Geometry*, 6(1):9–31, 1991.

[3] M. Dror, A. Efrat, A. Lubiw, J.S.B. Mitchell. Touring a Sequence of Polygons. In *Proc. 35th STOC*, p 473–482, 2003.

[4] J.S.B. Mitchell. Personal communication, 2017.

[5] T. Kamphans, E. Langetepe. Inspecting a Set of Strips Optimally. In *Proc. 11th WADS*, p 423–434. LNCS 5664, 2009.

[6] B.J. Nilsson, E. Packer. Weighted Discrete Surveillance Tours in Simple Polygons. In *Proc. 33rd EuroCG*, 2017.

[7] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, 1987.

[8] J. O'Rourke. Visibility. In J.E. Goodman, J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 28. CRC Press, 1997, 2nd ed. 2004.

[9] T.C. Shermer. Recent Results in Art Galleries. *Proceedings of the IEEE*, p 1384–1399, 1992.

[10] X.-H. Tan. Fast Computation of Shortest Watchman Routes in Simple Polygons. *Information Processing Letters*, 77(1):27–33, 2001.

[11] J. Urrutia. Art Gallery and Illumination Problems. In J.R. Sack, J. Urrutia, editors, *Handbook on Computational Geometry*, chapter 22. Elsevier, 1999.

Guarding Monotone Polygons with Half-Guards

Matt Gibson* Erik Krohn† Matthew Rayford‡

Abstract

We consider a variant of the art gallery problem where all guards are limited to seeing to the right inside of a monotone polygon. We provide a polynomial-time 4-approximation algorithm for a version of the problem where we wish to point-guard the vertices of the polygon. We then extend this algorithm and provide a $O(1)$-approximation to point-guard the boundary of the polygon and ultimately the entire polygon.

1 Introduction

In the geometric set cover problem, we are given some set of points P and a set S where each $s \in S$ can cover some subset of P. The subset of P is generally induced by some geometric object. For example, P might be a set of points in the plane, and s consists of the points contained within some disk in the plane. For most variants, the problem is NP-hard and can easily be reduced to an instance of the combinatorial set cover problem which has a polynomial-time $O(\log n)$-approximation algorithm—the best possible approximation under standard complexity assumptions [5, 18, 10, 12, 17]. The main question is to then determine which of the geometric set cover problem variants we can obtain polynomial-time approximation algorithms with approximation ratio $o(\log n)$, as any such algorithm must exploit the geometry of the problem to achieve the result. This area has been studied extensively, see for example [4, 21, 3], and much progress has been made utilizing algorithms that are based on solving the standard linear programming relaxation.

Unfortunately, these techniques do not work for set cover variants based on visibility, such as the well-known *art gallery problem*. An instance of the art gallery problem takes as input a simple polygon P. The polygon P is defined by a set of points $V = \{v_1, v_2, \ldots, v_n\}$. There are edges connecting $\{v_i, v_{i+1}\}$ where $i = 1, 2, \ldots, n-1$ and an edge connecting $\{v_n, v_1\}$. If these edges do not intersect other than at the points in V, then P is called a simple polygon. The edges of a simple polygon give us two disjoint regions: inside the polygon and outside the polygon. For any two points $p, q \in P$, we say that p sees q if the line segment $\overline{pq}$ does not go outside of P. The art gallery problem seeks to find a set of points $G \subseteq P$ such that every point $p \in P$ is seen by some point in G. We call this set G a guarding set. In the point-guarding problem, guards can be placed anywhere inside of P. In the vertex guarding problem, guards are only allowed to be placed at vertices in V. The optimization problem is thus defined as finding the smallest such G in each case.

These problems are motivated by applications such as line-of-sight transmission networks in terrains, signal communications and broadcasting, cellular telephony systems and other telecommunication technologies as well as placement of motion detectors and security cameras.

1.1 Previous Work

The question of whether guarding simple polygons is NP-hard was independently confirmed by Aggarwal [2] and Lee and Lin [16]. They showed that the problem is NP-hard for both vertex guarding and point-guarding.

Along with being NP-hard, Brodén et al. [6] and Eidenbenz [9] independently proved that point-guarding simple polygons is APX-hard. This means that there exists a constant $\epsilon > 0$ such that no polynomial-time algorithm can guarantee an approximation ratio of $(1+\epsilon)$ unless P = NP. Ghosh provides a $O(\log n)$-approximation for the problem of vertex guarding an n-vertex simple polygon in [11]. This result can be improved for simple polygons using randomization, giving an algorithm with expected running time $O(nOPT^2 \log^4 n)$ that produces a vertex guard cover with approximation factor $O(\log OPT)$ with high probability, where OPT is the smallest vertex guard cover for the polygon [8]. Whether a polynomial time constant factor approximation algorithm can be obtained for vertex guarding a simple polygon is a longstanding and well-known open problem. Deshpande et al. [7] present a pseudopolynomial randomized algorithm for finding a point-guard cover with approximation factor $O(\log OPT)$. King and Kirkpatrick provide a $O(\log \log OPT)$-approximation algorithm for the problem of guarding a simple polygon with guards on the perimeter in [13]. The point-guarding problem seems to be much more difficult and little is known about it [7].

Additional Polygon Structure. Due to the inherent difficulty in fully understanding the art gallery problem for simple polygons, there has been some work done guarding polygons with some additional structure. A simple polygon P is x-monotone (or simply monotone) if any vertical line intersects the boundary of P at most

*University of Texas at San Antonio, gibson@cs.utsa.edu
†University of Wisconsin - Oshkosh, krohne@uwosh.edu
‡University of Wisconsin - Oshkosh, rayfom16@uwosh.edu

twice. Let l and r denote the leftmost and rightmost vertices of P, respectively. Consider the "top half" of the boundary of P by walking along the boundary clockwise from l to r. We call this the *ceiling* of P. Similarly, we obtain the *floor* of P by walking clockwise along the boundary from r to l. Notice that both the ceiling and the floor are x-monotone polygonal chains—that is a vertical line intersects it in at most one point. Krohn and Nilsson [15] give a polynomial-time constant factor approximation algorithm for point-guarding monotone polygons. They also proved point-guarding and vertex guarding a monotone polygon is NP-hard [14, 15].

α-Floodlights. Motivated by the fact that many cameras and other sensors generally are not able to sense in 360°, previous works have considered the problem when guards have a fixed sensing angle α for some $0 < \alpha \leq 360$. This problem is often referred to as the *α-floodlight problem*. 180°-floodlights are sometimes referred to as *half-guards*. Some of the work on this problem has involved proving necessary and sufficient bounds on the number of α-floodlights required to guard (or illuminate) an n vertex simple polygon P, where floodlights are anchored at vertices in P and no vertex is assigned more than one floodlight, see for example [19, 20]. It is known that computing a minimum cardinality set of α-floodlights to illuminate a simple polygon P is APX-hard for both the point-guard and vertex guard variants [1].

1.2 Our Contribution

In this paper, we consider guarding monotone polygons with half-guards that can see in one direction, namely to the right. Let $p.x$ denote the x-coordinate of a point p. We modify visibility in that the definition of *sees* is changed to: a point p sees a point q if the line segment $\overline{pq}$ does not go outside of P and $p.x \leq q.x$.

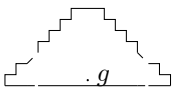


Figure 1

Our main result is to give a polynomial-time constant factor approximation algorithm for point-guarding monotone polygons with half-guards that see to the right. Krohn and Nilsson [15] obtained a similar result using full guards, but the algorithms are quite different and many new observations are needed to obtain the algorithm given in this paper. Indeed, note that there are monotone polygons P that can be covered with one full guard that require $\Omega(n)$ guards considered in this paper (see for example, Figure 1).

The remainder of the paper is sectioned as follows: in Section 2, we provide a 4-approximation for point-guarding a monotone polygon using half-guards where we wish to guard only the vertices of the polygon. In Section 3, we extend the algorithm given in Section 2 to provide a 20-approximation for guarding the entire boundary of the polygon. In Section 4, we extend the algorithm given in Section 3 to provide a 40-approximation for guarding the entire polygon.

2 Guarding the Vertices

In this section, we give a polynomial-time 4-approximation algorithm for guarding the vertices of a monotone polygon P with guards that see to the right. We do this by first giving a 2-approximation algorithm for guarding the vertices of the ceiling. We then have the algorithm for the entire polygon by symmetrically applying the ceiling algorithm to the vertices of the floor, giving a 4-approximation for guarding all vertices of P.

Before we describe the algorithm, we provide some preliminary definitions. The rightmost vertex that a point p sees on the ceiling is denoted $R_c(p)$. A vertical line that goes through a point p is denoted l_p. Given two points p, q in P such that $p.x < q.x$, we use (p, q) to denote the points r such that $p.x < r.x < q.x$. Similarly, we use $(p, q]$ to denote points r such that $p.x < r.x \leq q.x$, etc.

2.1 Ceiling Guard Algorithm

We first give a high level overview of our algorithm for guarding the vertices of the ceiling. Any feasible solution must place a guard at the leftmost vertex of the ceiling (or this vertex will not be covered). We begin by placing a guard here, and we iteratively place guards from left to right. When placing a new guard, we let S denote the guards we have already placed, and we let p denote the leftmost vertex on the ceiling that is not seen by a guard in S. The next guard we place, g, will lie somewhere on the line l_p. We initially place g at the intersection of l_p and the floor, and we slide g vertically along l_p until some condition holds. Let $C(S)$ denote the set of ceiling vertices seen by S, and let $C(g)$ denote the set of ceiling vertices seen by g. Note that as g slides up l_p, ceiling vertices may join and leave $C(g)$ as the vertices on the ceiling that g sees may change. Our algorithm locks in a final position for the guard g by sliding it vertically along l_p until moving it any higher will cause g to no longer see some vertex in $C(g) \setminus C(S)$ (the ceiling vertices seen by g that are not seen by any previously placed guard). See, for example, Figure 2. In this figure, initially g does not see v, but as we slide g up the line l_p, v becomes a new vertex in $C(g) \setminus C(S)$. If we slide g up any higher than as depicted in the figure, then g would no longer see v, and therefore we lock in the position of g. We then add g to S, and we repeat this procedure until all vertices on the ceiling are guarded. The formal ceiling guarding algorithm is shown in Algorithm 1.

This algorithm clearly returns a set of guards that sees every vertex on the ceiling. All steps, except the sliding step, can be trivially done in polynomial time. Since the polygon is simple, any vertex that is seen by a point on l_p must by seen by a contiguous line segment of l_p. We

Algorithm 1 Ceiling Guard

1: **procedure** CEILING GUARD(monotone polygon P)
2: $S \leftarrow \{s\}$ such that s is placed at the leftmost point l.
3: **while** there is an unseen ceiling vertex **do**
4: Let p be the leftmost ceiling vertex that is currently unseen by any guards in S. Initially place a guard g where l_p intersects the floor. Slide g up until it stops seeing some vertex $v \in C(g) \setminus C(S)$ on the ceiling. Place g at the point on l_p just before it stopped seeing v.
5: $S \leftarrow S \cup \{g\}$.
6: **end while**
7: **return** S
8: **end procedure**

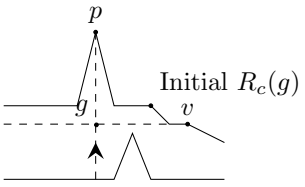


Figure 2: Moving g vertically on l_p until it stops seeing some ceiling vertex v.

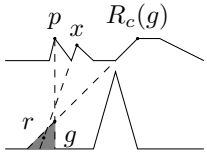


Figure 3: A point r is g-ceiling-dominated.

compute these line segments for each vertex that is seen from l_p. This takes $O(n^2)$ time. In the sliding step, we only need to consider the top and bottom points of these line segments. There are at most $2n$ of these points on l_p to consider during the sliding step. Therefore, Algorithm 1 runs in polynomial time. It remains to prove the approximation ratio.

2.2 Proof of Approximation

In this subsection, we will show that Algorithm 1 will place no more than 2 times the number of guards in the optimal solution. An optimal solution $\mathcal{O}$ is a minimum cardinality guard set such that for any vertex v on the ceiling of P, there exists some $g \in \mathcal{O}$ that sees v. The argument will be a charging argument; every guard we place will be charged to a guard in $\mathcal{O}$ in a manner such that each guard in $\mathcal{O}$ will be charged at most twice.

We now provide a key lemma that will be used to show that we do not charge a guard of $\mathcal{O}$ more than twice. Consider some guard g chosen by the algorithm, and let S_g denote the set of guards consisting of g and every guard that we chose prior to g. For any point r in P, we say that r is *g-ceiling-dominated* if every ceiling vertex to the right of g seen by r is also seen by some $g' \in S_g$.

Lemma 1 *Consider a guard g placed in step 5 of the algorithm. A point r that is to the left of g and below the ray $\overrightarrow{R_c(g)g}$ is g-ceiling-dominated.*

Proof. Let x denote a ceiling vertex that is seen by r to the right of p (inclusive). The proof will consider two cases depending on if the line segment $\overline{rx}$ intersects l_p below or above g. In both scenarios, we prove that x must be seen by some guard in S_g. The lemma immediately follows. We will again let S denote the guards placed prior to g (i.e., $S = S_g \setminus \{g\}$).

First suppose that $\overline{rx}$ intersects l_p at g or below g. While sliding g up l_p, it would have passed through the intersection point of $\overline{rx}$ and l_p, and therefore g saw x at this point in time. If x is not seen by some guard in S, then the final placement of g must see x as well. If the final placement of g is such that g does not see x, then it must be that x was already seen by some guard in S. Therefore it must be that x is seen by some guard in S_g.

Now let x be such that $\overline{rx}$ intersects l_p strictly above the final placement of g. For this to be the case, it must be that x is in $[p, R_c(g)]$ since r is left of g and is below $\overrightarrow{R_c(g)g}$. We will show that g must also see x. If g does not see x then either the floor must "pierce" $\overline{gx}$ from below or the ceiling must pierce $\overline{gx}$ from above. The floor cannot block g from x because it would also block g from $R_c(g)$, and the ceiling cannot block g from x because otherwise it would also block r from x. Therefore g sees x. See Figure 3. □

We now describe our method of charging the guards chosen by our algorithm to the guards in $\mathcal{O}$. When we place a guard g, we will charge g to some guard in $\mathcal{O}$ to the left of g. We prove by induction that when we place our guard g, we can charge g to a guard in $\mathcal{O}$ that has previously been charged at most once.

Base case: Our algorithm places a guard at the leftmost point and there must also be an optimal guard at this point. If this were not the case, then the optimal solution would not have guarded the leftmost point. We charge our guard to this optimal guard. Our base case then considers the first guard our algorithm places in the while loop. Consider the placement of this guard g. Let p be the first ceiling vertex not seen by the initial guard; the initial optimal guard also does not see p. Therefore, the optimal solution must have an uncharged guard o on l_p or to the left of l_p, and we can charge g to o.

Inductive Step: We assume the inductive hypothesis holds true for the first $k-1$ iterations of the while loop and we are on the k^{th} iteration. We consider the placement of guard g and the vertical line l_p that it is on. Let f denote the guard placed in iteration $k-1$, and let l_f denote the vertical line it is on, see Figure 4. We consider two cases depending on whether there is a guard in $\mathcal{O}$ that is in $(f, g]$.

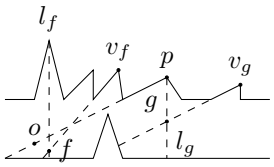


Figure 4: An optimal guard o to the left of l_f that can see p.

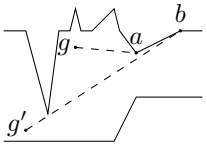


Figure 5: Guard g does not see b since a is blocking it; $\{a, b\}$ is not entirely seen.

Case 1: There is an $o \in \mathcal{O}$ in $(f, g]$. All previously charged guards $o' \in \mathcal{O}$ satisfy $o'.x \leq f.x$, and therefore o has not had a guard charged to it. We charge g to o.

Case 2: There is no optimal guard in $(f, g]$. In this case, we show that an optimal guard o' is f-ceiling-dominated when f was placed and o' was not f'-ceiling-dominated for any guard f' placed by the algorithm prior to placing f. We charge g to o'. An optimal guard o' can only be charged by this procedure in the iteration *immediately after* it becomes dominated, and therefore it can only be charged once by this case.

Since there is no optimal guard in $(f, g]$, p must be seen by an optimal guard o such that $o.x \leq f.x$. We will show that the line segment $\overline{op}$ must cross the line l_f strictly above f. If it crosses l_f through or below the final placement of f, then f must have seen p at some point while sliding vertically. It follows that either f sees p or some previous guard sees p, a contradiction. Therefore it must be that $\overline{op}$ crosses l_f strictly above f.

When we placed f, it slid vertically until it would have lost sight of some ceiling vertex v_f that was not seen by any previous guard. We will first show that v_f must be to the left of p. Since f does not see p, some part of P must block f from seeing p. The ceiling cannot block f from p because it would also block o from seeing p (since $\overline{op}$ crosses above f). If v_f were to the right of p, then the floor would not be able to block f from p either because it would also block f from seeing v_f. Therefore it must be that v_f is to the left of p.

Now let $o' \in \mathcal{O}$ denote an optimal guard that sees v_f. We will show that o' is f-ceiling-dominated by Lemma 1. Since we are in Case 2, it must be that $o'.x \leq f.x$. It remains to show that o' is below the ray $\overrightarrow{R_c(f)f}$. We will do this by showing that o' is below the ray $\overrightarrow{v_f f}$, and therefore must also be below $\overrightarrow{R_c(f)f}$ if $v_f \neq R_c(f)$. It follows that o' is below $\overrightarrow{v_f f}$ due to the manner in which the location of f was chosen. The point f stopped sliding when it would have lost sight of v_f, and in particular, it must be that the *ceiling* would prevent f from seeing v_f because f is only sliding vertically. Therefore any point in P that is to the left of f and is above the ray $\overrightarrow{v_f f}$ must also be blocked from seeing v_f. Since o' sees v_f, it must be that o' is below the ray $\overrightarrow{v_f f}$, satisfying the conditions of Lemma 1, and therefore is f-ceiling-dominated.

We charge g to o'. Since o' sees v_f (v_f was not guarded by our algorithm until we placed f), it must be that o' was not dominated prior to the placement of f. Thus o' can be charged a guard by this procedure only once.

This completes our charging scheme, which charges each guard g picked by our algorithm to an optimal guard in $\mathcal{O}$. We have shown that each guard in $\mathcal{O}$ can be charged at most once in Case 1 and at most once in Case 2, and therefore our algorithm returns a set of guards of size at most $2|\mathcal{O}|$. By running this algorithm on the ceiling and symmetrically applying the algorithm on the floor, we have the following theorem.

Theorem 2 *There is a polynomial-time 4-approximation algorithm for point-guarding the vertices of a monotone polygon with half-guards.*

3 Guarding the Boundary

In the previous section, we provided an algorithm to guard the vertices of the polygon with at most 4OPT guards. However, the algorithm is not guaranteed to guard the entire boundary, see Figures 5 and 6 for example. We will now provide a modification of Algorithm 1 to ensure that the entire boundary is seen. Similar to the last section, we begin by providing a polynomial-time 10-approximation algorithm that will cover the entire ceiling. This algorithm can be symmetrically applied to the floor to then give a polynomial-time 20-approximation algorithm that covers the entire boundary of P.

Suppose we have a guard set S that covers all of the vertices of the ceiling, and consider some edge $\{a, b\}$ on the ceiling such that a is to the left of b. If one guard $g \in S$ sees both a and b, then it is easy to see that g sees the entire edge $\{a, b\}$. Therefore, if some edge is not completely covered by S, then it must be that every guard that sees a does not see b (and vice versa). At a high level, our algorithm for guarding the entire ceiling begins by covering the vertices of the ceiling similarly to Algorithm 1. If at some point in time during this process we have that our current guard set sees both vertices of an edge but does not cover the entire edge, then we place additional guards to ensure that the entire edge is indeed covered.

For ease of description, we maintain two different sets of guards: S and S'. S is the set of guards chosen to cover vertices (similar to Algorithm 1), and S' is the set of guards chosen to fill in a missing gap on some edge. To prove the approximation ratio, we charge each guard in S' to one of the guards in S. We prove that each guard of S will have at most four guards of S' charged to it. Since each guard of $\mathcal{O}$ has at most two guards of S charged to it, we then have that each guard in $\mathcal{O}$ has at most 10 guards of $S \cup S'$ charged to it, giving us the approximation ratio.

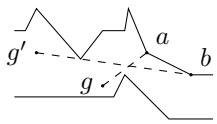


Figure 6: Guard g does not see b since the floor is blocking it; $\{a, b\}$ is not completely seen.

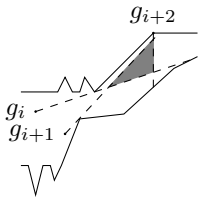


Figure 7: An example of a middle pocket.

Suppose we have just added a guard g to S so as to cover some vertices of the ceiling. Let E denote the set of edges such that for every edge in the set, the endpoints are seen by some guard in S but part of the edge is unseen (see step 6 of Algorithm 2). We prove that each edge $e = \{a, b\} \in E$ will fall into one of the four following cases, and each case is handled differently.
Case 1: g sees a and some guard in S sees b. a is blocking g from b. We note in this case that no other ceiling point can block g from b, see Figure 5.
Case 2: g sees a and some guard in S sees b. g does not see b because the floor is blocking g from b, see Figure 6.
Case 3: g sees b and some guard in S sees a. g does not see a because a is to the left of g.
Case 4: g sees b and some guard in S sees a. g does not see a because the ceiling is blocking g from a.

Due to lack of space, we omit the proofs that these four cases are exhaustive, and the algorithm places guards into S' in a way so that every edge in E is completely seen by S'. Moreover, we prove we can charge each guard of S' to a guard of S such that any guard in S is charged at most one guard for each of the four cases.

Now we have that each edge will be covered as soon as S sees both endpoints of the edge. At the end of the algorithm we have that every vertex on the ceiling is seen by a guard in S, we will have that $S \cup S'$ covers the entire ceiling. Each guard in S is charged at most one guard in S' per case, and therefore $|S'| \leq 4|S|$. We already had that $|S| \leq 2|\mathcal{O}|$, and thus $|S'| \leq 8|\mathcal{O}|$. Our final guarding set then satisfies $|S \cup S'| \leq 10|\mathcal{O}|$. By applying the algorithm on the ceiling and the floor, we have a 20-approximation algorithm for guarding the entire boundary of P.

Theorem 3 *There is a polynomial-time 20-approximation algorithm for point-guarding the boundary of a monotone polygon with half-guards.*

4 Guarding the Entire Polygon

Algorithm 2 ensures that the entire boundary of the polygon is seen. It is possible that parts of the interior of the polygon are unseen, see Figure 7. After Algorithm 2 is run, we run the final algorithm to ensure that the entire polygon is guarded. In this algorithm, we let S denote all guards returned by Algorithm 2. For any point p on the boundary of P, we let p^- denote a point on the boundary to the left of p that is "infinitesimally close" to p. A *middle pocket* is defined as an unseen part of the polygon that is not touching the boundary of the polygon. See Figure 7. The final algorithm will ensure that all middle pockets are guarded. The algorithm processes S from left to right and considers two consecutive guards. The algorithm places a guard at a strategic location ensuring

Algorithm 2 Modified Ceiling Guard

1: $S \leftarrow \{s\}$ such that s is placed at leftmost point l.
2: Let S' denote the initially empty set of "extra" guards we add to cover edges.
3: **while** there exists an unseen point on the ceiling of P from our guards in S **do**
4: Let p be the leftmost ceiling vertex that is currently unseen by any guards in S. Place a guard g where l_p intersects the floor. Slide g up until it stops seeing some vertex $v \in C(g)$ on the ceiling. Place g at the point just before it stopped seeing v. $S \leftarrow S \cup \{g\}$.
5: Let o be the vertex to the left of p on the ceiling.
6: Let E be the set of ceiling edges such that for every edge $e = \{a, b\} \in E$, g sees exactly one vertex on the edge, S only sees the other vertex on the edge, and S does not see the entire edge e.
7: **if** g sees some vertex a such that a is the leftmost vertex of some edge in E **then**
8: Let a_r be the rightmost a vertex that g sees, such that a is the leftmost vertex of some edge $e \in E$. Let b_r be vertex immediately to the right of a_r on the ceiling.
9: **if** a blocks g from b_r **then** ▷ Case 1
10: Let $g_{b_r} \in S$ be the leftmost guard that sees b_r. Draw a line l from g_{b_r} to b_r and place a guard g' at the point where l intersects l_p. Remove all edges in E that g' sees. $S' \leftarrow S' \cup \{g'\}$.
11: **end if**
12: **if** the floor blocks g from b_r **then** ▷ Case 2
13: Remove all edges in E that g' sees, and place a guard g' at a_r. $S' \leftarrow S' \cup \{g'\}$.
14: **end if**
15: **end if**
16: **if** a part of $\overline{ab}$ is not seen by S **then** ▷ Case 3
17: Remove $\{a, b\}$ from E, and place a guard g' at a. $S' \leftarrow S' \cup \{g'\}$.
18: **end if**
19: **while** E is not empty **do** ▷ Case 4
20: Remove $e = \{a, b\}$ from E and place a guard g' at a. $S' \leftarrow S' \cup \{g'\}$.
21: **end while**
22: **end while**
23: **return** $S \cup S'$.

that any part of the polygon that is unseen between the consecutive guards is now seen. The following lemma is proved in [15].

Lemma 4 *Consider a middle pocket p of a partial guard set S in a monotone polygon. Let r be the leftmost point in p. Not all guards of S can be to the left of r.*

Note that Lemma 4 is proved in [15] for guards that see in all directions, but it also trivially applies to our scenario since it deals with a region of P that lies entirely to the right of a set of guards.

Lemma 5 *Any middle pockets between 2 consecutive guards can be guarded with 1 guard.*

After the final algorithm terminates, the entire boundary is seen and all middle pockets are guarded; thus the entire polygon is seen. Note that each extra guard that the final algorithm places can be charged to S_i, and therefore each guard output by Algorithm 2 will be charged at most one guard placed by the final algorithm. We have the following theorem.

Theorem 6 *There is a 40-approximation algorithm for point-guarding a monotone polygon with half-guards.*

References

[1] Ahmed Abdelkader, Ahmed Saeed, Khaled A. Harras, and Amr Mohamed. The inapproximability of illuminating polygons by α-floodlights. In *CCCG*, pages 287–295, 2015.

[2] Alok Aggarwal. *The art gallery theorem: its variations, applications and algorithmic aspects*. PhD thesis, 1984.

[3] Greg Aloupis, Jean Cardinal, Sébastien Collette, Stefan Langerman, David Orden, and Pedro Ramos. Decomposition of multiple coverings into more parts. In *SODA*, pages 302–310, 2009.

[4] Boris Aronov, Esther Ezra, and Micha Sharir. Small-size epsilon-nets for axis-parallel rectangles and boxes. *SIAM J. Comp.*, 39(7):3248–3282, July 2010.

[5] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient probabilistically checkable proofs and applications to approximations. In *STOC*, pages 294–304, 1993.

[6] Björn Brodén, Mikael Hammar, and Bengt J. Nilsson. Guarding lines and 2-link polygons is APX-hard. In *CCCG*, pages 45–48, 2001.

[7] Ajay Deshpande, Taejung Kim, Erik D. Demaine, and Sanjay E. Sarma. A pseudopolynomial time O(log n)-approximation algorithm for art gallery problems. In *WADS*, pages 163–174, 2007.

[8] Alon Efrat and Sariel Har-Peled. Guarding galleries and terrains. *Information Processing Letters*, 100(6):238–245, 2006.

[9] Stephan Eidenbenz. Inapproximability results for guarding polygons without holes. In *ISAAC*, pages 427–436, 1998.

[10] Uriel Feige, Magnús M. Halldórsson, Guy Kortsarz, and Aravind Srinivasan. Approximating the domatic number. *SIAM Journal on Computing*, 32(1):172–195, 2003.

[11] Subir Kumar Ghosh. On recognizing and characterizing visibility graphs of simple polygons. *Discrete & Computational Geometry*, 17(2):143–162, 1997.

[12] David S. Johnson. Approximation algorithms for combinatorial problems. STOC, pages 38–49. ACM, 1973.

[13] James King and David G. Kirkpatrick. Improved approximation for guarding simple galleries from the perimeter. *Discrete & Computational Geometry*, 46(2):252–269, 2011.

[14] Erik Krohn and Bengt J. Nilsson. The complexity of guarding monotone polygons. In *CCCG*, pages 167–172, 2012.

[15] Erik Krohn and Bengt J. Nilsson. Approximate guarding of monotone and rectilinear polygons. *Algorithmica*, 66(3):564–594, 2013.

[16] D. T. Lee and A. K. Lin. Computational complexity of art gallery problems. *IEEE Trans. Inform. Theory*, 32(2):276–282, March 1986.

[17] Carsten Lund and Mihalis Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, September 1994.

[18] Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. STOC, pages 475–484. ACM, 1997.

[19] Bettina Speckmann and Csaba D. Tóth. Allocating vertex π-guards in simple polygons via pseudo-triangulations. *Discrete & Computational Geometry*, 33(2):345–364, 2005.

[20] Csaba D. Tóth. Art galleries with guards of uniform range of vision. *Computational Geometry*, 21(3):185 – 192, 2002.

[21] Kasturi R. Varadarajan. Epsilon nets and union complexity. In *Symposium on Computational Geometry*, pages 11–16, 2009.

Sharing a pizza: bisecting masses with two cuts

Luis Barba* † Patrick Schnider*

Abstract

Assume you have a pizza consisting of four ingredients (e.g. bread, tomatoes, cheese and olives) that you want to share with your friend. You want to do this fairly, meaning that you and your friend should get the same amount of each ingredient. How many times do you need to cut the pizza so that this is possible? We will show that two straight cuts always suffice. More formally, we will show the following extension of the well-known Ham-sandwich theorem: Given four mass distributions in the plane, they can be simultaneously bisected with two lines. That is, there exist two oriented lines with the following property: let R_1^+ be the region of the plane that lies to the positive side of both lines and let R_2^+ be the region of the plane that lies to the negative side of both lines. Then $R^+ = R_1^+ \cup R_2^+$ contains exactly half of each mass distribution. Additionally, we prove that five mass distributions in $\mathbb{R}^3$ can be simultaneously bisected by two planes.

1 Introduction

The famous *Ham-sandwich theorem* (see e.g. [11, 14]) states that any d mass distributions in $\mathbb{R}^d$ can be simultaneously bisected by a hyperplane. In particular, a two-dimensional sandwich consisting of bread and ham can be cut with one straight cut in such a way that each side of the cut contains exactly half of the bread and half of the ham. However, if two people want to share a pizza, this result will not help them too much, as pizzas generally consist of more than two ingredients. There are two options to overcome this issue: either they don't use a straight cut, but cut along some more complicated curve, or they cut the pizza more than once. In this paper we investigate the latter option. In particular we show that a pizza with four ingredients can always be shared fairly using two straight cuts. See Figure 1 for an example.

To phrase it in mathematical terms, we show that four mass distributions in the plane can be simultaneously bisected with two lines. A precise definition of what bisecting with n lines means is given in the Preliminaries. We further show that five mass distributions in $\mathbb{R}^3$ can be simultaneously bisected by two planes. These two main results are proven in Section 2. In Section 3 we go back to the two-dimensional case and add more restrictions on the lines. In Section 4 we look at the general case of bisecting mass distributions in $\mathbb{R}^d$ with n hyperplanes, and show an upper bound of nd mass distributions that can be simultaneously bisected this way. We conjecture that this bound is tight, that is, that any nd mass distributions in $\mathbb{R}^d$ can be simultaneously bisected with n hyperplanes. For $d = 1$, this is the well-known *Necklace splitting problem*, for which an affirmative answer to our conjecture is known [6, 11]. So, our general problem can be seen as both a generalization of the Ham-sandwich theorem for more than one hyperplane, as well as a generalization of the Necklace splitting problem to higher dimensions.

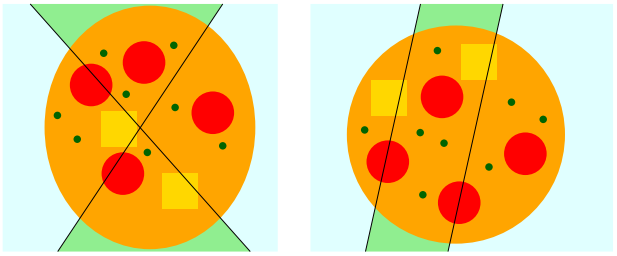


Figure 1: Sharing a (not necessarily round) pizza fairly with two cuts. One person gets the parts in the light blue region, the other person gets the parts in the green region.

Additionally, our results add to a long list of results about partitions of mass distributions, starting with the already mentioned Ham-sandwich theorem. A generalization of this is the polynomial Ham-sandwich theorem, which states that any $\binom{n+d}{d} - 1$ mass distributions in $\mathbb{R}^d$ can be simultaneously bisected by an algebraic surface of degree n [14]. Applied to the problem of sharing a pizza, this result gives an answer on how complicated the cut needs to be, if we want to use only a single (possibly self-intersecting) cut.

Several results are also known about equipartitions of mass distributions into more than two parts. A straightforward application of the 2-dimensional Ham-sandwich theorem is that any mass distribution in the plane can be partitioned into four equal parts with 2 lines. It is also possible to partition a mass distribution in $\mathbb{R}^3$ into 8 equal parts with three planes, but for $d \geq 5$, it is not always possible to partition a mass distribution

*Department of Computer Science, ETH Zürich, {luis.barba, patrick.schnider}@inf.ethz.ch

†Partially supported by the ETH Postdoctoral Fellowship

into 2^d equal parts using d hyperplanes [5]. The case $d = 4$ is still open. A result by Buck and Buck [4] states that a mass distribution in the plane can be partitioned into 6 equal parts by 3 lines passing through a common point. Several results are known about equipartitions in the plane with *k-fans*, i.e., k rays emanating from a common point. Note that 3 lines going through a common point can be viewed as a 6-fan, thus the previously mentioned result shows that any mass partition in the plane can be equipartitioned by a 6-fan. Motivated by a question posed by Kaneko and Kano [8], several authors have shown independently that 2 mass distributions in the plane can be simultaneously partitioned into 3 equal parts by a 3-fan [2, 7, 12]. The analogous result for 4-fans holds as well [1]. Partitions into non-equal parts have also been studied [15]. All these results give a very clear description of the sets used for the partitions. If we allow for more freedom, much more is possible. In particular, Soberón [13] and Karasev [9] have recently shown independently that any d mass distributions in $\mathbb{R}^d$ can be simultaneously equipartitioned into k equal parts by k convex sets. The proofs of all of the above mentioned results rely on topological methods, many of them on the famous Borsuk-Ulam theorem and generalizations of it. For a deeper overview of these types of arguments, we refer to Matoušek's excellent book [11].

Preliminaries

A *mass distribution* μ on $\mathbb{R}^d$ is a measure on $\mathbb{R}^d$ such that all open subsets of $\mathbb{R}^d$ are measurable, $0 < \mu(\mathbb{R}^d) < \infty$ and $\mu(S) = 0$ for every lower-dimensional subset S of $\mathbb{R}^d$. Let $\mathcal{L}$ be a set of oriented hyperplanes. For each $\ell \in \mathcal{L}$, let ℓ^+ and ℓ^- denote the positive and negative side of ℓ, respectively (we consider the sign resulting from the evaluation of a point in these sets into the linear equation defining ℓ). For every point $p \in \mathbb{R}^d$, define $\lambda(p) := |\{\ell \in \mathcal{L} \mid p \in \ell^+\}|$ as the number of hyperplanes that have p in their positive side. Let $R^+ := \{p \in \mathbb{R}^d \mid \lambda(p) \text{ is even}\}$ and $R^- := \{p \in \mathbb{R}^d \mid \lambda(p) \text{ is odd}\}$. We say that $\mathcal{L}$ *bisects* a mass distribution μ if $\mu(R^+) = \mu(R^-)$. For a family of mass distributions $\mu_1, \ldots, \mu_k$ we say that $\mathcal{L}$ *simultaneously bisects* $\mu_1, \ldots, \mu_k$ if $\mu_i(R^+) = \mu_i(R^-)$ for all $i \in \{1, \ldots, k\}$.

More intuitively, this definition can also be understood the following way: if C is a cell in the hyperplane arrangement induced by $\mathcal{L}$ and C' is another cell sharing a facet with C, then C is a part of R^+ if and only if C' is a part of R^-. See Figure 2 for an example.

Let $g_i(x) := a_{i,1}x_1 + \ldots + a_{i,d}x_d + a_{i,0} \geq 0$ be the linear equation describing ℓ_i^+ for $\ell_i \in \mathcal{L}$. Then the following is yet another way to describe R^+ and R^-: a point $p \in \mathbb{R}^d$ is in R^+ if $\prod_{\ell_i \in \mathcal{L}} g_i(p) \geq 0$ and it is in R^- if $\prod_{\ell_i \in \mathcal{L}} g_i(p) \leq 0$. That is, if we consider the union of the hyperplanes in $\mathcal{L}$ as an oriented algebraic surface of degree $|\mathcal{L}|$, then R^+ is the positive side of this surface and R^- is the negative side.

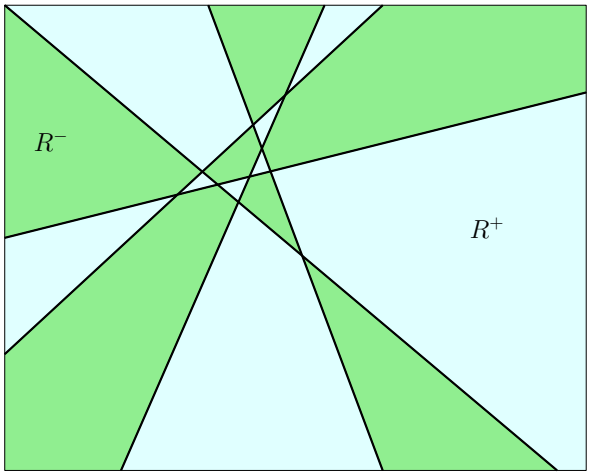


Figure 2: The regions R^+ (light blue) and R^- (green).

Note that reorienting one line just maps R^+ to R^- and vice versa. In particular, if a set $\mathcal{L}$ of oriented hyperplanes simultaneously bisects a family of mass distributions $\mu_1, \ldots, \mu_k$, then so does any set $\mathcal{L}'$ of the same hyperplanes with possibly different orientations. Thus we can ignore the orientations and say that a set $\mathcal{L}$ of (undirected) hyperplanes simultaneously bisects a family of mass distributions if some orientation of the hyperplanes does.

2 Two Cuts

In this section we will look at simultaneous bisections with two lines in $\mathbb{R}^2$ and with two planes in $\mathbb{R}^3$. Both proofs rely on the famous Borsuk-Ulam theorem [3], which we will use in the version of *antipodal mappings*. An antipodal mapping is a continuous mapping $f : S^d \to \mathbb{R}^d$ such that $f(-x) = -f(x)$ for all $x \in S^d$.

Theorem 1 (Borsuk-Ulam theorem [11]) *For every antipodal mapping $f : S^d \to \mathbb{R}^d$ there exists a point $x \in S^d$ satisfying $f(x) = 0$.*

The proof of the Ham-sandwich theorem can be derived from the Borsuk-Ulam theorem in the following way. Let μ_1 and μ_2 be two mass distributions in $\mathbb{R}^2$. For a point $p = (a, b, c) \in S^3$, consider the equation of the line $ax + by + c = 0$ and note that it defines a line in the plane parametrized by the coordinates of p. Moreover, it splits the plane into two regions, the set $R^+(p) = \{(x, y) \in \mathbb{R}^2 : ax + by + c \geq 0\}$ and the set $R^-(p) = \{(x, y) \in \mathbb{R}^2 : ax + by + c \leq 0\}$. Thus, we can define two functions $f_i := \mu_i(R^+(p)) - \mu_i(R^-(p))$ that together yield a function $f : S^2 \to \mathbb{R}^2$ that is continuous and antipodal. Thus, by the Borsuk-Ulam theorem, there is a point $p = (a, b, c) \in S^2$, such that

$f_i(-p) = -f_i(p)$ for $i \in \{1, 2\}$, which implies that the line $ax+by+c = 0$ defined by p is a Ham-sandwich cut. In this paper, we use variants of this proof idea to obtain simultaneous bisections by geometric objects that are parametrized by points in S^d. The main difference is that we replace some of the f_i's by other functions, whose vanishing enforces specific structural properties on the resulting bisecting object. We are now ready to prove our first main result:

Theorem 2 *Let $\mu_1, \mu_2, \mu_3, \mu_4$ be four mass distributions in $\mathbb{R}^2$. Then there exist two lines ℓ_1, ℓ_2 such that $\{\ell_1, \ell_2\}$ simultaneously bisects $\mu_1, \mu_2, \mu_3, \mu_4$.*

Proof. For each $p = (a, b, c, d, e, g) \in S^5$ consider the bivariate polynomial $c(p)(x, y) = ax^2+by^2+cxy+dx+ey+g$. Note that $c(p)(x, y) = 0$ defines a conic section in the plane. Let $R^+(p) := \{(x, y) \in \mathbb{R}^2 \mid c(p)(x, y) \geq 0\}$ be the set of points that lie on the positive side of the conic section and let $R^-(p) := \{(x, y) \in \mathbb{R}^2 \mid c(p)(x, y) \leq 0\}$ be the set of points that lie on its negative side. Note that for $p = (0, 0, 0, 0, 0, 1)$ we have $R^+(p) = \mathbb{R}^2$ and $R^-(p) = \emptyset$, and vice versa for $p = (0, 0, 0, 0, 0, -1)$. Also note that $R^+(-p) = R^-(p)$. We now define four functions $f_i : S^5 \to \mathbb{R}$ as follows: for each $i \in \{1, \ldots, 4\}$ define $f_i := \mu_i(R^+(p)) - \mu_i(R^-(p))$. From the previous observation it follows immediately that $f_i(-p) = -f_i(p)$ for all $i \in \{1, \ldots, 4\}$ and $p \in S^5$. It can also be shown that the functions are continuous, but for the sake of readability we postpone this step to the end of the proof. Further let

$$A(p) := \det \begin{pmatrix} a & c/2 & d/2 \\ c/2 & b & e/2 \\ d/2 & e/2 & g \end{pmatrix}.$$

It is well-known that the conic section $c(p)(x, y) = 0$ is degenerate if and only if $A(p) = 0$. Furthermore, being a determinant of a 3×3-matrix, A is continuous and $A(-p) = -A(p)$. Hence, setting $f_5(p) := A(p)$, $f := (f_1, \ldots, f_5)$ is an antipodal mapping from S^5 to $\mathbb{R}^5$, and thus by the Borsuk-Ulam theorem, there exists p^* such that $f(p^*) = 0$. For each $i \in \{1, \ldots, 4\}$ the condition $f_i(p^*) = 0$ implies by definition that $\mu_i(R^+(p^*)) = \mu_i(R^-(p^*))$. The condition $f_5(p^*) = 0$ implies that $c(p)(x, y) = 0$ describes a degenerate conic section, i.e., two lines, a single line of multiplicity 2, a single point or the empty set. For the latter three cases, we would have $R^+(p^*) = \mathbb{R}^2$ and $R^-(p^*) = \emptyset$ or vice versa, which would contradict $\mu_i(R^+(p^*)) = \mu_i(R^-(p^*))$. Thus $f(p^*) = 0$ implies that $c(p)(x, y) = 0$ indeed describes two lines that simultaneously bisect $\mu_1, \mu_2, \mu_3, \mu_4$.

It remains to show that f_i, is continuous for $i \in \{1, \ldots, 4\}$. To that end, we will show that $\mu_i(R^+(p))$ is continuous. The same arguments apply to $\mu_i(R^-(p))$, which then shows that f_i being the difference of two continuous functions is continuous. So let $(p_n)_{n=1}^{\infty}$ be a sequence of points in S^5 converging to p. We need to show that $\mu_i(R^+(p_n))$ converges to $\mu_i(R^+(p))$. If a point q is not on the boundary of $R^+(p)$, then for all n large enough we have $q \in R^+(p_n)$ if and only if $q \in R^+(p)$. As the boundary of $R^+(p)$ has dimension 1 and μ_i is a mass distribution we have $\mu_i(\partial R^+(p)) = 0$ and thus $\mu_i(R^+(p_n))$ converges to $\mu_i(R^+(p))$ as required. □

Using similar ideas, we can also prove a result in $\mathbb{R}^3$. For this we first need the following lemma:

Lemma 3 *Let $h(x, y, z)$ be a quadratic polynomial in 3 variables. Then there are antipodal functions $g_1, \ldots, g_4$, each from the space of coefficients of h to $\mathbb{R}$, whose simultaneous vanishing implies that h factors into linear polynomials.*

Proof. Write h as

$$h = (x, y, z, 1) \cdot A \cdot (x, y, z, 1)^T,$$

where A is a 4×4-matrix depending on the coefficients of h. It is well-known that h factors into linear polynomials if and only if the rank of A is at most 2. A well-known sufficient condition for this is that the determinants of all (3×3)-minors of A vanish. There are $\binom{4}{3} = 4$ different (3×3)-minors and for each of them the determinant is an antipodal function. □

With this, we can now prove the following:

Theorem 4 *Let $\mu_1, \ldots, \mu_5$ be five mass distributions in $\mathbb{R}^3$. Then there exist two planes ℓ_1, ℓ_2 such that $\{\ell_1, \ell_2\}$ simultaneously bisects $\mu_1, \ldots, \mu_5$.*

Proof. Similar to the proof of Theorem 2, we map a point $p \in S^9$ to a quadratic polynomial $h(p)(x, y, z)$ (note that a quadratic polynomial in three variables has 10 coefficients). Let $R^+(p) := \{(x, y, z) \in \mathbb{R}^2 \mid h(p)(x, y, z) \geq 0\}$ be the set of points that lie on the positive side of the conic section and let $R^-(p) := \{(x, y, z) \in \mathbb{R}^3 \mid h(p)(x, y) \leq 0\}$ be the set of points that lie on the negative side. For each $i \in \{1, \ldots, 5\}$ define $f_i := \mu_i(R^+(p)) - \mu_i(R^-(p))$. Analogous to the proof of Theorem 2, these functions are continuous and $f_i(-p) = -f_i(p)$. Further let $g_1, \ldots, g_4$ be the four functions constructed in Lemma 3. Then $f := (f_1, \ldots, f_5, g_1, \ldots, g_4)$ is a continuous antipodal mapping from S^9 to $\mathbb{R}^9$. Thus, by the Borsuk-Ulam theorem there exists a point $p^* \in S^9$ such that $f(p^*) = 0$. Analogous to the proof of Theorem 2, the existence of such a point implies the claimed result. □

3 Putting more restrictions on the cuts

In this section, we look again at bisections with two lines in the plane. However, we enforce additional conditions on the lines, at the expense of being only able to simultaneously bisect fewer mass distributions.

Theorem 5 *Let μ_1, μ_2, μ_3 be three mass distributions in $\mathbb{R}^2$. Given any line ℓ in the plane, there exist two lines ℓ_1, ℓ_2 such that $\{\ell_1, \ell_2\}$ simultaneously bisects μ_1, μ_2, μ_3 and ℓ_1 is parallel to ℓ.*

Proof. Assume without loss of generality that ℓ is parallel to the x-axis; otherwise rotate μ_1, μ_2, μ_3 and ℓ to achieve this property. Consider the conic section defined by the polynomial $ax^2+by^2+cxy+dx+ey+g$. If $a = 0$ and the polynomial decomposes into linear factors, then one of the factors must be of the form $\beta y + \gamma$. In particular, the line defined by this factor is parallel to the x-axis. Thus, we can modify the proof of Theorem 2 in the following way: we define f_1, f_2, f_3 and f_5 as before, but set $f_4 := a$. It is clear that f still is an antipodal mapping. The zero of this mapping now implies the existence of two lines simultaneously bisecting three mass distributions, one of them being parallel to the x-axis, which proves the result. □

Another natural condition on a line is that it has to pass through a given point.

Theorem 6 *Let μ_1, μ_2, μ_3 be three mass distributions in $\mathbb{R}^2$ and let q be a point. Then there exist two lines ℓ_1, ℓ_2 such that $\{\ell_1, \ell_2\}$ simultaneously bisects μ_1, μ_2, μ_3 and ℓ_1 goes through q.*

Proof. Assume without loss of generality that q coincides with the origin; otherwise translate μ_1, μ_2, μ_3 and q to achieve this. Consider the conic section defined by the polynomial $ax^2 + by^2 + cxy + dx + ey + g$. If $g = 0$ and the polynomial decomposes into linear factors, then one of the factors must be of the form $\alpha x + \beta y$. In particular, the line defined by this factor goes through the origin. Thus, we can modify the proof of Theorem 2 in the following way: we define f_1, f_2, f_3 and f_5 as before, but set $f_4 := g$. It is clear that f still is an antipodal mapping. The zero of this mapping now implies the existence of two lines simultaneously bisecting three mass distributions, one of them going through the origin, which proves the result. □

We can also enforce the intersection of the two lines to be at a given point, but at the cost of another mass distribution.

Theorem 7 *Let μ_1, μ_2 be two mass distributions in $\mathbb{R}^2$ and let q be a point. Then there exist two lines ℓ_1, ℓ_2 such that $\{\ell_1, \ell_2\}$ simultaneously bisects μ_1, μ_2, and both ℓ_1 and ℓ_2 go through q.*

Proof. Assume without loss of generality that q coincides with the origin; otherwise translate μ_1, μ_2 and q to achieve this. Consider the conic section defined by the polynomial $ax^2 + by^2 + cxy$, i.e., the conic section where $d = e = g = 0$. If this conic section decomposes into linear factors, both of them must be of the form $\alpha x + \beta y = 0$. In particular, both of them pass through the origin. Furthermore, as $d = e = g = 0$, the determinant $A(p)$ vanishes, which implies that the conic section is degenerate. Thus, we can modify the proof of Theorem 2 in the following way: we define f_1, f_2 as before, but set $f_3 := d$, $f_4 := e$ and $f_5 := g$. It is clear that f still is an antipodal mapping. The zero of this mapping now implies the existence of two lines simultaneously bisecting two mass distributions, both of them going through the origin, which proves the result. □

4 The general case

In this section we consider the more general question of how many mass distributions can be simultaneously bisected by n hyperplanes in $\mathbb{R}^d$. We introduce the following conjecture:

Conjecture 1 *Any $n \cdot d$ mass distributions in $\mathbb{R}^d$ can be simultaneously bisected by n hyperplanes.*

For $n = 1$ this is equivalent to the Ham-sandwich theorem. Theorem 2 proves this conjecture for the case $d = n = 2$. We first observe that the number of mass distributions would be tight:

Observation 1 *There exists a family of $n \cdot d + 1$ mass distributions in $\mathbb{R}^d$ that cannot be simultaneously bisected by n hyperplanes.*

Proof. Let $P = \{p_1, \ldots, p_{nd+1}\}$ be a finite point set in $\mathbb{R}^d$ in general position (no $d + 1$ of them on the same hyperplane). Let ϵ be the smallest distance of a point to a hyperplane defined by d other points. For each $i \in \{1, \ldots, nd + 1\}$ define μ_i as the volume measure of $B_i := B_{p_i}(\frac{\epsilon}{2})$. Note that any hyperplane intersects at most d of the B_i's. On the other hand, for a family of n hyperplanes to bisect μ_i, at least one of them has to intersect B_i. Thus, as n hyperplanes can intersect at most $n \cdot d$ different B_i's, there is always at least one μ_i that is not bisected. □

A possible way to prove the conjecture would be to generalize the approach from Section 2 as follows: Consider the n hyperplanes as a highly degenerate algebraic surface of degree n, i.e., the zero set of a polynomial of degree n in d variables. Such a polynomial has $k := \binom{n+d}{d}$ coefficients and can thus be seen as a point on S^{k-1}. In particular, we can define $\binom{n+d}{d} - 1$ antipodal mappings to $\mathbb{R}$ if we want to apply the Borsuk-Ulam theorem. Using $n \cdot d$ of them to enforce the mass distributions to be bisected, we can still afford $\binom{n+d}{d} - nd - 1$ antipodal mappings to enforce the required degeneracies of the surface. There are many conditions known to enforce such degeneracies, but they all require far

too many mappings or use mappings that are not antipodal. Nonetheless the following conjecture implies Conjecture 1:

Conjecture 2 *Let C be the space of coefficients of polynomials of degree n in d variables. Then there exists a family of $\binom{n+d}{d} - nd - 1$ antipodal mappings $g_i : C \to \mathbb{R}$, $i \in \{1, \ldots, \binom{n+d}{d} - nd - 1\}$ such that $g_i(c) = 0$ for all i implies that the polynomial defined by the coefficients c decomposes into linear factors.*

5 Algorithmic remarks

Going back to the planar case, instead of considering four mass distributions $\mu_1, \ldots, \mu_4$, one can think of having four sets of points $P_1, \ldots, P_4 \subset \mathbb{R}^2$. Thus, our problem translates to finding two lines that simultaneously bisect these point sets. The existence of such a bisection follows Theorem 2 as we can always replace each point by a sufficiently small disk and consider their area as a mass distribution.

An interesting question is then to find efficient algorithms to compute such a bisection given any four sets $P_1, \ldots, P_4$ with a total of n points. It is known that linear time algorithms exist for Ham-sandwich cuts of two sets of points in $\mathbb{R}^2$. However, we have not been able to obtain similar results for simultaneous bisections using two lines. A trivial $O(n^5)$ time algorithm can be applied by looking at all pairs of combinatorially different lines. While this running time can be reduced using known data structures, it still goes through $\Theta(n^4)$ different pairs of lines. Finding a better algorithm remains an interesting open question.

Using the algorithms for Ham-sandwich cuts from Lo, Steiger and Matoušek [10], and a Veronese map one can compute a conic section that simultaneously bisects $P_1, \ldots, P_4$ in $O(n^{4-\varepsilon})$ time. It remains open whether this algorithm can be modified to use the last degree of freedom to guarantee the degeneracy of the conic section and achieve $o(n^4)$ time.

References

[1] I. Bárány and J. Matoušek. Equipartition of two measures by a 4-fan. *Discrete & Computational Geometry*, 27(3):293–301, 2002.

[2] S. Bespamyatnikh, D. Kirkpatrick, and J. Snoeyink. Generalizing Ham Sandwich Cuts to Equitable Subdivisions. *Discrete & Computational Geometry*, 24(4):605–622, 2000.

[3] K. Borsuk. Drei Sätze ber die n-dimensionale euklidische Sphäre. *Fundamenta Mathematicae*, 20(1):177–190, 1933.

[4] R. C. Buck and E. F. Buck. Equipartition of convex sets. *Mathematics Magazine*, 22(4):195–198, 1949.

[5] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[6] C. R. Hobby and J. R. Rice. A moment problem in L_1 approximation. *Proceedings of the American Mathematical Society*, 16(4):665–670, 1965.

[7] H. Ito, H. Uehara, and M. Yokoyama. 2-dimension ham sandwich theorem for partitioning into three convex pieces. In *Revised Papers from the Japanese Conference on Discrete and Computational Geometry*, JCDCG '98, pages 129–157, London, UK, 2000. Springer-Verlag.

[8] A. Kaneko and M. Kano. Balanced partitions of two sets of points in the plane. *Computational Geometry*, 13(4):253 – 261, 1999.

[9] R. N. Karasev. Equipartition of several measures. *ArXiv e-prints*, 1011.4762, Nov. 2010.

[10] C.-Y. Lo, J. Matoušek, and W. Steiger. Algorithms for ham-sandwich cuts. *Discrete & Computational Geometry*, 11(1):433–452, 1994.

[11] J. Matoušek. *Using the Borsuk-Ulam Theorem: Lectures on Topological Methods in Combinatorics and Geometry*. Springer Publishing Company, Incorporated, 2007.

[12] T. Sakai. Balanced convex partitions of measures in $\mathbb{R}^2$. *Graphs and Combinatorics*, 18(1):169–192, 2002.

[13] P. Soberón. Balanced convex partitions of measures in $\mathbb{R}^d$. *Mathematika*, 58(01):71–76, 2012.

[14] A. H. Stone and J. W. Tukey. Generalized "sandwich" theorems. *Duke Math. J.*, 9(2):356–359, 06 1942.

[15] R. T. Zivaljevic. Combinatorics and Topology of partitions of spherical measures by 2 and 3 fans. *ArXiv e-prints*, math/0203028, Mar. 2002.

Balanced k-Center Clustering When k Is A Constant

Hu Ding*

Abstract

The problem of constrained k-center clustering has attracted significant attention in the past decades. In this paper, we study balanced k-center cluster where the size of each cluster is constrained by the given lower and upper bounds. The problem is motivated by the applications in processing and analyzing large-scale data in high dimension. We provide a simple nearly linear time 4-approximation algorithm when the number of clusters k is assumed to be a constant. Comparing with existing method, our algorithm improves the approximation ratio and significantly reduces the time complexity. Moreover, our result can be easily extended to any metric space.

1 Introduction

The k-center clustering is a fundamental problem in computer science and has numerous applications in real world. Given a set of points in Euclidean space and a positive integer k, the problem seeks k balls to cover all the points such that the maximum radius of the balls is minimized. Another variant of k-center clustering considers the case that all the given points (vertices) form a metric graph and the centers of the balls are chosen from the vertices. The optimal approximation results appeared in the 80's: Gonazlez [11] and Hochbaum and Shmoys [12] provided a 2-approximation and proved that any approximation ratio $c < 2$ would imply $P = NP$. Besides the classic problem, several variants of k-center clustering with upper [3, 5, 7, 14, 15] or lower [1, 2, 10] bounds on cluster sizes have been extensively studied in recent years. In particular, Ding et al. [9] studied k-center clustering with both upper and lower bounded cluster sizes which is also called *Balanced k-Center Clustering*. Most of existing methods model these constrained k-center clustering problems as linear integer programming and design novel rounding algorithms to obtain constant approximations.

Besides the well studied applications in data analysis and facility location, balanced k-center clustering is particularly motivated by the arising problems in *big data* [4, 6, 8]. For example, we need to dispatch data to multiple machines for processing if the data scale is extremely large; at the same time we have to consider the balancedness, because the machines receiving too much data could be the bottleneck of the system and the ones receiving too little data is not sufficiently energy-efficient.

In this paper, we consider the balanced k-center clustering problem in high dimension and assume that k is a constant. The rationale for the assumption is twofold: k is usually not large in practice (e.g., the data is distributed over less than 10 machines); even if k is large, we can first partition the data into multiple groups and perform balanced k-center clustering for each group with a much smaller k (similar to the manner of *hierarchical clustering* [13]).

Our main result. Given an instance of k-center clustering with upper and lower bounds on cluster sizes, we develop a nearly linear time 4-approximation algorithm. We assume that the dimensionality d is large and the number of clusters k is a constant. The key techniques contains two parts. First, we observe that Gonazlez's algorithm [11] could provide a set of candidates for the k cluster centers and at least one candidate yields 4-approximation (Lemma 1). Secondly, we develop a novel rounding procedure to select the qualified candidate and generate a feasible solution for the balanced k-center clustering (Lemma 2); note that a straightforward idea for the selection task is modeling it as a maximum flow problem but the running time would be at least quadratic. Comparing with the existing method for balanced k-center clustering [9], we improve the approximation ratio from 6 to 4 and significantly reduce the running time via avoiding to solve the large-scale linear programming.

Also, our result can be easily extended to any metric space and the running time depends on the complexity for acquiring the distance between any two points (e.g., the complexity is $O(d)$ in Euclidean space).

Notation. Throughout the paper we denote the input as a set of n points P in $\mathbb{R}^d$ and an integer $k \geq 1$; we further constrain the size of each cluster by the lower and upper bounds L and $U \in \mathbb{Z}^+$ (to ensure that a feasible solution exists, we assume $1 \leq L \leq \lfloor \frac{n}{k} \rfloor \leq \lceil \frac{n}{k} \rceil \leq U \leq n$).

For k-center clustering in $\mathbb{R}^d$, the k cluster centers could be any points in the space (though our 4-approximation solution comes from the input points); for the problem in abstract metric space, the cluster centers are restricted inside the input points.

*Department of Computer Science and Engineering, Michigan State University, huding@msu.edu

2 Our Algorithm

2.1 Finding The Candidates For Cluster Centers

Gonazlez's seminal paper [11] provided a very simple 2-approximation algorithm for k-center clustering in any dimension. Basically, the algorithm iteratively selects k points from the input, where the initial point is arbitrarily selected, and each following j-th step ($2 \leq j \leq k$) chooses the point which has the largest minimum distance to the already selected $j-1$ points. Finally, it is able to show that these k points induce a 2-approximation for k-center clustering if each input point is assigned to its nearest neighbor of these k points.

We denote these ordered k points selected by Gonazlez's algorithm as $S = \{s_1, s_2, \cdots, s_k\}$, and define the Cartesian product $\underbrace{S \times \cdots \times S}_{k}$ as S^k, i.e., $\{(s'_1, s'_2, \cdots, s'_k) \mid s'_j \in S, 1 \leq j \leq k\}$. Then we have the following lemma.

Lemma 1 *There exists a k-tuple points from S^k yielding a 4-approximation for balanced k-center clustering.*

Proof. Suppose the unknown k optimal balanced clusters are $C_1, C_2, \cdots, C_k$, and the optimal radius is r_{opt}. If the selected k points of S luckily fall to these k clusters separately, it is easy to obtain a 2-approximation via triangle inequality (we will discuss that how to assign the input points to the k cluster centers for satisfying the requirement of balance in Section 2.2).

Now, we consider the other case. Without loss of generality, we assume that s_{j_1} and s_{j_2} is the firstly appeared pair belonging to a same optimal cluster and $j_1 < j_2$. For the sake of simplicity, we assume that $s_j \in C_j$ for $1 \leq j \leq j_2 - 1$. Due to the nature of Gonazlez's algorithm, we know that

$$\max_{p \in \cup_{j=j_2}^{k} C_j} \{ \min_{1 \leq l \leq j_2 - 1} ||p - s_l|| \} \leq ||s_{j_1} - s_{j_2}|| \leq 2r_{opt}. \quad (1)$$

Note even for the points from a same cluster C_j where $j \geq j_2$, their nearest neighbors from $\{s_1, \cdots, s_{j_2-1}\}$ are not necessarily same. Moreover, because of the requirement of balance, we cannot simply assign them to their nearest neighbors to generate a 2-approximation by (1); actually, this is also the major difference between ordinary and balanced k-center clustering. Instead, for each $j \geq j_2$ we arbitrarily select a point $p \in C_j$, and assign the whole C_j to p's nearest neighbor of $\{s_1, \cdots, s_{j_2-1}\}$ which is denoted as $s_{l(j)}$. Correspondingly, for any $p' \in C_j$ we have

$$||p' - s_{l(j)}|| \leq ||p' - p|| + ||p - s_{l(j)}|| \leq 4r_{opt} \quad (2)$$

due to triangle inequality and the fact that both $||p'-p||$ and $||p - s_{l(j)}||$ are no larger than $2r_{opt}$. Thus, the k-tuple points $\{s_1, s_2, \cdots, s_{j_2-1}, s_{l(j_2)}, \cdots, s_{l(k)}\}$ yields a 4-approximation if each optimal cluster C_j is assigned to the j-th point in the tuple for $1 \leq j \leq k$. □

2.2 Finding A Feasible Solution

Next, we answer the question that how to assign the input points to a fixed k-tuple points to satisfy the requirement of balance. To show its generalization, we denote the given k-tuple as $\{q_1, q_2, \cdots, q_k\}$ which is not necessarily from S^k. It is easy to know that the qualified radii must come from the kn distances $\{||p - q_j|| \mid p \in P, 1 \leq j \leq k\}$. As a consequence, we can apply binary search to find the smallest qualified radius. For each candidate radius r, we draw k balls centered at the k-tuple points and with the radius r respectively. We denote the k balls as $\mathcal{B}_1, \cdots, \mathcal{B}_k$. Thus, the only remaining problem is determining that whether there exists a balanced clustering on P to be covered by such k balls. We call such a clustering as a *feasible solution* if it exists.

A straightforward way to find a feasible solution is building a bipartite graph between the n points of P and the k balls, where a point is connected to a ball if it is covered by the ball; each ball has a capacity U and demand L, and the maximum flow from the points to balls is n if and only if a feasible solution exists. The existing maximum flow algorithms, such as Ford-Fulkerson algorithm or the new Orlin's algorithm [16], costs at least $O(nm)$ time. Recall that k is constant, and below we will show that the problem can be solved by **a system of linear equations and inequalities (SoL)** with the size independent of n.

The region $\cup_{j=1}^{k} \mathcal{B}_j$ divides the space into $2^k - 1$ parts (we ignore the region outside the union of the balls, since no point locates there; otherwise, we can simply reject this candidate r). We use $\mathcal{R}_{(j_1, j_2, \cdots, j_t)}$ with $1 \leq j_1 < j_2 < \cdots < j_t \leq k$ to indicate the region

$$(\mathcal{B}_{j_1} \cap \cdots \cap \mathcal{B}_{j_t}) \setminus (\cup_{j \notin \{j_1, \cdots, j_t\}} \mathcal{B}_j).$$

We calculate the total number of points covered by $\mathcal{R}_{(j_1, j_2, \cdots, j_t)}$ which is denoted as $n_{(j_1, j_2, \cdots, j_t)}$, and assign t non-negative variables $x^{j_1}_{(j_1, j_2, \cdots, j_t)}, \cdots, x^{j_t}_{(j_1, j_2, \cdots, j_t)}$ where each $x^{j_l}_{(j_1, j_2, \cdots, j_t)}$ indicates the number of points assigned to the j_l-th cluster from $\mathcal{R}_{(j_1, j_2, \cdots, j_t)}$. Thus, we have the following two types of linear constraint.

$$x^{j_1}_{(j_1, j_2, \cdots, j_t)} + \cdots + x^{j_t}_{(j_1, j_2, \cdots, j_t)} = n_{(j_1, j_2, \cdots, j_t)}, \quad (3)$$

$$L \leq \sum_{(j_1, j_2, \cdots, j_t) \in \pi_{j_l}} x^{j_l}_{(j_1, j_2, \cdots, j_t)} \leq U. \quad (4)$$

Here π_{j_l} is the set of all the possible subsets containing j_l of $\{1, \cdots, k\}$. There are at most $k2^k$ variables and $O(2^k)$ linear constraints in the whole SoL. Since k is a constant, the time complexity for building such a SoL is $O(nd)$ which is dominated by computing the distances between the n points and k ball centers. Further, the time complexity for solving the SoL is $O(poly(2^k))$ via Gaussian elimination.

Once obtaining a feasible solution of the above SoL, we still need to check that whether the solution is an integer solution for generating a clustering result.

Lemma 2 *If there exists a feasible solution of the above SoL, we can always transform it to an integer solution in $O(poly(2^k))$ time.*

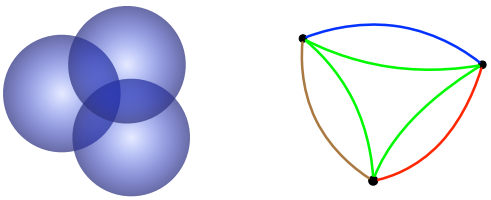


Figure 1: An illustration for building the multigraph G. Suppose $k = 3$ and the three balls locate as the left figure shows. For the sake of simplicity, we assume that all the variables corresponding to the overlapping areas are fractional. The colored multigraph G is in the right. The green edges correspond to the intersection of the three balls; any two vertices have another individually colored edge corresponding to their own intersection.

Proof. Suppose we have a fractional feasible solution denoted as $\Gamma = \{x^{j_l}_{(j_1,j_2,\cdots,j_t)} \mid j_l \in (j_1, j_2, \cdots, j_t), (j_1, j_2, \cdots, j_t) \in \pi\}$ where $\pi = 2^{\{1,\cdots,k\}}$. To help our analysis, we also construct a colored multigraph $G(V, E)$, where V contains k vertices $\{v_j \mid 1 \le j \le k\}$ corresponding to the k balls $\{\mathcal{B}_j \mid 1 \le j \le k\}$ respectively. Moreover, for any region $\mathcal{R}_{(j_1,j_2,\cdots,j_t)}$ and any pair $j_l, j_{l'}$ with $1 \le l < l' \le t$, we add an edge between v_{j_l} and $v_{j_{l'}}$ if both $x^{j_l}_{(j_1,j_2,\cdots,j_t)}$ and $x^{j_{l'}}_{(j_1,j_2,\cdots,j_t)}$ are fractional values. Thus, it is possible to have multiple edges between two vertices. Also, the edges corresponding to each $\mathcal{R}_{(j_1,j_2,\cdots,j_t)}$ share the same color (see Figure 1). Consider the following three cases.

Case I. If there is a circle having at least two different colors in G, we denote it as $v_1 \to v_2 \to \cdots \to v_h \to v_1$ w.l.o.g. From the construction of G, we know that for any two neighborhoods in the circle, there are two corresponding numbers from Γ which are both fractional and share the same region. Let the couples of numbers be

$$(x^1_{*_1}, x^2_{*_1}), (x^2_{*_2}, x^3_{*_2}), \cdots, (x^h_{*_h}, x^1_{*_h}). \tag{5}$$

Here we denote the foot subscripts by $*_j$ to simplify our analysis. If there exist two consecutive edges sharing the same color, e.g., $\overline{v_1v_2}$ and $\overline{v_2v_3}$, from the construction of G we know that v_1 and v_3 are connected by an edge with the same color as well. Hence we can always delete v_2 from the circle and add the edge $\overline{v_1v_3}$. Therefore we can assume that any neighbor edges have different colors in the circle, i.e., the following claim.

Claim. *$*_{j-1} \neq *_j$ for $2 \le j \le h$ and $*_h \neq *_1$.*

Meanwhile, we choose the small positive value

$$\delta = \min\{x^j_{*_j} - \lfloor x^j_{*_j} \rfloor, \lceil x^j_{*_{j-1}} \rceil - x^j_{*_{j-1}} \mid 1 \le j \le h\} \tag{6}$$

where $x^1_{*_0}$ represents $x^1_{*_h}$ for convenience. Together with the above claim we know that the following numbers

$$x^1_{*_1} - \delta, x^2_{*_1} + \delta, x^2_{*_2} - \delta, x^3_{*_2} + \delta, \cdots, x^h_{*_h} - \delta, x^1_{*_h} + \delta \tag{7}$$

contain at least one integer and all the others remain non-negative (see Figure 2). More importantly, no constraint of the SoL is violated after this adjustment. Since this operation adds new integers to Γ, we have to remove some edges of G due to the rule of its construction. If we keep adjusting the fractional values of Γ by this way, the edges of G will become fewer and fewer. After finite steps, there will be no circle or each circle has only one color, i.e., one of the next two cases happens. Actually the following two cases can be handled by similar manners. In order to show our idea more clearly, we discuss the simpler one, Case II, first.

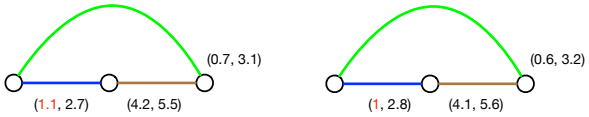


Figure 2: An illustration for adjusting the fractional numbers for Case I. The left shows an example circle with $h = 3$ and the couples of numbers. The right shows the couples of numbers after the adjustment with $\delta = 0.1$.

Case II. Now, we consider the second case that no circle exists in G; in other words, G is a forest. Different to the first case, we arbitrarily pick a leaf-to-leaf path in G and denote it as $v_1 \to v_2 \to \cdots \to v_h$, i.e., v_1 and v_h are two leaves in G (see Figure 3). Also from the construction of G, we have the following couples of fractional values

$$(x^1_{*_1}, x^2_{*_1}), (x^2_{*_2}, x^3_{*_2}), \cdots, (x^{h-1}_{*_{h-1}}, x^h_{*_{h-1}}). \tag{8}$$

Moreover, it is easy to know that $*_j \neq *_{j+1}$ for $1 \le j \le h-2$; otherwise, there will be a circle $v_j \to v_{j+1} \to v_{j+2} \to v_j$ due to the construction of G (which is contradict to the definition of Case II). Because v_1 is a leaf, we know that only one number of $\{x^1_* \mid * \in \pi_1\}$ is fractional and thus $\sum_{*\in\pi} x^1_*$ is fractional. Note that both L and U are integers, so the constraint (4) is not tight in both sides for $j_l = 1$, and similarly for $j_l = h$ too. We choose $\delta = \min\{x^j_{*_j} - \lfloor x^j_{*_j} \rfloor, \lceil x^{j+1}_{*_j} \rceil - x^{j+1}_{*_j} \mid 1 \le j \le h-1\}$. Through the same manner for analyzing the first case, we know that the following numbers

$$x^1_{*_1} - \delta, x^2_{*_1} + \delta, x^2_{*_2} - \delta, x^3_{*_2} + \delta, \cdots, x^{h-1}_{*_{h-1}} - \delta, x^h_{*_{h-1}} + \delta \tag{9}$$

contain at least one integer and all the others remain non-negative, while no constraint of the SoL is violated. In particular, the constraint (4) for $j_l = 1$ and h still holds since they are not tight before. Then we update G by removing some edges. If we keep performing this adjustment finite times, G will contain no edge. That is, we obtain an integer solution for the SoL.

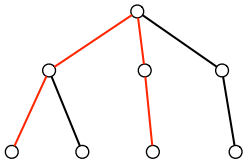


Figure 3: The red edges indicate a leaf-to-leaf path in the tree. The original edge colors are omitted here for the sake of simplicity.

Case III. The third case is that G only contains the circles having single color. We will show that this case can be handled by a similar way of Case II. First, we know that there are the following two different types of vertices in G. **Type** i: the vertex not belonging to any circle; **Type** ii: the vertex belonging to some circle. Due to the construction of G we know that the vertices belonging to a circle actually form a clique, and all of them are type ii. So we build a pseudo tree for G recursively as follows.

Pseudo-tree(G)

1. Initially, pick a vertex v arbitrarily from G.
2. If v is type i, take it as the root. Else, take the whole clique $\mathcal{C}$ containing v as the root.
3. Delete v (if type i) or $\mathcal{C}$ (if type ii) and its induced edges. If the remaining of G is not empty, it will become a set of disjoint components $\{G_1, \cdots, G_t\}$.
4. For each component G_i, add Pseudo-tree(G_i) as a child of v or $\mathcal{C}$.

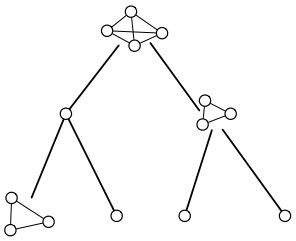


Figure 4: An example of pseudo tree. The edge colors are omitted here for the sake of simplicity.

Pseudo-tree(G) returns a pseudo tree where each node is either a type i vertex or a clique of type ii vertices (see Figure 4). Similar to Case II, we take an arbitrary leaf-to-leaf path of G. If both of the two leaves are type i vertices, we can adjust the fractional numbers along the path as same as Case II, and update G by removing some edges. Otherwise, we focus on the leaf that is a clique $\mathcal{C}$ of type ii vertices. Note that $\mathcal{C}$ contains at least three vertices, and only one of them has an outward edge from the clique (because it is a leaf). Suppose that the two vertices having no outward edge are v_1 and v_2, and the corresponding two fractional numbers are $x^1_{*_1}$ and $x^2_{*_1}$ respectively. Let $\delta = \min\{x^1_{*_1} - \lfloor x^1_{*_1} \rfloor, \lceil x^2_{*_1} \rceil - x^2_{*1}\}$. Then at least one of

$$x^1_{*_1} - \delta \quad \textit{and} \quad x^2_{*_1} + \delta \tag{10}$$

is an integer, and the other remains non-negative. Similar to Case II, we know that all the constraints of the SoL are not violated, and thus an update of G follows. After finite times of such an adjustment, G will become either Case II or a graph containing no edge (i.e., an integer solution of the SoL is obtained).

Finally, because the complexity of the initial G is $O(poly(2^k))$, the whole adjustment costs $O(poly(2^k))$ time as well and is independent of n. □

2.3 The 4-Approximation Algorithm

Combining Section 2.1 & 2.2, we have Algorithm 1. Step 1 & 2 cost $O(knd + nk\log(nk))$ time, and step 3 runs at most $O(k^k \log n)$ rounds where each round costs $O(n + poly(2^k))$ times. Thus, the total running time is $O(n(\log n + d))$ if k is a constant.

Theorem 3 *Algorithm 1 yields a 4-approximation of balanced k-center clustering, and the running time is $O(n(\log n + d))$ when k is a constant.*

Corollary 4 *Suppose the given instance locates in a metric space, and the time complexity for acquiring the distance between any two points is $O(D)$. Algorithm 1 yields a 4-approximation of balanced k-center clustering, and the running time is $O(n(\log n + D))$ when k is a constant.*

3 Other Issues

Finally, we address two questions: (1) is the approximation ratio 4 tight enough, and (2) why should we use S^k rather than S directly?

For the first question, we consider the following example. Let $n = 6$ points locate on a line, $k = 3$, and $L = U = 2$. See Figure 5. It is easy to know that the optimal solution is $C_1 = \{p_1, p_2\}$, $C_2 = \{p_3, p_4\}$, and $C_3 = \{p_5, p_6\}$ with $r_{opt} = 1$. Suppose that the first point selected by Gonazlez's algorithm is p_2, then the induced $S = \{p_2, p_5, p_1\}$ which results in a $(4 - \delta)$-approximation, no matter which 3-tuple is chosen from S^3. Since δ can be arbitrarily small, the approximation ratio 4 is tight.

Algorithm 1 4-Approximation Algorithm

Input: $P = \{p_i, \mid 1 \leq i \leq n\} \subset \mathbb{R}^d$, an integer $k \geq 1$, and integer lower and upper bounds $1 \leq L \leq U \leq n$.

1. Run Gonazlez's algorithm and output k points $S = \{s_1, s_2, \cdots, s_k\}$.
2. Compute the nk distances from P to S, and sort them in an increasing order. The set of distances is denoted as $\mathcal{R}$. Initialize the optimal radius $r_{opt} = \max \mathcal{R}$.
3. For each k-tuple $(s'_1, \cdots, s'_k)$ from S^k, binary search on $\mathcal{R}$. For each step with $r \in \mathcal{R}$, do the following steps.
   (a) Draw the k balls with radii r and centered at $(s'_1, \cdots, s'_k)$ separately.
   (b) If the SoL is feasible,
   - update r_{opt} to be r and record the feasible solution if $r < r_{opt}$;
   - if r is not a leaf, continue the binary search to the left side. Else, stop binary search.

   (c) Else,
   - if r is not a leaf, continue the binary search to the right side. Else, stop binary search.
4. Return the k-tuple from S^k with the smallest r_{opt} associating the corresponding feasible solution.

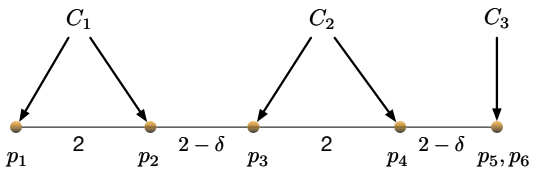


Figure 5: $||p_1 - p_2|| = ||p_3 - p_4|| = 2$ and $||p_2 - p_3|| = ||p_4 - p_5|| = 2 - \delta$ with a small positive δ; p_5 and p_6 overlap.

We construct another example to answer the second question. See Figure 6. It is easy to know $r_{opt} = r$. Suppose that the first point selected by Gonazlez's algorithm is p_1, then the induced $S = \{p_1, p_5, p_6\}$. If we take these 3 points as the cluster centers, the obtained radius is at least h (since p_3 and p_4 have to be assigned to p_6). Consequently, the approximation ratio is h/r which can be arbitrarily large. Hence we need to search the k-tuple points from S^k rather than S.

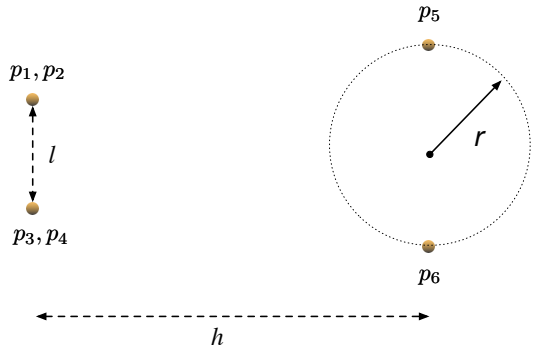


Figure 6: Let the 6 points locate in a plane, $k = 3$, and $L = U = 2$. p_1 and p_2 overlap, p_3 and p_4 overlap, and these 4 points locate on the same vertical line while p_5 and p_6 locate on another vertical line; $||p_1 - p_3|| = l$, $||p_5 - p_6|| = 2r$, and their horizontal distance is h; $l < 2r \ll h$.

4 Acknowledgements

The author was supported by a start-up fund from Michigan State University and CCF-1656905 from NSF. Part of the work was done when the author was in IIIS, Tsinghua University and Simons Institute, UC Berkeley. The author also wants to thank Jian Li, Lingxiao Huang, and Yu Liu for their helpful discussion.

References

[1] G. Aggarwal, R. Panigrahy, T. Feder, D. Thomas, K. Kenthapadi, S. Khuller, and A. Zhu, Achieving anonymity via clustering. ACM Trans. Algorithms 6(3), 49:1-49:19 (2010).

[2] S. Ahmadian and C. Swamy, Approximation algorithms for clustering problems with lower bounds and outliers. arXiv preprint arXiv:1608.01700 (2016).

[3] H. C. An, A. Bhaskara, C. Chekuri, S. Gupta, V. Madan, and O. Svensson, Centrality of trees for capacitated k-center. Mathematical Programming 154(1-2), 29-53 (2015).

[4] K. Aydin, M. Bateni, and V. S. Mirrokni, Distributed Balanced Partitioning via Linear Embedding. WSDM 2016: 387-396.

[5] J. Barilan, G. Kortsarz, and D. Peleg, How to allocate network centers. Journal of Algorithms 15(3), 385-415 (1993).

[6] M. Bateni, A. Bhaskara, S. Lattanzi, and V. S. Mirrokni, Distributed Balanced Clustering via Mapping Coresets. NIPS 2014: 2591-2599.

[7] M. Cygan, M. Hajiaghayi, and S. Khuller, Lp rounding for k-centers with non-uniform hard capacities. 53rd Annual Symposium on Foundations of Computer Science. pp. 273-282. IEEE Computer Society (2012).

[8] T. Dick, M. Li, V. Pillutla, C. White, M.-F. Balcan, and A. J. Smola, Data Driven Resource Allocation for Distributed Learning. CoRR abs/1512.04848 (2015).

[9] H. Ding, L. Hu, L. Huang, and J. Li, Capacitated Center Problems with Two-Sided Bounds and Outliers. CoRR abs/1702.07435 (2017).

[10] A. Ene, S. Har-Peled, and B. Raichel, Fast clustering with lower bounds, No customer too far, no shop too small. arXiv preprint arXiv:1304.7318 (2013).

[11] T. Gonzalez, Clustering to minimize the maximum intercluster distance. Theoret. Comput. Sci., 38:293-306, 1985.

[12] D. S. Hochbaum and D. B. Shmoys, A best possible heuristic for the k-center problem. Mathematics of operations research 10(2), 180-184 (1985).

[13] L. Kaufman and P. J. Roussew, Finding Groups in Data - An Introduction to Cluster Analysis. A Wiley-Science Publication John Wiley & Sons, 1990.

[14] S. Khuller and Y. J. Sussmann, The capacitated k-center problem. SIAM Journal on Discrete Mathematics 13(3), 403-418 (2000).

[15] T. Kociumaka and M. Cygan, Constant factor approximation for capacitated k-center with outliers. arXiv preprint arXiv:1401.2874 (2014).

[16] J. B. Orlin, Max flows in O(nm) time, or better. STOC 2013: 765-774.

2D Closest Pair Problem: A Closer Look

Ovidiu Daescu* Ka Yaw Teo*

Abstract

A closer look is taken at the well-known divide-and-conquer algorithm for finding the closest pair of a set of points in the plane under the Euclidean distance. An argument is made that it is sufficient, and sometimes necessary, to check only the next three points following the current point associated with the y-sorted array in the combine phase of the algorithm.

1 Introduction

The *closest pair of points* problem is a fundamental problem in computational geometry and has received significant attention over the years. The input for the problem consists of a set $P = \{p_1, p_2, \ldots, p_n\}$ of n points in R^d, where d is typically treated as a constant, and the objective is to find two points p and q in P such that $d(p, q) = min\{d(p_i, p_j)|p_i, p_j \in P, p_i \neq p_j\}$, where $d(p, q)$ represents the Euclidean distance between points p and q. It is well known that the problem can be solved optimally in $O(n \log n)$ time for any constant dimension d using a divide and conquer approach.

In this paper, the two-dimensional (2D) case of the problem is considered, and a tight geometric bound is derived on a specific step of the combine phase. The description of the algorithm is given in [1, 2, 3], and it can be summarized as follows: (1) Divide P into two equal parts, P_L and P_R, by a vertical line $l : x = x_m$, where x_m is the median x-coordinate of the points in P; (2) Recursively find the closest pair of points in P_L and P_R, respectively; (3) Let δ be the minimum of the two distances returned in the previous step; that is, no two points in P_L or P_R can be closer than δ. Then, the distance for the closest pair in P is either δ or given by a pair of points (p, q) where $p \in P_L$ and $q \in P_R$. In order to find the distance in the later case, denoted as $\delta_{L,R}$, let Y_δ represent the points p in P, with $x_p \in [x_m - \delta, x_m + \delta]$, sorted in non-decreasing order by y-coordinate. Then, $\delta_{L,R}$ can be found by traversing Y_δ in the sorted order and, for each point $p_i \in Y_\delta$, computing the distances from p_i to the next five points following p_i in Y_δ (see Exercise 33.4 in [2]). It has been posed to the authors of the current paper, as an open problem, to prove or disprove whether it suffices to check only the next four points following p_i in Y_δ. The main result is the following theorem.

Theorem 1 *It is sufficient, and sometimes necessary, to check only the next three points following p_i in Y_δ.*

Note that a similar and easier analysis follows if any point $p \in Y_\delta$ is required to satisfy $x_p \in (x_m - \delta, x_m + \delta)$, since we are looking for a pair of points that gives $\delta_{L,R} < \delta$. Nevertheless, for the sake of argument, the subsequent analysis has been performed in line with the original algorithm proposed in [1, 2, 3], which considers all points p in Y_δ with $x_p \in [x_m - \delta, x_m + \delta]$.

2 A Closer Look

In the current section, we prove Theorem 1. We start by taking care of inputs that have overlapping points, defined as points with the same x- and y-coordinates. Overlapping points imply that the closest pair has a zero distance.

Consider a $2\delta \times \delta$ axis-aligned box B centered at x_m, and assume that B is placed with its bottom edge on the x-axis and $x_m = 0$. As shown in [2], at most four points in B belong to P_L, and at most four to P_R.

Lemma 2 *Let p_i and p_j be two overlapping points in Y_δ, with $i < j$. Then, for each point in Y_δ, it is necessary and sufficient to check the next three points in Y_δ to detect the overlapping pair of points.*

Proof. Let p_i be the current point in Y_δ, and assume that $p_i \in P_L$. There can be at most three other points in B with the same y-coordinate as p_i. That happens when the x-coordinate of p_i is x_m, there is a point in P_L at $(x_m - \delta, 0)$, and two points in P_R at $(x_m + \delta, 0)$ and $(x_m, 0)$, respectively. Thus, it suffices to check the next three points following p_i in Y_δ to find the overlapping points. On the other hand, if p_{i+1}, p_{i+2}, and p_{i+3} are in the aforementioned order, then p_{i+3} has to be checked in order to identify the overlapping points. □

Notice that if B is defined as an open box, where any point $p \in Y_\delta$ satisfies $x_p \in (x_m - \delta, x_m + \delta)$, then the overlapping points can be found by checking only the next point following p_i in Y_δ.

From now on, assume that the input set P contains no overlapping points. Let B_L and B_R denote the left and right sides of B, respectively, as partitioned by vertical

*Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA. {daescu, ka.teo}@utdallas.edu

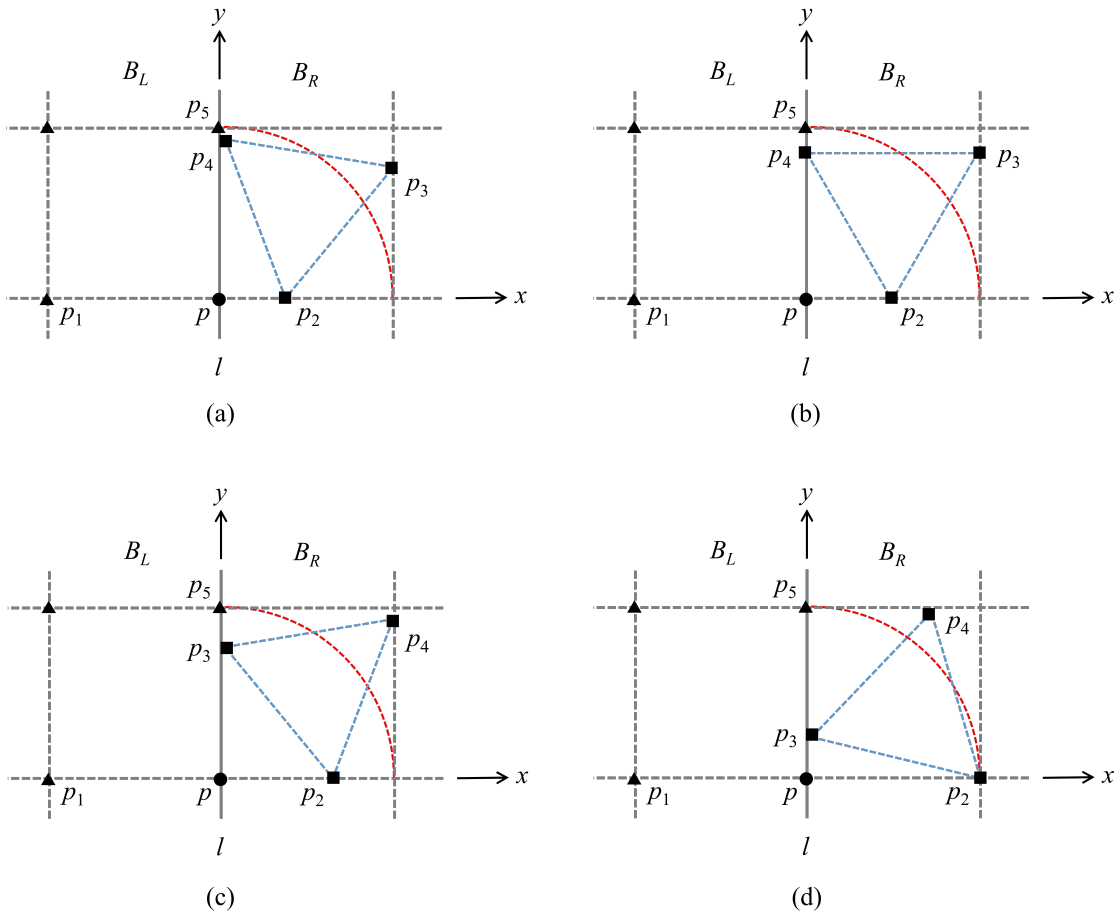


Figure 1: Illustration for Subcase A of Case I. (a)-(d) The positions of points p_2, p_3, and p_4 in B_R change as point p_2 varies from (0, 0) to (δ, 0). The circular arc (of radius δ) centered at p indicates that only two points in B_R are located $\leq \delta$ from p.

line $y = x_m$. B_L and B_R are $\delta \times \delta$ squares. Let $p = p_i$ be the current point in Y_δ, and assume that $p \in P_L$.

Given that the maximum number of points of separation of at least δ in B_L (or B_R) is four, at most three points in Y_δ with an array index greater than i lie within B_L. In other words, at most three points coming after p in Y_δ, not necessarily in a consecutive order, are in B_L.

This observation results in four different cases that must be considered separately - (I) three points after p are in B_L, (II) two points after p are in B_L, (III) one point after p is in B_L, and (IV) no point after p is in B_L (this case is trivial and omitted herein). In each of these cases, the worst-case scenario is determined, and that is the minimum number of points following p in Y_δ that must be examined in order to identify the closest pair of points correctly. For simplicity of notation, Y is used instead of Y_δ, and $p_1, p_2, \ldots$ in place of $p_{i+1}, p_{i+2}, \ldots$ hereafter.

As shall be seen shortly, by looking at the four cases, p_5 cannot be a candidate for the closest pair with p. It is then required to prove that either (i) one of $\{p_1, p_2, p_3\}$ is closer to p than p_4, or (ii) if p_4 is closer to p than any one of $\{p_1, p_2, p_3\}$, then one of $\{p_1, p_2, p_3\}$ is closer to p_4 than p, and so (p, p_4) cannot be the closest pair.

Case I: Three points after p are in B_L

Suppose that p is at the bottom right corner of B_L (the case where p is at the bottom left corner is trivial). We have the following subcases.

Subcase A: Three points after p are in B_R. Consider Figure 1, where p_2, p_3, and p_4 define an equilateral triangle of side length δ (i.e., worst case, in which the points are at their closest distance from each other in B_R). The first point in B_R can have the same y-coordinate as p (i.e., worst case, given that choosing a larger y-coordinate for the first point in B_R would simply increase the distances between the points in B_R and p), and it can be either the first or second point following

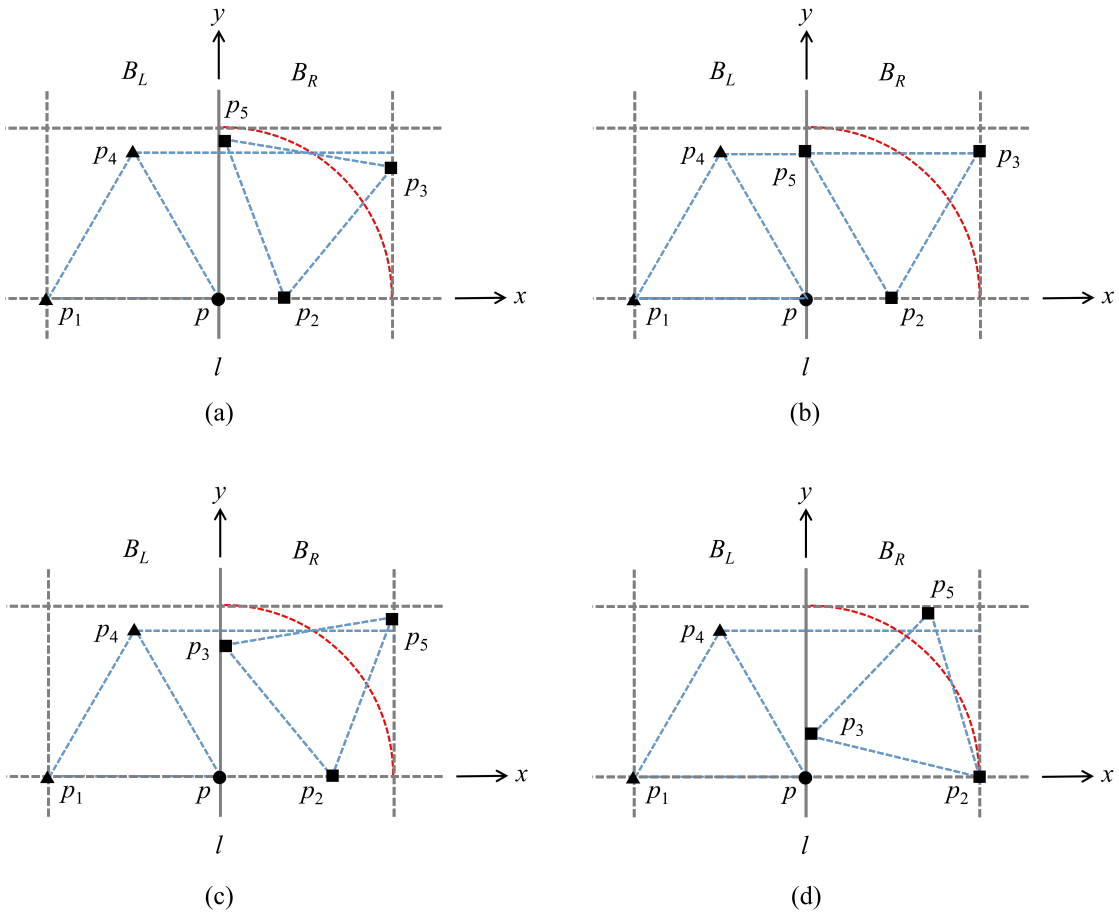


Figure 2: Illustration for Subcase B of Case II when $p = (0, 0)$. (a)-(d) The locations of points p_2, p_3,and p_5 in B_R change as point p_2 varies from $(0, 0)$ to $(\delta, 0)$. Based on the circular arc (of radius δ) centered at p, only two points in B_R are located $\leq \delta$ from p.

p in Y (i.e., labeled as p_1 or p_2). Since it is irrelevant to the subsequent argument, assume that it is p_2. When p_2 varies from $(0, 0)$ to $(\delta/2, 0)$, $d(p, p_3)$ is always greater than or equal to δ, and $d(p, p_4)$ is always greater than $d(p, p_2)$ (Figure 1 (a), (b)). As p_2 varies from $(\delta/2, 0)$ to $(\delta, 0)$, $d(p, p_4)$ is always greater than or equal to δ, but $d(p, p_3)$ can become smaller than $d(p, p_2)$ (Figure 1 (c), (d)). Thus, in Subcase A, p has to be compared with the three following points in Y.

Subcase B: Two points after p are in B_R. In this case, p_2 and p_3 can be located within the interior of B_R to be deemed competitive, and thus p must be compared with the next three following points in Y.

Overall, in Case I, only three points following p in Y need to be taken into consideration.

Case II: Two points after p are in B_L

In this case, p can be located anywhere along the bottom edge of B_L.

Subcase A: Four points after p are in B_R. This case is trivial; the point at the leftmost bottom corner of B_R is the closest point to p.

Subcase B: Three points after p are in B_R. Refer to Figure 2, where the triangles defined by (p, p_1, p_4) and (p_2, p_3, p_5), respectively, are equilateral with side length δ.

First, assume that p is at the lower right corner of B_L. The worst case occurs when the first point in B_R, either p_1 or p_2 (say p_2), has the same y-coordinate as p. When p_2 varies from $(0, 0)$ to $(\delta/2, 0)$, $d(p, p_3)$ is always greater than or equal to δ, and $d(p, p_5)$ is always greater than $d(p, p_2)$ (Figure 2 (a), (b)). As p_2 varies from $(\delta/2, 0)$ to $(\delta, 0)$, $d(p, p_5)$ is always greater than or equal to δ, but $d(p, p_3)$ can become smaller than $d(p, p_2)$ (Figure 2 (c), (d)). Thus, p has to be compared with the next three following points in Y.

When p is located somewhere on the bottom edge of B_L other than the bottom right corner of B_L, points

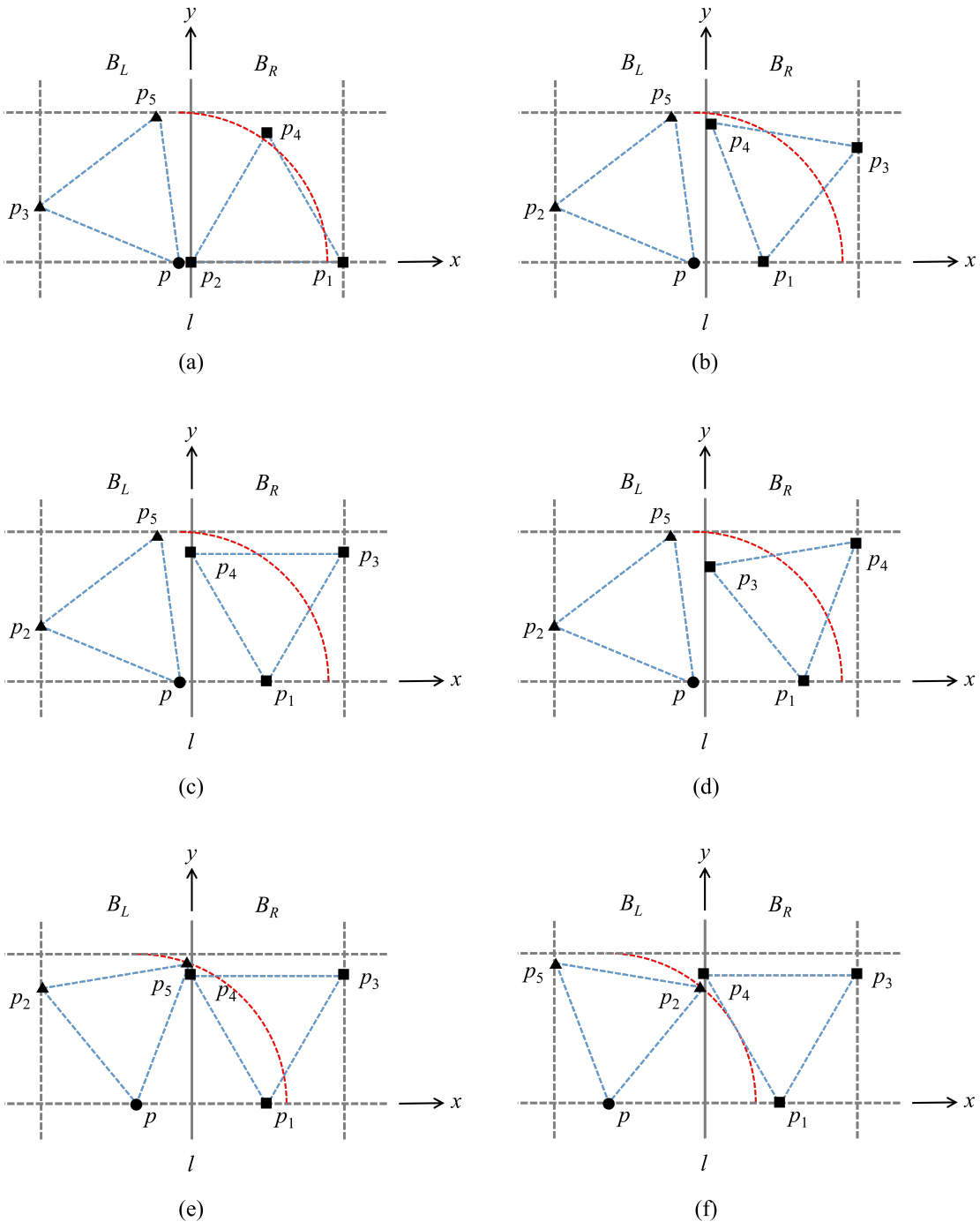


Figure 3: Illustration for Subcase B of Case II when $p \neq (0,\ 0)$. (a) When the first two points in B_R are located on the bottom edge, $d(p, p_2)$ can be $\leq \delta$. (b)-(d) When p_1 is between $(0,\ 0)$ and $(\delta/2, 0)$, $d(p, p_4) \geq (p, p_1)$. When p_1 is between $(\delta/2, 0)$ and $(\delta, 0)$, $d(p_3, p)$ can be $\leq d(p_1, p)$. (e)-(f) p_4 is at its closest to p. When p is between $(0, 0)$ and $(-\delta/2, 0)$, $d(p, p_4) \geq d(p, p_1)$. When p is between $(-\delta/2, 0)$ and $(-\delta, 0)$, $d(p, p_4) \geq \delta$. Thus, Only three points following p in Y need to be considered in the worst case.

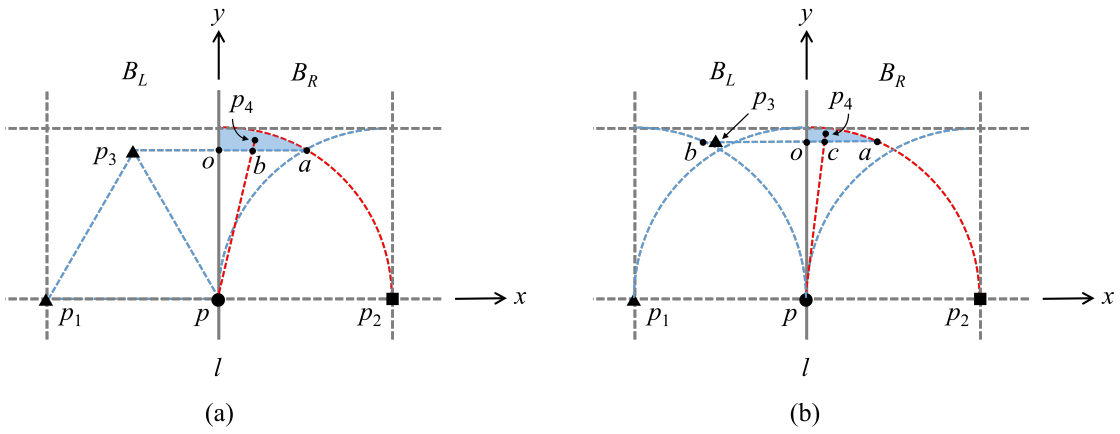


Figure 4: Illustration for Subcase C of Case II when $p = (0, 0)$. (a) p_3 is located exactly δ from p and p_1, respectively. (b) p_3 is placed slightly higher than and to the left of that in (a). In both scenarios, p_4 does not need to be taken into consideration, given that $d(p_3, p_4) \leq d(p, p_4)$.

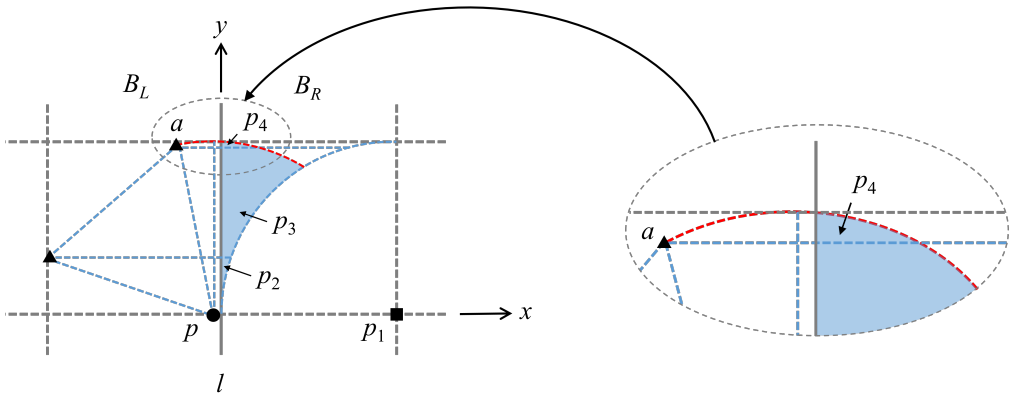


Figure 5: Illustration for Subcase C of Case II when $p \neq (0, 0)$. The shaded region indicates the possible location of the second point in B_R.

p_1 and p_2 can be placed at the corners on the bottom edge of B_R, and p_4 is always farther than δ from p (Figure 3 (a)). When p_1 varies from $(0, 0)$ to $(\delta/2, 0)$, as illustrated in Figure 3 (b) and (c), $d(p, p_4)$ is always greater than $d(p, p_1)$. As p_1 changes from $(\delta/2, 0)$ to $(\delta, 0)$, $d(p, p_4)$ is always greater than or equal to δ, but $d(p, p_3)$ can become smaller than $d(p, p_1)$ (Figure 3 (d)). The worst case happens when p_4 is located such that it has the smallest x- and y-coordinates possible (i.e., p_4 is at its closest to p), as shown in Figure 3 (e). In such case, when p is between $(0, 0)$ and $(-\delta/2, 0)$, $d(p, p_4) \geq d(p, p_1)$. When p is between $(-\delta/2, 0)$ and $(-\delta, 0)$, $d(p, p_4)$ is always greater than or equal to δ (Figure 3 (f)).

Altogether, in Subcase B, only three following points after p in Y need to be examined.

Subcase C: Two points after p are in B_R. As shown in Figure 4, with the assumption that p is located at the bottom right corner of B_L, the shaded region corresponds to possible locations of p_4 such that $d(p, p_4) \leq \delta$. The chosen location of p_2 is of the smallest y-coordinate, so that the area of the shaded region is maximized (i.e., worst case). A different location of p_2 would only diminish the shaded region. In addition, p_3 is placed such that $d(p, p_3) \geq \delta$ and $d(p_1, p_3) \geq \delta$.

Lemma 3 *There exists a configuration of points in Y such that p_4 is closer to p than any of $\{p_1, p_2, p_3\}$.*

Proof. Refer to Figure 4. □

Lemma 4 *If p_4 is closer to p than any of $\{p_1, p_2, p_3\}$ then $d(p_3, p_4) \leq d(p, p_4)$.*

Proof. At first, consider the scenario when p, p_1, and p_3 form an equilateral triangle of side δ, as shown in Figure 4 (a). Let p_4 be any point in the shaded region,

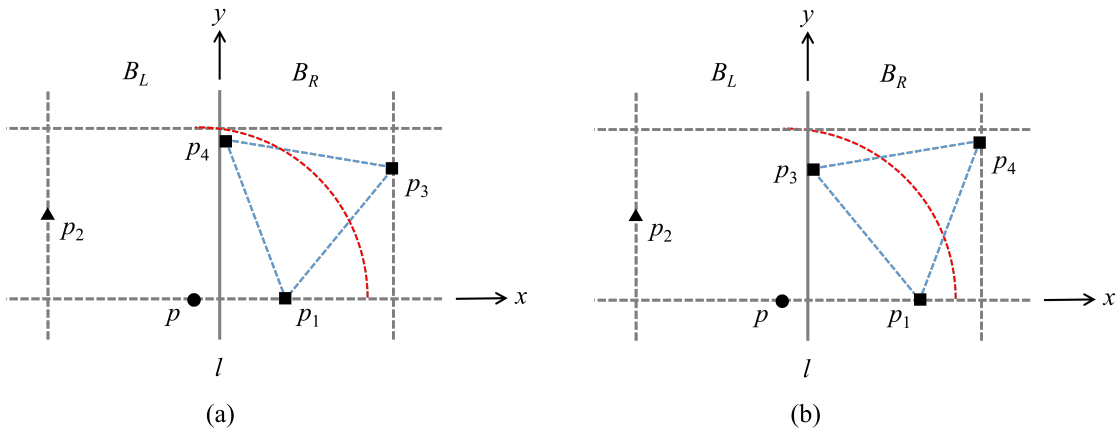


Figure 6: Illustration for Case III. (a) When p_1 is between (0, 0) and $(\delta/2, 0)$, $d(p, p_1) \leq d(p, p_4)$. (b) When p_1 is between $(\delta/2, 0)$ and $(\delta, 0)$, $d(p, p_3)$ can be $< \delta$.

let a be the intersection point of the two circular arcs (of radius δ) in B_R, and let o be the intersection between line segment p_3a and l. Obviously, $d(p_3, a) = \delta$, and, for any point b on the open line segment oa, $d(p_3, b) \leq d(p, b)$. Let b be the intersection point between pp_4 and p_3a. Then, $d(p, p_4) = d(p, b) + d(b, p_4) \geq d(p_3, b) + d(b, p_4) \geq d(p_3, p_4)$. As a result, $(p,\ p_4)$ does not need to be considered when the current point in Y is p.

Assume now that p_3 is moved upwards and to the left, while having $d(p_1, p_3) \geq \delta$ (Figure 4 (b)). Consider a horizontal line passing through p_3. Point o is the intersection of the horizontal line with l, point b is where $d(p_1, b) = \delta$, and point a is where $d(p, a) = \delta$. Let c be a point on line segment oa. Notice that $\angle p_3po \leq \pi/4$. Thus, $d(p_3, o) \leq d(p, o)$. Given that $d(p, p_3) \geq \delta$ and ab is parallel with pp_1, $d(a, b) = d(p, p_1) = \delta$ (since $d(p_1, b) = d(p, a) = \delta$). So, $d(p_3, a) \leq \delta$. As a result, for any point c on the open line segment oa, $d(p_3, c) \leq d(p, c)$. This implies that, for any point p_4 in the shaded region, $d(p_3, p_4) \leq d(p, p_4)$. Consequently, $(p,\ p_4)$ does not need to be checked. □

When p is placed to the left of the lower right corner of B_L, as illustrated in Figure 5, the second point in B_R can be located in the shaded region with a distance $\leq \delta$ (i.e., p_2, p_3, or p_4). Consider the case that the second point in B_R is p_4, and a is then p_3. We claim that $d(p_3, p_4) \leq d(p, p_4)$, which can be proven using a similar argument as that in Lemma 4.

Hence, in Subcase C, the current point p has to be compared to only the next three points in Y.

Case III: One point after p is in B_L

If p is situated at the lower right corner of B_L, the argument is essentially the same as that in Case I, but without the two points at the top edge of B_L. Thus, only three points following p in Y have to be examined.

Consider that p is located away from the bottom right corner of B_L. If there are four points in B_R, only two following points after p need to be checked, given that the first two points in B_R are located on the bottom edge of B_R (i.e., one at each lower corner). Suppose that there are three points in B_R. As shown in Figure 6 (a), when p_1 is located between (0, 0) and $(\delta/2,\ 0)$, $d(p, p_1) \leq d(p, p_4)$. When p_1 is between $(\delta/2,\ 0)$ and $(\delta,\ 0)$, p_3 can be less than δ from p. Hence, only the three following points after p in Y must be checked in the worst case.

This concludes the proof of Theorem 1.

3 Acknowledgment

The authors would like to thank Bogdan Armaselu and Shane St. Luce for their insightful comments on the problem. They also thank Balaji Raghavachari for raising the question of whether the fifth point after the current point in the y-sorted array can ever be relevant. This work has been partially supported by NSF award IIP1439718 and CPRIT award RP150164.

References

[1] J. L. Bentley and M. I. Shamos. Divide-and-conquer in multidimensional space. In *Proceedings of the Eighth Annual ACM Symposium on Theory of Computing*, pages 220–230, 1976.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2009.

[3] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag New York, Inc., 1985.

Supporting Ruled Polygons

Nicholas J. Cavanna* Marc Khoury† Donald R. Sheehy‡

Abstract

We explore several problems related to ruled polygons. Given a ruling of a polygon P, we consider the Reeb graph of P induced by the ruling. We define the Reeb complexity of P, which roughly equates to the minimum number of points necessary to support P. We give asymptotically tight bounds on the Reeb complexity that are also tight up to a small additive constant. When restricted to the set of parallel rulings, we show that the Reeb complexity can be computed in polynomial time.

1 Introduction

Gauss's Theorema Egregium states that any isometric embedding of a surface preserves the (Gaussian) curvature everywhere on the surface [8]. A particularly important example of this is the case of flat, or *rectifiable*, surfaces. The Gaussian curvature is the product of the so-called principal curvatures, and so, zero-curvature implies that at every point, some direction lies in a straight line (a principal curvature of zero). Ruled surfaces are one such example, but the most well-known example is pizza. If one rolls a (flat) triangular piece of pizza in one direction, the curvature in that direction is non-zero and thus, unless the pizza stretches or tears, the curvature in the orthogonal direction must be zero. This is what keeps the tip of the pizza from flopping downward.

In this paper, we attempt to establish a theoretical foundation for algorithmic problems on isometric embeddings of rectifiable polygons in $\mathbb{R}^3$. These are planar polygons that have embedded in $\mathbb{R}^3$ so that the curvature stays zero everywhere locally, but the embedding may not lie in a plane. Our original motivation came from the question of how many single points of contact where necessary to support a polygon so that none of the corners can "flop". A similar problem was studied in robotics for holding cloth [2], in which the researchers rediscovered the Art Gallery problem [7]. To correctly state such problems in the zero-curvature case, we first establish a vocabulary for *rulings* and give a description of the intrinsic topology. The ruling is the set of lines of zero curvature in some isometric embedding. This allows us to abstract away issues of physics (gravity, for example) and embeddings (self-intersection). Along the way, we connect these rulings to a generalization of Reeb graphs that allows us to connect ruled polygons to an art gallery-type theorem, phrased in terms of topological simplification.

For a ruled polygon P with no holes, cutting along a ruling line divides it into two pieces. The ruling is *supported* by a set of points $S \subset P$ if every line ℓ of the ruling has a point of S on both pieces of $P \setminus \ell$. For example, two points suffice to support a triangle (slice of pizza). We give the formal definition of the *Reeb complexity* of a polygon in Section 4, but roughly, it corresponds to the minimum size support set over all possible rulings. In Section 4, we prove that all n-gons have Reeb complexity at most $\frac{n}{2} + 1$, and we give a family of polygons with Reeb complexity $\frac{n}{2} - 4$.

We conclude with a collection of open problems and research directions that we believe could be of interest as they provide connections between classic problems in computational geometry such as monotone polygons, Hamiltonian triangulations, and art galleries and growing new areas such as Reeb graphs and topological simplification.

2 Definitions

Let P be a simple polygon in the plane. Let $\hat{P}$ be an isometric embedding of P into $\mathbb{R}^3$. This is one for which the distance between any pair of points on the embedded surface is the same as in the plane. Every point on $\hat{P}$ will have a principle curvature of zero in some direction. We will limit ourselves to *nondegenerate* embeddings, in which there is a unique such direction at each point. These correspond to line segments covering the polygon. Rather than working directly with embeddings $\hat{P}$ of P, we will look at the patterns induced by these line segments on P itself. Thus, we use Gauss's theorem to make statements about nondegenerate isometric embeddings by reasoning directly about planar polygons.

A *ruling* of P is a set of line segments in P with both endpoints on the boundary, whose interiors partition the interior of P. Moreover, we require that no two distinct segments in a ruling are collinear and intersect. This last condition may seem strange at first, but is a fundamental issue in rulings, particularly in defining a topology on the rulings. According to this definition, the

*University of Connecticut, nicholas.j.cavanna@uconn.edu
†University of California, Berkeley, khoury@cs.berkeley.edu
‡University of Connecticut, don.r.sheehy@gmail.com

segment through the reflex vertex of the polygon cannot be replaced by two shorter line segments. Segments that contain a reflex vertex in their relative interior are called *branch segments*.

A degenerate embedding of P does not have a corresponding ruling. This happens both for the extreme cases where the entire embedding lies in a plane, but also in other more interesting cases.

We say that a ruling is *simple* if it has no branch segments. A ruling is *parallel* if every pair of segments are parallel. A ruling is *Morse* if no branch segment contains more than one reflex vertex in its relative interior.

3 The Topology of Rulings

Given a simple polygon $P \subset \mathbb{R}^2$ with or without holes, the *Reeb graph* [9] with respect to a continuous function $f : P \to \mathbb{R}$ is the quotient space $\mathcal{R}(f) := P/\sim_f$ where $x \sim_f y$ if and only if x and y are in the same connected component of $f^{-1}(c)$, where $f(x) = f(y) = c$. For an accessible introduction to Reeb graphs, see [6]. We can construct a similar space from the ruling lines of P. That is we define the *Reeb graph of a ruling* S of P, $\mathcal{R}(S)$, to be the quotient space $P/\sim$, where $x \sim y$ if and only if $x, y \in s$ for some segment $s \in S$. The branch segments of S decompose the polygons into pieces which correspond to edges in the Reeb graph, glued together at internal nodes corresponding to the branch segments themselves. $\mathcal{R}(S)$ has a natural graph metric (i.e., is a 1-dimensional stratified metric space) where the distance between two equivalence classes $[x], [y]$ is the Hausdorff distance between the line segments containing x and y. Recall that the Hausdorff distance between two compact subsets A and B of a metric space is defined as

$$\mathrm{d}_H(A, B) = \max\{\max_{a\in A}\min_{b\in B}\mathrm{d}(a, b), \max_{b\in B}\min_{a\in A}\mathrm{d}(a, b)\}$$

The constructions of a Reeb graph of a ruling and the Reeb graph constructions align for particularly nice rulings, hence the naming convention. If we have a parallel ruling S, then the Reeb graph of the ruling is equivalent to the Reeb graph formed from the height function orthogonal to the ruling. Alternatively, if we have a simple ruling S on P, then we may consider the midpoint m_s of each line segment $s \in S$, which collectively trace out a path $\gamma : [0, 1] \to P$. We can then define a continuous function $f : P \to \mathbb{R}_{\geq 0}$ by $f(p) = \int_0^{t_0} |\gamma'(t)|dt$, where if p is on line segment s, $f^{-1}(m_s) = t_0$, yielding $\mathcal{R}(S) = \mathcal{R}(f)$.

For completeness' sake, the remainder of this section will explore the relations between the homotopy type and homology of a polygon P and its Reeb graph of a ruling S, $\mathcal{R}(S)$. A known result in [5] states that if a space X is locally path connected and is partitioned into connected equivalence classes by $\sim$ and $X/\sim$ is semilocally simply connected, then $q_* : \pi_1(X) \to \pi_1(X/\sim)$ is surjective, where q is the topological quotient map. Since each $q^{-1}([p])$ are path-connected, and all other conditions are satisfied, we have that $q_* : \pi_1(P) \to \pi_1(\mathcal{R}(S))$ is a surjection. Note that we use the fact that $\mathrm{int}(P)$ is homotopy equivalent to P.

With regards to homotopy, if P has h holes then it is homotopy equivalent to $\bigvee_h S^1$, where $\bigvee$ is the wedge sum formed by taking the disjoint union of h copies of S^1 and adjoining them each at a single point. The fundamental group of P, $\pi_1(P)$, is then equal to $\mathbb{Z} * \ldots * \mathbb{Z}$, the free product on h generators. If P has no holes, then it is contractible, and we have that P and $\mathcal{R}(S)$ are homotopy equivalent.

With regards to homology, the Hurewicz Theorem states there is an isomorphism between $\pi_1^{\mathrm{ab}}(P)$ and $H_1(P)$, where the former is the abelianization of $\pi_1(\cdot)$. Since $(*_{i=1}^{h}\mathbb{Z})^{\mathrm{ab}} = \mathbb{Z}^h$, then $H_1(P) = \mathbb{Z}^h$ as expected.

The following theorem due to Vietoris [10] allows us to provide an isomorphism between the homology of P and that of $\mathcal{R}(P)$.

Theorem 1 (Vietoris-Begle Mapping Theorem) *Given compact metric spaces X and Y and surjective map $f : X \to Y$, if for all $0 \leq k \leq n-1$, for all $y \in Y$, $\tilde{H}_k(f^{-1}(y)) = 0$, then $f_* : \tilde{H}_k(X) \to \tilde{H}_k(Y)$ is an isomorphism for $k \leq n-1$ and a surjection for $k = n$.*

Consider the quotient map $q : P \to \mathcal{R}(S)$, which is surjective by definition. Given $[p] \in \mathcal{R}(S)$, each fiber $q^{-1}([p])$ is the line segment corresponding to the equivalence class $[p]$, thus contractible, so $\tilde{H}_k(q^{-1}([p]))$ is acyclic for all dimensions k. Theorem 1 then implies that $\tilde{H}_*(P) = \tilde{H}_*(\mathcal{R}(S))$.

4 Asymptotically Tight Bounds

Let P be a simple polygon with n vertices and h holes. We define the *Reeb complexity* of P as the minimum number of leaves in $\mathcal{R}(S)$ over all possible rulings S of P. In this section we show that the Reeb complexity of P is upper bounded by $\frac{n}{2}+1$ and can be as large as $\frac{n}{2}-4$. To show the upper bound we consider the special case of parallel rulings, whereas to show the lower bound we construct a family of polygon for which any ruling must induce a Reeb graph with $\Omega(n)$ leaves.

4.1 Upper Bound

For any vector v, there is a parallel ruling S defined by sweeping the line ℓ orthogonal to v across $\mathbb{R}^2$. We think of v as the "height" direction, and the function $f_v : P \to \mathbb{R}$ maps x to its "height", $\langle v, x\rangle$, in the direction v. Here $\langle\cdot,\cdot\rangle$ denotes the dot product. The Reeb graph $\mathcal{R}(f_v)$ is the quotient space constructed by contracting the connected components of $P \cap \ell$ to single points as ℓ sweeps through $\mathbb{R}^2$ in the direction v. We

denote by b the number of branch (internal) nodes in $\mathcal{R}(f_v)$ and by l the number of leaves.

A *reflex vertex* p of a polygon P is a vertex whose interior angle is strictly greater than 180°; see Figure 1. We denote by $R(P)$ the set of reflex vertices of P and define $k = |R(P)|$. Note that a reflex vertex cannot be on the convex hull of P. The reflex vertices play an important role in determining the number of leaves in a Reeb graph induced by a parallel ruling. Each reflex vertex p bears witness to a closed set of vectors which, if the rulings are induced by any vector v in the set, eliminate p from the Reeb graph. That is p is not a critical point of f_v and does not correspond to a node of $\mathcal{R}(f_v)$.

Let $p \in R(P)$ be a reflex vertex of P. Denote by n_1 and n_2 the normals to the edges e_1 and e_2 adjacent to p, respectively. We will consider two double cones defined by the normals n_1, n_2 at apex p. Consider the vector $\overline{n} = (n_1 + n_2)/||(n_1 + n_2)||$ and notice that $\angle(\overline{n}, n_1) = \angle(\overline{n}, n_2)$. Define the closed double cone $C_p = \{v \in \mathbb{R}^2 : \angle(\overline{n}, v) \geq \angle(\overline{n}, n_1)\}$; see Figure 1.

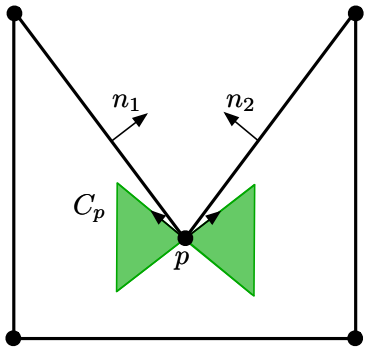


Figure 1. A polygon with a reflex vertex at p. The cone C_p is shown in green and is defined by the lines with directions n_1 and n_2 and intersecting p

Lemma 2 *Let P be a simple polygon with or without holes and v be an arbitrary direction. Consider an arbitrary leaf node $q \in \mathcal{R}(f_v)$. Then for all $p \in R(P)$, $p \notin f_v^{-1}(q)$.*

Proof: Suppose, for the sake of contradiction, that there exists a reflex vertex $p \in f_v^{-1}(q)$. Consider the ruling line ℓ at p, and divide ℓ into two rays r_1, r_2 with base point p. Since q is a leaf node, the ruling line ℓ at p locally intersects the interior of P in a single connected component. Thus one of the two rays, say r_1, points into the exterior of P between the two edges adjacent to p. However, this implies that we can perturb ℓ in directions v and $-v$ while still locally intersecting the interior of P. This contradicts that fact that q is a leaf node. □

Lemma 3 *Let P be a simple polygon with or without holes, $p \in R(P)$ be a reflex vertex, and v be an arbitrary direction. Then p creates a branch node in $\mathcal{R}(f_v)$ if and only if $v \notin C_p$.*

Proof: Note that, as a consequence of Lemma 2, p can only create a branch node in $\mathcal{R}(f_v)$. Suppose that p creates a branch node in $\mathcal{R}(f_v)$. For this to occur, the ruling line ℓ at p locally intersects the interior of P in two connected components. This happens if and only if the vector orthogonal to ℓ is not in the set C_p. □

From Lemmas 2 and 3, we see that C_p defines precisely the set of vectors v such that the ruling f_v eliminates p from $\mathcal{R}(f_v)$. In Section 5, we will use the cones C_p to compute the Reeb complexity of a polygon when restricted to the set of parallel rulings.

When f_v is Morse, every branch node of $\mathcal{R}(f_v)$ has degree 3. In this case we have that $2|E| = \sum_{u \in \mathcal{R}(f_v)} \deg(u) = 3b + l$, where $|E|$ is the total number of edges in $\mathcal{R}(f_v)$. Since the holes are disjoint, each hole creates a cycle in $\mathcal{R}(f_v)$ adding one edge to the total number of edges, giving $|E| = b + l - 1 + h$. Combining the expressions we get the relation $l = b + 2 - 2h$. Note that when $h = 0$ we recover the relationship between the number of internal nodes and the number of leaves in a tree. Furthermore when f_v is not Morse, the equality becomes the inequality $l \geq b + 2 - 2h$.

Lemma 4 *Let P be a simple polygon with h holes, k be the number of reflex vertices, and v be an arbitrary direction. If f_v is Morse, then $\mathcal{R}(f_v)$ has at most $k + 2 - 2h$ leaves.*

Proof: In the worst case, the vector v is not in C_p for any $p \in R(P)$. By Lemma 3 every reflex vertex creates a branch node in $\mathcal{R}(f_v)$. Since f_v is Morse, we have the relationship $l = b + 2 - 2h \leq k + 2 - 2h$. □

This result is tight in that there exists a polygon P and a direction v such that $\mathcal{R}(f_v)$ has exactly $k + 2$ leaves; see Figure 2. However it is easy to construct polygons where all but a constant number of vertices are reflex vertices. In such cases we can bound the number of branch nodes in the Reeb graph much more tightly.

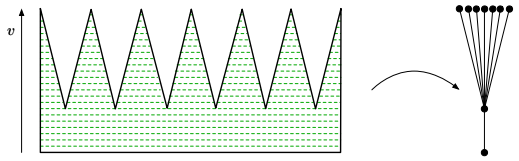


Figure 2. A polygon with 6 reflex vertices and a parallel ruling for which the Reeb graph has 8 leaves. However had we taken v to be the x-direction, the Reeb graph would only have 2 leaves, and the ruling would be simple.

Lemma 5 *Let P be a simple polygon with h holes and n vertices, and let v be an arbitrary direction. Then number of branch nodes $b \leq \lfloor \frac{n}{2} - 1 + h \rfloor$.*

Proof: Let m denote the number of non-reflex vertices and note that $n = k + m$. By Lemma 3, a reflex vertex either forms a branch node or is eliminated from $\mathcal{R}(f_v)$; it follows that $b \leq k$. Similarly, as a consequence of Lemma 2, every leaf node is created by the ruling passing over one or more non-reflex vertices and it follows that $l \leq m$. Combining these inequalities with the inequality $l \geq b + 2 - 2h$, we have that

$$\begin{aligned} b + 2 - 2h &\leq l \\ &\leq m \\ 2b + 2 - 2h &\leq m + b \\ &\leq m + k \\ &= n \\ b &\leq \frac{n - 2 + 2h}{2}. \end{aligned}$$

□

When f_v is Morse, we can use the relation $l = b + 2 - 2h$ to bound the Reeb complexity of any ruling as $\lfloor \frac{n}{2} + 1 - h \rfloor \leq \lfloor \frac{n}{2} + 1 \rfloor$. Notice that this bound holds for any arbitrary ruling, not just parallel rulings, as increasing the set of rulings considered only decreases the Reeb complexity.

Theorem 6 *Let P be a simple polygon with h holes and n vertices. Let S be any ruling of P. Then the Reeb complexity is at most $\lfloor \frac{n}{2} + 1 \rfloor$.*

4.2 Lower Bound

Consider once again the example shown in Figure 2. Had we chosen the direction orthogonal to v, the Reeb graph would have only 2 leaves. To establish a lower bound, and thus show that our result is asymptotically tight, we construct a family of polygons whose Reeb complexity is $\Omega(n)$.

Consider two concentric circles C_1, C_2 centered at the origin and with radii r_1, r_2 respectively, where $r_1 \gg r_2$. We parameterize $C_1(\theta) = r_1(\sin\theta, \cos\theta)$ and $C_2(\phi) = r_2(\sin\phi, \cos\phi)$. For each n, we construct a set of $2n$ vertices, n of which will be placed on C_1, with the remaining n vertices being placed on C_2. The first set of n vertices are placed on C_1 at $\theta_i = \frac{2\pi i}{n}$ for $i \in [n-1]$. The second set of n vertices are placed on C_2 at $\phi_i = \frac{(2i+1)\pi}{n}$ for $i \in [n-1]$. The edges of the polygon are constructed by connecting the ith vertex of C_1 to the $i-1$ and ith vertices of C_2. See Figure 3. Notice that every vertex on C_2 is a reflex vertex of the polygon.

Now consider a vertex p on C_1, and suppose that some (not necessarily parallel) ruling S eliminates p from

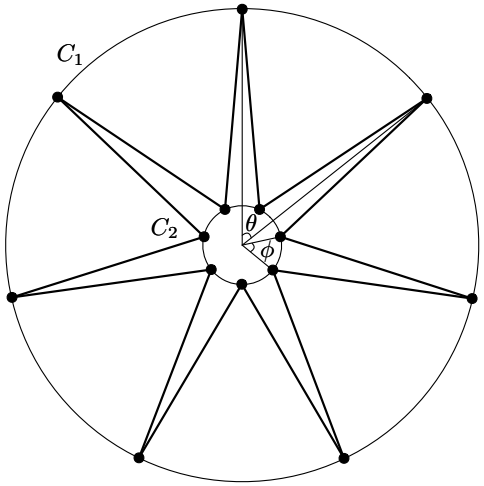


Figure 3. Our lower bound construction for $n = 7$.

$\mathcal{R}(S)$. Then there exists some line segment $s = (p, q)$ of the ruling S with endpoint p. The other endpoint of s can be contained in one of only two (when n is odd) or three (when n is even) other spikes of the polygon, due to the limited visibility at p. We prove this statement in the following paragraphs. Crucial to this argument is the fact that q must be on the boundary of P in a spike different than that of p for S to be a valid ruling.

Let p_1, p_2 be the vertices on C_2 that are adjacent to p. The length of the segment $|p_1p_2| \leq \frac{2\pi}{n} r_2$, the length of the arc connecting p_1 and p_2. The affine hulls of the edges pp_1 and pp_2 each intersect C_2 in two points. Let p_1' and p_2' be the intersections not equal to p_1 and p_2; see Figure 4. We will show that the length of the segment $|p_1'p_2'| \leq \frac{2\pi}{n} \frac{r_1+1}{r_1-1}$, which, for an appropriate choice of r_1, covers at most 2 intervals when n is odd and 3 intervals when n is even.

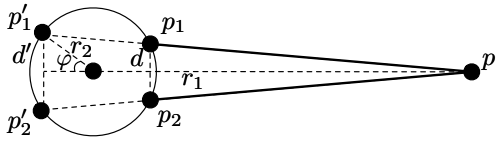


Figure 4. For the spike at p we consider the affine hulls of the edges adjacent to p. The intersection of these affine hulls with C_2 define p_1' and p_2'. The point q can lie in any of the spikes spanned by the segment $p_1'p_2'$.

Let m and m' be the midpoints of the segments p_1p_2 and $p_1'p_2'$ respectively. Notice that the triangles $\triangle p_1mp$ and $\triangle p_1'm'p$ are similar. Define the lengths $d = |p_1m|$ and $d' = |p_1'm'|$. Since $\triangle p_1mp$ and $\triangle p_1'm'p$ are similar we have the relationship $\frac{d'}{d} = \frac{|pm'|}{|pm|}$. We can write the length $|pm| = r_1 - r_2 + \delta$ for some $\delta > 0$. Here δ is

the distance between m and C_2. Similarly we can write $|pm'| = r_1 + r_2 \sin\varphi$; see Figure 4. Then we have that

$$\begin{aligned} d' &= d\frac{|pm'|}{|pm|} \\ &= d\frac{r_1 + r_2\sin\varphi}{r_1 - r_2 + \delta} \\ &\leq \frac{\pi}{n}\frac{r_1 + r_2\sin\varphi}{r_1 - r_2 + \delta} \\ &\leq \frac{\pi}{n}\frac{r_1 + r_2}{r_1 - r_2 + \delta} \\ &\leq \frac{\pi}{n}\frac{r_1 + r_2}{r_1 - r_2}, \end{aligned}$$

where the first inequality follows from $|p_1p_2| \leq \frac{2\pi}{n}r_2$, the second from $\sin\varphi \leq 1$, and the third from $\delta > 0$. Then taking $r_2 = 1$ we have that

$$\begin{aligned} |p_1'p_2'| &= 2d' \\ &\leq \frac{2\pi}{n}\frac{r_1 + 1}{r_1 - 1}. \end{aligned}$$

The vertices on C_2 are spaced so that each interval has arc-length $\frac{2\pi}{n}$. All that remains is to compute the number of intervals a segment of length $\frac{2\pi}{n}\frac{r_1+1}{r_1-1}$ can cover. When $r_2 = 1$, this amounts to computing the angle 2φ, shown in Figure 4, when d' is at its maximum value. The angle $2\varphi = 2\arcsin\frac{d'}{r_2} \leq 2\arcsin\left(\frac{\pi}{n}\frac{r_1+1}{r_1-1}\right)$. Then the maximum number of intervals spanned by the segment $|p_1'p_2'|$ is

$$\frac{2\arcsin\left(\frac{\pi}{n}\frac{r_1+1}{r_1-1}\right)}{\frac{2\pi}{n}} = \frac{n}{\pi}\arcsin\left(\frac{\pi}{n}\frac{r_1+1}{r_1-1}\right),$$

which approaches $\frac{r_1+1}{r_1-1}$ as $n \to \infty$. For the purposes of this construction we take $r_1 = 4$. When $r_1 = 4$, for all $n \geq 7$, the segment $|p_1'p_2'|$ can span at most 2 intervals when n is odd and at most 3 when n is even.

Suppose that p corresponds to the ith spike. When n is odd q must intersect one of only two possible intervals, those corresponding to the spikes $i+\lfloor n/2\rfloor$ and $i+\lceil n/2\rceil$. The ith and $i+1$th spike have overlapping regions of a single interval. However if a ruling line attempts to eliminate the vertex on the $i+2$th spike, the resulting line segment must intersect s. This is impossible in any valid ruling, and it follows that at most 4 spikes can be eliminated from $\mathcal{R}(S)$. Thus the Reeb complexity of this polygon is at least $n-4$. Note that the polygon has $2n$ vertices in total, so this matches our upper bound. In conjunction with Theorem 6, we've established the following theorem.

Theorem 7 *Let P be a simple polygon with h holes and with n vertices. Let S be any ruling of P. Then the Reeb complexity of P is upper bounded by $\frac{n}{2}+1$. Furthermore there exists simple polygons for which the Reeb complexity is at least $\frac{n}{2}-4$.*

While our bound is asymptotically tight, the additive difference between the example used to establish the lower bound and the upper bound proved in Theorem 6 is 5. It remains open whether there exists a polygon P for which every direction induces a Reeb graph with exactly $\frac{n}{2}+1$ leaves.

5 Computing the Reeb Complexity for Parallel Rulings

Given a simple polygon P with h holes and n vertices we wish to compute the Reeb complexity of P. In Section 6, we conjecture that this problem is NP-complete for general rulings. In the special case of parallel rulings, we show that the problem can be solved in $O(n\log n)$ time.

By Lemma 3, finding a parallel ruling of minimum Reeb complexity is equivalent to finding a vector v that is contained in the maximum number of cones C_p. We use the standard duality transform that maps a point (a,b) to the line $\ell = \{(x,y) : y = ax - b\}$. In the dual plane, a parallel ruling S dualizes to a vertical line, because each line in S has the same slope. Similarly, the two lines ℓ_p, ℓ_p' that bound the cone C_p dualize to two points $(m_p, c_p), (m_p', c_p')$ where m_p, m_p' are the slopes of ℓ_p, ℓ_p' and $-c_p, -c_p'$ are the y-intercepts.

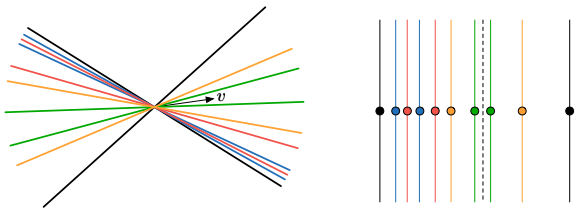


Figure 5. The set of cones for each reflex vertex translated to the origin. The boundaries of these cones dualize to points in the dual plane. We disregard the y-coordinate of the dualized points and consider the resulting list of slopes on the x-axis. Then finding a vector in the maximum number of cones is equivalent to finding a line in the maximum number of intervals. Dualizing the entire dotted line as a set of points, gives the desired ruling.

The algorithm begins by computing the cone C_p for each reflex vertex $p \in R(P)$. The duality transform is applied to the set of lines $\cup_p\{\ell_p, \ell_p'\}$, giving the set of points $\cup_p\{(m_p, c_p), (m_p', c_p')\}$. Notice that the y-intercept values can be disregarded, as we are interested in a vector v based at the origin that lies in the maximum number of cones translated to the origin. The set of slopes $I = \cup_p\{(m_p, m_p')\}$ define a set of intervals. Sort the endpoints of the intervals and call the resulting list L. Finding a vector v that is contained in the maximum number of cones is equivalent to finding the vertical line that lies in the maximum number of intervals in I.

There is one remaining caveat. Notice that traversing the list of slopes L in increasing order corresponds, in the primal plane, to traversing the cones in rotary order starting with the vector $v = (0, -1)$ and performing a rotation of 180°. The vector v may already lie in a subset of the cones $\mathcal{C}_v$. Consider $C_p \in \mathcal{C}_v$ and its corresponding pair (m_p, m'_p). In this case, the first endpoint of the interval (m_p, m'_p) that the traversal encounters in L is an exit event, not an entry event. The pair (m_p, m'_p) corresponds to the interval $(-\infty, m_p] \cup [m'_p, \infty)$. To account for this, the algorithm first computes $\mathcal{C}_v$, and labels the elements of L with the correct entry/exit labels. The algorithm keeps a counter c, initialized with the value $|\mathcal{C}_v|$, and traverses L incrementing c on each entry event, and decrementing c on each exit event. The maximum value of this counter $c_{\max}$ gives the minimum number of leaves $k - c_{\max} + 2 - 2h$, where k is the number of reflex vertices.

The runtime of the algorithm is dominated by sorting L, which takes $O(n \log n)$ time. The other steps of the algorithm – computing the cones C_p, dualizing the boundary lines $\cup_p \{\ell_p, \ell'_p\}$, computing the subset $\mathcal{C}_v$, assigning the correct labels to the intervals, and computing $c_{\max}$ – can all be done in $O(n)$ time. The correctness of the algorithm follows from Lemma 3. We have established the following theorem.

Theorem 8 *Let P be a simple polygon with h holes and n vertices. The Reeb complexity of P, restricted to the set of parallel rulings, can be computed in $O(n \log n)$ time.*

Note that the algorithm presented in this section is equivalent to an algorithm in the primal space where a vector is rotated once around the origin. As the vector rotates around the origin, the algorithm keeps track of entry and exit events defined by each cone. We chose to present the algorithm in the dual space because future extensions to more general classes of rulings will likely operate in the dual space. As shown in Figure 5, a parallel ruling corresponds to a vertical line in the dual space, since each line segment of the ruling has identical slope. More general rulings correspond to curves in the dual space. Characterizing the set of curves that correspond to valid rulings of a polygon is likely to be an important first step to settling algorithmic questions related to Reeb complexity.

6 Conclusions and Open Problems

Many problems on Reeb complexity of polygons and rulings remain open.

1. Give an algorithm to determine if the Reeb complexity of a polygon is at most a given bound b. We conjecture that this problem is NP-Complete.
2. A special case of the problem above is to test if a polygon admits a simple ruling. When this problem was posed at the open problem session of CCCG 2016, David Eppstein observed that a polygon admits a simple ruling if and only if some subdivision of the edges results in a polygon that admits a Hamiltonian triangulation [1]. It may be possible to adapt the algorithm in that paper to this problem. There is likely also a connection to sweepable polygons [4], 2-walkable [3] polygons, and algorithms for detecting them.
3. What rulings correspond to physically realizable rulings? A similar problem is to characterize the rulings that result from a given support set under the effects of gravity.
4. Is every Reeb graph of a ruling on P also the Reeb graph of a continuous function on P?
5. A ruling is called proper if no two line segments share an endpoint. Can the Reeb complexity of a polygon change if we only permit proper rulings?

References

[1] E. M. Arkin, M. Held, J. S. B. Mitchell, and S. Skiena. Hamiltonian triangulations for fast rendering. *The Visual Computer*, 12(9), 1996.

[2] M. P. Bell and D. J. Balkcom. Grasping non-stretchable cloth polygons. *International Journal of Robotics Research*, 29(6), 2010.

[3] B. Bhattacharya, A. Mukhopadhyay, and G. Narasimhan. Optimal algorithms for two-guard walkability of simple polygons. In *Proceedings of the 7th International Workshop on Algorithms and Data Structures*, 2001.

[4] P. Bose and M. van Kreveld. Generalizing monotonicity: On recognizing special classes of polygons and polyhedra. *International Journal of Computational Geometry and Applications*, 15(6), 2005.

[5] J. S. Calcut, R. E. Gompf, and J. D. McCarthy. Quotient maps with connected fibers and the fundamental group. *arXiv preprint arXiv:0904.4465*, 2009.

[6] H. Edelsbrunner and J. Harer. *Computational Topology - an Introduction*. American Mathematical Society, 2010.

[7] J. O'Rourke. *Art Gallery Theorems and Algorithms*, volume 57. Oxford University Press Oxford, 1987.

[8] A. Pressley. *Elementary Differential Geometry*. Springer Science & Business Media, 2010.

[9] G. Reeb. Sur les points singuliers dune forme de pfaff completement intégrable ou dune fonction numérique. *CR Acad. Sci. Paris*, 222(847-849), 1946.

[10] L. Vietoris. Über den höheren Zusammenhang kompakter Räume und eine Klasse von zusammenhangstreuen Abbildungen. *Mathematische Annalen*, 97(1), 1927.

Tilers, Tilemakers, Transformers!

Stefan Langerman*

A *tiling* is a covering of the plane with copies of a geometric shape (*tiles*) without gaps or overlaps. A *tiler* is a shape that tiles the plane.

An *unfolding* is obtained by cutting along the surface of a polyhedron through all its vertices, and opening all the dihedral angles between adjacent faces to obtain a single flat nonoverlapping geometric shape.

A *dissection* is a decomposition of a shape into pieces that, can be rearranged to form another shape.

In this hands-on talk, I will explore connections between these fascinating concepts, in an attempt to shed some light on several still unsolved algorithmic problems, among them:

How easy (or hard) is it to determine if a given geometric shape can tile the plane?

*Département d'Informatique, Université Libre de Bruxelles, stefan.langerman@ulb.ac.be. Directeur de recherches du F.R.S.-FNRS.

A problem on track runners

Adrian Dumitrescu* Csaba D. Tóth†

Abstract

Consider the circle C of length 1 and a circular arc A of length $\ell < 1$. It is shown that there exists $k = k(\ell) \in \mathbb{N}$, and a schedule for k runners along the circle with k distinct but constant positive speeds so that at any time $t \geq 0$, at least one of the k runners is *not* in A.

Keywords: Kronecker's theorem, rational independence, track runners, multi-agent patrolling, idle time.

1 Introduction

In the classic *lonely runner conjecture*, introduced by Wills [11] and Cusick [4], k agents run clockwise along a circle of length 1, starting from the same point at time $t = 0$. They have distinct but constant speeds. A runner is called *lonely* when he/she is at distance of at least $\frac{1}{k}$ from all other runners (along the circle). The conjecture asserts that each runner a_i is lonely at some time $t_i \in (0, \infty)$. The conjecture has only been confirmed for up to $k = 7$ runners [1, 2]. A recent survey [7] lists a few other related problems.

Recently, some problems with similar flavor have appeared in the context of *multi-agent patrolling*, in some one-dimensional scenarios [3, 5, 6, 9, 10]. Suppose that k mobile agents with (possibly distinct) maximum speeds v_i $(i = 1, \ldots, k)$ are in charge of *patrolling* a closed or open fence (modeled by a circle or a line segment). The movement of the agents over the time interval $[0, \infty)$ is described by a *patrolling schedule* (or *guarding schedule*), where the speed of the ith agent, $(i = 1, \ldots, k)$, may vary between zero and its maximum value v_i in any of the two directions along the fence. Given a closed or open fence of length ℓ and maximum speeds $v_1, \ldots, v_k > 0$ of k agents, the goal is to find a patrolling schedule that minimizes the *idle time*, defined as the longest time interval in $[0, \infty)$ during which some point along the fence remains unvisited, taken over all points. Several basic problems are open, such as the following: It is *not* known how to efficiently decide, given $v_1, \ldots, v_k > 0$, and $\ell, \tau > 0$ whether k agents with these maximum speeds can ensure an idle time at most τ when patrolling a segment of length ℓ.

This note is devoted to a question on track runners. As customary, we consider the unidirectional circular track; for convenience we assume runners run clockwise. In the spirit of the lonely runner conjecture, we posed the following question in [7]:

> Assume that k runners $1, 2, \ldots, k$, with distinct but constant speeds, run clockwise along a circle of length 1, starting from arbitrary points. Assume also that a certain half of the circular track (or any other fixed circular arc) is in the shade at all times. Does there exist a time when all runners are in the shade along the track?

Here we answer the question in the negative: the statement does not hold even if the shaded arc almost covers the entire track, e.g., has length 0.999, provided k is large enough.

Notation and terminology. We parameterize a circle of length ℓ by the interval $[0, \ell]$, where the endpoints of the interval $[0, \ell]$ are identified. A *unit* circle is a circle of unit length $C = [0, 1] \mod 1$. A *schedule* of k agents consists of k functions $f_i : [0, \infty] \to [0, \ell]$, for $i = 1, \ldots, k$, where $f_i(t) \mod \ell$ is the position of agent i at time t. Each function f_i is continuous, piecewise differentiable, and its derivative (speed) is bounded by $|f_i'| \leq v_i$. The k agents have *constant speeds* $v_1, \ldots, v_k$, with starting points $\beta_1, \ldots, \beta_k$ when $f_i(t) = v_i t + \beta_i \mod \ell$ for all $i = 1, \ldots, k$. A schedule is called *periodic* with period $T > 0$ if $f_i(t) = f_i(t + T) \mod \ell$ for all $i = 1, \ldots, k$ and $t \geq 0$. $H_n = \sum_{i=1}^{n} 1/i$ denotes the nth *harmonic number*; and $H_0 = 0$. If I is an interval, $|I|$ denotes its length.

2 Track runners in the shade

We first show that the answer to the question posed in [7] is negative in general:

Theorem 1 *Consider a circle C of unit length and a circular arc $A \subset C$ of length $\ell = |A| < 1$. Then there exists $k = k(\ell) \in \mathbb{N}$, and a schedule for k runners with k distinct constant speeds and suitable starting points, so that at any time $t \geq 0$, at least one of the k runners is in the complement $C \setminus A$.*

*Department of Computer Science, University of Wisconsin–Milwaukee, WI, USA. Email: dumitres@uwm.edu

†Department of Mathematics, California State University Northridge, Los Angeles, CA, and Department of Computer Science, Tufts University, Medford, MA, USA. Email: cdtoth@acm.org

Proof. Set $v_i = i$ as the speed of agent i, for $i = 1, \ldots, k$, where $k = k(\ell) \in \mathbb{N}$ will be specified later. Assume, as we may, that $C \setminus A = [0, a]$, for some $a \in (0, 1)$. Let $t_0 = 0$. Since the speed of each agent is an integer (and thereby multiple of the circle length $\mathrm{len}(C) = 1$), the resulting schedule is periodic and the period is 1. To ensure that at any $t \geq 0$, at least one agent is in $[0, a]$, it suffices to ensure this *covering condition* on the time interval $[0, 1]$, i.e., one period of the schedule. All agents start at time $t = 0$; however, it is convenient to specify their schedule with their positions at a later time.

Agent 1 starts at point 0 at time 0; at time a, its position is at a (exiting $[0, a]$). Agent 2 is at point 0 at time a; at time $a + a/2$, its position is at a (exiting $[0, a]$). Agent 3 is at point 0 at time $a + a/2$; at time $a + a/2 + a/3$, its position is at a (exiting $[0, a]$). Subsequent agents are scheduled according to this pattern. For $i = 1, \ldots, k$, agent i is at point 0 at time aH_{i-1}; at time aH_i, its position is at a (exiting $[0, a]$). The schedules are given by the functions $f_i(t) = it - iaH_{i-1}$ for $i = 1, \ldots, k$.

The construction ensures that

1. agent i is in $[0, a]$ during the time interval $[aH_{i-1}, aH_i]$, for $i = 1, \ldots, k$.
2. $\bigcup_{i=1}^{k} [aH_{i-1}, aH_i] \supseteq [0, 1]$.

Indeed, condition 2 is $aH_k \geq 1$, or equivalently $H_k \geq 1/a$. Since $\ln k \leq H_k$, it suffices to have $\ln k \geq 1/a$, or $k \geq \exp(1/a)$, and the theorem is proved. □

The result extends to any finite number of circular arcs $A_1, A_2, \ldots, A_m \subset C$. Stating the results for the complements $B_i = C \setminus A_i$, for $i = 1, 2, \ldots, m$, we can schedule k mobile agents with distinct integer speeds so that at any time $t \geq 0$, each interval B_i contains at least one of the agents.

Theorem 2 *Consider a circle C of unit length and m circular arcs $B_1, B_2, \ldots, B_m \subset C$, for some $m \in \mathbb{N}$. Then there exists $k \in \mathbb{N}$, and a schedule for k runners with k distinct constant speeds and suitable starting points, so that at any time $t \geq 0$, each of the arcs $B_1, B_2, \ldots, B_m$ contains at least one of the k runners.*

Proof. In the proof of Theorem 1, we constructed a schedule of $k(\ell)$ agents with speeds $1, 2, \ldots, k(\ell)$. Note, however, that for any $s \in \mathbb{N}$, we could have used agents of speeds $s + 1, s + 2, \ldots, s + k(\ell, s)$, such that

$$\bigcup_{i=s+1}^{s+k(\ell,s)} [aH_{i-1}, aH_i] \supseteq [0, 1]. \tag{1}$$

Indeed, for every $s \in \mathbb{N}$ there exists $k(\ell, s) \in \mathbb{N}$ satisfying (1), since $\lim_{i\to\infty} H_i = \infty$.

We can schedule $k_1 = k(|B_1|, 0)$ agents with speeds $1, 2, \ldots, k_1$ such that at any time $t \geq 0$, the arc $B_1 = [0, a_1]$ contains at least one of these agents. For arc B_2, we can schedule $k_2 = k(|B_2|, k_1)$ agents with speeds $k_1 + 1, k_2 + 2, \ldots, k_1 + k_2$ such that at any time $t \geq 0$, the arc B_2 contains at least one of them. In general, if the first $i - 1$ intervals are covered, let $s_i = \sum_{j=1}^{i-1} k_i$. Then we can schedule $k_i = k(|B_i|, s_i)$ agents with speeds $s_i + 1, s_i + 2, \ldots, s_i + k_i$ such that at any time $t \geq 0$, the arc B_i contains at least one of them. □

Now that we have seen that the answer to our question is negative in general, it is however interesting to exhibit some scenarios (i.e., conditions) under which the the answer is positive.

A set of real numbers $\xi_1, \xi_2, \ldots, \xi_k$ is said to be *rationally independent* if no linear relation

$$a_1\xi_1 + a_2\xi_2 + \cdots + a_k\xi_k = 0,$$

with integer coefficients, not all of which are zero, holds. In particular, if $\xi_1, \xi_2, \ldots, \xi_k$ are rationally independent, then they are pairwise distinct. Recall now Kronecker's theorem; see, e.g., [8, Theorem 444, p. 382]. (Although inessential, for conformity with the above formulation, our inequalities in Theorems 4 through 6 are strict.)

Theorem 3 (Kronecker, 1884) *If $\xi_1, \xi_2, \ldots, \xi_k \in \mathbb{R}$ are rationally independent, $\alpha_1, \alpha_2, \ldots, \alpha_k \in \mathbb{R}$ are arbitrary, and T and ε are positive reals, then there is a real number $t > T$, and integers $p_1, p_2, \ldots, p_k$, such that*

$$|t\xi_m - p_m - \alpha_m| \leq \varepsilon \quad (m = 1, 2, \ldots, k).$$

As a corollary, we obtain the following result.

Theorem 4 *Assume that k runners $1, 2, \ldots, k$, with constant rationally independent (thus distinct) speeds $\xi_1, \xi_2, \ldots, \xi_k$, run clockwise along a circle of length 1, starting from arbitrary points. For every circular arc $A \subset C$ and for every $T > 0$, there exists $t > T$ such that all runners are in A at time t.*

Proof. Assume, as we may, that $A = [0, a]$, for some $a \in (0, 1)$. Let $0 \leq \beta_i < 1$, be the start position of runner i, for $i = 1, 2, \ldots, k$. Set $\alpha_i = a/2 + 1 - \beta_i$, for $i = 1, 2, \ldots, k$, set $\varepsilon = a/3$, and employ Theorem 3 to finish the proof. □

Remark. It is interesting to note that Theorem 1 gives a negative answer regardless of how long the shaded arc is, while Theorem 4 gives a positive answer regardless of how short the shaded arc is *and* for how far in the future one desires.

Observe that if $\xi_1, \xi_2, \ldots, \xi_k$ are rationally independent reals, then at least one ξ_i must be irrational (in fact, all but at most one ξ_i must be irrational). To obtain the conclusion of Theorem 4 neither the condition that the speeds $\xi_1, \xi_2, \ldots, \xi_k$ are rationally independent, nor the condition that at least one ξ_i is irrational

is necessary. For instance, by controlling the relative speeds allows one to obtain the same result with rational speeds, as shown in the following.

Theorem 5 *Assume that k runners $1, 2, \ldots, k$, with constant but distinct speeds run clockwise along a circle of length 1, starting from arbitrary points. For every circular arc $A \subset C$, there exist distinct rational speeds $v_1, v_2, \ldots, v_k > 0$, so that for every $T > 0$, there exists $t > T$ such that all runners are in A at time t.*

Proof. Assume, as we may, that $[0, a] \subseteq A$, for some rational $a \in (0, 1)$. Let $\beta_1, \beta_2, \ldots, \beta_k$ be the starting points of the runners, where $0 \leq \beta_i < 1$, for $i = 1, 2, \ldots, k$. We proceed by induction on the number of runners k, and with a stronger induction hypothesis extending to every arc A. The base case $k = 1$ is satisfied by setting $v_1 = 1$ for any arc A. The subsequent speeds will be set to increasing values, so that $v_1 < v_2 < \cdots < v_k$.

For the induction step, assume that the statement holds for runners $1, 2, \ldots, k-1$, the arc $A' = [0, a/2]$ and T, and we need to prove it for runners $1, 2, \ldots, k$, the arc $A = [0, a]$ and T. By the induction hypothesis, there exists $t > T$ so that runners $1, 2, \ldots, k-1$, are in A' at time t. Set $v_k = \frac{2}{a} v_{k-1}$; since $a, v_{k-1} \in \mathbb{Q}$, we have $v_k \in \mathbb{Q}$. Observe that runner k will enter the arc A at point 0 before any of the first $k-1$ runners exits A at point a, regardless of his or her starting point. Hence all k runners will be in A at some time in the interval $[t, t + 1/v_k]$, completing the induction step, and thereby the proof of the theorem. □

In Theorem 5, the speeds $v_1, v_2, \ldots, v_k$ ensure that k runners are in a given circular arc $A \subset C$ infinitely many times. Different intervals may require different speeds (based on the relative position of A and the k starting positions). The next theorem shows that, in fact, the same k speeds ensure this property for all circular arcs of a given length $a > 0$. Its proof is very similar to that of Theorem 5; for clarity we include both proofs.

Theorem 6 *Assume that k runners $1, 2, \ldots, k$, with constant but distinct speeds run clockwise along a circle of length 1, starting from arbitrary points. For every $a \in (0, 1)$ there exist distinct rational speeds $v_1, v_2, \ldots, v_k > 0$, so that for every $T \geq 0$ and every circular arc $A \subset C$ of length a, there exists $t > T$ such that all runners are in A at time t.*

Proof. Let $\beta_1, \beta_2, \ldots, \beta_k$ be the starting points of the runners, where $0 \leq \beta_i < 1$, for $i = 1, 2, \ldots, k$. We proceed by induction on the number of runners k. The base case $k = 1$ is satisfied by setting $v_1 = 1$ for any $a > 0$.

For the induction step, assume that the statement holds for $k - 1$ runners, and we need to prove it for k runners. Let an arc length length $a > 0$ and k starting positions $\beta_1, \ldots, \beta_k$ be given. By the induction hypothesis, for the arc length $a' = a/2$ and $k-1$ starting points $\beta_1, \ldots, \beta_{k-1}$ there exist speeds $v_1, \ldots v_{k-1}$ so that for any $T \geq 0$ and any arc $A' \subset C$ of length $a' = a/2$, all runners $1, 2, \ldots, k-1$ are in A' at some time $t > T$.

Set $v_k = \frac{2}{a} v_{k-1}$. Consider an arbitrary arc $A = [\alpha, \alpha + a] \subset C$ of length a. Denote the first half of the arc by $A' = [\alpha, \alpha + a/2]$. At time t, runners $1, 2, \ldots, k-1$ are in A'. Observe that runner k will enter the arc A at point α before any of the first $k-1$ runners exits A at point $\alpha + a$, regardless of his or her starting point. Hence all k runners will be in A at some time in the interval $[t, t + 1/v_k]$, completing the induction step, and thereby the proof of the theorem. □

The speeds of the agents in Theorems 5 and 6 can be chosen as integers if desired, by setting $v_k = \lceil \frac{2}{a} v_{k-1} \rceil$.

3 Conclusions and Open Problems

It is interesting to point out a connection between runners in the shade and idle time (as defined in Section 1). Assume that k runners $1, 2, \ldots, k$, with constant rationally independent (thus distinct) speeds $0 < \xi_1, \xi_2, \ldots, \xi_k \leq 1$, run clockwise along a circle of length 1, starting from arbitrary points. Further assume that $\sum_{i=1}^{k} \xi_i = S$, where $S \leq k$ is large, say, close to k. A straightforward volume argument [5] yields the lower bound $\tau \geq 1/\sum_{i=1}^{k} \xi_i = 1/S \geq 1/k$ on the idle time. On the other hand, by Theorem 3, for every circular arc $A \subset C$ and for every $T > 0$, there exists $t > T$ such that all runners are in A at time t; pick an arbitrary interval A of length $|A| = \varepsilon$, where ε is small. Since the maximum speed of the agents is at most 1, the idle time τ must be at least $|C \setminus A| = 1 - \varepsilon$. The example shows that the volume-based lower bound for the idle time can sometimes be very weak for large k.

The problems we have studied also suggest a few algorithmic questions for a circle C of unit length that we list below.

Problem 1 *Given k runners with speeds $v_1, \ldots, v_k > 0$ and a circular arc $A \subset C$, decide whether there exist starting points $\beta_1, \ldots, \beta_k$ for the k runners, such that at any time $t \geq 0$, at least one of the runners is in A.*

Even if all runners start from the same point (say, $\beta_i = 0$ for all $i = 1, 2, \ldots, k$), it is unclear how to test whether some runner will be in the shade at all times, or all runners will be out of the shade infinitely often.

Problem 2 *Given k runners with speeds $v_1, \ldots, v_k > 0$ starting at 0 and a circular arc $A \subset C$, decide whether there exists $T \geq 0$, such that at any time $t \geq T$, at least one of the runners is in A.*

Problem 3 *Given k runners with speeds $v_1, \ldots, v_k > 0$ starting at 0, and an circular arc $B \subset C$, decide whether for any $T \geq 0$ there exists $t \geq T$ such that all k runners are in B at time t.*

The following two problems are in some sense the "inverses" of Problem 1.

Problem 4 *Given k runners with speeds $v_1, \ldots, v_k > 0$ starting from points $\beta_1, \ldots, \beta_k \in C$, respectively, and an arc length $\ell > 0$, decide whether there exist a circular arc $A \subset C$ of length ℓ and a time $T \geq 0$ such that at any time $t \geq T$ at least one of the runners is in A.*

Problem 5 *Given k runners with starting points $\beta_1, \ldots, \beta_k \in C$, a circular arc $A \subset C$, and a parameter $v > 0$, decide whether there exist rational speeds $v_1, \ldots, v_k \in (0, v)$ and a time $T \geq 0$ such that at any time $t \geq T$ at least one of the runners is in A.*

References

[1] J. Barajas and O. Serra, The lonely runner with seven runners, *Electron. J. Combin.* **15** (2008), R48.

[2] T. Bohman, R. Holzman, and D. Kleitman, Six lonely runners, *Electron. J. Combin.* **8** (2001), R3.

[3] Y. Chevaleyre, Theoretical analysis of the multi-agent patrolling problem, in *Proc. Int. Conf. Intelligent Agent Technology (IAT 2004)*, IEEE, 2004, pp. 302–308.

[4] T. W. Cusick, View-obstruction problems, *Aequationes Math.* **9** (1973), 165–170.

[5] J. Czyzowicz, L. Gasieniec, A. Kosowski, and E. Kranakis, Boundary patrolling by mobile agents with distinct maximal speeds, in *Proc. 19th European Sympos. Algorithms (ESA 2011)*, LNCS 6942, Springer, 2011, pp. 701–712.

[6] A. Dumitrescu, A. Ghosh, and Cs. D. Tóth, On fence patrolling by mobile agents, *Electron. J. Combin.* **21(3)** (2014), P3.4.

[7] A. Dumitrescu and Cs. D. Tóth, Computational Geometry Column 59, *SIGACT News Bulletin* **45(2)** (2014), 68–72.

[8] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers*, fifth edition, Oxford University Press, 1979.

[9] A. Kawamura and Y. Kobayashi, Fence patrolling by mobile agents with distinct speeds, *Distributed Computing* **28** (2015), 147–154.

[10] A. Kawamura and M. Soejima, Simple strategies versus optimal schedules in multi-agent patrolling, in *Proc. 9th International Conference on Algorithms and Complexity (CIAC 2015)*, LNCS 9079, Springer, 2015, pp. 261–273; full paper available at arXiv:1411.6853.

[11] J. M. Wills, Zwei Sätze über inhomogene diophantische Approximation von Irrationalzahlen, *Monatsh. Math.* **71** (1967), 263–269.

Common Development of Prisms, Anti-Prisms, Tetrahedra, and Wedges

Amartya Shankha Biswas* Erik D. Demaine*

Abstract

We construct an uncountably infinite family of unfoldings, each of which can be folded into twelve distinct convex solids, while also tiling the plane.

1 Introduction

The problem of folding polygons into convex polyhedra was posed by Lubiw and O'Rourke [5]. Since then there have been several results about polygons that can fold into multiple distinct polyhedra. One of the first results in this field was by Mitani and Uehara [6], who construct a countably infinite family of unfoldings, each of which can fold into two different orthogonal boxes with integer sides. This was further expanded in [1] and [8] to produce countably infinite families that fold into three different boxes. Other results investigate unfoldings between Platonic solids [7] and between the regular tetrahedron and each Johnson-Zalgaller solid [3].

All of these common unfoldings, however, fold into only a small number of polyhedra. A notable exception is the two examples in [4, sec. 25.6–25.7]: the Latin Cross folds into 23 distinct convex polyhedra, while the square folds into six uncountable families of convex polyhedra. These case studies, however, do not easily generalize to families of unfoldings. It is also relatively easy to make a common unfolding of infinitely many tetrahedra, from any rectangle, but this relies on the simple mechanism of rolling belts and all resulting polyhedra are combinatorially equivalent. Another result that concerns a large number of polyhedra is the common development of 22 pentacubes [2]; however, most of these polycubes are non-convex. This still leaves open the problem of finding large families of common developments of a large number of convex polyhedra; see Sections 2.1–2.2 for further discussion.

We construct a common development that can fold into **twelve** different convex polyhedra, in five different combinatorial classes. Additionally, we show that there is an ***uncountably*** **infinite** family of such developments, each giving rise to twelve different convex polyhedra.

In particular, two of these polyhedra are orthogonal boxes (specifically square prisms). So, if we consider only rational edge lengths, this results in a new infinite family of developments that are common unfoldings of two different (integer-sided) boxes. This is very similar to the results in [6], since we will only be cutting along grid lines.

Another useful property considered in [6, 1, 8, 3] is whether the development tiles the plane. This is a practically important consideration, because it makes the development simple and efficient to fabricate from a sheet of material (no wastage). Our development does in fact tile the plane (Figure 1).

2 Development

The construction of our development starts with a rectangle of paper with size $L \times W$. We assume without loss of generality that $L > W$ and $W = 1$. All instances of $W \neq 1$ can be obtained by scaling the construction appropriately.

We then add square tabs to each set of opposite sides, such that each side has four equally spaced tabs (Figure 1). The tabs on the side with length L are squares of length $L/8$, and the ones on the adjacent sides are of length $W/8 = 1/8$. The tabs on the longer (L) side are shifted by a certain length in order to leave space for the smaller set of tabs (Figure 1). The shift has to be at least $1/8$ to accommodate this, and the maximum possible shift is $L/8$. This means that we require $L > 1$ for the construction to be feasible. On the other hand, the larger tabs extend to a distance $L/8$ into the paper, which also requires that $L/8 < 1 \implies L < 8$. This allows us to bound the aspect ratio $L = L/W$ of the development:

$$1 < L < 8.$$

In our construction, we set the shift distance L_{shift} as

$$L_{shift} = \frac{L/8 + W/8}{2}.$$

The final construction is shown in Figure 1.

The complementary tabs ensure that the pattern can still tile the plane. As an important consequence, this also allow us to "stitch" two opposite sides together without any gaps. This will allow us to pick either pair of opposite sides, and glue them together to form two different cylinders.

We will refer to the cylinder formed by folding around the L side as the L-cylinder, and the one folded around

*MIT Computer Science and Artificial Intelligence Laboratory. {asbiswas,edemaine}@mit.edu.

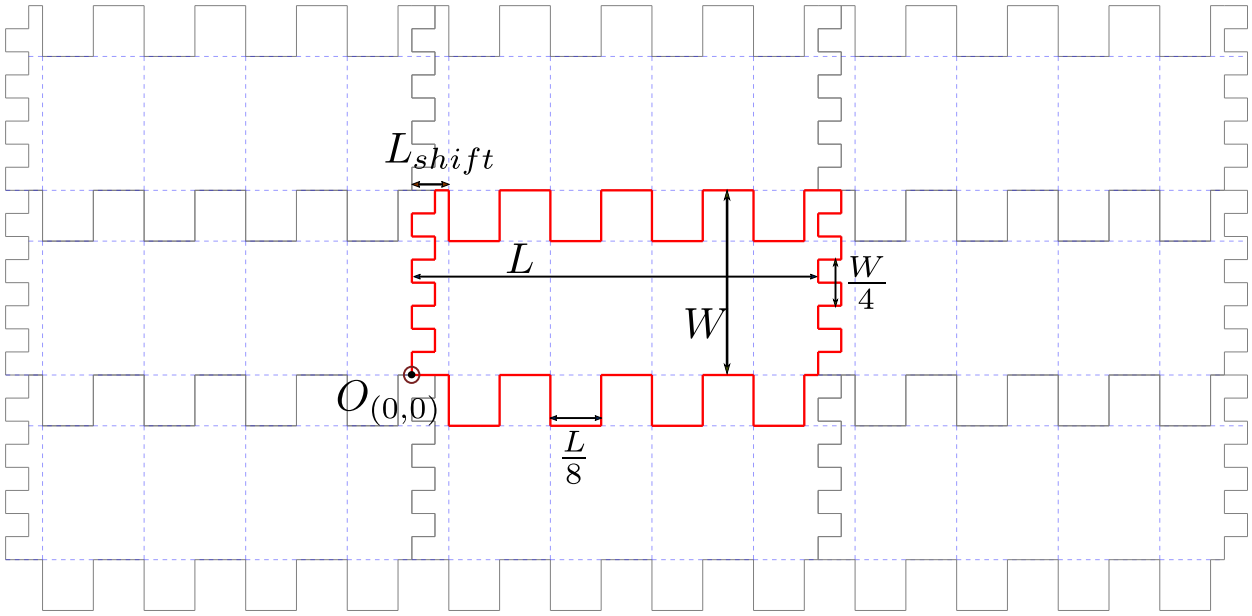
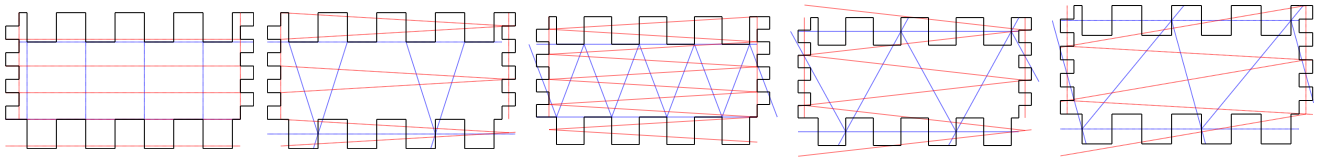


Figure 1: The development is constructed by adding tabs to an $L \times W$ rectangle—four tabs on each side, where opposite sides have complementary tabs. It tiles the plane. The blue dotted creases form one of the possible prisms.



(a) Square Prisms (b) Wedges (c) Square Anti-Prisms (d) Isosceles Tetrahedra (e) Rhombic Disphenoids

Figure 2: Crease patterns for each of the possible foldings of Figure 1. The blue creases correspond to the solid formed by staring with the L-cylinder, and the red creases correspond to the solid formed by staring with the W-cylinder.

the W side as the W-cylinder. These cylinders have complementary sets of tabs on either end. We can close the ends in two different ways. The obvious closing is performed by folding down each of the square tabs by 90° to form a square end cap (Figure 3b). Note that there are actually two different ways to close the square. We can rotate the corners by 45°, and obtain a reflected version of the fold pattern (Figure 3c). The different possible orientations of the square are shown in Figure 5a.

Another way to close the end is to fold the tabs in half and form a straight line (Figure 3a). There are four possible orientations of the line zip, which are formed by varying the endpoints of the zip line (Figure 5b).

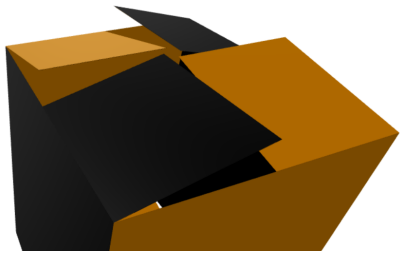
By closing the ends in different ways, we can obtain a large class of convex polyhedra. Each of these will be explained in detail in the following sections.

- **Square prism** — We can close both ends of the cylinder into squares that line up.
- **Square anti-prism** — Same as above, but one of the squares is rotated by 45°.
- **Isosceles tetrahedra** — We close the ends of the cylinders by zipping them into orthogonal lines.
- **Rhombic disphenoids** — We zip the two ends into non-orthogonal lines. Since this solid is chiral, there are two possible foldings. This is essentially a tetrahedron with congruent scalene faces.
- **Obtuse wedges** — We close one of the ends into a square and the other one into a line.

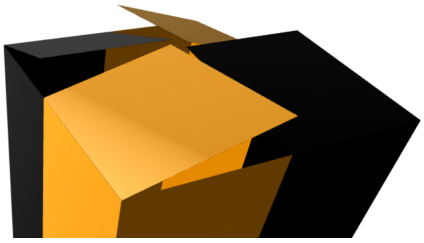
Each of these constructions can be performed by starting with either cylinder. So, we obtain a total of $2 \times 5 = 10$ different convex polyhedra from this unfolding.

Further, the rhombic disphenoids have distinct mirror images which can also be constructed (by turning the folding inside out). This brings the total number of possible foldings to 12. Figure 2 gives the crease patterns for each of these shapes.

(a) Zippng the end of the cylinder into a line. See also Figure 9b.



(b) Closing end into square



(c) Square rotated by $\pi/4$

Figure 3: The three different ways to close the end of a cylinder. Note that the line zipping can also be performed in four different locations (Figure 5b).

Theorem 1. *There is an uncountably infinite family of common developments for the aforementioned twelve polyhedra. Moreover, each member of this family can tile the plane.*

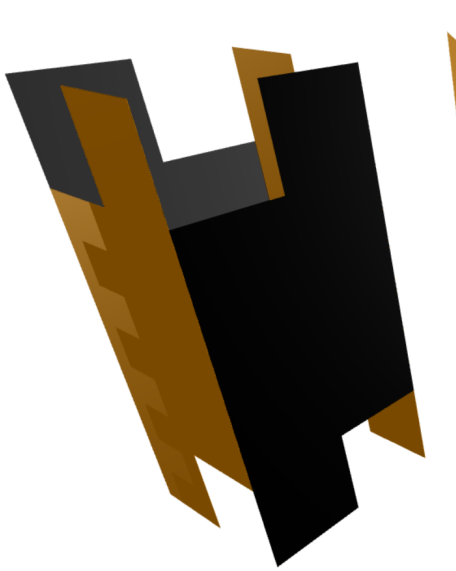
Proof. The development tiles the plane because we start with a rectangle (which tiles the plane), and add tabs, where each tab is added along with its complement. This preserves the tiling property. Since we can vary the length L continuously in the interval $(1, 8)$, this is an uncountable family. The subsequent sections elaborate on the construction of the twelve solids. □

2.1 Comparison to Rolling Belts

Rolling belts offer a trivial way to obtain uncountably infinite polyhedra from the same unfolding. Start with an arbitrary rectangle, and glue opposite sides to form a cylinder. Then the two ends of the cylinder can be zipped in (uncountable) infinitely many ways, to obtain an infinite family of tetrahedron foldings.

This construction is somewhat "uninteresting" because it relies on rolling belts. One way to formalize this is to consider the gluing tree [4] corresponding to each folding, which is the same for all of the tetrahedra gluings. Another property is that all of the resulting polyhedra are combinatorially equivalent, in the sense that their 1-skeleton graphs are identical (K_4), except for two gluings into degenerate doubly covered rectangles.

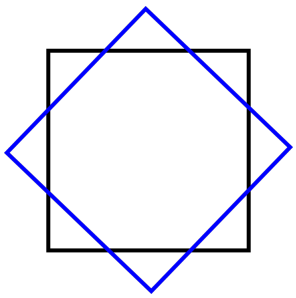
In our results, as well as in past common unfolding results [6, 1, 8], the constructed polyhedra all have different gluing trees, and do not use continuous rolling belts. This is an indicator of the non-triviality of these solutions.



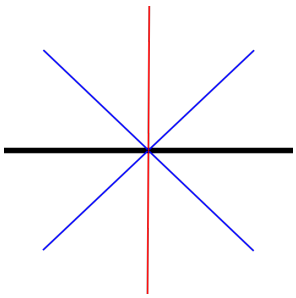
(a) L-cylinder formed by folding along the L direction.

(b) W-cylinder.

Figure 4: We start the construction by folding the development into a cylinder, and attaching the two ends using one set of complementary tabs.



(a) Different orientations to close a square.



(b) Different orientations to zip to a line.

Figure 5: Different ways to close the end of a cylinder. As a convention, the black line indicates the base (bottom side) of the solid.

2.2 Comparison to Box Unfoldings

We claim that the box unfoldings in [6, 1, 8] are all countably infinite, up to scaling.

For instance, consider the construction in [8], which results in a common development of two boxes of size $a \times b \times 8a$ and $a \times 2a \times (2a + 3b)$. An important point to note here is that the construction requires tabs of a specific size. These tabs need to exactly divide both a and b into an integral number of pieces. Thus b/a has to be rational.

Because we are ignoring scale factors, we can set $a = 1$ without loss of generality. So, the number of common developments possible in this setting is just the number of possible values of b, which is a subset of the rationals. Therefore, we obtain a countable family of developments. (Of course, if we reintroduce scale factors, each member of this family will correspond to an uncountably infinite number of scaled copies, one for each positive real number a.)

3 Square Prism

A square prism is a cuboid where one set of opposite faces are squares. So, a square prism is a cuboid of size $a \times a \times b$. For the remainder of this paper, we will abbreviate this as an $a \times b$ prism.

Definition 3.1. The *aspect* ratio of an $a \times b$ prism is defined as b/a.

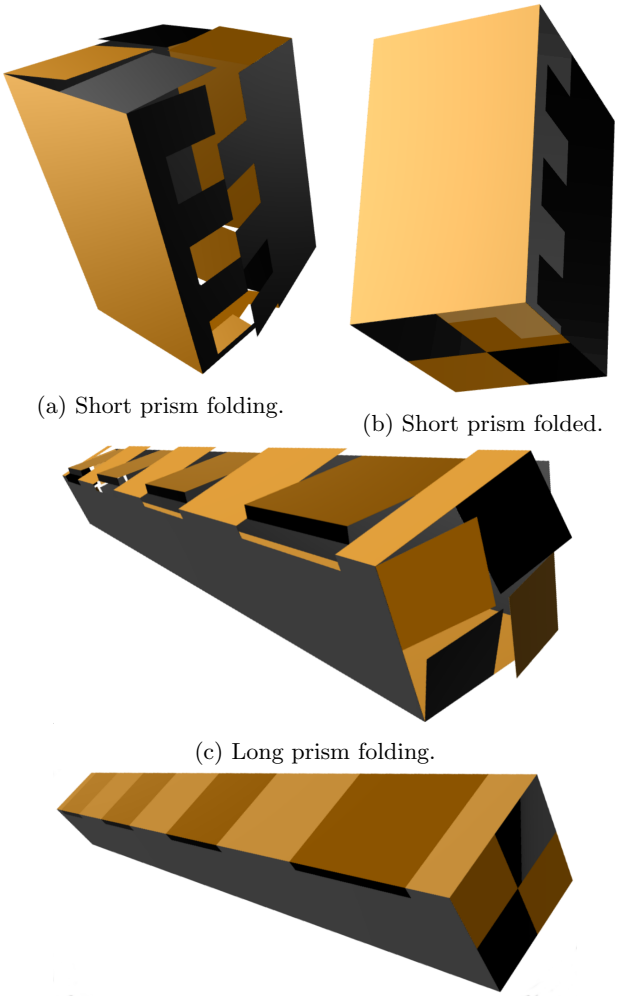
Starting with the two possible cylinders (Figure 4), we can close both ends to make corresponding squares (as in Figure 3b) to obtain two square prisms with different aspect ratios (Figure 6). The crease patterns are in Figure 2a.

- The prism resulting from closing the L-cylinder has aspect ratio $(W - L/8) \times (L/4) = \left(\frac{4}{L} - \frac{1}{2}\right) \times 1$.
- The prism resulting from closing the W-cylinder has aspect ratio $(L - W/8) \times (W/4) = \left(4L - \frac{1}{2}\right) \times 1$

We can compare the two prisms formed by plotting their aspect ratios with respect to the aspect ratio of the starting development; see Figure 7. This gives us the following theorem.

Theorem 2. *Given any aspect ratio $\alpha \in (0, 31.5) \setminus \{3.5\}$, we can construct an unfolding of a prism with aspect ratio α such that the unfolding also folds into a prism with a different aspect ratio. This results in an uncountably infinite family of common unfoldings.*

Proof. If $\alpha \in (0, 3.5)$, then we set $L = \frac{4}{\alpha+0.5}$, and if $\alpha \in (3.5, 31.5)$, then we set $L = \frac{\alpha+0.5}{4}$. This ensures that $1 < L < 8$. Since $L \neq 1$, we can ensure that the two prisms formed have distinct aspect ratios ($4L - 0.5$ and $4/L - 0.5$). Recall that we ignore scale factors by setting $W = 1$. □



(a) Short prism folding.

(b) Short prism folded.

(c) Long prism folding.

(d) Long prism folded.

Figure 6: Two different square prisms from a common development.

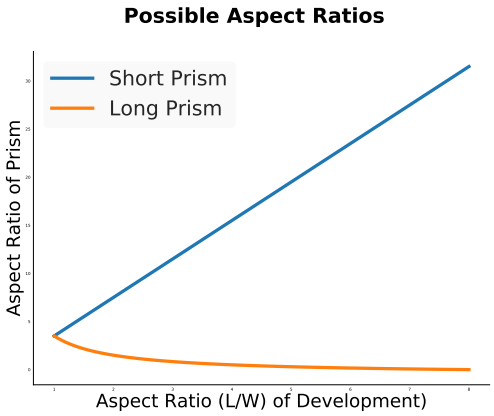
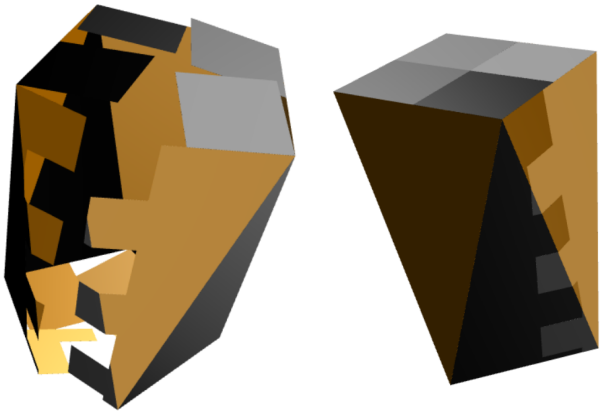


Figure 7

3.1 Anti-prisms

We saw in Figure 3c that we can close a cylinder end into a square that is rotated by 45°. This implies that we can close both end-caps into squares that are offset by a "half-turn". This construction results in a square anti-prism. The two square faces of the anti-prism will be oriented as the blue and black squares in Figure 5a.



(a) Antiprism folding.

(b) Antiprism Folded

Figure 8: Folding the short anti-prism.

As before, we can obtain a short anti-prism by starting with the L-cylinder and a long one by starting with the W-cylinder. A partially folded anti-prism is shown in Figure 8a and the final folded form is in Figure 8b. Both crease patterns are shown in Figure 2c.

4 Isosceles Tetrahedra

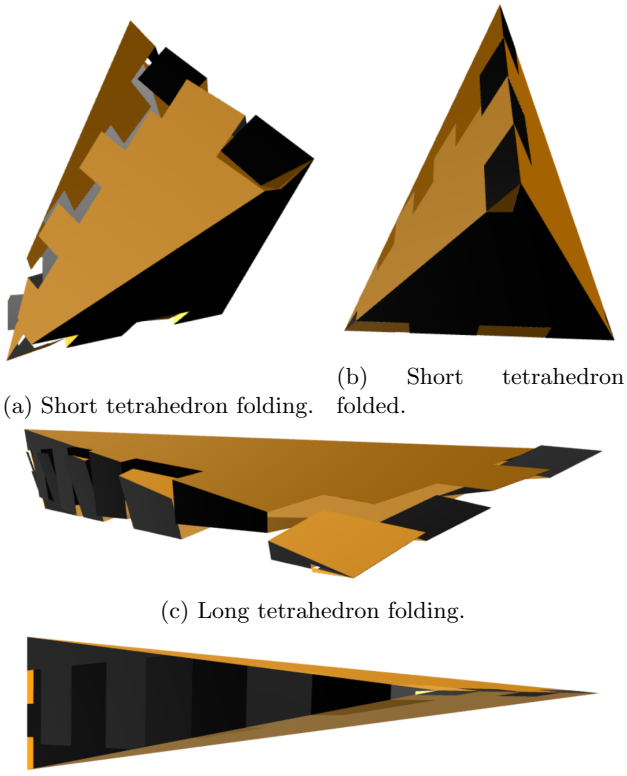
Next, we will consider the solids that are formed by zipping the ends of a cylinder into a line (Figure 3a). Note that we can zip the line in one of four different orientations (Figure 5b). If we let the two ends zip according to the black and red lines in Figure 5b, we obtain a tetrahedron with isosceles faces.

We can construct two different sizes of tetrahedra by starting with either the L or the W cylinder. Both of the possible tetrahedra along with their partially folded states are shown in Figure 9. The crease patterns are in Figure 2d.

Definition 4.1. The *aspect ratio* of an isosceles tetrahedron is defined as the ratio of the height of the isosceles triangle to the length of its base.

The short tetrahedron has an aspect ratio of $L/2 \times W = L \times 2$ and the long tetrahedron has an aspect ratio of $W/2 \times L = 2 \times L$

Theorem 3. *For any aspect ratio $\alpha \in (0.25, 4) \setminus \{2\}$, there is a common unfolding of an α-tetrahedron and a distinct tetrahedron (having different aspect ratio).*



(a) Short tetrahedron folding.

(b) Short tetrahedron folded.

(c) Long tetrahedron folding.

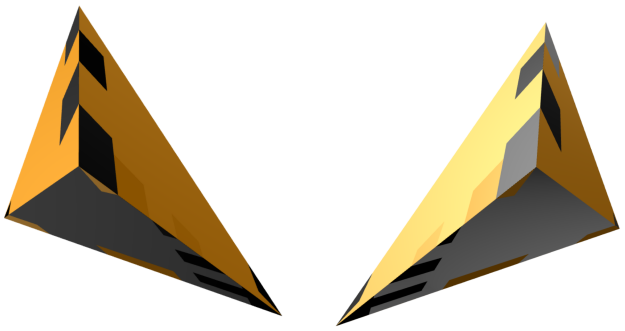
(d) Long tetrahedron folded.

Figure 9: Folding tetrahedra

Proof. We set $L = 2/\alpha$ if $\alpha < 2$ and $L = 2\alpha$ if $\alpha > 2$. Since $L \neq 2$, this results in two different prisms ($L/2$ and $2/L$). □

4.1 Rhombic Disphenoid

We can also obtain non-isosceles tetrahedra by zipping the two ends of a cylinder into non-orthogonal lines. So, we can zip the two ends according to the black and blue lines in Figure 5b.



(a) Rhombic Disphenoid (scalene faces).

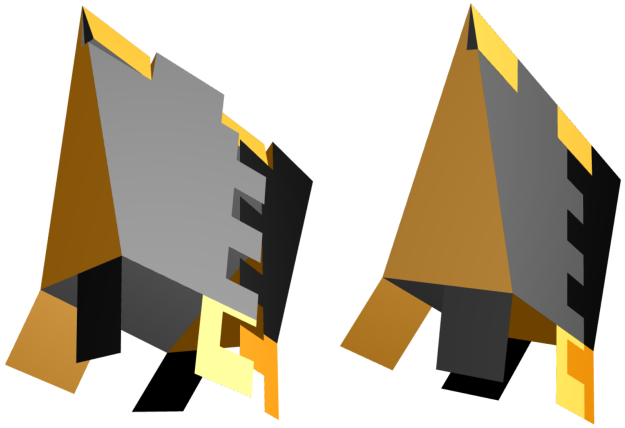
(b) Mirror image.

Figure 10

This construction results in a tetrahedron with congruent scalene triangle faces. This is also called a *rhombic disphenoid*. This is the only polyhedron in this paper that is chiral, and we can form the mirror image by turning the unfolding "inside-out". Both versions of the disphenoid are shown in Figure 10. The crease patterns for both the long and the short disphenoid are in Figure 2e.

5 Obtuse Wedges

In addition to zipping both ends of the cylinder in an equivalent way, we can also zip one end to a line and the other end to a square. This gluing results in a polyhedron with a square base, two triangular side faces, and two trapezoidal side faces (Figure 11). This solid is an obtuse wedge. Both of the crease patterns are shown in Figure 2b.



(a) Wedge folding. (b) Wedge Folded

Figure 11: Zipping two ends differently results in a wedge (half a tetrahedron). The four bottom tabs have to be folded up to complete the square base.

The wedge can also be thought of as a "half tetrahedron": when we extend four side edges, we eventually obtain a tetrahedron (Figure 12). The aspect ratio (Definition 4.1) of this tetrahedron extension is $(W - L/16) \times (L/4) = (16 - L) \times 4L$ for the short wedge, and $(L - W/16) \times (W/4) = (16L - 1) \times 4$ for the long wedge (using $W = 1$).

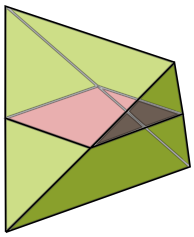


Figure 12: Two wedges forming a tetrahedron.

6 Conclusion

In this paper, we constructed an uncountable family of common developments. Unlike the majority of previous results, these developments fold to more than three convex polyhedra. It may be possible to extend the basic ideas from the tab construction to other types of polygons and obtain more interesting unfolding families. As a bonus, our developments tile the plane, which has practical implications.

Acknowledgements

This work was initiated during an open problem session in the MIT course 6.849 on Geometric Folding Algorithms: Linkages, Origami, Polyhedra in Spring 2017. We thank the other participants for providing a stimulating research environment.

References

[1] Z. Abel, E. Demaine, M. Demaine, H. Matsui, G. Rote, and R. Uehara. Common developments of several different orthogonal boxes. In *The 23rd Canadian Conference on Computational Geometry (CCCG'11)*, 2011.

[2] G. Aloupis, P. K. Bose, S. Collette, E. D. Demaine, M. L. Demaine, K. Douïeb, V. Dujmović, J. Iacono, S. Langerman, and P. Morin. Common unfoldings of polyominoes and polycubes. In *Computational Geometry, Graphs and Applications*, pages 44–54. Springer, 2011.

[3] Y. Araki, T. Horiyama, and R. Uehara. Common unfolding of regular tetrahedron and Johnson-Zalgaller solid. In *Journal of Graph Algorithms and Applications, Vol.20, No.1*, pages 101–114, 2016.

[4] E. D. Demaine and J. O'Rourke. *Geometric Folding Algorithms*. Cambridge University Press, 2007.

[5] A. Lubiw and J. O'Rourke. When can a polygon fold to a polytope. In *Technical Report Technical Report 048, Department of Computer Science, Smith College*, 1996.

[6] J. Mitani and R. Uehara. Polygons folding to plural incongruent orthogonal boxes. In *Proc. CCCG*, pages 31–34, 2008.

[7] T. Shirakawa, T. Horiyama, and R. Uehara. On common unfolding of a regular tetrahedron and a cube. pages 47–50, 2011.

[8] T. Shirakawa and R. Uehara. Common developments of three incongruent orthogonal boxes. In *International Journal of Computational Geometry and Applications, Vol. 23, No. 1*, pages 65–71, 2013.

Computing 3SAT on a Fold-and-Cut Machine

Byoungkwon An* Erik D. Demaine* Martin L. Demaine* Jason S. Ku*

Abstract

This paper introduces a computational model called a *fold-and-cut machine* which allows as operations simple folds and unfolds, straight-line cuts, and inspection for a *through-hole* (hole through all the layers of paper). We show that a (deterministic) fold-and-cut machine can decide a 3SAT instance with n variables and m clauses using $O(nm+m^2)$ operations (with just one cut and inspection), showing that the machine is at least as powerful as a nondeterministic Turing machine.

1 Introduction

Computational origami [DO08] is usually about using algorithms (on traditional computers) to design/analyze paper foldings. But what if we view the piece of paper as the computer itself, with folding as one of the operations provided by the model of computation? In this paper, we initiate the study of what computation is possible in such folding models.

A *fold-and-cut machine* operates on a polygonal 2D piece of paper, supporting four operations—Fold, Unfold, Cut, Look—that modify the current flat folded state of the piece of paper (initially flat) and the piece of paper itself.

1. A Fold operation is any simple fold [ADK, ABD+04]: a fold of some number of layers along a straight line by $\pm 180°$.

2. An Unfold operation undoes a Fold operation.

3. A Cut operation is a complete straight-line cut through (all layers of) the current flat folding, discarding all but one specified piece.

4. A Look operation decides whether the current flat folding has a *through-hole*, i.e., has nonzero genus (a hole) when imagining all touching layers to be fused together.

Each line (in a Fold or Cut operation) is specified by a distinct pair of points with integer coordinates using a polynomial number of bits. The initial polygon of paper is similarly described by integer vertex coordinates each using a polynomial number of bits. (In fact, in our constructions, the paper will be a rectangle and the lines will all be horizontal, vertical, or 45° diagonal.)

Models involving just Fold and Unfold operations have been considered before [CDD+11, Ueh11], but where the goal was to achieve certain geometric folding properties instead of computation. We add the ability to make cuts, though in fact we will use just a single cut, in the style of the fold-and-one-cut problem [DO08, DDL98, BDEH01], also previously considered in the context of simple folds [DDH+10], but with geometric instead of computational goals.

A *3SAT instance* is a Boolean formula over n variables $X = \{x_1, x_2, \ldots, x_n\}$ in conjunctive normal form (CNF):

$$\bigwedge_{i=1}^{m} (a_i \vee b_i \vee c_i), \tag{1}$$

where a_i, b_i, c_i are called literals, each corresponding to some variable in X, or its negation. Each term $(a_i \vee b_i \vee c_i)$ is called a *CNF clause*. A 3SAT instance is *satisfiable* if there exists an assignment of each variable as either 0 or 1 such that the Boolean formula evaluates to 1. Deciding satisfiability of a given 3SAT instance is a classic NP-complete problem [GJ79].

Theorem 1 *A fold-and-cut machine can decide 3SAT in $O(nm+m^2)$ operations, using just one* Cut *and just one* Look.

As a consequence, all decision problems in NP can be solved by a polynomial number of operations on a fold-and-cut machine, making it at least as powerful as a nondeterministic Turing machine.

Inspired by the recently introduced *fold-and-one-punch problem* [ADD+], we also consider an alternative folding model of computation that replaces the Cut operation as follows:

3′. A Punch operation cuts a point hole (or a small circular hole) through (all layers of) the current flat folding.

(Again the point has integer coordinates specified by a polynomial number of bits.) We prove that Theorem 1 also holds on this *fold-and-punch machine*. In fact, this construction works even with "1.5D paper": a narrow rectangular strip that can be folded only perpendicular to the strip direction, but which remains connected after punching a hole.

*CSAIL, Massachusetts Institute of Technology, {dran, edemaine,mdemaine,jasonku}@csail.mit.edu

This paper is organized as follows. Section 2 introduces the structure of our approach, representing a 3SAT instance as a DNF formula with a symmetric clause ordering. Section 3 describes a set of holes in a piece of paper that correspond to a 3SAT instance. Section 4 shows that the resulting paper can be folded so that the outcome of a single Look operation is equivalent to the solution to the 3SAT instance. Section 5 shows how to produce the described hole pattern using polynomially many Fold operations and one Cut operation. Section 6 puts all of these pieces together to prove the theorem. We conclude in Section 7 with open problems for future work.

2 Approach

Conceptually, we use the known conversion from any 3CNF formula into disjunctive normal form (DNF) [GJ79], which results in an OR of 3^m DNF clauses, where each DNF clause is an AND of m of the CNF literals, one from each CNF clause. Our reduction from 3SAT cannot afford to actually compute this DNF formula, but the fold-and-cut machine we produce will end up physically constructing a truth table for the DNF formula where holes represent 1s.

Deciding whether the 3SAT instance has a solution is equivalent to finding an assignment of the variables $\alpha : X \to \{0,1\}$ for which some DNF clause evaluates to 1. There are 3^m DNF clauses and 2^n possible assignments of variables. Let $d(i,j,k)$ represent the Boolean value of the ith CNF clause's literal present in the jth DNF clause when using the kth assignment for the variables, with $i \in [1,m]$, $j \in [1,3^m]$, and $k \in [1,2^n]$. In particular, if the literal from the ith CNF clause present in the jth DNF clause is variable x_r, then $d(i,j,k) = \alpha_k(x_r)$, while if the literal is the negation of x_r, then $d(i,j,k) = \neg\alpha_k(x_r)$. Thus, there are $m(3^m)(2^n)$ possible $d(i,j,k)$. Then the DNF formula associated with a 3SAT instance is equivalent to evaluating the exponentially sized Boolean formula:

$$\bigvee_{k=1}^{2^n} \bigvee_{j=1}^{3^m} \bigwedge_{i=1}^{m} d(i,j,k). \tag{2}$$

To prove Theorem 1, we will use a fold-and-cut machine to operate on a rectangular strip of paper P with unit width and length $m(3^m)(2(2^n)+2)$. We will conceptually divide P evenly into unit-square *cells*. We will associate two cells with each possible $d(i,j,k)$, with the $m(3^m)2$ remaining cells not associated with any $d(i,j,k)$. We will cut a hole in the center of $m(3^m)(2^n)$ cells of P, resulting in a modified paper P', cutting a hole exactly when a cell has a $d(i,j,k)$ associated with it which evaluates to 1.

To decide whether the 3SAT instance has a satisfying assignment, we will give a folding of P' such that every

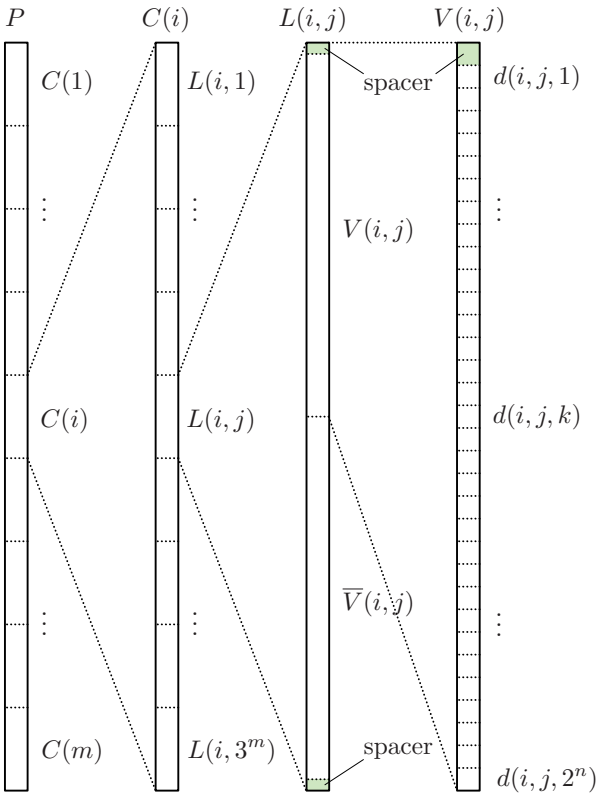


Figure 1: Layout of blocks and cells of P.

cell overlaps with exactly $m-1$ other cells. Further, any cell associated with $d(i,j,k)$ overlaps with cells associated with each $d(i',j,k)$ for $i' \in [1,m]$. Thus, if the folding has a through-hole passing through a cell corresponding to $d(i,j,k)$, then DNF clause j evaluates to 1 under variable assignment k. And if a cell associated with $d(i,j,k)$ does not have a through-hole, then DNF clause j evaluates to 0 under variable assignment k. A Look operation restricted to the folded location of a cell associated with $d(i,j,k)$ will evaluate the formula

$$\bigwedge_{i=1}^{m} d(i,j,k), \tag{3}$$

while the entire Look operation performs the same test in parallel for all DNF clauses j, over all variable assignments k. We will explicitly evaluate each of the exponentially many $d(i,j,k)$ values in some cell, but we will be able to produce them using only polynomially many operations.

3 Hole Locations

First, we will describe how cells of P are associated with each $d(i,j,k)$; see Figure 1. Conceptually, we will divide P into consecutive equally-sized sets of cells called

blocks. Dividing P into a different number of equally-sized sets will partition P at different levels of detail, into blocks of different size. At the coarsest level, the paper is divided into m *CNF blocks*, one for each 3CNF clause. Each CNF block $C(i)$ contains 3^m literal blocks, one for each DNF clause. Each literal block $L(i,j)$ contains two variable blocks $V(i,j)$ and $\overline{V}(i,j)$, which are mirror reflections of each other. Each variable block $V(i,j)$ is made up of 2^n+1 cells.

The first cell $\sigma(i,j,1)$ in a variable block is a blank cell not associated with any $d(i,j,k)$, which we call a *spacer*. The remaining 2^n cells in variable block $V(i,j)$ are associated with $d(i,j,k)$, one for each variable assignment, with cell $\sigma(i,j,k+1)$ associated with $d(i,j,k)$. The desired hole pattern P' contains a hole in a cell exactly when it is associated with a $d(i,j,k)$ equal to 1. We call a cell associated with a $d(i,j,k)$ equal to 1 a *hole cell*, with *blank cells* referring to any other cell. We will show in Section 4 that we can fold P' so that a Look operation can decide the 3SAT instance, and in Section 5 we can produce P' from P in polynomially many operations.

We have not yet fully specified the hole pattern for P' because we have not fixed an ordering for the DNF clauses or the variable assignments. For our construction, the DNF clauses and the variable assignments must be ordered in a highly symmetric way to allow blocks to be aligned with a polynomial number of folds, though the CNF clauses may be ordered arbitrarily. We order the DNF clauses in the following way. Let DNF clause $\mathcal{D}(j)$ be defined as:

$$\mathcal{D}(j) = \bigwedge_{i=1}^{m} l(i,j), \tag{4}$$

where $l(i,j)$ represents the literal from CNF clause i appearing in DNF clause j, according to:

$$l(i,j) = \begin{cases} a_i & \text{if } \left\lfloor \frac{j-1}{3^{m-i}} \right\rfloor \equiv 0 \text{ or } 5 \mod 6 \\ b_i & \text{if } \left\lfloor \frac{j-1}{3^{m-i}} \right\rfloor \equiv 1 \text{ or } 4 \mod 6 \\ c_i & \text{if } \left\lfloor \frac{j-1}{3^{m-i}} \right\rfloor \equiv 2 \text{ or } 3 \mod 6 \end{cases}. \tag{5}$$

Conceptually, this order corresponds to the following layout. Each CNF block $C(i)$ contains 3^{i-1} *unit blocks* made up of three adjacent *subunit blocks*, one for each literal in $\{a_i, b_i, c_i\}$. Each subunit block contains 3^{m-i} literal blocks, all of which are associated with the same literal. Further, every unit block is a reflection of each adjacent unit block.

For variable assignments, we represent an assignment as a binary string and list them in lexicographical order, with the first variable being the left most digit in the binary string. So for a 3SAT instance with 5 variables, the first variable assignment among the 2^n assignments will be 00000, while the tenth variable assignment will be 01001.

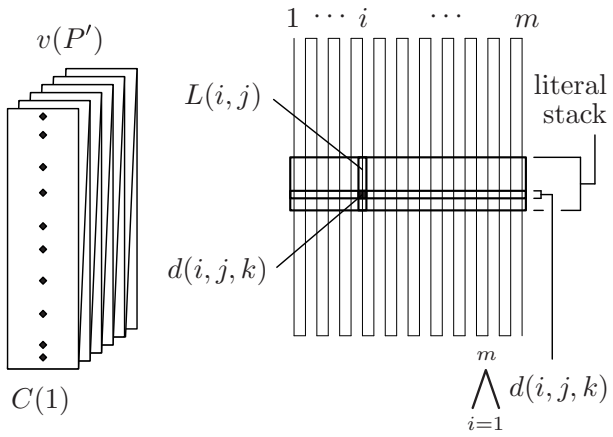


Figure 2: Flat folding $v(P')$ aligns cell stacks containing cells corresponding to $d(i,j,k)$ for all $i \in [1,m]$, and in doing so, enables verification of the instance using a single Look operation.

4 Verification

Let $v(P')$ be a flat folding of P' produced by pleating the m CNF blocks back and forth on top of each other. This folding can easily be produced by a sequence of simple folds by folding each crease in order from top to bottom.

Flat-folding $v(P')$ has m layers uniformly at every point. In particular, every cell folds onto $m-1$ other cells, while each literal block overlaps and aligns with $m-1$ other literal blocks, one from each CNF block. We call a set of aligned and overlapping blocks of the same size a *stack*. For example, we call m overlapping literal blocks a literal stack.

Lemma 2 *Flat-folding $v(P')$ has a through-hole if and only if the 3SAT instance associated with P' is satisfiable.*

Proof. We first prove a bijection between literal stacks and DNF clauses, by showing that, for any DNF clause $\mathcal{D}(j)$, there exists a literal stack containing literal blocks corresponding to the CNF clause literals present in $\mathcal{D}(j)$. Let $l(i,j)$ be the literal in $\mathcal{D}(j)$ associated with CNF clause i. CNF block $C(1)$ contains a single unit block with three subunit blocks, one for each literal. If some literal stack represents $\mathcal{D}(j)$, it must contain a literal block from the subunit block corresponding to literal $l(1,j)$. Now consider CNF block $C(i)$ for $i > 1$. By construction, every unit block of $C(i)$ is exactly the same size as a subunit block of $C(i-1)$, and every subunit block of $C(i-1)$ will exactly align and overlap with some unit block of $C(i)$ in $v(P')$. Thus by induction, there exists a literal stack containing each $l(i,j)$ in $\mathcal{D}(j)$. There are 3^m DNF clauses and 3^m literal stacks, so there is a bijection between them.

Lastly, given a literal stack corresponding to DNF clause $\mathcal{D}(j)$, we show that there is a through-hole in the stack if and only if the DNF clause is satisfiable. Each literal block is mirror symmetric containing a variable block $V(i,j)$ and its reflection $\overline{V}(i,j)$, and variable blocks exactly overlap other variable blocks in two symmetric stacks. In particular, within a variable stack, each cell associated with $d(i,j,k)$ overlaps a cell corresponding to every other $d(i',j,k)$ for $i' \in [1,m]$, and $d(i,j,k)$ has a hole exactly when $d(i,j,k)$ is 1 by construction. Thus, if $\mathcal{D}(j)$ evaluates to 1 using variable assignment k, then there will be a through-hole, and there will be no through-hole if some $d(i,j,k)$ in the variable stack is 0, completing the proof. □

5 Making Holes

In this section, we show how to produce the holes in P' from paper P in $O(nm+m^2)$ FOLD operations and one CUT operation. We will fold P in two stages. First, we will show how to fold a CNF block $C(i)$ into three literal stacks, with each literal stack containing the 3^{m-1} literal blocks in $C(i)$ corresponding to the same literal. Second, we will show how to fold a literal block $L(i,j)$ to align all hole cells to the same location.

5.1 Align Literals

Given a CNF block $C(i)$, we align literal blocks corresponding to the same literal by folding along a sequence of *symmetric two-fold pleats*. A symmetric two-fold pleat folds a block along two lines dividing the block into equal thirds. The upper crease will be a valley fold, and the lower crease will be a mountain fold.

Lemma 3 *We can fold any CNF block $C(i)$ into three adjacent literal stacks using a sequence of $2(i-1)+6(m-i)$ simple folds, with each literal stack containing the 3^{m-1} literal blocks corresponding to the same literal.*

Proof. The folding follows directly from the order of $\mathcal{D}(j)$ defined by $l(i,j)$; see Figure 3. First, we overlap every unit block on top of one another by repeatedly folding through all layers along $i-1$ symmetric two-fold pleats. These folds overlap the unit blocks in the same orientation because unit blocks alternate in orientation in the layout defined by $l(i,j)$. Then, we overlap the literal blocks in each of the three adjacent subunit blocks by using $m-i$ symmetric two-fold pleats. Each symmetric two-fold pleat can be folded using two simple folds by first valley folding along the higher fold, then folding back down along the lower fold. The first step uses $2(i-1)$ FOLD operations, while the second step uses $3(2)(m-i)$. □

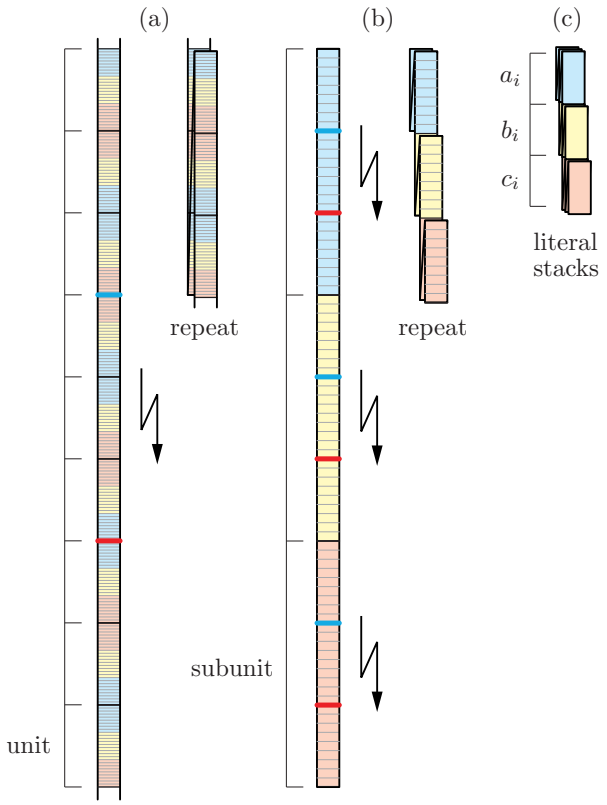


Figure 3: Aligning literal blocks associated with the same literal in the CNF clause corresponding with CNF block $C(i)$. (a) Align unit blocks from $C(i)$ on top of each other using $i-1$ pleats. (b) Collapse each subunit stack into one literal stack using $m-i$ pleats. (c) The folding after aligning the literal blocks into three stacks.

5.2 Align Hole Locations

We align hole locations of a literal block using the following procedure; see Figure 4. First, valley fold the literal block down along its lower boundary, and fold back up along the line separating $V(i,j)$ and $\overline{V}(i,j)$. Because $V(i,j)$ and $\overline{V}(i,j)$ are symmetric (recall that they are mirror reflections of each other), the resulting flat-folded pleat will have through-holes at the same locations as $V(i,j)$.

Lemma 4 *We can fold any variable block $V(i,j)$ onto the first three cell locations of the block using at most $2n$ simple folds, such that a cell folds to the second cell location from the top if and only if it is a hole cell.*

Proof. We prove by constructing such a folding. By definition, each variable block comprises a spacer cell followed by 2^n cells, with cell $\sigma(i,j,k+1)$ associated with $d(i,j,k)$. Also, each variable block is associated with some variable x_r or its negation. Because of the lexicographical order of the variable assignments, the

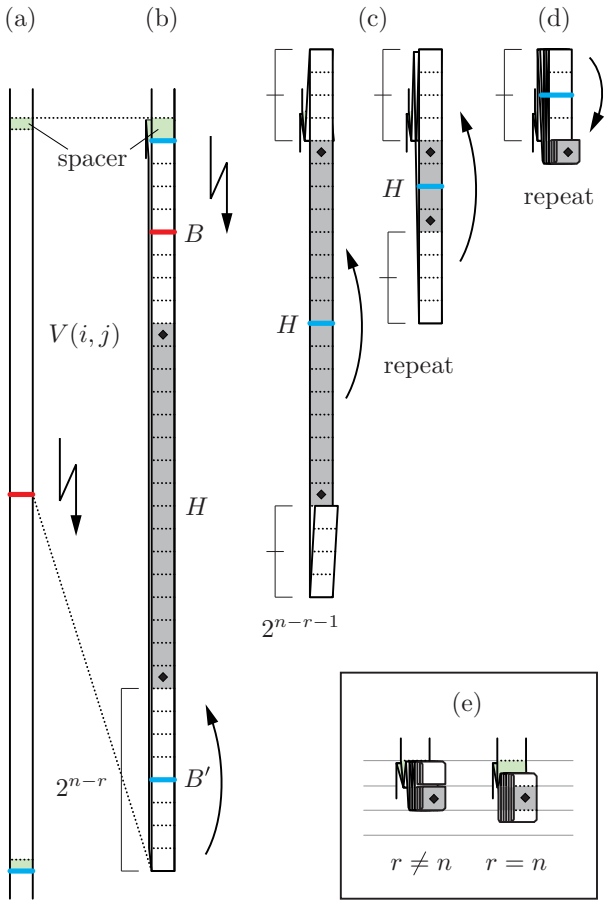


Figure 4: Aligning hole cells in literal block $L(i, j)$. (a) Fold $L(i, j)$ to align variable blocks. (b) Pre-process leading or trailing blank cells B or B' of $V(i, j)$. (c) Align hole cells by folding H in half $n - 1$ times. (d) Get rid of extra blank cells by folding the excess with height 2^{n-r-1} in half $n-r-1$ times. (e) Final state for two cases, when $r \neq$ and when $r = n$.

hole pattern described by $d(i, j, k)$ within the variable block is an alternating sequence of 2^{n-r} hole cells and 2^{n-r} blank cells. If the variable block is associated with variable x_r, then cell $\sigma(i, j, 1)$ is a blank cell, while if the block is associated with $\neg x_r$, then $\sigma(i, j, 1)$ is a hole cell.

First, we perform a preprocessing step. Let B be the block of blank cells between the spacer and the first hole cell exclusive. If B is nonempty, it has length 2^{n-r}. Valley fold B up along its top boundary line, and valley fold it back down along the line dividing its cells equally. This aligns the first hole cell onto the second cell location. Otherwise, let B' be the block of blank cells after the last hole cell. If B' is nonempty, it has length 2^{n-r}. Valley fold B' in half, bringing its bottom edge to its top. Note that exactly one of B and B' will

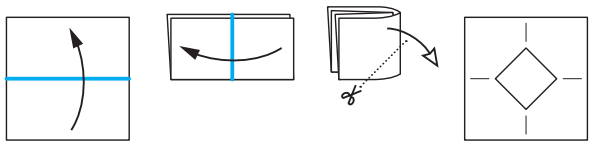


Figure 5: How to cut a diamond-shaped hole.

be nonempty.

Second, we perform hole alignment. Let H be the block of cells between the first hole cell and the last hole cell inclusive. Because the hole pattern is an alternating sequence of 2^{n-r} hole cells and blank cells, H is symmetric. Valley fold H in half by folding the bottom half of H along with any attached blank cells upward. Because H is symmetric, hole cells of this variable block will only overlap other hole cells from the block. Because of the alternation and symmetry of the hole pattern, the block of cells on the top-most layer of this resulting flat folding between the first hole cell and the last hole cell will again be symmetric. So we can repeatedly fold the block containing the hole cells in half until all hole cells are aligned in the second cell location. After repeating this procedure, there will either be no paper below the second cell, or if $r = n$, half a cell of paper will extend below the second cell. In any case, no paper from $V(i, j)$ will extend below the third cell location.

Next, we fold away blank cells extending above the first cell location. After we align the hole cells, excess blank cells will exist above the second cell location. These cells extend above the second hole location by length 2^{n-r-1}, equal to half the width of a 2^{n-r} block of blank cells. We can fold all empty cells extending above the second cell location onto the spacer cell by folding the excess cells in half $n-r-1$ times, or not at all if $n - r - 1 < 1$.

The preprocessing step requires either 1 or 2 simple folds, the hole alignment steps require $n-1$ simple folds, while collapsing empty cells requires at most $n - r - 1$ simple folds. Since $r \in [1, n]$, this folding uses at most $2n$ folds. □

5.3 Cutting a Hole

Lemma 5 *We can make a diamond-shaped hole centered on a hole location on the interior of a flat folding using two* Fold *operations and one* Cut *operation.*

Proof. See Figure 5. Fold in half horizontally through the hole location and then vertically though the same point. Then cut off the paper containing the hole location with a cut line at 45°. □

On a fold-and-punch machine, this lemma can be replaced by a direct Punch operation.

6 Proof of Theorem

Now we are ready to prove Theorem 1.

Proof. We produce hole pattern P' from P according to a given 3SAT instance by doing the following. Fold each CNF block of P to align literal blocks as described in Lemma 3, which folds CNF block $C(i)$ using $2(i-1)+6(m-i)$ FOLD operations, for a total of $4(m^2-m)$. Then, for each of the $3m$ resulting literal stacks, pleat it in half to align each variable block $V(i,j)$ and $\overline{V}(i,j)$ onto each other, and align hole locations as described in Lemma 4, using at most $2n$ FOLD operations per literal stack. Perform this step on literal stacks starting from the lowest literal stack up to the highest. This ordering ensures that we are always performing simple folds on a set of topmost layers. The result is a flat folding contained within the space of three cells, with all hole cells overlapping at the second cell location. Then use the construction in Lemma 5 to construct a hole centered on the second cell location, using two FOLD operations and one straight CUT. This folding uses $O(nm+m^2)$ FOLD operations.

Now unfold to P' using $O(nm+m^2)$ UNFOLD operations. Then, folding P' to $v(P')$ using $m-1$ FOLD operations results in a flat folding which, by Lemma 2, has a through-hole if and only if the original 3SAT instance is satisfiable. Thus a single LOOK operation completes the computation. □

7 Open Problems

We suggest some open problems for future study. What is the full computation power of a fold-and-cut machine? We can simulate t time units of a fold-and-cut machine on a RAM in $2^{O(t)}$ time. Can a polynomial-time fold-and-cut machine simulate all of EXPTIME, or at least PSPACE? The model seems related to parallel computation; a natural goal would be to solve problems in EXPTIME representable by a polynomial-depth circuit. More generally, if we allow input coordinates to be arbitrary real numbers, do we get $\mathbb{R}$ versions of complexity classes such as $\mathrm{NP}_{\mathbb{R}}$ [BCSS98]? Is it possible to solve NP with just FOLD/UNFOLD operations, without a CUT? (This may require adding a different query operation to the model.)

Acknowledgments The authors would like to thank Alex Cornejo, Eli Groban, Owen Macindoe, Shuhei Miyashita, Jimmy Wong, and Damien Woods for their insightful discussions and support.

References

[ABD+04] Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Martin L. Demaine, Joseph S. B. Mitchell, Saurabh Sethia, and Steven Skiena. When can you fold a map? *Computational Geometry: Theory and Applications*, 29(1):23–46, 2004.

[ADD+] Yasuhiko Asao, Erik D. Demaine, Martin L. Demaine, Hideaki Hosaka, Akitoshi Kawamura, Tomohiro Tachi, and Kazune Takahashi. Folding and punching paper. *Journal of Information Processing*. To appear. Special issue of papers from the 19th Japan Conference on Discrete and Computational Geometry, Graphs, and Games, September 2016.

[ADK] Hugo Akitaya, Erik D. Demaine, and Jason S. Ku. Simple folding is really hard. *Journal of Information Processing*. To appear. Special issue of papers from the 19th Japan Conference on Discrete and Computational Geometry, Graphs, and Games, September 2016.

[BCSS98] Lenore Blum, Felipe Cucker, Michael Shub, and Steve Smale. *Complexity and Real Computation*. Springer-Verlag, 1998.

[BDEH01] Marshall Bern, Erik Demaine, David Eppstein, and Barry Hayes. A disk-packing algorithm for an origami magic trick. In *Origami^3: Proceedings of the 3rd International Meeting of Origami Science, Math, and Education*, pages 17–28. A K Peters, 2001.

[CDD+11] Jean Cardinal, Erik D. Demaine, Martin L. Demaine, Shinji Imahori, Tsuyoshi Ito, Masashi Kiyomi, Stefan Langerman, Ryuhei Uehara, and Takeaki Uno. Algorithmic folding complexity. *Graphs and Combinatorics*, 27(3):341–351, 2011.

[DDH+10] Erik D. Demaine, Martin L. Demaine, Andrea Hawksley, Hiro Ito, Po-Ru Loh, Shelly Manber, and Omari Stephens. Making polygons by simple folds and one straight cut. In *Revised Papers from the China-Japan Joint Conference on Computational Geometry, Graphs and Applications*, Lecture Notes in Computer Science, pages 27–43, Dalian, China, November 2010.

[DDL98] Martin L Demaine, Erik D Demaine, and Anna Lubiw. Folding and cutting paper. In *Revised Papers from the Japan Conference on Discrete and Computational Geometry*, volume 1763 of *Lecture Notes in Computer Science*, pages 104–118, December 1998.

[DO08] Erik D Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra*. Cambridge University Press, August 2008.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[Ueh11] Ryuhei Uehara. Stamp foldings with a given mountain-valley assignment. In *Origami^5: Proceedings of the 5th International Meeting of Origami Science, Math, and Education*, pages 585–597. A K Peters/CRC Press, November 2011.

Data Structures for Fréchet Queries in Trajectory Data*

Mark de Berg Ali D. Mehrabi Tim Ophelders

Abstract

Let π be a trajectory in the plane, represented as a polyline with n edges. We show how to preprocess π into a data structure such that for any horizontal query segment σ in the plane and a subtrajectory between two vertices of π, one can quickly determine the Fréchet distance between σ and that subtrajectory. We provide data structures for these queries that need $O(n^2 \log^2 n)$ preprocessing time, $O(n^2 \log^2 n)$ space, and $O(\log^2 n)$ query time. If we are interested only in the Fréchet distance between the complete trajectory π and a horizontal query segment σ, we can answer these queries in $O(\log^2 n)$ time using only $O(n^2)$ space.

1 Introduction

Comparing the shapes of polygonal trajectories—or time series in general—is an important task that arises in many contexts and is an active line of research in computational geometry and several other research areas [12]. A basic question here is how one can formally compare two given trajectories and to measure how similar they are to each other. To this end, several similarity measures have been developed in the past, and the Fréchet distance [1] is probably the most popular one: it has been used in various applications including speech recognition [11], signature and handwriting recognition [13], geographic applications such as map-matching of vehicle tracking data [5], and moving object analysis [6]. The Fréchet distance is commonly described using the following "leash" metaphor: a man walks on one trajectory and has a dog on a leash on the other trajectory. Both man and dog can vary their speeds, but they may not walk backwards. The Fréchet distance between the two trajectories is the length of the shortest leash with which man and dog can walk from the beginning to the end of the respective trajectories.

There has been a vast amount of work on algorithmic aspects of similarity measures and the Fréchet distance in particular, and a complete review is beyond our possibilities here. (We refer an interested reader to [12] for a comprehensive discussion on this topic.)

*Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands. The authors are supported by the Netherlands Organization for Scientific Research (NWO) under grants 024.002.003, 612.001.118, and 639.023.208, respectively.

Most of the existing works dealing with similarity measures for trajectories study the following algorithmic question: how quickly can one compute or approximate the similarity measure for two given trajectories? However, in several applications it is helpful to store the trajectories into a data structure that allows a user to quickly compute the similarity between trajectories and a query trajectory. The results in this paper contribute to this research direction.

Background. There are several papers that study the problem of designing data structures for querying a trajectory (or a set of trajectories), to find subtrajectories (or the subset of trajectories) that are similar to a given query trajectory with respect to Fréchet distance. Due to space limitations we are able to only highlight a few of the works.

De Berg *et al.* [3] showed how to store a trajectory π into a data structure such that, given a query segment σ and a threshold δ_{max}, one can count all subtrajectories of π whose Fréchet distance to σ is at most δ_{max}. However, their work has several drawbacks: (i) in addition to all the correct subtrajectories their data structure may include additional subtrajectories whose Fréchet distance to σ can be up to a factor $2+3\sqrt{2}$ times larger than δ_{max}, (ii) their data structure is a complicated multi-level structure which is difficult to implement and unlikely to be efficient in practice, and finally (iii) it is unclear how to actually report the subtrajectories in an efficient manner. In a closely related work Gudmundsson and Smid [10] studied a more general version of the problem (where the data structure stores a geometric tree instead of a trajectory and the query is also a trajectory), but their solution makes several assumptions on the input: the tree must be c-packed and the edges of the tree and query must be relatively long compared to δ_{max}. Along the same line of research, De Berg and Mehrabi [4] presented data structures for preprocessing a given trajectory π in the plane representing the movement of a player during a game, such that the following queries can be answered: given two points s and t in the plane, report all subtrajectories of π in which the player has moved in a more or less straight line from s to t. They consider two measures of straightness, namely *dilation* and *direction deviation*, and presented efficient and easy-to-implement data structures with fast construction procedures and provable space

and error guarantees. As far as we are aware, the work by Driemel and Har-Peled [8] is the most related work to our work. They presented a data structure for preprocessing a trajectory π such that given a query trajectory σ with k vertices and two vertices p and q of π, one is able to approximate the Fréchet distance between σ and the subtrajectory of π from p to q, up to a constant factor. In this paper we study the same problem as Driemel and Har-Peled, but our goal is to compute the *exact* Fréchet distance. To be able to still obtain fast query times, we restrict our attention to the special case where σ is a horizontal segment.

Contributions. We study the problem of preprocessing a trajectory π with n edges in the plane into a data structure that is able to quickly answer the following type of queries: given a horizontal line segment σ in the plane and two vertices p, q of π, compute the Fréchet distance between σ and the subtrajectory of π from p to q. Our main result states that such a π can be preprocessed, affording $O(n^2 \log^2 n)$ time, into a data structure such that the desired queries can be answered in $O(\log^2 n)$ time. The data structure needs $O(n^2)$ space if p, q are the endpoints of π, and needs $O(n^2 \log^2 n)$ space otherwise.

2 Preliminaries

For a point p in the plane, we denote its coordinates by $(p.x, p.y)$. For two point sets P and Q, let $d_{\overrightarrow{H}}(P, Q) := \sup_{p\in P} \inf_{q\in Q} \|p-q\|$ be the directional Hausdorff distance from P to Q.

Let $p_0, p_1, \ldots, p_n$ be a sequence of $n+1$ points in the plane. We denote the polyline (or trajectory) defined by this sequence by $\mathcal{P}(p_0, \ldots, p_n)$. We generally view a polyline $\pi := \mathcal{P}(p_0, \ldots, p_n)$ as a piecewise-linear function. Namely, the function $\pi : [0, n] \to \mathbb{R}^2$ such that $\pi(i+t) := (1-t)p_i + tp_{i+1}$ for all $i \in \{0, \ldots, n-1\}$ and all $0 \le t \le 1$. With a slight abuse of notation, we sometimes also use π to denote the image of this function, which is a point set in $\mathbb{R}^2$; namely, the union over i of the segments between p_i and p_{i+1}.

For two polylines π and σ of n and m edges, respectively, their Fréchet distance is defined as

$$d_F(\pi, \sigma) := \inf_{\substack{\alpha:\,[0,1]\to[0,n] \\ \beta:\,[0,1]\to[0,m]}} \sup_{t\in[0,1]} \|\pi(\alpha(t)) - \sigma(\beta(t))\|,$$

where α and β range over continuous nondecreasing surjections. Given such α and β, the point $\pi(\alpha(t))$ is said to be *matched* with $\sigma(\beta(t))$.

Let ℓ_y be the horizontal line $\mathbb{R}\times\{y\}$. For two points p and q in the plane, the point on ℓ_y minimizing the maximum distance to p or q is the point where ℓ_y intersects the perpendicular bisector of the line segment from p to q, if the intersection lies inside the vertical strip $[p.x, q.x]$; see Figure 1. Otherwise, it is either $(p.x, y)$ or $(q.x, y)$. Furthermore, we define the distance

$$B_{(p,q)}(y) = \min_{x\in\mathbb{R}} \max\{\|p - (x, y)\|, \|q - (x, y)\|\}$$

as the minimum distance over all points on ℓ_y to the farthest of p and q.

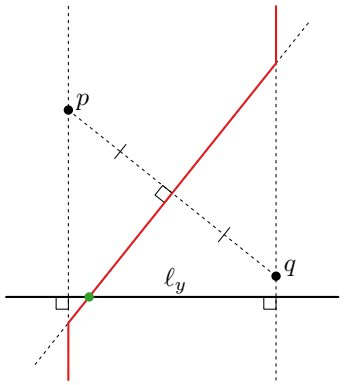


Figure 1: The point (green) on the line $\ell_y := \mathbb{R} \times \{y\}$ minimizing the distance to the farthest of p and q, and its trajectory (red) as y varies.

3 Fréchet distance to a segment

In this section, we show that the Fréchet distance between π and σ can be captured in an alternative expression $F_y(\pi, \sigma)$ in the special case where σ is a single (horizontal) segment. This alternative expression forms the basis for our data structures.

Let $\pi := \mathcal{P}(p_0, \ldots, p_n)$ be a polygonal trajectory of n edges in the plane. Let $x_0 \le x_1$ and $y \in \mathbb{R}$, and define $s_0 := (x_0, y)$ and $s_1 := (x_1, y)$, so that $\sigma := \mathcal{P}(s_0, s_1)$ is a horizontal segment in the plane. We can now define $F_y(\pi, \sigma)$ as follows:

$$\begin{aligned} F_y(\pi, \sigma) := \max\{&\|p_0 - s_0\|, \\ &\|p_n - s_1\|, \\ &d_{\overrightarrow{H}}(\pi, \sigma), \\ &\max_{i\le j,\ p_i.x \ge p_j.x} B_{(p_i,p_j)}(y)\}. \end{aligned} \tag{1}$$

The last term in this equation involves the term $B_{(p_i,p_j)}$ between pairs of vertices of π with $p_j.x \ge p_i.x$ even though p_j appears later than p_i along the trajectory (as $i \le j$). Note that σ is directed to the right, since we assume that $x_0 \le x_1$; so in a sense, these pairs (p_i, p_j) are those that "go backwards" relative to σ.

To show that $d_F(\pi, \sigma) = F_y(\pi, \sigma)$, we first show that $d_F(\pi, \sigma) \ge F_y(\pi, \sigma)$.

Lemma 1 $d_F(\pi,\sigma) \geq \max_{i\leq j,\ p_i.x\geq p_j.x} B_{(p_i,p_j)}(y)$.

Proof. Suppose not, then $d_F(\pi,\sigma) < B_{(p_i,p_j)}(y)$ for some $i \leq j$ with $p_i.x \geq p_j.x$. Let $d = d_F(\pi,\sigma)$, and let $e = B_{(p_i,p_j)}(y)$ for such i and j. Let $x^* = \arg\min_{x\in\mathbb{R}} \max\{\|p_i-(x,y)\|, \|p_j-(x,y)\|\}$. Then $e = \max\{\|p_i-(x^*,y)\|, \|p_j-(x^*,y)\|\} > d$. We have $p_i.x \geq x^* \geq p_j.x$ (otherwise one can decrease e by replacing x^* by $p_i.x$ or $p_j.x$). We have $\|p_i-(x',y)\| > d$ for all $x' \leq x^*$ and $\|p_j-(x',y)\| > d$ for all $x' \geq x^*$. Therefore, for any α and β, with $\|\pi(\alpha(t))-\sigma(\beta(t))\| \leq d$, we must have $\beta(t) > \beta(t')$ when $\alpha(t) = i$ and $\alpha(t') = j$. Hence α and β cannot both be monotone nondecreasing surjections. This contradicts that $d < e$. □

Lemma 2 $d_F(\pi,\sigma) \geq F_y(\pi,\sigma)$.

Proof. For all nondecreasing surjections α and β, the point p_0 is matched with s_0, and p_n is matched with s_1. Therefore $d_F(\pi,\sigma) \geq \|p_0-s_0\|$ and $d_F(\pi,\sigma) \geq \|p_n-s_1\|$. As all points of π are matched with some point of σ, we have $d_F(\pi,\sigma) \geq d_{\overrightarrow{H}}(\pi,\sigma)$. Combining this with Lemma 1, we get that $d_F(\pi,\sigma) \geq F_y(\pi,\sigma)$. □

To show that $d_F(\pi,\sigma) \leq F_y(\pi,\sigma)$ we use free space diagrams, which are a tool commonly used to compute the Fréchet distance. For a given distance threshold ε, the free space diagram $\mathcal{F}_\varepsilon(\pi,\sigma) = \{(a,b) \in [0,n]\times[0,1] \mid \|\pi(a)-\sigma(b)\| \leq \varepsilon\}$ indicates the pairs of points on π and σ that are at most distance ε apart. The product parameter space $[0,n]\times[0,1]$ consists of a row of n cells $[i,i+1]\times[0,1]$, each of which has a convex intersection with $\mathcal{F}_\varepsilon(\pi,\sigma)$. The Fréchet distance between π and σ is at most ε if and only if $(0,0) \in \mathcal{F}_\varepsilon(\pi,\sigma)$, $(n,1) \in \mathcal{F}_\varepsilon(\pi,\sigma)$ and for each $i<j$, there exists $0 \leq b \leq b' \leq 1$ such that (i,b) and (j,b') lie in $\mathcal{F}_\varepsilon(\pi,\sigma)$.

Lemma 3 $d_F(\pi,\sigma) \leq F_y(\pi,\sigma)$.

Proof. Let $d = F_y(\pi,\sigma)$ and let $\mathcal{F} = \mathcal{F}_d(\pi,\sigma)$. It suffices to show that $(0,0) \in \mathcal{F}$, $(n,1) \in \mathcal{F}$ and for each $i<j$, there exists $0 \leq b \leq b' \leq 1$ such that (i,b) and (j,b') lie in $\mathcal{F}$. Indeed, because $d \geq \|p_0-s_0\|$ and $d \geq \|p_n-s_1\|$, both $(0,0)$ and $(n,1)$ lie in $\mathcal{F}$.

We will show that for all $i<j$, there exist $0 \leq b \leq b' \leq 1$ for which (i,b) and (j,b') lie in $\mathcal{F}$. Because $d_{\overrightarrow{H}}(\pi,\sigma) \leq d$, we have for each i that some $(i,b) \in \mathcal{F}$. So let $b \geq 0$ be the minimum value for which $(i,b) \in \mathcal{F}$ and let $b' \leq 1$ be the maximum value for which $(j,b') \in \mathcal{F}$. We show that $b \leq b'$. Suppose for a contradiction that $b' < b$, then $p_j.x \leq \sigma(b').x < \sigma(b).x \leq p_i.x$. But then since $i<j$, we have $d \geq B_{(p_i,p_j)}(y)$. However, by convexity of free space cells, there is no value b'' for which both (i,b'') and (j,b'') lie in $\mathcal{F}$, so $B_{(p_i,p_j)}(y) > d$, which is a contradiction. □

Theorem 4 follows readily from Lemmas 2 and 3.

Theorem 4 *Let $x_0 \leq x_1$ and $y \in \mathbb{R}$. Let $s_0 = (x_0,y)$ and $s_1 = (x_1,y)$ so that $\sigma = \mathcal{P}(s_0,s_1)$ is a horizontal segment in the plane. Let π be an arbitrary polygonal trajectory in the plane. Then $d_F(\pi,\sigma) = F_y(\pi,\sigma)$.*

4 The basic data structure

In this section we describe our data structure for the case where we want to compute the Fréchet distance from a horizontal query segment Q to the entire trajectory π. In the next section we then show how to generalize the solution to the case where a query also specifies two indices q, q', with $0 \leq q \leq q' \leq n$, and we want to compute the Fréchet distance between Q and the subtrajectory $\pi_{q,q'}$ of π. We use $\pi_{q,q'}$, for $0 \leq q \leq q' \leq n$, to denote the subtrajectory of π from vertex p_q to vertex $p_{q'}$.

Our data structure consists of three components each of which is based on one of the terms in Equation 1. For the first two terms of Equation 1, we simply store the endpoints p_0 and p_n of π, so that we can compute their distance to respectively s_0 and s_1 in constant time during the query procedure. To handle the third term of Equation 1, we provide the following lemma.

Lemma 5 *For a polyline $\pi := \mathcal{P}(p_0,\ldots,p_n)$ and a horizontal segment $\sigma = \mathcal{P}(s_0,s_1)$ with $s_0 := (x_0,y)$, $s_1 := (x_1,y)$ and $x_0 \leq x_1$, we have*

$$d_{\overrightarrow{H}}(\pi,\sigma) = \max\{\max_{p_i.x\in(-\infty,x_0]} \|s_0-p_i\|, \max_{p_i.x\in[x_1,+\infty)} \|s_1-p_i\|, \max_i |y-p_i.y|\}. \quad (2)$$

Proof. Recall that the directed Hausdorff distance from π to σ is the distance from the point on π farthest from σ. This distance is attained at a vertex p_i of π, because one of the endpoints of each edge of π is at least as far from σ as all points interior to that edge. If $p_i.x \leq x_0$, then its distance to σ is $\|p_i-s_0\|$. If $p_i.x \geq x_1$, then its distance to σ is $\|p_i-s_1\|$. Otherwise, the point on σ closest to p_i is $(p_i.x,y)$, at distance $|p_i.y-y|$. Since $|p_i.y-y| \leq \|p_i-s_0\|$ and $|p_i.y-y| \leq \|p_i-s_1\|$, the claim follows. □

Lemma 5 suggests we compute $d_{\overrightarrow{H}}(\pi,\sigma)$ using a data structure D with the following components. The proof of Theorem 7 shows how these components are combined to answer a given query.

- We store the vertices of π, ordered by x-coordinate, in a balanced binary tree $T(\pi)$. For each node ν in $T(\pi)$, we store a farthest-point Voronoi diagram $FVD(\nu)$ on the vertices of π in the subtree rooted at ν.

- We let $top(\pi)$ and $bottom(\pi)$, respectively, store the topmost and the bottommost vertices of π.

It remains to deal with the last term in Equation 1. To this end, we first observe that for a fixed pair (p_i, p_j) of vertices of π with $i \leq j$ and $p_i.x \geq p_j.x$, the function $B_{(p_i,p_j)}(y)$ consists of two half-lines with slopes -1 and 1, respectively, and possibly a hyperbolic arc connecting their endpoints; see Figure 2. The hyperbolic arc captures the distances from p_i to the intersection of ℓ_y with the perpendicular bisector of the line segment from p_i to p_j. The two half-lines correspond to the distance to p_i and p_j, respectively. The endpoint of such a half-line lies at the value of y for which the perpendicular bisector intersects the vertical line through p_i, or p_j, respectively. As a consequence, computing the last term in Equation 1 for all possible values of y-coordinates of σ corresponds to computing the upper envelope of quadratically many hyperbolic arcs (and line segments); see Figure 3.

We let $\mathcal{E}(\pi)$ denote the upper envelope and we store it into a list $\mathcal{L}$. The list $\mathcal{L}$ represents $\mathcal{E}(\pi)$ as an ordered list of t pieces, where we next show that $t = O(n^2)$.

Lemma 6 *The complexity of $\mathcal{E}(\pi)$ is $O(n^2)$.*

Proof. We show that any two functions $B_{(p_i,p_j)}$ and $B_{(p_k,p_l)}$ intersect at most twice. Then according to Davenport-Schinzel sequences [9, Chapter 21] the complexity of $\mathcal{E}(\pi)$ will be linear in the number of hyperbolic arcs (and line segments), which is $O(n^2)$.

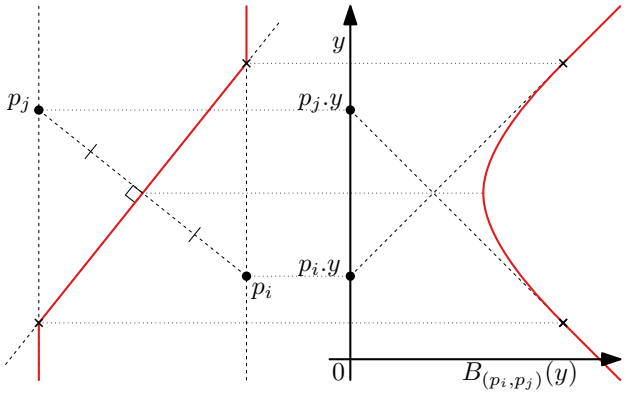


Figure 2: Left: the trajectory of the point on ℓ_y minimizing the distance to the farthest of p_i and p_j as y varies. Right: the distance from this point to the farthest of p_i and p_j as a function of y.

First recall that each function $B_{(p,q)}$ consists of three pieces: two half-lines of slopes of -1,+1, and one hyperbolic arc. The two hyperbolic arcs $arc(p_i, p_j)$ and $arc(p_k, p_l)$ of any two functions $B_{(p_i,p_j)}$ and $B_{(p_k,p_l)}$ intersect at most twice as they are quadratic functions. In

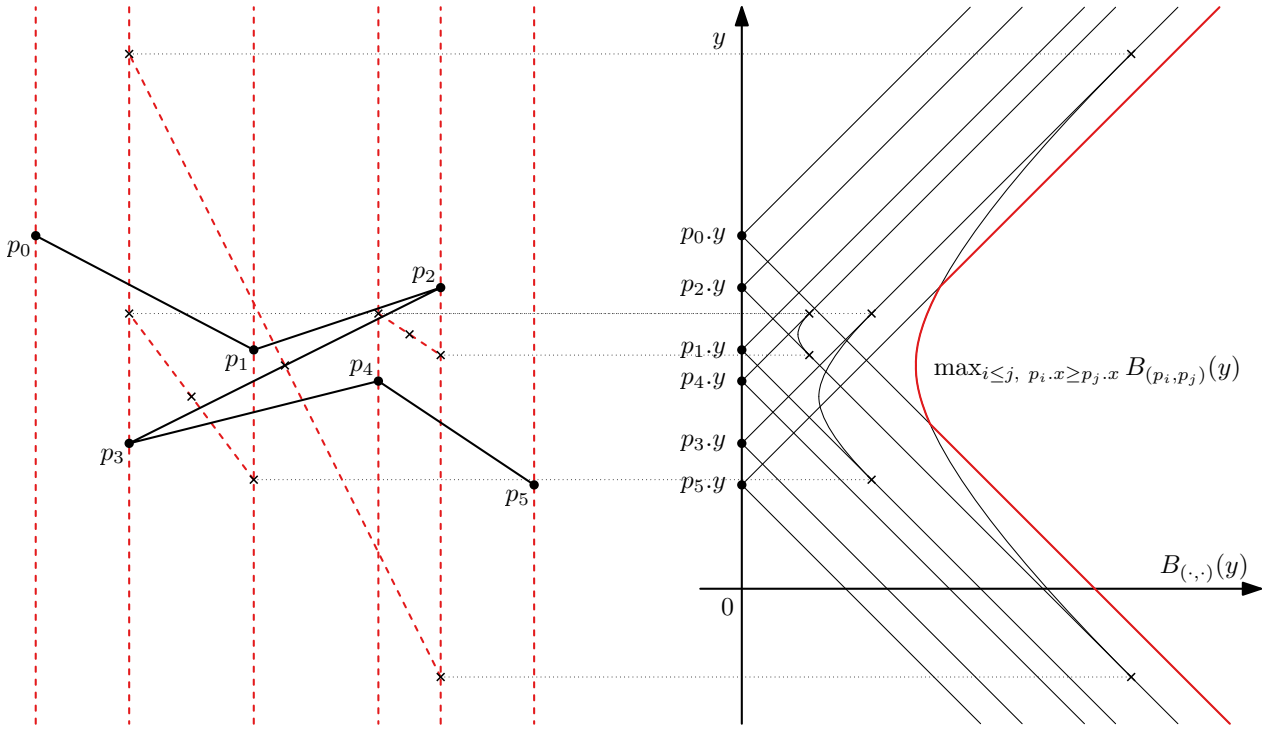


Figure 3: Left: a trajectory and all the perpendicular bisectors for pairs (p_i, p_j) with $i \leq j$ and $p_i.x \geq p_j.x$. Right: the corresponding hyperbolic arcs of the pairs p_i, p_j with $i \leq j$ and $p_i.x \geq p_j.x$. The arcs in red form the upper-envelope.

addition, the half-lines have slopes -1 and +1 and thus one can easily show that any two functions $B_{(p_i,p_j)}$ and $B_{(p_k,p_l)}$ intersect at most twice. □

Putting everything together we obtain the following result.

Theorem 7 *Given a trajectory π with n edges in the plane, one can preprocess π into a data structure of size $O(n^2)$ such that queries that ask for the Fréchet distance between π and a given horizontal query line segment in the plane, can be answered in $O(\log^2 n)$ time. The preprocessing time of the data structure is $O(n^2 \log n)$.*

Proof. The preprocessing time of the data structure comes from the fact that it takes $O(t \log t)$ time to compute the upper envelope of t hyperbolas in the plane [9, Chapter 21]. The data structure consists of two data structures D and $\mathcal{L}$. The data structure D is essentially a balanced binary search tree in which each node ν stores a farthest-point Voronoi diagram on the vertices of π stored in the subtree rooted at ν. Since a farthest-point Voronoi diagram structure needs $O(n)$ space [2], for a point set of size $O(n)$, each level of the binary search tree uses $O(n)$ space and therefore D uses $O(n \log n)$ space in total as the binary search tree has $O(\log n)$ levels. In addition, since the list $\mathcal{L}$ uses $O(n^2)$ space, the overall space requirement of our data structure is $O(n^2)$.

A query with a segment $\sigma = [s_0, s_1]$ is answered as follows. First, compute the the distance between corresponding endpoints of π and σ. Second, compute the Hausdorff distance between π and σ using the different components of D: (i) Query the $FVD(\nu)$'s with point s_0, for canonical nodes ν in $T(\pi)$ whose union covers the range $(-\infty, s_0.x]$. Similarly query the $FVD(\nu)$'s with point s_1, for canonical nodes ν in $T(\pi)$ whose union covers the range $[s_1.x, +\infty)$. Maintain the maximum distance returned from the $O(\log n)$-many such queries. And, (ii) compute the maximum distance between the topmost and the bottommost vertices of π (stored in $top(\pi)$ and $bottom(\pi)$) and σ. Third, compute the intersection of ℓ_y and $\mathcal{E}(\pi)$ using a binary search on the quadratically many pieces of $\mathcal{E}(\pi)$ stored in $\mathcal{L}$. As the answer to the given query, return the maximum of the three values computed in the three steps mentioned above.

The correctness of the query procedure follows from Equation 1 and the query time is dominated by querying $O(\log n)$ farthest-point Voronoi diagrams each of which takes $O(\log n)$ time. □

5 Querying Fréchet distance to subtrajectories

We will now refine our data structure to support queries for the Fréchet distance $d_F(\pi_{q,q'}, \sigma)$ between σ and a subtrajectory $\pi_{q,q'}$ between two vertices $p_q, p_{q'}$ of π. The query will, in addition to the horizontal segment σ, take two indices q, q' with $0 \leq q \leq q' \leq n$.

In order to answer such a query, we build a data structure that, for each subtrajectory $\pi_{q,q'}$, can compute the terms of Equation 1 efficiently. The first two terms of this equation become $\|p_q - s_0\|$ and $\|p_{q'} - s_1\|$, respectively, and can, given a query (σ, q, q'), be answered in constant time.

For the third term $d_{\overrightarrow{H}}(\pi_{q,q'}, \sigma)$, we build a balanced binary tree whose leaves represent the edges of π, ordered as they appear on π. Each node of this tree represents a subtrajectory $\pi_{i,j}$ of π consisting of the edges of π in that subtree. We store the indices i and j of the endpoints of this subtrajectory with the node. In addition, for each node, reuse the data structure we built based on Equation 2 for computing the Hausdorff distance from this subtrajectory to σ.

Given the indices q and q', one can then compute the Hausdorff distance from $\pi_{q,q'}$ to σ as follows. Search for the *maximal* nodes in the tree whose representative subtrajectory is contained in $\pi_{q,q'}$. Here, a node is maximal if there is no ancestor whose representative subtrajectory is contained in $\pi_{q,q'}$. The tree contains $O(\log n)$ such maximal nodes, and one can find them in $O(\log n)$ time given q and q'. Now, for each such node, query the Hausdorff distance from its representative subtrajectory to σ in $O(\log^2 n)$ time, and take the maximum of all the answers. We claim that this is the Hausdorff distance from $\pi_{q,q'}$ to σ. Indeed, each term of the maximum is a lower bound on $d_{\overrightarrow{H}}(\pi_{q,q'}, \sigma)$, and since the union of the subtrajectories represented by these nodes is exactly $\pi_{q,q'}$, the maximum is also an upper bound on $d_{\overrightarrow{H}}(\pi_{q,q'}, \sigma)$. So $d_{\overrightarrow{H}}(\pi_{q,q'}, \sigma)$ can be queried in $O(\log^2 n)$ time, using $O(n \log n)$ space and $O(n \log^2 n)$ preprocessing time.

It remains to build a data structure for the last term of Equation 1. For this we use a 2D range tree on the set $C := \{(i,j) \mid i \leq j \text{ and } p_i.x \geq p_j.x\}$. That is, a balanced tree on the first coordinate of pairs in C, where for each subtree (say it is rooted at v), we store an additional tree $T'(v)$ on the second coordinate for the pairs of C in that subtree. $T'(v)$ is a balanced tree on the second coordinate, whose nodes store the upper envelope of the functions $B_{(p_i,p_j)}$ for the pairs $(i,j) \in C$ in their subtree. The upper envelope stored at the root v of a subtree $T'(v)$ of size k, requires $O(k)$ space, and it can be computed in $O(k)$ time by merging the envelopes of its children.

As $|C| = O(n^2)$, the complete data structure for the last term of Equation 1 uses $O(n^2 \log^2 n)$ space and one can build it in $O(n^2 \log^2 n)$ time.

Given a query (σ, q, q'), we can now use a range query that, in $O(\log^2 n)$ time, reports the upper envelopes of $O(\log^2 n)$ nodes, the upper envelope of whose union

is exactly the upper envelope of the functions $B_{(p_i,p_j)}$ with $q \leq i \leq j \leq q'$. Instead of computing this envelope explicitly, we query each of the envelopes in $O(\log n)$ time at coordinate y, and take the maximum over the results for a total of $O(\log^3 n)$ time. The query time can be improved to $O(\log^2 n)$ time by applying fractional cascading [7].

Therefore, using $O(n^2 \log^2 n)$ preprocessing time and $O(n^2 \log^2 n)$ space, we can compute $d_F(\pi_{q,q'}, \sigma)$ in $O(\log^2 n)$ time for any query (σ, q, q').

6 Discussion

In this paper we showed how to preprocess a trajectory π with n edges in the plane into a data structure that is able to answer the following type of queries in $O(\log^2 n)$ time. Given two vertices p, q of π and a horizontal line segment σ in the plane, report the Fréchet distance between σ and the subtrajectory of π from p to q. The data structure can be constructed in $O(n^2 \log^2 n)$ time, it needs $O(n^2)$ space if p, q are the endpoints of π, and it needs $O(n^2 \log^2 n)$ space otherwise.

We conclude the paper by stating some interesting open questions:

- Can the quadratic upper bound of Lemma 6 for the complexity of the upper envelope be realized? If this upper bound is not tight, we might be able to reduce the space complexity of our data structure in Section 4.
- Can the data structure in Section 5 be extended to handle the case where p and q indicate a subtrajectory of π whose endpoints can lie in the interior of some edge of π?
- Can the data structure in Section 4 (or in Section 5) be extended to handle arbitrarily-oriented query segments in the plane?

References

[1] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5:75–91, 1995.

[2] M. de Berg and O. Cheong and M.v. Kreveld and M. Overmars. Computational Geometry: Algorithms and Applications (3rd edition). Springer-Verlag, 2008.

[3] M. de Berg and A.F. Cook and J. Gudmundsson. Fast Fréchet queries. *Computational Geometry: Theory and Applications*, 46:747–755, 2013.

[4] M. de Berg and A.D. Mehrabi. Straight-path queries in trajectory data. *Journal of Discrete Algorithms*, 36: 27–38, 2016.

[5] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. *In Proc. 31th International Conference on Very Large Data Bases*, 853–864, 2005.

[6] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler and J. Luo. Detecting commuting patterns by clustering subtrajectories. *In Proc. 19th Annual International Symposium on Algorithmis and Computations*, 644–655, 2008.

[7] B. Chazelle and L. J. Guibas. Fractional Cascading: I. A data structuring technique. *Algorithmica*, 1:133–162, 1986.

[8] A. Driemel and S. Har-Peled. Jaywalking your dog: computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42:1830–1866, 2013.

[9] J.E. Goodman and J. O'Rourke. Handbook of Discrete and Computational Geometry. Second Edition, Chapman & Hall/CRC, 2004.

[10] J. Gudmundsson and M.H.M. Smid. Fast algorithms for approximate Fréchet matching queries in geometric trees. *Computational Geometry: Theory and Applications*, 48:479–494, 2015.

[11] S. Kwong and Q.H. He and K.F. Man and K.S. Tang and C.W. Chau. Parallel generic-based hybrid pattern matching algorithm for isolated word recognition. *Journal of Pattern Recognition and Artificial Intelligence*, 12:573–594, 1998.

[12] W. Meulemans. Similarity measures and algorithms for cartographic schematization. PhD Thesis. TU Eindhoven, 2014.

[13] E. Sriraghavendra and K. Karthik and C. Bhattacharayya. Fréchet distance based approach for searching online handwriting documents. *In Proc. 9th International Conference on Document Analysis and Recognition*, 461–465, 2007.

Discretized Approaches to Schematization*

Maarten Löffler† Wouter Meulemans‡

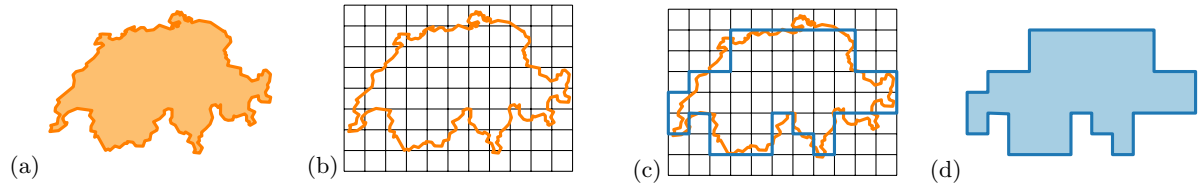


Figure 1: Discretized schematization. (a) Simple polygon P to be schematized (Switzerland). (b) Grid graph G placed on P. (c) *Simple* cycle S in G that best resembles P. (d) S is a rectilinear schematization of P.

Abstract

For both the Fréchet distance and the symmetric difference, we show that finding the simple polygon S restricted to a grid that best resembles a simple polygon P is NP-complete, even if: (1) we require that S and P have equal area; (2) we require turns to occur in a specified sequence for the Fréchet distance; (3) we permit S to have holes for the symmetric difference.

1 Introduction

Cartographic maps are an important tool for exploring, analyzing and communicating data in their geographic context. Effective maps show information as prominently as possible. In *schematic maps*, abstraction is taken to "extreme" levels, representing complex geographic elements with only few line segments. This highlights the primary aspects and avoids an "illusion of accuracy" [15]: the schematic appearance is a visual cue of distortion, imprecision or uncertainty. However, the low complexity must be balanced with recognizability.

Schematic maps tend to be stylized by constraining the permitted geometry. Orientations of line segments are often restricted to a small set $\mathcal{C}$. The typical example is a schematic transit map, in which all segments are horizontal, vertical or a 45-degree diagonal. A central problem in schematization is the following: given a simple polygon P, compute a simple $\mathcal{C}$-oriented polygon S with low complexity and high resemblance to P.

Here we investigate a discretized approach to schematization, characterized by placing a grid graph G over P that models our geometric style and requiring the boundary of S to coincide with a simple cycle in G (Fig. 1). Though it restricts the solution space, this approach readily offers some benefits.

- It can easily model a variety of constraints, even combining different geometry types.
- It promotes the use of collinear edges and provides a uniformity of edge lengths.
- Simplicity enforces a minimal width for narrow strips in P, leading to automated exaggeration—a main cartographic operator [16] for avoiding an undesirable visual collapse (see Fig. 2).
- It makes areas easy to assess [5] or subdivide [14].

Contributions. Focusing on grid graphs (a grid of unit squares), we consider two similarity metrics: the Fréchet distance and the symmetric difference. In Section 3 we prove that the problem is NP-complete under the Fréchet distance, even if we require area preservation and restrict valid solutions to those with a specific sequence of left and right turns. In Section 4 we prove that the problem is NP-complete also under the symmetric difference, even if we require area preservation and we permit the solution to be a polygon with holes. Though the problems are similar in setup, the very different natures of the metrics require different reductions.

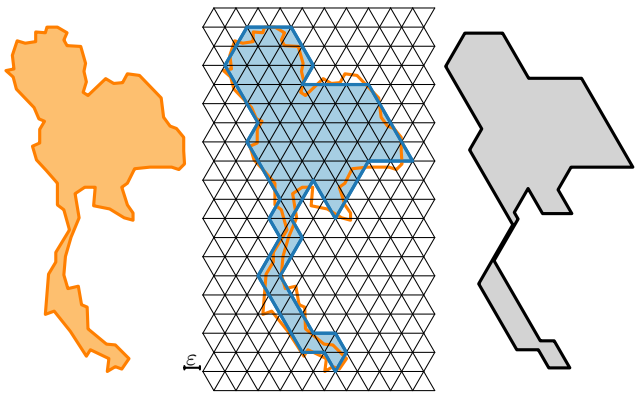


Figure 2: Discretization with simplicity (mid) exaggerates the narrow strip in Thailand (left). Result with a visual collapse, computed using [4] (right).

*An early version of this work appeared on arXiv [13].

†Utrecht University, the Netherlands, m.loffler@uu.nl

‡Eindhoven University of Technology, the Netherlands, w.meulemans@tue.nl

Related work. We highlight the most relevant related work; for a more complete treatment refer to [13]. Recently, schematizing geographic regions has gained increasing attention, e.g. [4, 17]. Our discretized approach is similar in nature to the octilinear schematization technique of Cicerone and Cermignani [6], though simplicity is of no concern in their work. A grid graph always admits a solution with Hausdorff distance at most $3\sqrt{2}/2$ and Fréchet distance at most $(\beta + \sqrt{2})/2$, for β-narrow polygons [2]. Minimizing the Hausdorff distance is NP-complete even on a grid graph [2].

Our problem using the Fréchet distance closely resembles "map matching", with applications in GIS [1, 11] Wylie and Zhu [18] prove independently that our problem is NP-hard under the *discrete* Fréchet distance, however, without requiring a simple input polygon nor a grid graph. A stronger result—with a simple input curve and a grid graph—follows directly from our proofs.

On the dual graph, the problem under the symmetric difference is a specialization of the known NP-hard maximum-weight-connected-subgraph problem [8, 12]. Our results readily imply that this dual problem remains NP-hard even in a constrained geometric setting.

2 Preliminaries

Polygons. A polygon P is defined by a cyclic sequence of vertices in $\mathbb{R}^2$. We use $|P|$ to refer to the area of polygon P and ∂P for its boundary. A polygon is *simple* if no two edges intersect, except at common vertices.

Grid graphs. A grid graph $G = (V, E)$ is a plane graph with all vertices positioned at integer coordinates within a rectangular region, with edges being all unit-length segments connecting pairs of vertices at distance 1.

Cycles. A *cycle* in a graph is a (cyclic) sequence of adjacent vertices; a cycle is *simple* if the sequence does not contain a vertex more than once. A simple cycle in a grid graph corresponds to a simple rectilinear polygon.

Faces. G has two types of faces: *cells* (unit squares), and an *outer face*. A set of faces in G is said to be *connected* if the corresponding induced subgraph of the dual graph G^* is connected; it is *simply connected* if the remaining faces are also connected. A simply connected face set corresponds to a simple rectilinear polygon.

Fréchet distance. Let $B_P \colon S^1 \to \partial P$ continuously map the unit circle onto the boundary of P. Let Ψ denote the set of all orientation-preserving homeomorphisms on S^1. The *Fréchet distance* between two polygons, $d_{\mathrm{F}}(P, Q)$, is defined as $\inf_{\psi \in \Psi} \max_{t \in S^1} \|B_P(t) - B_Q(\psi(t))\|$, where $\|\cdot\|$ denotes the Euclidean distance.

Symmetric difference. The symmetric difference between two polygons P and Q is defined as the area covered by precisely one of the polygons: $d_{\mathrm{SD}}(P, Q) = |(P \cup Q) \setminus (P \cap Q)| = |P \cup Q| - |P \cap Q| = |P| + |Q| - 2 \cdot |P \cap Q|$.

3 Using the Fréchet distance

Theorem 1 *Let G be a grid graph, let P be a simple polygon and let $\varepsilon > 0$. It is NP-complete to decide whether G contains a simple cycle C with $d_F(C, P) \leq \varepsilon$.*

We focus here on sketching a proof that it is indeed NP-hard. We assume $\varepsilon = 3.5$ throughout this proof.

We reduce from *planar monotone 3-SAT* [7]: the problem to decide whether a 3CNF formula F with clauses of either fully positive or fully negative literals is satisfiable, where F is embedded such that all variables lie on a single horizontal line, and clauses are positioned above (positive) or below (negative) this line and are connected to the variables using 1-bend orthogonal leaders (Fig. 3). We construct a simple polygon P and a grid graph G such that G contains a simple cycle C with $d_{\mathrm{F}}(C, P) \leq 3.5$ if and only if F is satisfiable. The construction, and therefore G, has polynomial complexity. Below, we sketch the necessary gadgets which lead to the construction shown in Fig. 4.

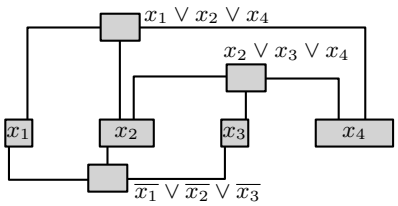


Figure 3: Instance of the constrainted 3CNF formula F.

3.1 Gadgets

We first review the shared general idea of all gadgets and their visual encoding in the corresponding figures. Graph G is shown by a grid of thin light-gray lines. Each gadget contains a part of P called the *local curve* (red), ending in two *gates* (red dots). To reason about gadgets, we also use a corresponding cycle in G which we call the *path boundary* (black) and a *curve area* containing the local curve (gray-filled). The gadgets interact via vertices and edges on shared path boundaries. There is no interaction based on the local curve: it is used only to force choices in using edges of G.

Pressure. If a cycle exists in the complete graph, a *local path* within or on the path boundary must have Fréchet distance at most 3.5 to the local curve. The local path "claims" its vertices: these can no longer be used by another gadget. This results in *pressure* on the other gadget to use a different path, if shared vertices (on the path boundary) are used. To support reasoning about interaction, a gadget has pressure *ports* (green): a sequence of edges on the path boundary that may be shared with another gadget. A port may *receive* pressure, indicating that the shared vertices may not be used in the gadget for its local path. Similarly, it may

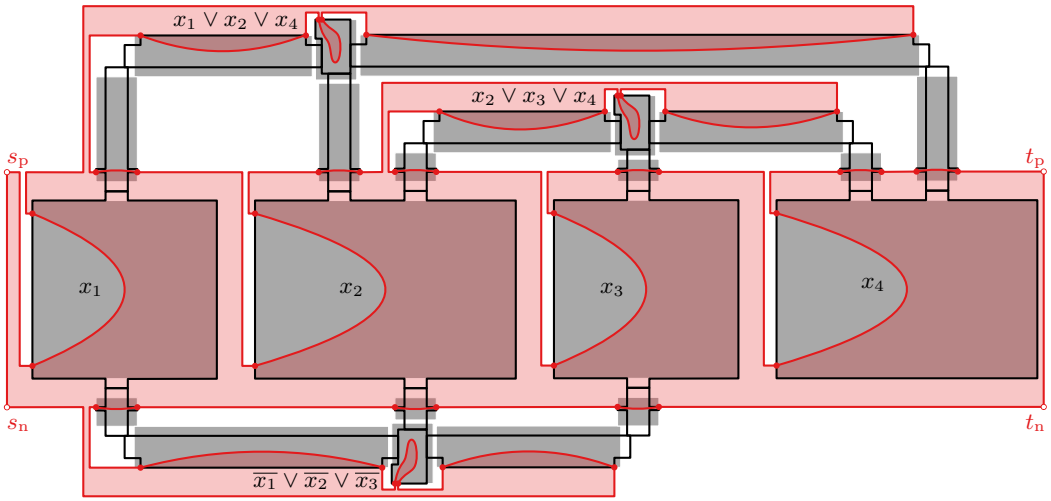


Figure 4: Construction sketch with gadgets for the formula in Fig. 3. Gadgets specify an area in which part of P is located (gray rectangle) and in which part of C must lie (black polygon). They interact via shared boundaries of the black polygons. The red lines connect the various gadgets to obtain a simple polygon, its interior is slightly shaded; the curves within the gadgets are abstracted.

give pressure, indicating that the shared vertices may not be used by an adjacent gadget.

Propagation gadget. A propagation gadget has exactly two ports. The gadget does not admit a path if both ports receive pressure. If one port receives pressure, the other must give pressure: in other words, it propagates pressure. The gadgets can be constructed with any length that is at least 12. Fig. 5 shows one of length (height) 12. The length of the gadget determines where its gates appear: on opposite sides for odd-length and on the same side for even-length gadgets. The complexity of the local curve is linear in its length.

The propagation works due to a local curve that zigzags back and forth, with a distance over $2\varepsilon = 7$ between the endpoints of the zigzags. Hence, the middle of the gadget must be crossed for each zigzag. By placing the right number of zigzags, and having the first and last at exactly distance 3.5 from the ports, we achieve that one of the ports must be overlapped by a local path. The positioning of the first and last further ensure that *all* edges of one port must be used. This is illustrated in Fig. 5: since the dotted variants do not work, the solid ones must be used. Taking an extra grid cell upwards to reach the endpoint leads to pressure on the other side, in which case the entire other port must be covered.

Clause gadget. A clause gadget is illustrated in Fig. 6. It has fixed dimensions. The gadget admits a local path only if one of its ports does *not* receive pressure. Any local path causes pressure on at least one port; for each port there is a path that causes pressure only on that port. The lack of external pressure on a port indicates that the value of the corresponding variable satisfies the clause. There is no local path that avoids all three ports: if all ports receive pressure, none of the variables satisfy the clause and the gadget does not admit a local path.

The construction roughly consists of three zigzags, with one extra spike nested inside the middle one. This extra spike is the crucial element: it can be positioned inside any of the other three zigzags (since ε is 3.5), mimicking that (at least) one of three variables satisfies the clause. The middle zigzag and the spike reach down to approximately the same y-coordinate. In particular, to cover the endpoints of the zigzag or spike, the local

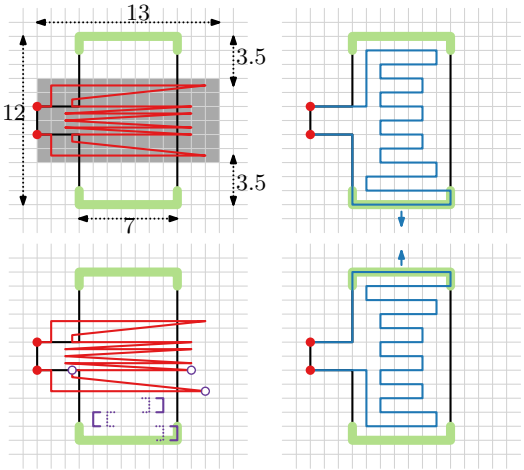


Figure 5: A propagation gadget. Specification and local curve (top left). Local paths with pressure on exactly one port (right). The first three endpoints (open dots) can be reached with the solid subpaths (purple), but not with the dotted variants (bottom left).

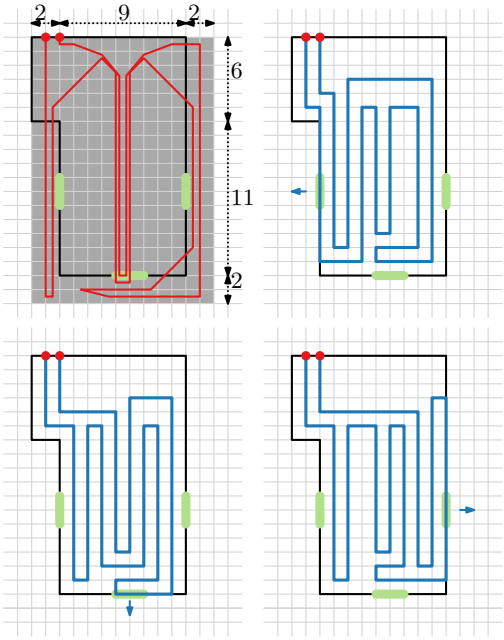


Figure 6: The specification and local curve (top left) of a clause gadget. The other three figures illustrate local paths; each gives pressure on at least one port.

path must reach down to distance 3 above the port. Without the spike inside, the zigzag can be reached by going to exactly this distance. However, if the spike is inside it, the middle zigzag is forced down one more space. The extra cover from the right zigzag propagates this towards the port. This extra cover is necessary due to the convexity of the Euclidean distance: without it, the outer zigzags must reach at least as far down as the middle one, when the spike is inside it; this would cover any potential port intended for the middle zigzag.

The curve area extends slightly outside of the path boundary. Hence, local paths exist with Fréchet distance at most 3.5 that lie outside the path boundary. However, this would claim more beyond the indicated ports, only further restricting any other nearby gadgets.

Variable gadget. Variable gadgets are obtained by making a cycle of propagation gadgets, such that they must all find a local path in the same direction. This direction readily represents the truth value of the variable, which can be transferred to other propagation gadgets.

3.2 Construction with gadgets

We now construct polygon P based on formula F (Fig. 4). First, we place all variable gadgets next to one another, in the order determined by F, with a distance of 11 between consecutive variables. This placement ensures that the ports of variables start on an even and end on an odd x-coordinate.

Using the y-coordinates in the embedding of F, we sort the positive clauses to define a *positive order* $\langle c_1, \ldots, c_k \rangle$. We place the gadget for clause c_j such that the bottom side of its path boundary is at a distance $13 + 24(j-1)$ above the variables. Analogously, we use a *negative order* to place the negative clauses below the variables. Horizontally, the clause gadgets are placed such that the right side of the path boundary lines up with the right side of the appropriate port on the variable gadget of the middle literal. Finally, we place propagation gadgets for each link in F to connect the clause and variable gadgets.

Any overlap in the curve areas would imply that the provided embedding for F—which structures the layout—is not planar. Thus, all gadgets have disjoint curve areas: local curves do not intersect.

Connecting gadgets. We have composed the various gadgets in polynomial time. However, we do not yet have a simple polygon. We must "stitch" the local curves together (in any order) to create polygon P. To this end we first create two subcurves: P_{p} for the variables, positive clause gadgets and their propagation gadgets; and P_{n} for the negative clause gadgets and their propagation gadgets. Fig. 4 visually illustrates the construction of these three subcurves, each with endpoints s_* and t_*, and how to connect them.

Proving the theorem. We now have a simple polygon P; with G implicitly defined as a large enough grid graph. We must argue that the complexity is polynomial and that F is satisfiable if and only if a simple cycle C exists in G with $d_{\mathrm{F}}(C, P) \leq 3.5$. We sketch the main argument of the proof.

A satisfying assignment is derived from C by inspecting the local paths of the variable gadgets: is the pressure clockwise or counterclockwise? Similarly, a satisfying assignment leads to a cycle in G, by concatenating the appropriate local paths given with the gadgets.

Let n denote the number of variables, and m the number of clauses in formula F. Variable gadgets have width linear in their degree and are placed with $O(1)$ distance between them: the entire width is bounded by $O(n+m)$. Variable and clause gadgets have constant height and are placed with $O(1)$ distance between them: the entire height is bounded by $O(m)$. Hence, the polygon's coordinates remain polynomial.

4 Using the symmetric difference

Theorem 2 *Let G be a grid graph, let P be a simple polygon and let $D > 0$. It is NP-complete to decide whether G contains a connected face set S with $d_{SD}(S, P) \leq D$.*

Here we focus on sketching a proof to show that the problem is NP-hard.

We reduce from the *rectilinear Steiner tree problem* [9]: given a set X of n points in $\mathbb{R}^2$, is there a tree T

of total edge length at most L that connects all points in X, using only horizontal and vertical line segments? Vertices of T are not restricted to X. An optimal result must be contained in graph $H(X)$ corresponding to the arrangement of horizontal and vertical lines through each point in X [10]; $H(X)$ is illustrated in Fig. 7(a). We call a vertex of $H(X)$ a *node* if it corresponds to a point in X and a *junction* otherwise. As the problem is scale invariant, we assume $L = 1$ and thus all edges in $H(X)$ must be shorter than 1.

We transform point set X into a grid graph G, a polygon P and a value $D > 0$. We construct G such that each cell corresponds to a vertex (*node-cell* or *junction-cell*), an edge (*edge-cell*), or a bounded face (*face-cell*) of $H(X)$; see Fig. 7(b). Polygon P is constructed to partially overlap all cells, except for the face-cells. To structure P we use a *skeleton* ς, a tree spanning the non-face-cells in the dual of G.

Weights. The symmetric difference between P and a face set $S = \{c_1, c_2, \ldots\}$ may be computed as $|P| + \sum_{c\in S}(|c| - 2 \cdot |P \cap c|)$, since the faces in S are interior-disjoint. As $|c| = 1$, we define the *weight* of a cell c in G as $w(c) = 1-2\cdot|P\cap c|$. Hence, the symmetric difference is $|P| + \sum_{c\in S} w(c)$. We set the desired weight $w(c)$ for cell c to: $-\frac{3}{4}$ if c is a node-cell; 0 if c is a junction-cell; $\|e\|/2$ if c is an edge-cell, where $\|e\|$ is the length of the corresponding edge e in $H(X)$; and 1 if c is a face-cell.

Given $w(c)$ for cell c, the area of overlap $A(c)$ is $|P \cap c| = \frac{1-w(c)}{2}$. Every cell is covered by $A(c)$ area of P; $|P|$ equals $\sum_{c\in G} A(c)$. We call $P \cap c$ the *local polygon* of c. We set $D = |P| - \frac{3}{4}n + \frac{1}{2} = (\sum_{c\in G} A(c)) - \frac{3}{4}n + \frac{1}{2}$.

Designing cells. We design every cell such that the desired weight is achieved. For a face-cell, $w(c) = 1$ and $A(c) = 0$: P does not overlap this cell. For all other cells the local polygon covers a fraction of its interior, as determined by $A(c)$. Skeleton ς dictates how to connect the local polygons; we ensure that at least the middle 25% of the shared edge (the *connector*) is covered. A local polygon never touches the corners of its cell.

Node- and junction-cells may have up to four neighbors in ς. A node-cell has weight $-\frac{3}{4}$; $A(c) = \frac{7}{8}$. A junction-cell has weight 0; $A(c) = \frac{1}{2}$. The local polygon can easily touch the connectors while covering exactly the prescribed area, see Fig. 8(a–b).

An edge-cell has weight $\|e\|/2$ and thus should cope with weights between 0 and 0.5; $A(c)$ lies between $\frac{1}{4}$ and $\frac{1}{2}$. Any edge-cell has degree 1 or 2 in ς; if it has degree 2, the neighboring cells are on opposite sides. The local polygon for $A(c) = \frac{1}{4}$ is a rectangular shape that touches exactly the necessary connectors; we widen this shape to cover the precise area needed if $A(c) > \frac{1}{4}$. This is illustrated in Fig. 8(c).

Proving the theorem. The reduction is polynomial, as P has $O(1)$ complexity in each of the $O(n^2)$ cells. What remains is to prove equivalence of the answers.

Suppose we have a rectilinear Steiner tree T of length at most 1 in $H(X)$. We construct a face set S as the union of all cells corresponding to vertices and edges in T. By definition of T, this must contain all node-cells and cannot contain face-cells. As junction-cells have no weight, the total weight of S is $-\frac{3}{4}n + \sum_{e\in T} w(c_e) = -\frac{3}{4}n + \frac{1}{2}\sum_{e\in T}\|e\|$ where c_e is the cell of G corresponding to edge e. By assumption $\sum_{e\in T}\|e\| \le 1$: the total weight is at most $-\frac{3}{4}n + \frac{1}{2}$. Thus, the symmetric difference for S is at most $|P| - \frac{3}{4}n + \frac{1}{2} = D$.

Suppose we have a connected face set S in G such that $d_{\mathrm{SD}}(S, P) \le D$. The total weight is thus $D - |P| = -\frac{3}{4}n + \frac{1}{2}$. Since face-cells have weight 1 and only node-cells have negative weight, being $-\frac{3}{4}$, this can be achieved only if S contains all node-cells and no face-cells. In particular, the sum of the weights over all edge-cells is at most $\frac{1}{2}$. Thus, the subgraph of $H(X)$ described by the selected cells is connected, contain all nodes of X, and have total length at most 1. If this subgraph is not a tree, we can make it a tree, by leaving out edges (further reducing the total length), until the subgraph is a tree.

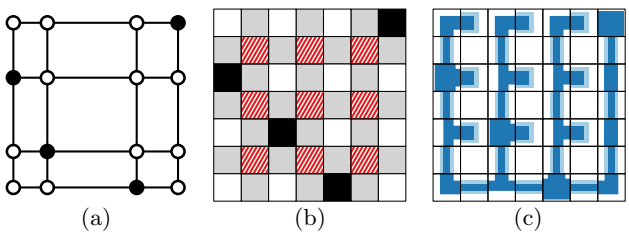


Figure 7: Sketch of the reduction. (a) Graph $H(X)$ where X is given by dark points. (b) Grid graph G: node-cells are black, junction-cells white, edge-cells gray; face-cells are hatched. (c) Constructed polygon P with respect to G.

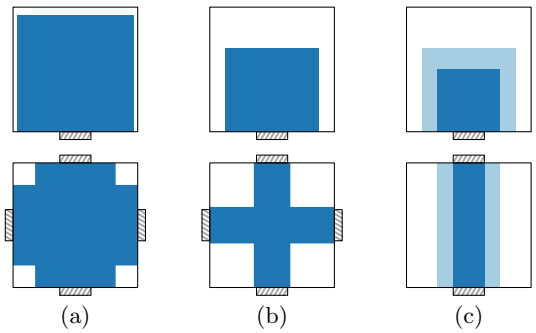


Figure 8: Local polygons, with connectors as hatched rectangles. (a) Node-cell, covered for 87.5%. (b) Junction-cell, covered for 50%. (c) Edge-cell, covered for 25% (dark) up to 50% (dark and light).

5 Conclusions

We studied discretized approaches to the construction of schematic maps, by restricting solutions to grid graph. This has several advantanges, such as promoting alignment and uniformity of edge lengths and avoiding the risk of a visual collapse. We considered two similarity metrics: the Fréchet distance and the symmetric difference. Unfortunately, both turn out to be NP-complete.

Implications. Our reductions imply several further results; refer to [13] for further details. Relevant for e.g. area-preserving schematization [4] and cartograms [5], computing the best *area-equivalent* shape is also NP-complete, under both similarity metrics. The Fréchet -distance variant admits no PTAS and its reduction extends to the *discrete* Fréchet distance, as well as to polygons with a given *bend profile*: the sequence of left and right bends in counterclockwise order along its boundary. The symmetric difference reduction also works for a *simply* connected face set. Finally, the problem remains NP-complete for graphs representing hexagonal and triangular tilings.

Open problems. In contrast to *partial* grid graphs [13], strict monotonicity in the Fréchet distance is crucial for this reduction with full grid graphs; the problem under the *weak* Fréchet distance on full grid graphs remains open. Moreover, it remains open whether the problem (general or grid-based) is fixed-parameter tractable—in e.g. the number of bends in the output polygon—for either similarity metric.

Though our problem under the Fréchet distance is now proven NP-complete, the construction requires that ε is 3.5. For $\varepsilon < 0.5$, the problem becomes trivial to solve: there is only one feasible sequence of vertices in G. For partial grid graphs hardness can be proven for $\varepsilon = 1$ [13]; but what is the complexity with full grid graphs for ε in between 0.5 and 3.5?

Do realistic input assumptions help to obtain efficient algorithms? A first result is known [2], finding a cycle that has Fréchet distance bounded in the "narrowness" of the input polygon. Can we obtain a complementary result, where the algorithm's running time rather than its Fréchet distance depends on the realism parameter?

Results obtained via the Fréchet distance may locally deviate more than necessary. Can we extend *locally correct Fréchet matchings* [3] to our setting?

Acknowledgments. The authors would like to particularly thank: Kevin Buchin, Bart Jansen, Arthur van Goethem, Marc van Kreveld, Aidan Slingsby, and Bettina Speckman for inspiring discussions on the topic of this paper. W. Meulemans was partially supported by Marie Skłodowska-Curie Action MSCA-H2020-IF-2014 656741, the Netherlands Organisation for Scientific Research (NWO) project 639.023.208 and the Netherlands eScience Center (NLeSC) project 027.015.G02.

References

[1] H. Alt, A. Efrat, G. Rote, and C. Wenk. Matching planar maps. *J. Algo.*, 49:262–283, 2003.

[2] Q. W. Bouts, I. Kostitsyna, M. van Kreveld, W. Meulemans, W. Sonke, and K. Verbeek. Mapping polygons to the grid with small Hausdorff and Fréchet distance. In *Proc. 24th ESA*, LIPIcs 57, Art. 22, 2016.

[3] K. Buchin, M. Buchin, W. Meulemans, and B. Speckmann. Locally correct Fréchet matchings. In *Proc. 20th ESA*, LNCS 7501, pages 229–240, 2012.

[4] K. Buchin, W. Meulemans, A. van Renssen, and B. Speckmann. Area-preserving simplification and schematization of polygonal subdivisions. *ACM TSAS*, 2(1):Art. 2, 2016.

[5] R. G. Cano, K. Buchin, T. Castermans, A. Pieterse, W. Sonke, and B. Speckmann. Mosaic drawings and cartograms. *CGF*, 34(3):361–370, 2015.

[6] S. Cicerone and M. Cermignani. Fast and simple approach for polygon schematization. In *Proc. 12th ICCSA*, LNCS 7333, pages 267–279, 2012.

[7] M. de Berg and A. Khosravi. Optimal binary space partitions in the plane. In *Proc. 16th COCOON*, LNCS 6196, pages 216–225, 2010.

[8] M. El-Kebir and G. W. Klau. Solving the maximum-weight connected subgraph problem to optimality. *CoRR*, abs/1409.5308, 2014.

[9] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM J. Appl. Math.*, 32(4):826–834, 1977.

[10] M. Hanan. Steiner's problem with rectilinear distance. *SIAM J. Appl. Math.*, 14(2):255–265, 1966.

[11] J.-H. Haunert and B. Budig. An algorithm for map matching given incomplete road data. In *Proc. 20th ACM SIGSPATIAL GIS*, pages 510–513, 2012.

[12] D. Johnson. The np-completeness column: an ongoing guide. *J. Alg.*, 6(1):145–159, 1985.

[13] W. Meulemans. Discretized approaches to schematization. *CoRR*, abs/1606.06488, 2016.

[14] W. Meulemans, J. Dykes, A. Slingsby, C. Turkay, and J. Wood. Small multiples with gaps. *IEEE TVCG*, 23(1):381–390, 2017.

[15] M. Monmonier. *How to Lie with Maps*. University of Chicago Press, 1996.

[16] N. Regnauld and R. B. McMaster. A synoptic view of generalisation operators. In *Generalisation of Geographic Information: Cartographic Modelling & Applications*, pages 37–66, 2007.

[17] A. van Goethem, W. Meulemans, B. Speckmann, and J. Wood. Exploring curved schematization of territorial outlines. *IEEE TVCG*, 21(8):889–902, 2015.

[18] T. Wylie and B. Zhu. Intermittent map matching with the discrete Fréchet distance. *CoRR*, abs/1409.2456, 2014.

On the minimum edge size for 2-colorability and realizability of hypergraphs by axis-parallel rectangles

Nirman Kumar *

Abstract

Given a hypergraph $\mathsf{H} = (\mathcal{V}, \mathcal{E})$ what is the minimum integer $\lambda(\mathsf{H})$ such that the sub-hypergraph with edges of size at least $\lambda(\mathsf{H})$ is 2-colorable? We consider the computational problem of finding the smallest such integer for a given hypergraph, and show that it is NP-hard to approximate it to within $\log m$ where $m = |\mathcal{E}|$. For most geometric hypergraphs, i.e., those defined on a set of n points by intersecting it with some shapes, it is well known that there is a coloring with 2 colors 'red' and 'blue', such that any hyperedge containing $c \log n$ points, for some constant c, is bi-chromatic, i.e., contains points of both colors. We observe that indeed, for several such hypergraph families, this is the best possible – i.e., there are some n points where there will always be a hyperedge with $\Omega(\log n)$ points that is mono-chromatic. These results follow from results on the indecomposability of coverings. We also show that a frequently used hypergraph, used in the literature on indecomposable coverings *cannot* be realized by axis-parallel rectangles in the plane. This problem was mentioned in a paper of Pach et al. on indecomposable coverings.

1 Introduction

Given a hypergraph $\mathsf{H} = (\mathcal{V}, \mathcal{E})$, suppose every element of $\mathcal{V}$ is assigned one of two colors 'red' and 'blue'. An edge $e \in \mathcal{E}$ is properly colored if not all elements of e receive the same color. The property of being 2-colorable is a well studied problem in combinatorics. It is also known as "Property B", a term coined by E.W. Miller [12] in honor of Felix Bernstein [1]. For many geometric hypergraphs, i.e., hypergraphs defined by a set of n points by intersecting it with some geometric shapes, it is known that the sub-hypergraph of all edges of size at least $t \geq c \log n$ for some constant c has property B. A natural question is to ask what is the smallest possible value of t with this property? Denote this minimum by $\lambda(\mathsf{H})$. In this paper we investigate this question, with a focus on some geometric hypergraphs.

Property B. A simple probabilistic argument of Erdős[4] shows that if each edge size is at least p, and the hypergraph has at most 2^{p-1} edges, it is 2-colorable. Erdős and Hajnal [5] asked: What is the smallest number of edges in a hypergraph each of which have size p, without Property B. They showed that $m(p) < 4^p$ [2]. In geometric hypergraphs on n points in $\mathbb{R}^d$, bounded VC-dimension arguments [10] often mean that the total number of edges is at most $O(n^d)$. Thus, if $p \approx c \log n$, for some constant c, since 2^{p-1} exceeds the number of edges, the sub-hypergraph with edge size at least p has property B.

Cover decomposability. The problem we study is related to the concept of cover decomposability, a topic that has been the subject of much recent research, see the survey [15]. Consider a family of sets $\mathcal{S}$ in the plane (or space). Now, given a finite set of points P, one can define a *primal* hypergraph $\mathsf{H} = (P, \mathcal{R})$ by intersecting the sets in $\mathcal{S}$ with P. Likewise, one can define a dual hypergraph H^* by restricting to a finite sub-family of $\mathcal{S}$, and an edge for each point of the plane (or space) defined by the sets in the sub-family that contain it. In cover decomposability one is interested in the problem : Is $\lambda(\mathsf{H}^*) = O(1)$? A result of Pach [13] implies that if one is looking at the family of translates of an open convex polygon, the problems for the primal and dual hypergraphs are equivalent, i.e., $\lambda(\mathsf{H}) = O(1)$ iff $\lambda(\mathsf{H}^*) = O(1)$. There are many positive and negative results known about cover decomposability, for example, the family of all translates of a given open convex polygon is cover decomposable [18], disks and other convex shapes with a smooth boundary are not cover decomposable [14], but if the convex set is unbounded then it is cover decomposable; translates of concave polygons (most of them) are known to be not cover decomposable [17]. The family of all homothets of a triangle is cover decomposable [6, 7] but homothets of any convex polygon with at least four sides are not cover decomposable [8]. The family of axis-parallel rectangles is also not cover decomposable [16]. The problem of when $\lambda(\mathsf{H})$ (i.e., for the primal hypergraphs) is $O(1)$ has also received recent attention - for example, it has been recently shown to be true for squares [1], while it is known to be false for axis-parallel rectangles [3]. As mentioned

*University of Memphis; nkumar8@memphis.edu; http://sites.google.com/site/nirman/.

[1] Sometimes, Property B is only defined for hypergraphs where each edge has the same size, but we use it for general hypergraphs.

[2] Better bounds are now known.

before, results for the dual setting also imply results for the primal setting for families which are translates of open convex polygons. Techniques for proving indecomposability of coverings are very useful for lower bounding $\lambda(\mathsf{H})$. Typically, this is done by proving that a certain hypergraph which is not 2-colorable can be realized by the dual (or primal) geometric hypergraph.

Our contributions. Our contributions can be summarized as follows:

(I) We show that for a given hypergraph $\mathsf{H} = (\mathcal{V}, \mathcal{E})$ it is NP-hard to approximately compute $\lambda(\mathsf{H})$ upto a factor of $\log m$, where $m = |\mathcal{E}|$.

(II) We show that a certain frequently used hypergraph, which has been used before in proving indecomposability of coverings, cannot be realized by axis-parallel rectangles. This problem was mentioned in a paper of Pach et al. [16].

Paper organization. In Section 2 we provide definitions, set up notation, and provide some simple results on $\lambda(\mathsf{H})$. We prove the computational hardness of approximating $\lambda(\mathsf{H})$ in Section 3. In Section 4 we observe that known constructions from the indecomposability of coverings literature imply lower bounds on $\lambda(\mathsf{H})$ for some primal and dual geometric hypergraphs. In Section 5 we prove our result on the non-realizability of a hypergraph from [16] as the primal hypergraph of axis-parallel rectangles.

2 Preliminaries and Basic Results

A hypergraph $\mathsf{H} = (\mathcal{V}, \mathcal{E})$ is a finite set $\mathcal{V}$ along with a collection $\mathcal{E}$ of subsets of $\mathcal{V}$. Given a hypergraph $\mathsf{H} = (\mathcal{V}, \mathcal{E})$, and an integer t with $0 \leq t \leq |\mathcal{V}|$, define the t-level hypergraph H_t to be the hypergraph $(\mathcal{V}, \mathcal{E}_t)$ where $\mathcal{E}_t = \{e \in \mathcal{E} \mid |e| \geq t\}$. We define the *property B threshold*, $\lambda(\mathsf{H})$, to be the minimum integer t with $2 \leq t \leq |\mathcal{V}|$ such that $(\mathcal{V}, \mathcal{E}_t)$ has property B. For a hypergraph $\mathsf{H} = (\mathcal{V}, \mathcal{E})$ where $|e| = k \geq 2$ for every $e \in \mathcal{E}$ we have $\lambda(\mathsf{H}) = 2$ iff $(\mathcal{V}, \mathcal{E})$ has property B. [3]

A family of hypergraphs. We often work with families of hypergraphs, which are collections of "related" hypergraphs, such that for every integer n we have (possibly several) hypergraphs $(\mathcal{V}, \mathcal{E})$ in the family with $|\mathcal{V}| = n$. In such cases, we will be interested in deriving lower bounds on $\lambda(\mathsf{H})$ as a function of n – such a bound will be worst case, i.e., for almost all values of n, there will be some hypergraph H with n vertices in the family with $\lambda(\mathsf{H})$ lower bounded by the function. Some important families are defined geometrically and are of special interest to us. Let $\mathcal{S}$ be a family of sets in $\mathbb{R}^d$ (for example, all axis-parallel boxes, or balls, or convex polytopes etc.) Two families can be defined as follows:

(A) **Primal geometric family.** A hypergraph in the family is defined by a set P of n points as $\mathsf{H} = (P, \mathcal{E})$ where the edges are defined by intersecting P with members of $\mathcal{S}$.

(B) **Dual geometric family.** A hypergraph H^* in the family is defined by a finite sub-family $\mathcal{S}^*$ of $\mathcal{S}$; $\mathsf{H}^* = (\mathcal{S}^*, \mathcal{E}^*)$ where for each point p of space, an edge $e^* \in \mathcal{E}^*$ is defined by looking at all the sets in $\mathcal{S}^*$ that contain p.

We now introduce some hypergraphs which are not 2-colorable. They have been used previously to prove indecomposability of coverings in [16, 17, 14]. Given a rooted tree $T = (V, E)$, define a hypergraph $\mathsf{H}(T) = (\mathcal{V}(T), \mathcal{E}(T))$, where $\mathcal{V}(T) = V$ and the set of edges $\mathcal{E}(T)$ is the union $\mathcal{E}_L(T) \cup \mathcal{E}_S(T)$. An edge in $\mathcal{E}_L(T)$ is the set of vertices on a root-to-leaf path, and an edge of $\mathcal{E}_S(T)$ is the set of vertices which are the children of some internal node. Observe that the number of edges of edges is precisely $|\mathcal{V}(T)| = |V|$. For a given integer $k \geq 2$, let T_k denote the rooted tree where every internal node has degree k and the height is $k-1$. It is clear that the hypergraph $\mathsf{H}(T_k)$ is k-regular and it has $\frac{k^k - 1}{k-1}$ vertices and edges.

Another commonly used hypergraph is $\mathsf{H}(k, \ell) = (\mathcal{V}(k, \ell), \mathcal{E}(k, \ell))$ for integers $k, \ell \geq 1$ [17, 14]. We do not need its exact definition here, but mention some basic properties relevant to us. The edge set $\mathcal{E}(k, \ell)$ is a disjoint union of 'red' edges and 'blue' edges where each red edge has k elements and each blue edge has ℓ elements. The hypergraph has the property that in any 2 coloring of $\mathcal{V}(k, \ell)$ with colors 'red' and 'blue' either, (1) all vertices in some red edge are colored red, or (2) all vertices in some blue edge are colored blue. The hypergraph $\mathsf{H}(k, \ell)$ has $|\mathcal{V}(k, \ell)| = \binom{k+\ell}{k} - 1$ while $|\mathcal{E}(k, \ell)| = \binom{k+\ell}{k}$. If $\ell = k$, the number of edges is $\binom{2k}{k} \leq 4^k$. We say that we can realize a family of hypergraphs $\mathsf{H}_n = (\mathcal{V}_n, \mathcal{E}_n)$ as the primal (resp. dual) hypergraph of a family of sets $\mathcal{S}$, if for each integer $t \geq 1$ and each hypergraph $\mathsf{H} = (\mathcal{V}, \mathcal{E})$ in the family with $|\mathcal{V}| = t$ there is a set of points P in the plane such that the primal (resp. dual) hypergraph induced by P and $\mathcal{S}$ is isomorphic to H.

2.1 Elementary properties of $\lambda(\mathsf{H})$

The following are elementary and we omit their proof.

Lemma 1 *For a hypergraph $\mathsf{H} = (\mathcal{V}, \mathcal{E})$ we have that $\lambda(\mathsf{H}) \leq 2 \log(2|\mathcal{E}| + 1)$.*

Lemma 2 *For a hypergraph $\mathsf{H} = (\mathcal{V}, \mathcal{E})$, $\lambda(\mathsf{H}) \leq \mathsf{disc}(\mathsf{H}) + 1$, where $\mathsf{disc}(\mathsf{H})$ is the combinatorial discrepancy.*

[3] Notice that we do not require that there is an edge $e \in \mathcal{E}$ with $|e| = \lambda(\mathsf{H})$. For example, the hypergraph with a single edge of large cardinality obviously has property B, and we have $\lambda(\mathsf{H}) = 2$, but there is no edge of size 2.

3 Computational Hardness Result

It is NP-hard to decide if a given k-uniform hypergraph is 2-colorable for $k \geq 3$ [9]. As such given a hypergraph $\mathsf{H} = (\mathcal{V}, \mathcal{E})$ deciding if $\lambda(\mathsf{H}) = 2$ is NP-complete and thus it is NP-hard to compute $\lambda(\mathsf{H})$ exactly. We show here that it is NP-hard to approximate it within a factor $\log m$ where $m = |\mathcal{E}|$.

Theorem 3 *There is a constant $c > 0$ such that the following problem is NP-hard: Given a hypergraph $\mathsf{J} = (\mathcal{U}, \mathcal{F})$ with $|\mathcal{F}| = m$ output a number α with $\lambda(\mathsf{J}) \leq \alpha \leq c \log m \cdot \lambda(\mathsf{J})$.*

Proof. Consider an instance of determining whether a given 4-uniform hypergraph $\mathsf{H} = (\mathcal{V}, \mathcal{E})$ is 2-colorable. Let $\mathcal{E} = \{e_1, e_2, \ldots, e_s\}$ and $|e_i| = 4$ for $1 \leq i \leq s$. Let $k = 2\lceil \log s \rceil$ (an even number). The hypergraph $\mathsf{H}' = \mathsf{H}(k,k) = (\mathcal{V}', \mathcal{E}')$, where $\mathcal{V}'$ is disjoint from $\mathcal{V}$, has at most $4^k \leq s^5$ edges for s large enough. Let $\mathsf{H}'' = (\mathcal{V}'', \mathcal{E}'')$ be defined on distinct elements but isomorphic to H'. Now consider the following three hypergraphs: (1) $\mathsf{H}_1 = (\mathcal{V} \cup \mathcal{V}', \mathcal{E}_1)$ where $\mathcal{E}_1 = \{e_1 \cup e_2 \mid e_1 \in \mathcal{E}, e_2 \in \mathcal{E}'\}$,

(2) $\mathsf{H}_2 = (\mathcal{V} \cup \mathcal{V}'', \mathcal{E}_2)$ where $\mathcal{E}_2 = \{e_1 \cup e_2 \mid e_1 \in \mathcal{E}, e_2 \in \mathcal{E}''\}$, and, (3) $\mathsf{H}_3 = (\mathcal{V}' \cup \mathcal{V}'', \overline{\mathcal{E}})$, where $\overline{\mathcal{E}}$ is defined as follows: For each $e' \in \mathcal{E}'$, choose arbitrarily $k/2$ elements of e' and call this set $\overline{e'}$. Similarly, define $\overline{e''}$ for each $e'' \in \mathcal{E}''$. Now, $\overline{\mathcal{E}} = \{\overline{e'} \cup \overline{e''} \mid e' \in \mathcal{E}', e'' \in \mathcal{E}''\}$.

Lemma 4 *The hypergraph $\mathsf{J} = (\mathcal{V} \cup \mathcal{V}' \cup \mathcal{V}'', \mathcal{E}_1 \cup \mathcal{E}_2 \cup \overline{\mathcal{E}})$ is 2-colorable iff $(\mathcal{V}, \mathcal{E})$ is 2-colorable.*

Proof. If $(\mathcal{V}, \mathcal{E})$ is 2-colorable, then choose the same coloring for elements of $\mathcal{V}$, color all elements of $\mathcal{V}'$ blue and all elements of $\mathcal{V}''$ red. It is easy to see that all the edges of $\mathcal{E}_1 \cup \mathcal{E}_2 \cup \overline{\mathcal{E}}$ are properly colored.

On the other hand, suppose that J can be 2-colored. Since $\mathcal{V}, \mathcal{V}', \mathcal{V}''$ are mutually disjoint one can look upon the coloring as a coloring on the three of those sets separately. We claim that the coloring colors $(\mathcal{V}, \mathcal{E})$ properly. Suppose, this is not true. Then there is an edge $e \in \mathcal{E}$ that is mono-chromatic, say all its elements are blue. Since we know that H' cannot be 2-colored there is some $e' \in \mathcal{E}'$ which is mono-chromatic. The color of e' cannot be blue otherwise the edge $e \cup e' \in \mathcal{E}_1$ would be all blue. So it must be red. Similarly there is a $e'' \in \mathcal{E}''$ that is all red. However then, $\overline{e'} \cup \overline{e''} \in \overline{\mathcal{E}}$ is all red and this contradicts the fact that the coloring is proper for J which includes this edge. It must therefore be true that $\mathcal{E}$ has been properly 2-colored by the induced coloring. □

For the hypergraph J, the total number of edges m is clearly polynomial in s each of cardinality at most $k + 4 = \Theta(\log s)$. Thus the total size of the new hypergraph is polynomial in s. Also, $\log m = O(\log s)$, where m is the number of edges of J. In particular, there is a constant c such that $2c \log m < k$.

Now, by Lemma 4, if $\lambda(\mathsf{H}) = 2$ then $\lambda(\mathsf{J}) = 2$ otherwise $\lambda(\mathsf{J}) \geq k$, since all edge sizes are at least k in J. Suppose we have an approximation algorithm that approximates $\lambda(\mathsf{J})$ to within $c \log m$, i.e., it outputs an α with $\lambda(\mathsf{J}) \leq \alpha \leq c \log m \cdot \lambda(\mathsf{J})$. Then, we can decide if H is 2-colorable as follows: If $\alpha \leq 2c \log m$, then output that H is 2 colorable, otherwise it is not. One can verify easily that the algorithm will correctly decide 2 colorability of H. The reduction is complete. □

In light of the above and Lemma 1, the algorithm which simply outputs $O(\log |\mathcal{E}|)$ as an approximation for $\lambda(\mathsf{H})$, is asymptotically an optimal approximation algorithm, assuming P≠NP.

4 Lower bound for some Geometric Hypergraphs.

Suppose that we can geometrically realize the graph $\mathsf{H}(T_k)$, for all large enough k, for some family of sets $\mathcal{S}$ as its primal (resp. dual) hypergraph. Then, for the family of hypergraphs induced by $\mathcal{S}$, it follows that $\lambda(\mathsf{H}) = \Omega(\frac{\log n}{\log \log n})$ (resp. $\lambda(\mathsf{H}^*) = \Omega(\log n / \log \log n)$), since for all large enough k, there is some hypergraph induced by $\mathcal{S}$ on an $n = \frac{k^k - 1}{k-1}$ point set P (resp. with a sub-family of size n) such that for any two coloring of P (resp. sets in the sub-family) some edges with size at least $k = \Omega(\frac{\log n}{\log \log n})$ are mono-chromatic. Similarly, if for all k, ℓ sufficiently large, we can realize $\mathsf{H}(k, \ell)$ then it will follow (letting $\ell = k$ and reasoning as before) that $\lambda(\mathsf{H}) = \Omega(\log n)$ (or $\lambda(\mathsf{H}^*) = \Omega(\log n)$). Results from the literature on indecomposability of coverings imply the following. In each case a realization of all $\mathsf{H}(T_k)$ or all $\mathsf{H}(k, \ell)$ has been shown. The following theorem summarizes such known constructions exhibiting lower bounds on $\lambda(\mathsf{H})$ (resp. $\lambda(\mathsf{H}^*)$) as a function of n where H (resp. H^*) is a geometric primal (resp. dual) hypergraph induced by a set of points P with $|P| = n$ (resp. induced by a sub-family of size n):

Theorem 5 *For the family of, (i) Translates of (open or closed) concave polygons with no parallel sides: $\lambda(\mathsf{H}), \lambda(\mathsf{H}^*) = \Omega(\log n)$ [17]. (ii) Open unit disks: $\lambda(\mathsf{H}), \lambda(\mathsf{H}^*) = \Omega(\log n)$ [14]. (iii) Homothets of any convex polygon with at least 4 sides, or a concave one with no parallel sides: $\lambda(\mathsf{H}^*) = \Omega(\log n)$ [8]. (iv) Open strips: $\lambda(\mathsf{H}), \lambda(\mathsf{H}^*) = \Omega(\log n / \log \log n)$ [16].*

The result in part (ii) of the above theorem, extends to a larger family, all (open or closed) disks as well.

We conjecture that for hypergraphs defined by the family of all axis-parallel rectangles we have $\lambda(\mathsf{H}) = \Omega(\log n)$. Interestingly, by Lemma 2 this would also imply a $\Omega(\log n)$ lower bound for the combinatorial discrepancy of axis-parallel rectangles, which is Tusnády's

problem in the plane, see [11]. This bound is already known from [2].

5 Non realizability of $\mathsf{H}(T_k)$ by axis-parallel rectangles

Pach et al. [16] mention (see page 6) that it was not known if the graph $\mathsf{H}(T_k)$ could be realized as the primal hypergraph of the family of axis-parallel rectangles. They were interested in the question if $\lambda(\mathsf{H}) = O(1)$ where H is from the family of primal hypergraphs of axis-parallel rectangles. It was shown in [3] that $\lambda(\mathsf{H}) \neq O(1)$. Here, we answer the question left undecided in [16]. We show that $\mathsf{H}(T_k)$ cannot be realized by axis-parallel rectangles in the plane, for sufficiently large k. We conjecture that $\mathsf{H}(k, \ell)$ is also not realizable (for large enough k, ℓ), and a proof similar to one we give below will probably suffice to show this.

In what follows, whenever we say rectangle we mean an axis-parallel rectangle. Let $\mathsf{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph and $\mathcal{V}' \subseteq \mathcal{V}, \mathcal{E}' \subseteq \mathcal{E}$. A sub-hypergraph H' of H defined by $\mathcal{V}', \mathcal{E}'$ is the hypergraph $(\mathcal{V}', \{e \cap \mathcal{V}' | e \in \mathcal{E}'\})$. We show that for all sufficiently large k, $\mathsf{H}(T_k)$ cannot be realized as the hypergraph of points induced by rectangles. The following is a simple observation, that follows by deleting rectangles and points given a realization of H.

Observation 6 *Let $\mathsf{H} = (\mathcal{V}, \mathcal{E})$ be a hypergraph that can be realized by points wrt rectangles, and let H' be a sub-hypergraph of H. Then H' can also be realized by points wrt rectangles.*

Another easy observation is the following.

Observation 7 *$\mathsf{H}(T_k)$ is a sub-hypergraph of $\mathsf{H}(T_m)$ for all $m \geq k$.*

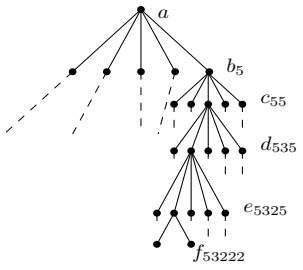


Figure 1: Some vertices and edges of T_0. The height is 5 and each internal node except those with depth 4 have 5 children each. Each node at depth 4 has two leaves as children. The naming of a few vertices is shown.

We omit the easy proof of the following lemma.

Lemma 8 *Let T be an arbitrary rooted tree. Then $\mathsf{H}(T)$ is a sub-hypergraph of $\mathsf{H}(T_k)$ for some k. Moreover, by Observation 7 above, it is a sub-hypergraph for $\mathsf{H}(T_k)$ for all large enough k.*

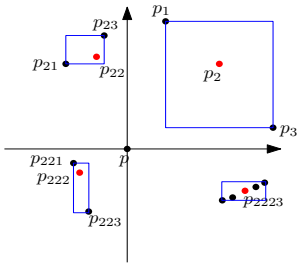


Figure 2: Some of the points shown are forced to be placed as above. The blue rectangles shown are contained in R, R^2, R^{22}, R^{222}.

Now, in order to prove that $\mathsf{H}(T_k)$ cannot be realized by points wrt rectangles, we consider the tree T_0 shown in Figure 1 and we show that there is no way to realize $\mathsf{H}(T_0)$ using rectangles.

For two points p, q in the same quadrant, we say p dominates q if any rectangle containing p and the origin, must contain q. For a set of points in the same quadrant, if none dominates any other, we say they are *on the skyline*.

Theorem 9 *For the tree T_0, $\mathsf{H}(T_0)$ cannot be realized by points wrt rectangles.*

Proof. The root is named a. The vertices at depth 1 are named as $b_1, \ldots, b_5$, those at depth 2 are named c_{ij}, where c_{ij} for fixed i are the children of b_i. Continuing, those at depth 3 are d_{ijk}, at depth 4 are e_{ijkl}. The nodes at depth 6 are f_{ijklm}. Here the indexes i, j, k, l vary from 1 to 5 while m varies in $1, 2$. The proof is by contradiction. Assume that there is a realization of $\mathsf{H}(T_0)$ by points wrt rectangles. Let the point corresponding to the root be p and let p_i be the points corresponding to b_i, p_{ij} for c_{ij} and so on. In the rest of the proof we talk of the points as if they are the vertices of the tree itself with the parent-child or sibling relationships, for brevity. For example, we will say p_{23} is a child of p_2 etc.

Let p be the origin. We may assume that the x and y coordinates of all the points are distinct and the rectangles that define the edges of $\mathsf{H}(T_0)$ contain the relevant points in their interior (as this can be ensured by infinitesimal shifts). The proof will proceed by forcing some points of depth 1 into the 1st quadrant (this is without loss of generality – the crucial point is that they are in the *same* quadrant), then some at depth 2 are forced into the 2nd quadrant etc. Ultimately, we run out of quadrants for the points at depth 5.

We let R denote the rectangle containing the hyperedge $\{p_1, p_2, p_3, p_4, p_5\}$ and in general the children of point, say p_{23}, which define a hyperedge in $\mathcal{E}_S(T_0)$ is realized by rectangle R^{23}. Now, for a hyperedge in $\mathcal{E}_L(T_0)$ defined by say the root-to-leaf path to p_{12445}, let rectangle R_{12445} realize it.

The following lemma follows directly from definitions and the pigeon-hole principle.

Lemma 10 *Let X be a set of points in the plane, $|X| \geq 5$. Suppose that there is a rectangle R_X containing all points in X but not the origin, and for each point $x \in X$, there is a rectangle R_x containing x and the origin, but no other point of X. Then, there are at least 3 points in X that lie in 1 quadrant and are on the skyline.*

Applying Lemma 10 with X as the children of p, i.e., $X = \{p_1, \ldots, p_5\}$, and $R_X = R$ while the rectangles R_x can be taken as any rectangle realizing one of the root-to-leaf hyperedges through one of the p_i we conclude that some three points among $p_1, \ldots, p_5$ lie in one quadrant, which is without loss of generality the first quadrant. Assume these are p_1, p_2, p_3 and that p_2 is the 'middle' point, see Figure 2. Fix indices j, k, l, m. We claim that none of the points $p_{2j}, p_{2jk}, p_{2jkl}, p_{2jklm}$ belong to the 1st quadrant. The following elementary lemma is required.

Lemma 11 *Let points x_1, x_2, x_3 be in the same quadrant on the skyline with x_2 in the middle. Let y be another point such that there exist the following rectangles: (i) R_X containing x_1, x_2, x_3 but not y (ii) R_1 containing the origin and x_1 but not y, (iii) R_3 containing the origin and x_3 but not y, and, (iv) R_y that contains the origin, x_2 and y but not x_1, x_3. Then, y cannot lie in the same quadrant as x_1, x_2, x_3.*

Now, we come to the claim above. To see that p_{2j} cannot belong to the 1st quadrant, we apply Lemma 11 by letting x_1, x_2, x_3 be p_1, p_2, p_3 respectively, y be p_{2j}, and letting $R_X = R, R_1 = R_{11111}, R_3 = R_{31111}, R_y = R_{2jklm}$. (These rectangle choices are not unique; other choices lead to the same conclusion.) Similarly, $p_{2jk}, p_{2jkl}, p_{2jklm}$ belong to different quadrants. By what we showed above, $p_{21}, \ldots, p_{25}$ do not lie in the 1st quadrant. The proof is now essentially successive repetition of the above arguments for the different levels of the tree. For example, applying Lemma 10, we conclude some three of $p_{21}, \ldots, p_{25}$ lie in 1 quadrant - wlog assume this is quadrant 2 and the points are p_{21}, p_{22}, p_{23}, all on the skyline, with p_{22} the middle point. By applying Lemma 11 above we can conclude (wlog) that the points $p_{221}, p_{222}, p_{223}$ must lie in 1 quadrant on the skyline (wlog 3rd quadrant), with p_{222} the middle point. Similarly, applying the combination of Lemma 10 and Lemma 11 to the descendants of p_{222} we conclude as before that all its descendants must lie in quadrant 4. Moreover, all the p_{222l}, for $1 \leq l \leq 5$ must all lie on the skyline and consider a 'middle' point of these - say this is p_{2223}. We have now run out of quadrants for the children at the next level of p_{2223}. More precisely, there is no way to place p_{22231} owing to Lemma 11. This is a contradiction. □

References

[1] E. Ackerman, B. Keszegh, and M. Vizer. Coloring points with respect to squares. In *32nd Int. Sympos. Comp. Geom. (SoCG 2016)*, volume 51, pages 5:1–5:16, 2016.

[2] J. Beck. Balanced two-colorings of finite sets in the square i. *Combinatorica*, 1(4):327–335, 1981.

[3] X. Chen, J. Pach, M. Szegedy, and G. Tardos. Delaunay graphs of point sets in the plane with respect to axis-parallel rectangles. *Random Struc. Algorithms*, 34:11–23, 2009.

[4] P. Erdős. On a combinatorial problem. *Nordisk Mat. Tidskr.*, 11:5–10, 1963.

[5] P. Erdős and A. Hajnal. On a property of families of sets. *Acta Math. Acad. Sci. Hungar.*, 12:87–123, 1961.

[6] B. Keszegh and D. Pálvölgyi. Octants are cover-decomposable. *Disc. and Comp. Geometry*, 47(3):598–609, 2012.

[7] B. Keszegh and D. Pálvölgyi. More on decomposition coverings by octants. *Jour. on Comp. Geom.*, 6(1):300–315, 2015.

[8] I. Kovács. Indecomposable coverings with homothetic polygons. *Disc. and Comp. Geometry*, 53(4):817–824, 2015.

[9] L. Lovász. Coverings and colorings of hypergraphs. In *Proc. 4th Southeast. Conf. on Comb., Graph Theory, and Comp.*, pages 3–12, 1973.

[10] J. Matousek. *Lectures on Discrete Geometry*. Springer-Verlag New York, Inc., 2002.

[11] J. Matousek. *Geometric Discrepancy: An Illustrated Guide*. Springer, 2010.

[12] E. Miller. On a property of families of sets. *Comptes Rendus Varsovie*, 30:31–38, 1937.

[13] J. Pach. Covering the plane with convex polygons. *Disc. and Comp. Geometry*, 1(1):73–81, 1986.

[14] J. Pach and D. Pálvölgyi. Unsplittable coverings in the plane. *Advances in Mathematics*, 302:433–457, 2016.

[15] J. Pach, D. Pálvölgyi, and G. Tóth. Survey on decomposition of multiple coverings. *Geometry–Intuitive, Discrete, and Convex(I. Bárány, K. J. Böröczky, G. F. Tóth, J. Pach eds.), Bolyai Soc. Math. Studies*, 24, 2014.

[16] J. Pach, G. Tardos, and G. Tóth. Indecomposable coverings. *Canad. Math. Bull.*, 52:451–463, 2009.

[17] D. Pálvölgyi. Indecomposable coverings with concave polygons. *Disc. and Comp. Geom.*, 44:577–588, 2010.

[18] D. Pálvölgyi and G. Tóth. Convex polygons are cover-decomposable. *Disc. and Comp. Geometry*, 43(3):483–496, 2010.

Upward order-preserving 8-grid-drawings of binary trees

Therese Biedl*

Abstract

This paper concerns upward order-preserving straight-line drawings of binary trees with the additional constraint that all edges must be routed along edges of the 8-grid (i.e., horizontal, vertical, diagonal) or some subset thereof. We give an algorithm that draws n-node trees with width $O(\log^2 n)$, while the previous best result were drawings of width $O(n^{0.48})$. If only some of the grid-lines are allowed to be used, then the algorithm gives (with minor modifications) the same upper bounds for the $\{\diagup, |, \diagdown\}$-grid and the $\{|, \diagdown, \overline{\ }\}$-grid. On the other hand, in the $\{\diagup, \diagdown, \overline{\ }\}$-grid sometimes $\Omega(\sqrt{n/\log n})$ width is required.

1 Introduction

There are many algorithms to draw trees, especially rooted trees, because different applications impose different requirements on the drawing. In this paper, the drawing should satisfy the following constraints:

- It is *planar*, i.e., no vertices or edges overlap unless the corresponding graph-elements do.
- It is *straight-line*, i.e., every edge is drawn as a straight-line segment that connects the corresponding vertices.
- It is *strictly-upward*, i.e., parents have larger y-coordinates that children. (For some of the results this is relaxed to *upward drawings* where edges may be horizontal.)
- It is *order-preserving*, i.e., a given left-to-right order of children at each node must be respected in the drawing.

Such drawings were called *ideal drawings* previously [3]. All drawings in this paper must be planar and straight-line, and this will not always be mentioned. Further, vertices must always be placed at grid-points, i.e., with integer coordinates. Any drawing is assumed (after possible translation) to reside within the $[1, W] \times [1, H]$-grid where W and H are the *width* and *height*. *Column* i consists of all grid-points with x-coordinate i; *row* j consists of all grid-points with y-coordinate j. Since the height may have to be $\Omega(n)$ in a strictly-upward drawing, the objective of this paper is to find ideal drawings of binary trees that have small width.

There are many results concerning how to draw rooted trees; see for example [5] for an overview, [2] for some recent results, and Table 1 for the results especially relevant to this paper. This paper focuses on *grid-drawings*, which means that the edges must be drawn along the lines of a grid. This is well-studied for so-called *orthogonal drawings*, where the grid is the rectangular grid (also called the *4-grid*) and hence all edges are horizontal or vertical. Creszenci et al. [4] showed that every binary n-node tree has an upward straight-line 4-grid-drawing in an $O(\log n) \times O(n)$-grid (the drawing need not be order-preserving). For complete binary trees as well as for Fibonacci trees, they achieve an $O(\sqrt{n}) \times O(\sqrt{n})$-grid. For order-preserving drawings, significantly more area may be needed: Frati [6] showed that $\Omega(n)$ width and height is necessary for some binary trees in an upward straight-line 4-grid drawing.

The focus of this paper is the *octagonal grid* or *8-grid* that has horizontal, vertical and diagonal lines in both directions. Drawings in the 8-grid could also be called *ASCII*-drawings, since they could easily be done in ASCII using characters `/ | _ \`. Creszenci et al. [4] argue that their upward 4-grid-drawings can easily be converted into strictly-upward 8-grid-drawings via a downward shear. This preserves the same width and gives asymptotically the same height, hence any binary tree has an (unordered) strictly-upward drawing in an $O(\log n) \times O(n)$-grid. For order-preserving drawings, only much weaker bounds are known. Chan [3] studied ideal drawings of binary trees (not necessarily with edges along the grid). As he points out, the first and second of his four algorithms adapt easily to create ASCII-drawings of binary trees. The width of these depends much on the chosen *spine* (a concept that will be used in Theorem 1 as well); with a suitable choice Chan achieves ideal 8-grid-drawings of width $O(n^{0.48})$ and height $O(n)$.

Results of this paper: In this paper, we show how to create ideal 8-grid-drawings of binary trees. The previous best known bounds here are drawings of width

*David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, Ontario N2L 1A2, Canada. biedl@uwaterloo.ca Supported by NSERC. The author would like to thank Timothy Chan and Stephanie Lee for inspiring discussions.

Grid-lines	Upward?	Order-preserving	width upper bound	width lower bound
$\{\|, \overline{\ }\}$	upward	no	$O(\log n)$ [4]	$\Omega(\log n)$ [4]
$\{\|, \overline{\ }\}$	upward	yes	$O(n)$ (folklore)	$\Omega(n)$ [6]
$\{\|, \diagdown\}$	strictly-upward	no	$O(\log n)$ [4]	$\Omega(\log n)$ [4]
$\{\diagup, \diagdown\}$	strictly upward	yes	$O(n)$ (folklore)	$\Omega(n)$ (Thm.2)
$\{\|, \diagdown\}$	strictly upward	yes	$O(n)$ (folklore)	$\Omega(n)$ (Thm.2)
$\{\diagup, \|, \diagdown\}$	strictly upward	yes	$O(n^{0.48})$ [3] $O(\log^2 n)$ (Thm.1)	$\Omega(\log n)$ [4]
$\{\|, \diagdown, \overline{\ }\}$	upward	yes	$O(\log^2 n)$ (Thm.3)	$\Omega(\log n)$ [4]
$\{\diagup, \overline{\ }, \diagdown\}$	upward	no	$O(n)$ (folklore)	$\Omega(\sqrt{n/\log n})$ (Thm.4)

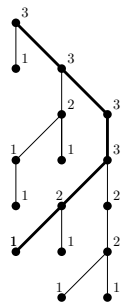
Table 1: Results for planar, upward, straight-line grid-drawings of binary trees. Some more results can be derived in the obvious way, e.g. the upper bound for the $\{\diagup, |, \diagdown\}$-grid also holds for the 8-grid.

$O(n^{0.48})$ [3]. This paper improves this to create drawings that have width $O(\log^2 n)$. In fact, the width is $rpw(T)^2$, where the *rooted pathwidth* $rpw(T)$ is a lower-bound on the width of *any* upward drawing of a tree T (even if it need not be order-preserving or straight-line). Since $rpw(T) \leq \log(n+1)$ [2], our algorithm can be viewed as an $(\log(n+1))$-approximation algorithm for the width of ideal 8-drawings. We also study what happens if one set of the parallel grid-lines is removed; depending on which set is removed we can either achieve the same width-bound or argue that a lower bound of $\Omega(\sqrt{n/\log n})$ holds on the width. See Table 1.

2 Background

Let T be a rooted tree with n nodes that is binary, i.e., every node has at most two children. For any node v, use T_v to denote the subtree rooted at v.

The *rooted pathwidth* $rpw(T)$ of a rooted tree is defined as follows [2]: If every node of T has at most one child, then $rpw(T) = 1$. (In other words, T is a path from the root to the unique leaf.) Otherwise, set $rpw(T) := 1 + \min_P \max_{T' \subset T-P} rpw(T')$. Here the minimum is taken over all paths P for which one end is at the root of T, and the maximum is taken over all subtrees that result when deleting all nodes of P from T. A path P where the minimum is achieved is sometimes called an *rpw-main-path*, though this paper uses the term *spine* to mimic the notations of [7]. The figure above shows a tree with the rooted pathwidth indicated for all nodes; one possible spine is bold.



3 Ideal 8-grid drawings of binary trees

Theorem 1 *Let T be a rooted binary tree. Then T has an ideal 8-grid-drawing of width at most $(rpw(T))^2$.*

Proof. The proof is strongly inspired by the algorithm of Garg and Rusu [7] that give ideal drawings of binary trees of width $O(\log n)$. (Their drawings are not necessarily in the 8-grid.) Their key idea was to use drawings that are "stretchable" in the sense that for any given $\alpha \geq 0$ one can prescribe the contents of the top α rows of the drawing. This then allows to merge drawings of subtrees in a recursive construction. For grid-drawings we need a slightly modified definition as follows:

Definition 1 *Let T be a rooted binary tree with $rpw(T) = r$, and let $\alpha \geq 0$ be given. An 8-grid-drawing of T is called a* left-α-drawing *if within the first α rows, all points in columns $r+1$ and further to the right are unused.*

Put differently, within the top α rows, only the leftmost r columns may be used for placing vertices and edges. Note that (in contrast to [7]) this definition of a left-α-drawing makes no restrictions where the root must be placed (other than that it must be within the leftmost r columns).[1] Define symmetrically a *right-α-drawing* to be one where within the first α rows only the rightmost r columns may be used.

We need two more types of drawings. Define a *left-corner-drawing* and a *right-corner-drawing* to be a drawing of T where the root is at the top-left (top-right) corner. The main claim, to be proved by induction on $rpw(T)$, is the following:

Claim 1 *Fix an arbitrary $\alpha \geq 0$. Then T has a left-α-drawing, a right-α-drawing, a left-corner-drawing and a right-corner-drawing, and all four drawings have width at most $(rpw(T))^2$.*

To prove this claim, consider the base case where $rpw(T) = 1$. This implies that T is a path from the root to a single leaf. Such a path can easily be drawn

[1] Inspection of the construction given below reveals that the root is always in column 1 or r, but we will not make use of this.

with width $1 = 1^2$, and this satisfies the conditions for all four drawings.

Now assume that $r := rpw(T) \geq 2$. From the definition of rooted pathwidth, we know that T has a spine P such that all subtrees T' of $T - P$ satisfy $rpw(T') \leq rpw(T) - 1$. Let the vertices of P be $v_0 v_1 \dots v_m$ where v_0 is the root.[2] For simplicity of description, assume that every spine-node $v_i \neq v_0$ has a sibling; if it does not then simply add a sibling and delete it in the obtained drawing later. Adding a sibling that is a leaf does not affect the rooted pathwidth since $rpw(T) \geq 2$, so this does not affect the width-bound. Thus from now on every spine-node except v_m has a left and a right child.

We explain here only how to create the left-α-drawing and the left-corner-drawing; the other two drawings are obtained in a symmetric fashion. There are three cases, depending on whether v_1 is the left or right child of v_0, and which type of drawing is desired.

Case 1: v_1 is the left child of v_0. In this case, the same construction works for both a left-α-drawing and a left-corner drawing (for the latter, use $\alpha := 1$ below). Place the root v_0 at the top-left corner. Let s_0 be the right child of v_0, and recursively obtain a left-α'-drawing $D(T_{s_0})$ of T_{s_0}, where $\alpha' = \alpha - 1$. Place $D(T_{s_0})$, flush left with column 2 and sufficiently far below such that the $\diagdown$-diagonal from v_0 ends exactly at s_0. Next obtain recursively a left-corner-drawing $D(T_{v_1})$ of T_{v_1}, and place it below $D(T_{s_0})$, flush left with column 1. Connect (v_0, v_1) vertically (both are in column 1). This finishes the construction.

Note that $rpw(T_{s_0}) \leq rpw(T) - 1 = r - 1$ by definition of rooted pathwidth and the spine. Therefore $D(T_{s_0})$ uses only the leftmost $r - 1$ columns within the first α' rows. So this gives a left-α-drawing with the root in the top left corner, as desired. As for the width, $D(T_{s_0})$ has width at most $(r-1)^2$ while $D(T_{v_1})$ has width at most r^2; therefore the width is at most $\max\{1+(r-1)^2, r^2\} = r^2$ as desired.

Case 2: v_1 is the right child of v_0, and we want a left-corner-drawing. Let s_0 be the left child of v_1. Recursively find a left-corner-drawing $D(T_{s_0})$ of T_{s_0}, say that it has height H'. $D(T_{s_0})$ has width at most $(r-1)^2$ since $rpw(T_{s_0}) < rpw(T)$. Recursively find a right-H'-drawing $D(T_{v_1})$ of T_{v_1} of width r^2. (If its width is smaller than r^2, then pad it with empty columns on the left.) Thus within the topmost H' rows of $D(T_{v_1})$, the leftmost $r^2 - r > (r-1)^2$ columns are empty. $D(T_{s_0})$ fits within this empty space; place it flush left with column 1. Finally place v_0 vertically above s_0 (i.e., in column 1) and high enough so that the $\diagdown$-diagonal from v_0 ends exactly at v_1. This gives a left-corner-drawing of width r^2 as desired.

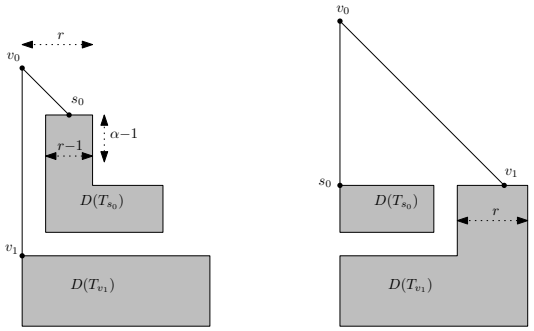


Figure 1: The construction in Case 1 and Case 2.

Case 3: v_1 is the right child of v_0, and we want an α-drawing. This is the most complicated case where a longer section of the spine may get drawn before recursing. Figure 2 illustrates the construction. Recall that every spine-vertex $v_i \neq v_0$ has a sibling by assumption; as in [7] let s_{i-1} be the sibling of v_i. Let $k \geq 1$ be the smallest integer such that v_k is either v_m or s_k is the *left* child of v_k.

First place vertices $v_1, \dots, v_k$ of the spine; vertex v_0 will be added later. Thus, place v_1 in column r. Now repeat for $1 \leq i \leq k-1$: recursively find a left-corner-drawing $D(T_{s_i})$ of T_{s_i}, place it flush left with column $r+1$ and one row below v_i, then place v_{i+1} in column r and in the last row used by $D(T_{s_i})$. This ends with vertex v_k having been placed in column r. Extend a $\diagdown$-diagonal from v_k; this will later be used to complete edge (v_k, v_{k+1}). Next, recursively obtain a left-corner-drawing $D(T_{s_k})$ of T_{s_k}, and place it, flush left with column r and $(r-1)^2$ rows below the row of v_k.

Note that $D(T_{s_k})$ has width at most $(r-1)^2$. Therefore (in the drawing of width r^2 that is being created) there are $r^2 - (r-1)^2 - (r-1) = r$ columns free to the right of $D(T_{s_k})$. These will be used for $T_{v_{k+1}}$ later. Also note that in the topmost row of $D(T_{s_k})$, the $\diagdown$-diagonal from v_k is within the rightmost r rows, and hence it does not interfere with $D(T_{s_k})$.

Let H' be the total number of rows that are used thus far, i.e., from the row of v_1 to the bottommost row of $D(T_{s_k})$. Note that columns $1, \dots, r-1$ are (thus far) entirely free. Recursively find a left-α'-drawing of T_{s_0}, where $\alpha' = H' + \alpha - 1$. Place it, starting $\alpha - 1$ rows above v_1 and flush left with column 1. Within the top α' rows this uses only columns $1, \dots, r-1$ by $rpw(T_{s_0}) < rpw(T)$, and hence this does not intersect the previously placed subtrees.

Place v_0 vertically above v_1 (i.e., in column r) and high enough so that the $\diagup$-diagonal from v_0 ends exactly

[2] The notation here is the same as in [7], though their spine is chosen differently as to always use the heaviest child, rather than the one that has the largest rooted pathwidth.

at s_1.

Let H'' be the number of rows from the row of s_k to the bottommost row of $D(T_{s_0})$. Recursively find a right-H''-drawing $D(T_{v_k})$ of T_{v_k}. Place it, flush right with the rightmost column, and in the row of s_k or below such that the $\diagdown$-diagonal extending from v_k exactly meets the point containing v_{k-1}. Within the topmost H'' rows, drawing $D(T_{s_0})$ uses only the rightmost r columns. Recall that r columns remained free next to $D(T_{s_k})$, and also r columns are free next to $D(T_{s_0})$ since this drawing has width at most $(r-1)^2$. Thus drawing $D(T_{v_k})$ does not interfere with previously placed drawings. This gives the desired left-α-drawing of width r^2.

This ends the construction for all cases and proves Theorem 1. □

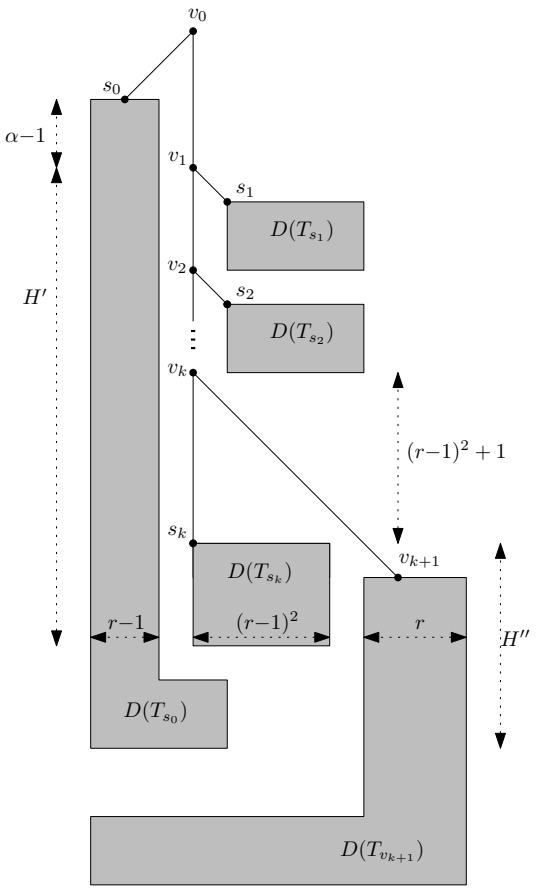


Figure 2: The construction in Case 3.

Height-consideration: In most previous tree-drawing papers, the height is easily shown to be $O(n)$, because all rows (or nearly all rows) intersect at least one vertex. In contrast to this, the construction here has many rows (e.g. most of the rows $1, \ldots, r^2$ in the construction for Case 2) that intersect only edges.

One can easily argue that the height is at most $O(n \cdot (rpw(T))^2)$, because (as one can see) any row without vertex in it intersects a diagonal edge, any such diagonal edge intersects at most $rpw(T)^2$ rows, and these rows can be assigned to the upper endpoint of the diagonal edge.

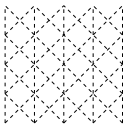
If one follows the construction exactly as described, then $\Omega(nr^2)$ height (for $r = rpw(T)$) may result. (For example, consider a tree where the spine has length $\Omega(n)$ and nearly all siblings of spine-vertices are leaves, but the last few siblings have big enough subtrees to force rooted pathwidth r.) However, there are some obvious possible improvements to the height. To give just one, in Case 2 the drawing $D(T_{s_0})$ could be moved much higher, directly under the $\diagdown$-diagonal, because due to the strict-upwardness of the drawing, the ith row of it is empty in column $i+1$ and farther right. This alone is not enough to ensure a smaller height, but we suspect that combining this with drawing the spine more carefully when some siblings have very small size may lead to a drawing of width $O(\log^2 n)$ and height $O(n)$. This remains for future work.

4 Ideal 6-grid drawings of binary trees

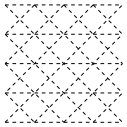
Now we turn to the 6-grid, which has grid-lines with angles of 60° between them. Frequently it is easier to think of it instead as a grid that has three of the four sets of grid-lines of the 8-grid (e.g., horizontal, rightward, and $\diagdown$-diagonals). Bachmeier et al. [1] studied 6-grid drawings of trees. Their drawing were not (necessarily) upward, and as such, it was irrelevant which of the grid-lines of the 8-grid are used for the 6-grid, since they are all the same after 90° rotation and/or a shear, and a shear does not affect the asymptotic area.

In contrast to this, we study here *upward* drawings of binary trees in the 6-grid, and as before, focus on keeping the width small. As will be seen, here it makes a difference exactly which grid-lines are used to represent the 6-grid.

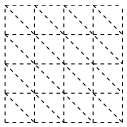
The following grids will be studied:



- The $\{\diagup, |, \diagdown\}$-grid: grid-lines are vertical or along a 45° diagonal in either direction.



- The $\{\diagup, \diagdown, \text{—}\}$-grid: grid-lines are horizontal or along a 45° diagonal in either direction.



- The $\{|, \diagdown, \text{—}\}$-grid: grid-lines are horizontal or vertical or along one of the 45° diagonals. (For the $\{\diagup, |, \text{—}\}$-grid a symmetric set of results is obtained by using a horizontal flip.)

We have thus far mostly studied ideal drawings, which must be strictly-upward and hence horizontal lines are disallowed. In particular, Theorem 1 created strictly-upward drawings in the 8-grid, which hence are automatically drawings in the $\{\diagup, |, \diagdown\}$-grid. Therefore every binary tree T has an ideal drawing that is an embedding in the $\{\diagup, |, \diagdown\}$-grid and has width at most $(rpw(T))^2$.

Now we turn to other types of 6-grids. Again, having an ideal drawing means being strictly-upward, so no horizontal lines can be used. We show that then no small width is possible.

Theorem 2 *There exists a binary tree T such that any ideal drawing of T in the $\{|, \diagdown\}$-grid or the $\{\diagup, \diagdown\}$-grid requires width and height $\Omega(n)$.*

Proof. Let T consist of a path of length $n/2$ and attach at each node a left child that is a leaf. For an order-preserving and strictly-upward drawing, the path must be drawn following the $\diagdown$-diagonals. This gives a width and height of at least $n/2 - 1$. □

Therefore, the remaining drawing-results will be in a relaxed model of ideal drawings where horizontal edges are allowed, hence the drawing is upward rather than strictly-upward. Call these *weakly-ideal* drawings. (As before all drawings must be planar, straight-line and order-preserving.)

Theorem 3 *Every binary tree T has a weakly-ideal drawing that is an embedding in the $\{|, \diagdown, \overline{\ }\}$-grid and has width at most $(rpw(T))^2$.*

Proof. The proof is very similar to the proof of Theorem 1. As before, define (left/right) corner-drawings and (left/right) α-drawings. Additionally now demand for all these drawings that in the topmost row no point to the right of the root is occupied (we say that the root is *right-free*).

Create left-corner-drawings and left-α-drawings almost exactly as before. The only difference is that at the places where a $\diagup$-diagonal was used before, we now use a horizontal edges instead; this is feasible because the root of corresponding subtree is right-free. For right-corner and right-α-drawings, the constructions are not entirely symmetric anymore, but again, by using horizontal edges rather than diagonal ones, drawings can be constructed. Figure 3 illustrates the constructions in all cases; the details are left to the reader. □

Finally consider the $\{\diagup, \diagdown, \overline{\ }\}$-grid, which is the same as the 8-grid where no vertical edges are allowed. Theorem 2 showed that ideal drawings have to have large width. We show here that even weakly-ideal drawings may require large with. In fact, the following bound holds for any planar straight-line drawing in the

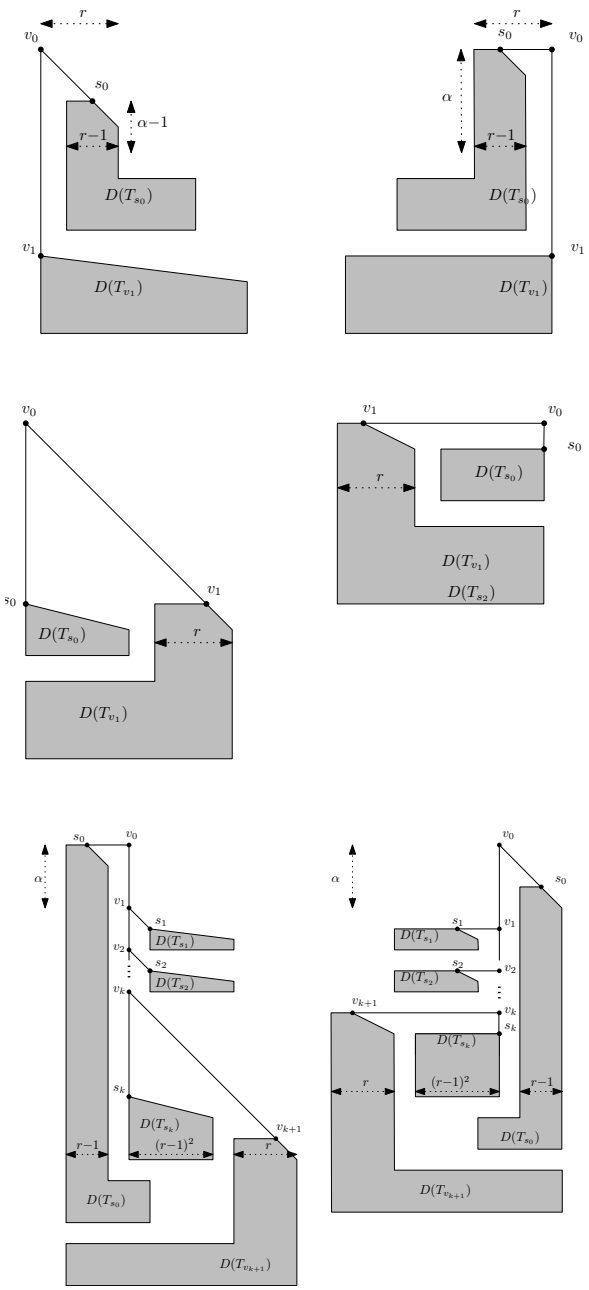


Figure 3: The construction for the $\{|, \diagdown, -\}$-grid.

$\{\diagup, \diagdown, \overline{\ }\}$-grid, even if it is not upward or not order-preserving.

Theorem 4 *The complete binary tree must have width $O(\sqrt{n/\log n})$ in any straight-line drawing in the $\{\diagup, \diagdown, \overline{\ }\}$-grid.*

Proof. The proof is very similar to the "simplest" method for obtaining a width-lower-bound for weakly-ideal drawings of the complete ternary tree, see [8]. We

repeat the argument here for completeness. Fix an arbitrary straight-line drawing of the complete binary tree in the $\{\diagup, \diagdown, \overline{\ }\}$-grid, say it has w columns. So any edge spans a horizontal distance of at most $w-1$. Since only horizontal and diagonal edges are allowed, therefore any edge spans a vertical distance of at most $w-1$.

Observe that T has height $h := \log\left(\frac{n+1}{2}\right)$, i.e., the path from the root to each leaf contains h edges. In consequence, any node has vertical distance at most $h(w-1)$ from the root. Therefore the entire drawing is contained within a rectangle that has w columns and up to $h(w-1)$ rows above and below the root, hence $2h(w-1)+1$ rows in total. Therefore the drawing resides in a grid with at most $2wh(w-1)+w$ grid points. Since all n nodes are placed on these grid-points, necessarily

$$n \leq 2wh(w-1) + w \in O(w^2 \log n),$$

which implies $w \in \Omega(\sqrt{n/\log n})$. □

We strongly suspect that a lower bound of $\Omega(\sqrt{n})$ on the width holds, but this remains for future work. For the complete binary tree, it is easy to find a construction that has width and height $O(\sqrt{n})$, and in fact, no horizontal edges are used.

Theorem 5 (based on [4]) *The complete binary tree has an ideal drawing in the $\{\diagup, \diagdown\}$-grid of grid-size $O(\sqrt{n}) \times O(\sqrt{n})$.*

Proof. Crescenzi et al. gave a simple recursive construction that draws the complete binary tree in a 4-grid of size $O(\sqrt{n}) \times O(\sqrt{n})$ [4]. Moreover, all edges go rightward or downward. Scale this drawing by $\sqrt{2}$ and then rotate it by 45° clockwise. Due to the scaling, this maps all vertices to grid-points, and all edges are now diagonal and downward as desired. □

5 Remarks

This paper developed algorithms for weakly-ideal 8-grid-drawings of binary trees, i.e., planar upward straight-line order-preserving drawings with edges drawn along grid-lines for the 8-grid (or some subset therefore). We gave constructions of width $O(\log^2 n)$ for a number of such grids. The height is rather large ($O(n \log^2 n)$), and improving this remains an open problem. We also showed that width $O(\sqrt{n/\log n})$ is required for the grid where no vertical lines are allowed.

A natural question is whether similar bounds could be proved for ternary trees. For unordered drawings, Bachmeier et al. [1] gave simple recursive constructions that achieve width $O(n^{\log_3 2}) \approx O(n^{0.631})$. In work done simultaneously with the current paper, Lee studied *ordered* drawings of ternary trees and proved that every ternary tree has such a weakly-ideal 8-grid drawing of width $\Omega(n^{0.68})$ [8]. Furthermore, the complete ternary tree requires width $\Omega(n^{0.411})$ in any upward octagonal-grid-drawing [8]. Both the constructions and the lower bounds in Lee's thesis are significantly more complicated than the ones given here, and will be published separately.

As for open problems, the obvious one is to close the "gap" between the width $O(\log^2 n)$ achieved with our algorithm and the lower bound of $\Omega(\log n)$ for the complete binary tree. Are there binary trees that require $\omega(\log n)$ width in ideal 8-grid drawings?

The other remaining gap concerns drawings in the $\{\diagup, \diagdown, \overline{\ }\}$-grid-grid. Can we achieve a width of $O(\sqrt{n})$ not just for complete binary trees but for all trees?

References

[1] Christian Bachmaier, Franz-Josef Brandenburg, Wolfgang Brunner, Andreas Hofmeier, Marco Matzeder, and Thomas Unfried. Tree drawings on the hexagonal grid. In Ioannis G. Tollis and Maurizio Patrignani, editors, *Graph Drawing (GD 2008)*, volume 5417 of *Lecture Notes in Computer Science*, pages 372–383. Springer, 2009.

[2] Therese Biedl. Ideal tree-drawings of approximately optimal width (and small height). *Journal of Graph Algorithms and Applications*, 21(4):631–648, 2017.

[3] Timothy M. Chan. A near-linear area bound for drawing binary trees. *Algorithmica*, 34(1):1–13, 2002.

[4] Pierluigi Crescenzi, Giuseppe Di Battista, and Adolfo Piperno. A note on optimal area algorithms for upward drawings of binary trees. *Comput. Geom.*, 2:187–200, 1992.

[5] Giuseppe Di Battista and Fabrizio Frati. A survey on small-area planar graph drawing, 2014. CoRR report 1410.1006.

[6] Fabrizio Frati. Straight-line orthogonal drawings of binary and ternary trees. In Seok-hee Hong, Takao Nishizeki and Wu Quan, editors, *Graph Drawing (GD 2007)*, volume 4875 of *Lecture Notes in Computer Science*, pages 76–87, Springer, 2007.

[7] Ashim Garg and Adrian Rusu. Area-efficient order-preserving planar straight-line drawings of ordered trees. *Int. J. Comput. Geometry Appl.*, 13(6):487–505, 2003.

[8] Stephanie Lee. Upward octagonal drawings of ternary trees. Master's thesis, University of Waterloo, August 2016. (Supervisors: T. Biedl and T. Chan.) Available at `https://uwspace.uwaterloo.ca/handle/10012/10832`.

Index