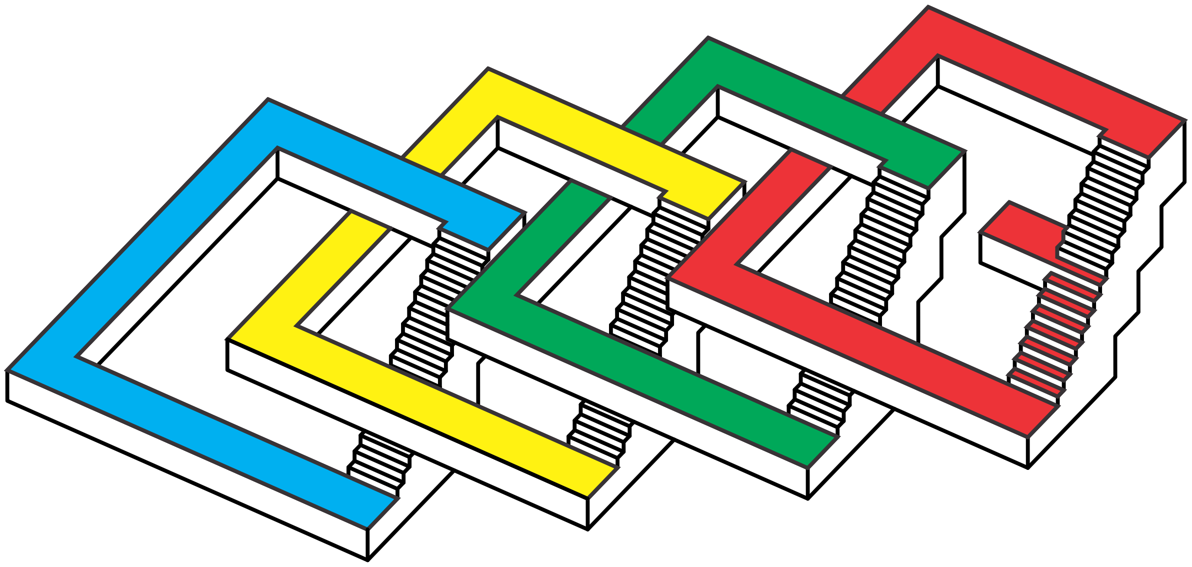
24th Canadian Conference on Computational Geometry

August 8-10, 2012
Charlottetown, Canada



Preface

This volume contains the official proceedings of the 24th Canadian Conference on Computational Geometry (CCCG'12), held in Charlottetown on August 8-10, 2012. These papers are also available electronically at `http://www.cccg.ca` and at `http://2012.cccg.ca`.

We thank the staff at Holland College, and in particular Tina Lesyk, Marsha Doiron and Tracey Campbell, for preparing the conference site.

We are grateful to the Program Committee for agreeing to a rigorous review process. They, and other reviewers, thoroughly examined all submissions and provided excellent feedback. Out of 75 papers submitted, 49 are contained in these proceedings. We thank the authors of all submitted papers, all those who have registered, and in particular Günter Ziegler, Pankaj Agarwal and Joseph Mitchell for presenting plenary lectures.

We also thank Sébastien Collette, who prepared these proceedings, as well as Perouz Taslakian and Narbeh Bedrossian who designed the conference logo.

Last but not least, we are grateful for sponsorship from *AARMS*, the *Mprime Network*, *PIMS* and the *Fields Institute*. Their financial support has helped us to cover many costs as well as provide significant funding to over 50 students and postdocs, including waivers of their registration fees.

Greg Aloupis and David Bremner
(Conference Organizers)

FIELDS mprime AARMS pims

Copyrights of the articles in these proceedings are maintained by their respective authors.

More information about this conference and about previous and future editions is available online at

`http://cccg.ca`

Invited Speakers

- Pankaj Agarwal (Duke U.)
- Joseph Mitchell (Stony Brook U.)
- Günter Ziegler (Freie Universität Berlin) – Erdős memorial lecture

Organizing Committee

- Greg Aloupis (U. Libre de Bruxelles)
- David Bremner (U. New Brunswick)

Program Committee

- Oswin Aichholzer (T.U. Graz)
- Greg Aloupis (U. Libre de Bruxelles)
- Therese Biedl (U. Waterloo)
- David Bremner (U. New Brunswick)
- Mark de Berg (TU Eindhoven)
- Jeff Erickson (U. Illinois at Urbana-Champaign)
- Ferran Hurtado (U. Politecnica de Catalunya)
- John Iacono (Polytechnic Institute of New York U.)
- Mark Keil (U. Saskatchewan)
- David Kirkpatrick (U. British Columbia)
- Stefan Langerman (U. Libre de Bruxelles)
- Alex López-Ortiz (U. Waterloo)
- Anil Maheshwari (Carleton U.)
- Michael McAllister (Dalhousie U.)
- Pat Morin (Carleton U.)
- Bradford Nickerson (U. New Brunswick)
- Diane Souvaine (Tufts U.)
- Csaba Tóth (U. Calgary)
- Godfried Toussaint (New York U. Abu Dhabi)
- Ryuhei Uehara (JAIST)
- Steve Wismath (U. Lethbridge)
- Hamid Zarrabi-Zadeh (Sharif U. of Technology)
- Norbert Zeh (Dalhousie U.)

Other Reviewers

- Mohammad Ali Abam
- Victor Alvarez
- Jasine Babu
- Aritra Banik
- Luis Felipe Barba Flores
- Gregory Bint
- Nicolas Broutin
- Javier Cano
- Paz Carmi
- Mirela Damian
- Pooya Davoodi
- Stephane Durocher
- Ehsan Emam-Jomeh-Zadeh
- Ruy Fabila
- Eli Fox-Epstein
- Robert Fraser
- Alfredo Garcia
- Amin Gheibi
- Omid Gheibi
- Alexander Gilbers
- Sathish Govindarajan
- Thomas Hackl
- Masud Hasan
- Meng He
- John Howat
- Minghui Jiang
- James King
- Rolf Klein
- Matias Korman
- Sonal Kumari
- Stuart MacGillivray
- Jonas Martinez
- Paul Nigsch
- Mostafa Nouri Baygi
- Belen Palop
- Alexander Pilz
- Bodhayan Roy
- Sasanka Roy
- Alejandro Salinger
- Stefan Schirra
- Raimund Seidel
- Rodrigo Silveira
- Matthew Skala
- Michiel Smid
- Bettina Speckmann
- Svetlana Stolpner
- Perouz Taslakian
- Constantinos Tsirogiannis
- Birgit Vogtenhuber
- Gernot Walzl
- Andrew Winslow
- David R. Wood
- Stefanie Wuhrer
- Sadra Yazdanbod

Contents

Algorithms for Geometric Similarity

Pankaj K. Agarwal[*]

A basic problem in classifying, or searching for similar objects in, a large set of geometric objects is computing similarity between two objects. This has led to extensive work on computing geometric similarity between two objects. This talk discusses some old and some new geometric-similarity algorithms, with an emphasis on transportation distance (also called earth mover's distance) and Frechet distance. The talk will also touch upon a few open problems in this area.

[*]Duke University.

Visibility-Monotonic Polygon Deflation*

Prosenjit Bose Vida Dujmović Nima Hoda Pat Morin

Abstract

A *deflated* polygon is a polygon with no visibility crossings. We answer a question posed by Devadoss et al. (2012) by presenting a polygon that cannot be deformed via continuous visibility-decreasing motion into a deflated polygon. In order to demonstrate non-deflatability, we use a new combinatorial structure for polygons, the directed dual, which encodes the visibility properties of deflated polygons. We also show that any two deflated polygons with the same directed dual can be deformed, one into the other, through a visibility-preserving deformation.

1 Introduction

Much work has been done on visibilities of polygons [6, 8] as well as on their convexification, including work on convexification through continuous motions [4]. Devadoss et al. [5] combine these two areas in asking the following two questions: (1) Can every polygon be convexified through a deformation in which visibilities monotonically increase? (2) Can every polygon be deflated (i.e. lose all its visibility crossings) through a deformation in which visibilities monotonically decrease?

The first of these questions was answered in the affirmative at CCCG 2011 by Aichholzer et al. [2]. In this paper we resolve the second question in the negative. We also introduce a combinatorial structure, the directed dual, which captures the visibility properties of deflated polygons and we show that a deflated polygon may be monotonically deformed into any deflated polygon with the same directed dual.

2 Preliminaries

We begin by presenting some definitions. Here and throughout the paper, unless qualified otherwise, we take *polygon* to mean simple polygon on the plane.

A *triangulation*, T, of a polygon, P, with vertex set V is a partition of P into triangles with vertices in V. The *edges* of T are the edges of these triangles and we call such an edge a *polygon edge* if it belongs to the polygon or, else, a *diagonal*. A triangle of T with exactly one diagonal edge is an *ear* and the *helix* of an ear is its vertex not incident to any other triangle of T.

Let w and uv be a vertex and edge, respectively, of a polygon, P, such that u and v are seen in that order in a counter-clockwise walk along the boundary of P. Then uv is *facing* w if (u, v, w) is a left turn. Two vertices or a vertex and an edge of a polygon are *visible* or *see* each other if there exists a closed line segment contained inside the closed polygon joining them. If such a segment exists that intersects some other line segment then they are visible *through* the latter segment. We say that a polygon is in *general position* if the open line segment joining any of its visible pairs of vertices is contained in the open polygon.

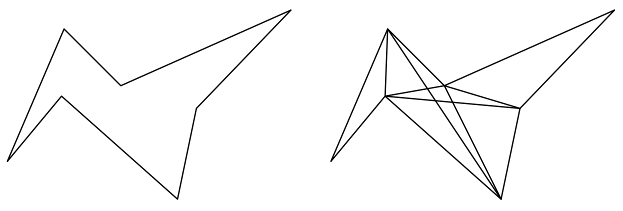


Figure 1: (a) A polygon and (b) its visibility graph.

The *visibility graph* of a polygon is the geometric graph on the plane with the same vertex set as the polygon and in which two vertices are connected by a straight open line segment if they are visible (e.g. see Figure 1).

2.1 Polygon Deflation

A *deformation* of a polygon, P, is a continuous, time-varying, simplicity-preserving transformation of P. Specifically, to each vertex, v, of P, a deformation assigns a continuous mapping $t \mapsto v^t$ from the closed interval $[0, 1] \subset \mathbb{R}$ to the plane such that $v^0 = v$. Additionally, for $t \in [0, 1]$, P^t is simple, where P^t is the polygon joining the images of t in these mappings as their respective vertices are joined in P.

A *monotonic deformation* of P is one in which no two vertices ever become visible, i.e., there do not exist u and v in the vertex set of P and $s, t \in [0, 1]$, with $s < t$, such that u^t and v^t are visible in P^t but u^s and v^s are not visible in P^s.

A polygon is *deflated* if its visibility graph has no edge intersections. Note that a deflated polygon is in general

*The full version of this paper is available at `http://arxiv.org/abs/1206.1982v1`. School of Computer Science, Carleton University, {`jit, vida, nhoda, morin`}`@scs.carleton.ca` position and that its visibility graph is its unique triangulation. Because of this uniqueness and for convenience, we, at times, refer to a deflated polygon and its triangulation interchangeably. A *deflation* of a polygon, P, is a monotonic deformation $t \mapsto P^t$ of P such that P^1 is deflated. If such a deformation exists, then P is *deflatable*.

2.2 Dual Trees of Polygon Triangulations

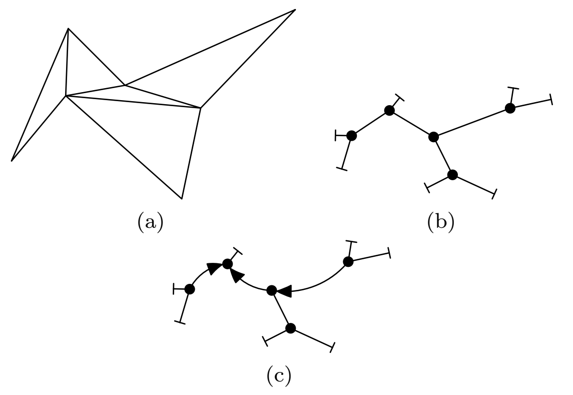


Figure 2: (a) A polygon triangulation, (b) its dual tree and (c) its directed dual. Triangle and terminal nodes are indicated with disks and tees, respectively.

The *dual tree*, D, of a polygon triangulation, T, is a plane tree with a *triangle node* for each triangle of T, a *terminal node* for each polygon edge of T and where two nodes are adjacent if their correspondents in T share a common edge. The dual tree preserves edge orderings of T in the following sense. If a triangle, a, of T has edges e, f and g in counter-clockwise order then the corresponding edges of its correspondent, a^D, in D are ordered e^D, f^D and g^D in counter-clockwise order (e.g. see Figure 2b).

Note that the terminal and triangle nodes of a dual tree have degrees one and three, respectively. We call the edges of terminal nodes *terminal edges*.

An ordered pair of adjacent triangles (a, b) of a polygon triangulation, T, is *right-reflex* if the quadrilateral union of a and b has a reflex vertex, v, situated on the right-hand side of a single segment path from a to b contained in the open quadrilateral. We call v the *reflex endpoint* of the edge shared by a and b (see Figure 3).

The *directed dual*, D, of a polygon triangulation, T, is a dual tree of T that is partially directed such that, for every right-reflex pair of adjacent triangles (a, b) in T, the edge joining the triangle nodes of a and b in D is directed $a \to b$ (e.g. see Figure 2c). Note that if P is deflated, then for every pair of adjacent triangles, (a, b), of T one of (a, b) or (b, a) is right-reflex and so every non-terminal edge in D is directed.

Throughout this paper, as above, we use superscripts

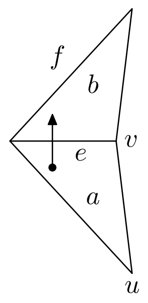


Figure 3: A pair of triangles, a and b, sharing an edge, e, such that their quadrilateral union has a reflex vertex and a single segment path from a to b contained in the open quadrilateral. The reflex endpoint, v, of e is to the right of the path and so the pair (a, b) is right-reflex.

to denote corresponding objects in associated structures. For example, if a is a triangle of the triangulation, T, of a polygon and b is a triangle node in the dual tree, D, of T then a^D and b^T denote the node corresponding to a in D and the triangle corresponding to b in T, respectively.

3 Directed Duals of Deflated Polygons

In this section, we derive some properties of deflated polygons and use them to relate the visibilities of deflated polygons to paths in their directed duals. We also show that two deflated polygons with the same directed dual can be monotonically deformed into one another. The proofs of Lemmas 1, 3 and 4 are not difficult and can be found in the full version of this paper [3].

Lemma 1 *Let P be a deflated polygon, let a be an ear of P and let P' be the polygon resulting from removing a from P. Then P' is deflated.*

Corollary 2 *If the union of a subset of the triangles of a deflated polygon triangulation is a polygon, then it is deflated.*

Lemma 3 *If u is a vertex opposite a closed edge, e, in a triangle of a deflated polygon triangulation, then u sees exactly one polygon edge through e.*

Let u be the vertex of a deflated polygon triangulation, T, and let e be an edge opposite u in a triangle of T. An *induced sequence* of u through e is the sequence of edges through which u sees a polygon edge, f, through e. This sequence is ordered by the proximity to u of their intersections with a closed line segment joining u and f that is interior to the open polygon everywhere but at its endpoints (e.g. see Figure 4b).

Lemma 4 *Suppose u is a vertex opposite a closed non-polygon edge, e, in a triangle, a, of a deflated polygon*

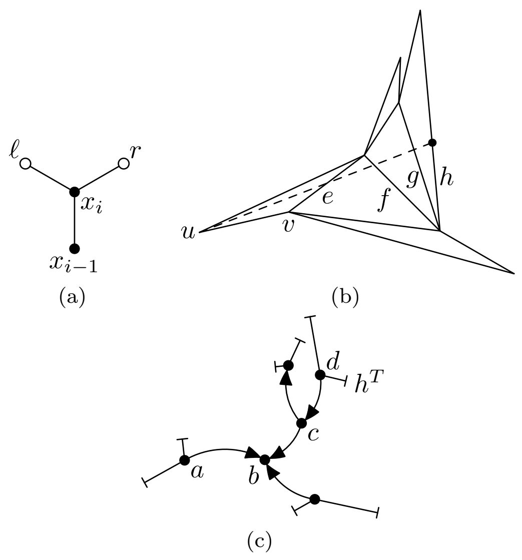


Figure 4: (a) A node x_i of a directed dual and its neighbours x_{i-1}, r and ℓ in an iteration of the construction of a visibility path, (b) a deflated polygon triangulation, T, wherein the induced sequence of the vertex u through the edge e is (e, f, g, h) and (c) the directed dual, T, in which the visibility path of the directed dual starting with nodes (a, b) is (a, b, c, d, h^T).

triangulation. Let v be the reflex endpoint of e and let f be the edge opposite v in the triangle sharing e with a (see Figure 3). Then u sees the same polygon edge through e as v sees through f.

Corollary 5 *If u, v, e and f are as in Lemma 4, then the induced sequence of u through e is equal to that of v through f prepended with e.*

3.1 Directed Duals and Visibility

A *visibility path*, (x_1, x_2, …, x_n), of the *directed dual*, D, of a deflated polygon is a sequence of nodes in D meeting the following conditions. x_1 is a triangle node adjacent to x_2 and, for $i \in \{2, \ldots, n\}$, if x_i is a terminal node, then it is x_n—the final node of the path. Otherwise, let the neighbours of x_i be x_{i-1}, r and ℓ in counter-clockwise order (see Figure 4a). Then

$$x_{i+1} = \begin{cases} r & \text{if edge } \{x_{i-1}, x_i\} \text{ is directed } x_{i-1} \leftarrow x_i \\ \ell & \text{if edge } \{x_{i-1}, x_i\} \text{ is directed } x_{i-1} \rightarrow x_i \end{cases}$$

(e.g. see Figure 4c).

Lemma 6 *Let (a, b, c) be a simple path in the directed dual, D, of a deflated polygon triangulation, T, where a and b are triangle nodes joined by the edge e. Let u be the vertex opposite e^T in a^T, let v be the reflex endpoint of e^T and let f be the edge opposite v in b^T (see Figure 4b). Then (a, b, c) is the substring of a visibility path if and only if f^D joins b and c in D.*

Proof. Suppose (a, b, c) is the substring of a visibility path and let x be the neighbour of b not a nor c and let x' be the edge of b^T not e^T nor f. We consider the case where the neighbours of b are a, x and c in counter-clockwise order—the argument is symmetric in the other case. Then (a, b) is right-reflex and so b^T has counter-clockwise edge ordering: e^T, x', f. Then, since edge orderings are preserved in the directed dual, f^D joins b and c as required. Reversing the argument gives the converse. □

Corollary 7 *Let D, T, a, b, e and u be as in Lemma 6. The induced sequence of u through e is equal to the sequence of correspondents in T of edges traversed by the visibility path starting with (a, b) in D. The final node of this visibility path corresponds to the edge u sees through e^T.*

Theorem 8 *A vertex, u, and edge, g, of a deflated polygon, P, are visible if and only if there is a visibility path in the directed dual, D, of the triangulation, T, of P starting on a triangle node corresponding to a triangle incident to u and ending on g^D.*

Proof. Assume u sees g. If g is an edge of a triangle, a, incident to u then (a^D, g^D) is the required visibility path. Otherwise u sees g through some edge, e, and the existence of the required visibility path follows from Corollary 7.

Assume, now, that the visibility path exists. If its triangle nodes all correspond to triangles incident to u then g is incident to one of these triangles and so visible to u. Otherwise, let e be the first edge the path traverses from a node, a, corresponding to a triangle incident to u to a node, b, corresponding to a triangle not incident to u.

Then, by Corollary 7, the induced sequence of u through e^T corresponds to a visibility path starting with (a, b) and this visibility path ends on a node corresponding to the edge u sees through e. Since two consecutive nodes of a visibility path determine all subsequent nodes, these visibility paths end on the same node, g^D, and so u sees g. □

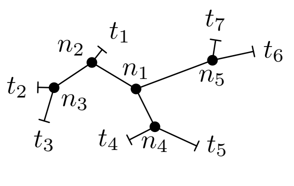


Figure 5: A plane tree with the following maximal outer paths: (t_7, n_5, n_1, n_2, t_1), (t_1, n_2, n_3, t_2), (t_2, n_3, t_3), (t_3, n_3, n_2, n_1, n_4, t_4), (t_4, n_4, t_5), (t_5, n_4, n_1, n_5, t_6), (t_6, n_5, t_7).

An *outer path* of a plane tree, D, is the sequence of nodes visited in a counter-clockwise walk along its outer face in which no node is visited twice. An outer path is *maximal* if it is not a proper substring of any other outer path (e.g. see Figure 5). Note that an outer path, $(x_1, x_2, \ldots, x_n)$, of the directed dual of a polygon triangulation, T, corresponds to a triangle fan in T where the triangles have clockwise order $x_1^T, x_2^T, \ldots, x_n^T$ about their shared vertex.

Theorem 9 *A pair of vertices, u and v, of a deflated polygon P are visible if and only if, in the directed dual, D, of the triangulation, T, of P, their corresponding maximal outer paths share a node.*

Proof. The maximal outer paths of u and v share a node in D if and only if they are incident to a common triangle in T and, since P is deflated, this is the case if and only if u and v are visible. □

3.2 Directed Dual Equivalence

In this section, we show that if two deflated polygons have the same directed dual, then one can be monotonically deformed into the other. First, we fully characterize the directed duals of deflated polygons.

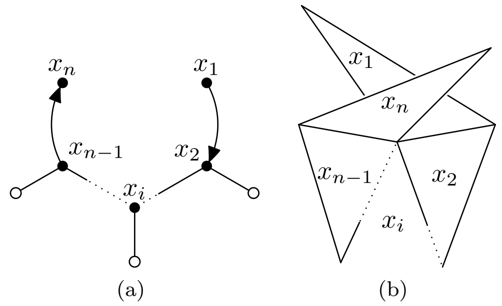


Figure 6: If (a) the tree with outer path $(x_1, x_2, \ldots, x_n)$ were a subtree of the directed dual of a polygon triangulation, T, then (b) the triangles corresponding to nodes x_1, x_2, x_{n-1} and x_n in T would overlap, contradicting the simplicity of the polygon.

Theorem 10 *A partially directed plane tree, D, in which every non-terminal node has degree three and where an edge is directed if and only if it joins two non-terminal nodes of degree three is the directed dual of a deflated polygon if and only if it does not contain an outer path, $(x_1, x_2, \ldots, x_n)$, with $n \geq 4$, such that the edges from x_1 and x_{n-1} are both forward directed (i.e. $x_1 \rightarrow x_2$ and $x_{n-1} \rightarrow x_n$).*

Henceforth, we call such a path an *illegal path*.

Proof. Suppose D contains an illegal path, $(x_1, x_2, \ldots, x_n)$. If D is the directed dual of a polygon triangulation, T, then x_1^T, x_2^T, x_{n-1}^T and x_n^T share a common vertex reflex in both quadrilaterals $x_1^T \cup x_2^T$ and $x_{n-1}^T \cup x_n^T$ (see

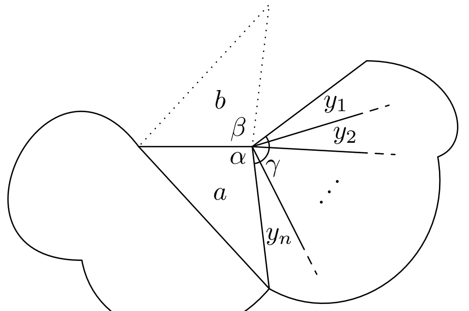


Figure 7: The inductive polygon in the proof of Theorem 10 or a polygon from the inductive deformation in the proof of Theorem 11.

Figure 6). This contradicts the disjointness of these quadrilaterals.

Suppose, now, that D has no illegal paths. We prove the converse with a construction of a polygon triangulation having D as its directed dual. Let b be a terminal node in the subtree of D induced by its non-terminal nodes. Then b has two terminal neighbours and one non-terminal neighbour, a. Let D' be the tree resulting from replacing a and its terminal neighbours with a single terminal node, x, connected to b with an undirected edge. By induction on the number of non-terminal nodes, there exists a deflated polygon triangulation, T, having D' as its directed dual.

Assume, without loss of generality, that the edge joining a and b is directed $a \rightarrow b$. Let u be the endpoint of x^T pointing in a clockwise direction in the boundary of T and let $(y_1, y_2, \ldots, y_n)$ be the outer path of D corresponding to the triangles other than b^T in T incident to u (see Figure 7). Note that $(y_i, y_{i+1}, \ldots, y_n, a, b)$ is an outer path of D and so, by hypothesis, for all $i \in \{1, 2, \ldots, n-1\}$, the edge joining y_i and y_{i+1} is directed $y_i \leftarrow y_{i+1}$.

Then, to show that a triangle may be appended to T to form the required triangulation, it suffices to show that the sum of the angles at u of the triangles y_1^T, y_2^T, $\ldots$, y_n^T is less than π, which, in turn, follows from the backward directedness of the edges of $(y_1, y_2, \ldots, y_n)$. □

Theorem 11 *If the deflated polygons P and P' have the same directed dual, D, then P can be monotonically deformed into P'.*

Proof. Let b be an ear of the triangulation, T, of P and let b' be the triangle corresponding to b^D in the triangulation, T', of P'. By induction on the number of triangles in T, there is a monotonic deformation $t \mapsto Q^t$ from $Q = P \setminus b$ to $Q' = P' \setminus b'$. Note that replacing b^D and its terminal nodes in D with a single terminal node gives the directed dual, D', of Q. Then, since Q

is deflated (Lemma 1) and $t \mapsto Q^t$ is monotonic, for all $t \in [0, 1]$, Q^t is deflated and has directed dual D'.

Let v be the helix of b, let a be the triangle sharing an edge, e, with b and let u be the reflex endpoint of e. We need to show that there is a continuous map $t \mapsto v^t$ that, combined with $t \mapsto Q$, gives a monotonic deformation of a polygon with directed dual D. For $t \in [0, 1]$, let α^t be the angle of a^t at u^t in Q^t and let γ^t be the sum of the angles at u^t of the triangles, y_1^t, y_2^t, $\ldots$, y_n^t, other than a^t of the triangulation of Q^t incident to u^t (see Figure 7).

Then, since v may be brought arbitrarily close to u in a monotonic deformation of P, it suffices to show that there is a continuous map $t \mapsto \beta^t$ specifying an angle for b^t at u^t such that, for all $t \in [0, 1]$, $0 < \beta^t < \pi$, $\alpha^t + \beta^t > \pi$ and $\alpha^t + \beta^t + \gamma^t < 2\pi$. The latter two conditions are equivalent to

$$\pi - \alpha^t < \beta^t < (\pi - \alpha^t) + (\pi - \gamma^t) .$$

It follows from Theorem 10 that the outer path ($y_1^{D'}$, $y_2^{D'}$, $\ldots$, $y_n^{D'}$) is left-directed and so that $\gamma^t < \pi$. Then $\beta^t = \pi - (\alpha^t + \gamma^t)/2$ satisfies all required conditions.

Now, let $t \mapsto R^t$ be the monotonic deformation from a polygon with directed dual D combining $t \mapsto Q^t$ and the map $t \mapsto v^t$ defined by a fixed distance between u^t and v^t of $r \in \mathbb{R}_{>0}$ and an angle for b^t at u^t of β^t.

Prepending $t \mapsto R^t$ with a deformation of P in which v is brought to the distance r from u and then rotated about u to an angle of β^0; then appending a deformation comprising similar motions ending at P'; and, finally, scaling in time gives a continuous map, $t \mapsto P^t$, with $P^0 = P$ and $P^1 = P'$. Since, for all $t \in [0, 1]$, Q^t is simple, a small enough r can be chosen such that $t \mapsto P^t$ is simplicity-preserving. Then, by the properties of $t \mapsto \beta^t$, $t \mapsto P^t$ is the required monotonic deformation. □

4 Deflatability of Polygons

In this section, we show how deflatable polygons may be related combinatorially to their deflation targets and use this result to present a polygon that cannot be deflated. We also show that vertex-vertex visibilities do not determine deflatability. These results depend on the following Lemma.

Lemma 12 *Let $t \mapsto P^t$ be a monotonic deformation of a polygon, P, in general position. Then a vertex and an edge are visible in P^1 only if they are visible in P.*

The proof, which is available in the full version of this paper [3], uses analytic arguments similar to those used by Ábrego et al. [1].

A *compatible directed dual* of a polygon, P, in general position is the directed dual of a deflated polygon, P', such that, under an order- and chirality-preserving bijection between the vertices of P and P', a vertex-edge or vertex-vertex pair are visible in P' only if their correspondents are visible in P. By *chirality*-preserving bijection, we mean one under which a counter-clockwise walk on the boundary of P corresponds to a counter-clockwise walk on the boundary of P'.

Theorem 13 *A polygon, P, in general position with no compatible directed dual is not deflatable.*

Proof. It follows from Lemma 12 that if P is monotonically deformable to a deflated polygon P', then the directed dual of P' is compatible with P. □

Lemma 14 *Suppose a polygon, P, in general position has a compatible directed dual, D. Let P' be the deflated polygon with directed dual D whose vertex-vertex and vertex-edge visibilities are a subset of those of P under an order- and chirality-preserving bijection. Then the unique triangulation, T', of P' is a triangulation, T, of P under the bijection and D can be constructed by directing the undirected non-terminal edges of the directed dual of T.*

Proof. Note that T' is the visibility graph of P'. Then, since P is in general position and has the same vertex count as P', it follows from the vertex-vertex visibility subset property of P' that T' triangulates P under the bijection.

It remains to show that, for every non-terminal edge of the directed dual of T, either the edge is undirected or it is directed as in D or, equivalently, that for every pair of adjacent triangles, a and b, in T corresponding to the triangles a' and b' in T', if (a, b) is right-reflex then so is (a', b'). Suppose, instead, that (b', a') is right-reflex. Let e' be the edge shared by a' and b', let u' be the vertex of a' opposite e' and let f' be the edge of b' opposite the reflex endpoint of e'. Then, by Lemma 4, u' sees an edge through f' but the corresponding visibility is not present in P, contradicting the vertex-edge visibility subset property of P'. □

Theorem 15 *There exists a polygon that cannot be deflated.*

Proof. We show that the general position polygon, P, in Figure 8a has no compatible directed dual and so, by Lemma 13, is not deflatable. Assume that the directed dual, D, of a deflated polygon, P', is compatible with P. Then, by Lemma 14, D can be constructed by directing the undirected non-terminal edges of the directed dual of some triangulation of P. Up to symmetry, P has a single triangulation, its directed dual has a single undirected non-terminal edge and there is a single way to direct this edge. Then we may assume, without loss of generality, that D is the tree shown in Figure 8b and, by

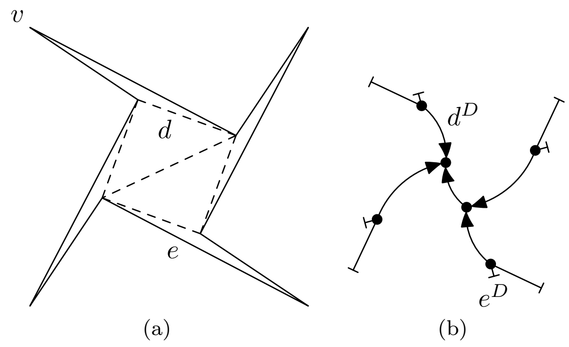


Figure 8: (a) A non-deflatable polygon, P, with its only triangulation, up to symmetry, indicated with dashed lines and (b) its only candidate for a compatible directed dual, D, up to symmetry.

Theorem 8, the correspondents of the vertex v and edge e in P' are visible. This contradicts the compatibility of D. □

Note that, although the non-deflatability of P can be shown using *ad hoc* arguments, the combinatorial technique used here can be applied to other polygons. See the full version of this paper [3] for examples.

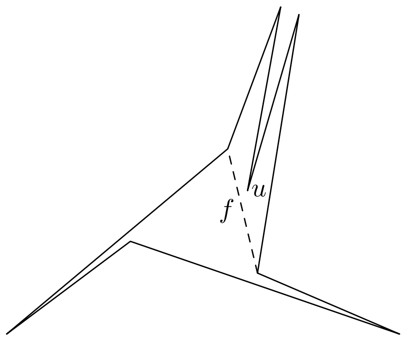


Figure 9: A deflatable polygon with the same vertex-vertex visibilities as the non-deflatable polygon shown in Figure 8a.

Theorem 16 *The vertex-vertex visibilities of a polygon do not determine its deflatability.*

Proof. The polygon in Figure 9 has the same vertex-vertex visibilities as the non-deflatable polygon in Figure 8a and yet can be deflated by moving the vertex u through the diagonal f. □

5 Summary and Conclusion

We presented the directed dual and showed that it captures the visibility properties of deflated polygons. We then showed that two deflated polygons with the same directed dual can be monotonically deformed into one another. Next, we showed that directed duals can be used to reason combinatorially, via directed dual compatibility, about the deflatability of polygons. Finally, we presented a polygon that cannot be deflated and showed that the vertex-vertex visibilities of a polygon do not determine its deflatability.

A full characterization of deflatable polygons still remains to be found. If the converse of Theorem 13 is true, then the existence of a compatible directed dual gives such a characterization. We conjecture the following weaker statement.

Conjecture 1 *The vertex-edge visibilities of a polygon in general position determine its deflatability.*

We conclude, however, by noting that, in light of Mnev's Universality Theorem [7], it is unknown if even the order type of a polygon's vertex set determines its deflatability.

6 Acknowledgements

This research was partly funded by NSERC and by Carleton University through an I-CUREUS internship. We would also like to thank Joseph O'Rourke for informing us of a property of point set order types.

References

[1] B. Ábrego, M. Cetina, J. Leaños, and G. Salazar. Visibility-preserving convexifications using single-vertex moves. *Information Processing Letters*, 112(5):161–163, 2012.

[2] O. Aichholzer, G. Aloupis, E. D. Demaine, M. L. Demaine, V. Dujmović, F. Hurtado, A. Lubiw, G. Rote, A. Schulz, D. L. Souvaine, and A. Winslow. Convexifying polygons without losing visibilities. In *Proc. 23rd Annual Canadian Conference on Computational Geometry (CCCG)*, pages 229–234, 2011.

[3] P. Bose, V. Dujmović, N. Hoda, and P. Morin. Visibility-monotonic polygon deflation. arXiv:1206.1982v1.

[4] R. Connelly, E. Demaine, and G. Rote. Straightening polygonal arcs and convexifying polygonal cycles. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 432–442, 2000.

[5] S. Devadoss, R. Shah, X. Shao, and E. Winston. Deformations of associahedra and visibility graphs. *Contributions to Discrete Mathematics*, 7(1):68–81, 2012.

[6] S. K. Ghosh. *Visibility Algorithms in the Plane.* Cambridge University Press, 2007.

[7] N. Mnev. The universality theorems on the classification problem of configuration varieties and convex polytopes varieties. In O. Viro and A. Vershik, editors, *Topology and Geometry – Rohlin Seminar*, volume 1346 of *Lecture Notes in Mathematics*, pages 527–543. Springer Berlin / Heidelberg, 1988. 10.1007/BFb0082792.

[8] J. O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, 1987.

Common Developments of Three Different Orthogonal Boxes

Toshihiro Shirakawa Ryuhei Uehara[*]

Abstract

We investigate common developments that can fold into plural incongruent orthogonal boxes. It was shown that there are infinitely many orthogonal polygons that fold into two incongruent orthogonal boxes in 2008. In 2011, it was shown that there exists an orthogonal polygon that folds into three boxes of size $1 \times 1 \times 5$, $1 \times 2 \times 3$, and $0 \times 1 \times 11$. It remained open whether there exists an orthogonal polygon that folds into three boxes of positive volume. We give an affirmative answer to this open problem: there exists an orthogonal polygon that folds into three boxes of size $7 \times 8 \times 56$, $7 \times 14 \times 38$, and $2 \times 13 \times 58$. The construction idea can be generalized, and hence there exists an infinite number of orthogonal polygons that fold into three incongruent orthogonal boxes.

1 Introduction

Since Lubiw and O'Rourke posed the problem in 1996 [5], polygons that can fold into a (convex) polyhedron have been investigated. In the book on geometric folding algorithms by Demaine and O'Rourke in 2007, many results about such polygons are given [4, Chapter 25]. Such polygons have many applications including toys and puzzles. For example, the puzzle "cubigami" (Figure 1) is developed by Miller and Knuth, and it is a common development of all tetracubes except one (since the last one has surface area 16, while the others have surface area 18). One of the many interesting problems in this area is whether there exists a polygon that folds into plural incongruent orthogonal boxes. Biedl et al. gave two polygons that fold into two incongruent orthogonal boxes [3] (see also [4, Figure 25.53]). Later, Mitani and Uehara constructed infinite families of orthogonal polygons that fold into two incongruent orthogonal boxes [6]. Last year, Abel et al. showed an orthogonal polygon that folds into three boxes of size $1 \times 1 \times 5$, $1 \times 2 \times 3$, and $0 \times 1 \times 11$ [1]. However, the last "box" has volume zero; this is a so called "doubly covered rectangle" (e.g., [2]). Therefore, it remains open to show whether there is a polygon that can fold into three or more boxes of positive volume.

We give an affirmative answer to this open problem; there exists an orthogonal polygon that can fold into

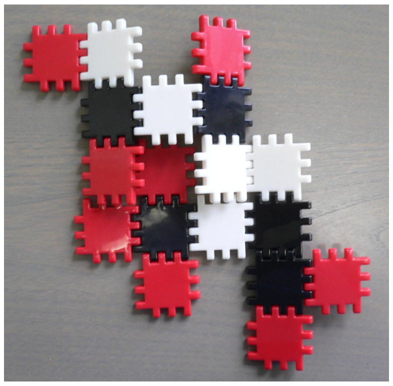


Figure 1: Cubigami.

three incongruent orthogonal boxes of size $7 \times 8 \times 56$, $7 \times 14 \times 38$, and $2 \times 13 \times 58$ (Figure 2)[1].

The construction idea can be generalized. Therefore, we conclude that there exist infinitely many orthogonal polygons that can fold into three incongruent orthogonal boxes.

2 Construction of the common development

The definition of the *development* of a solid can be found in [4, Chap. 21]. Roughly, the development is the unfolding obtained by slicing the surface of the solid, and it forms a single connected simple polygon without self-overlap. The *common development* of two (or more) solids is the development that can fold into the solids. In this paper, as developments, we only consider orthogonal polygons that consist of unit squares.

Intuitively, the basic construction idea is simple. We first choose a common development of two different boxes of size $a \times b \times c$ and $a' \times b' \times c'$. We select one of these two boxes; let it have size $a \times b \times c$. We cut the two rectangles of size $a \times b$ (one at the *top*, and another at the *bottom* of the box) into two pieces of size $a \times b/2$ each. Then we squash the box and make these two rectangles of size $a \times b$ into two rectangles of size $(a + b/2) \times b/2 = 2a \times b/2$ (Figure 3). However, this simple idea immediately comes to a dead end; this operation can be done properly if and only if $a = b/2$, and hence we only change the rectangle of size 1×2 into the

[*]School of Information Science, JAIST, Asahidai 1-1, Nomi, Ishikawa 923-1292, Japan. uehara@jaist.ac.jp

[1]This figure is also available at http://www.jaist.ac.jp/~uehara/etc/puzzle/nets/3box.pdf for ease to cut and fold.

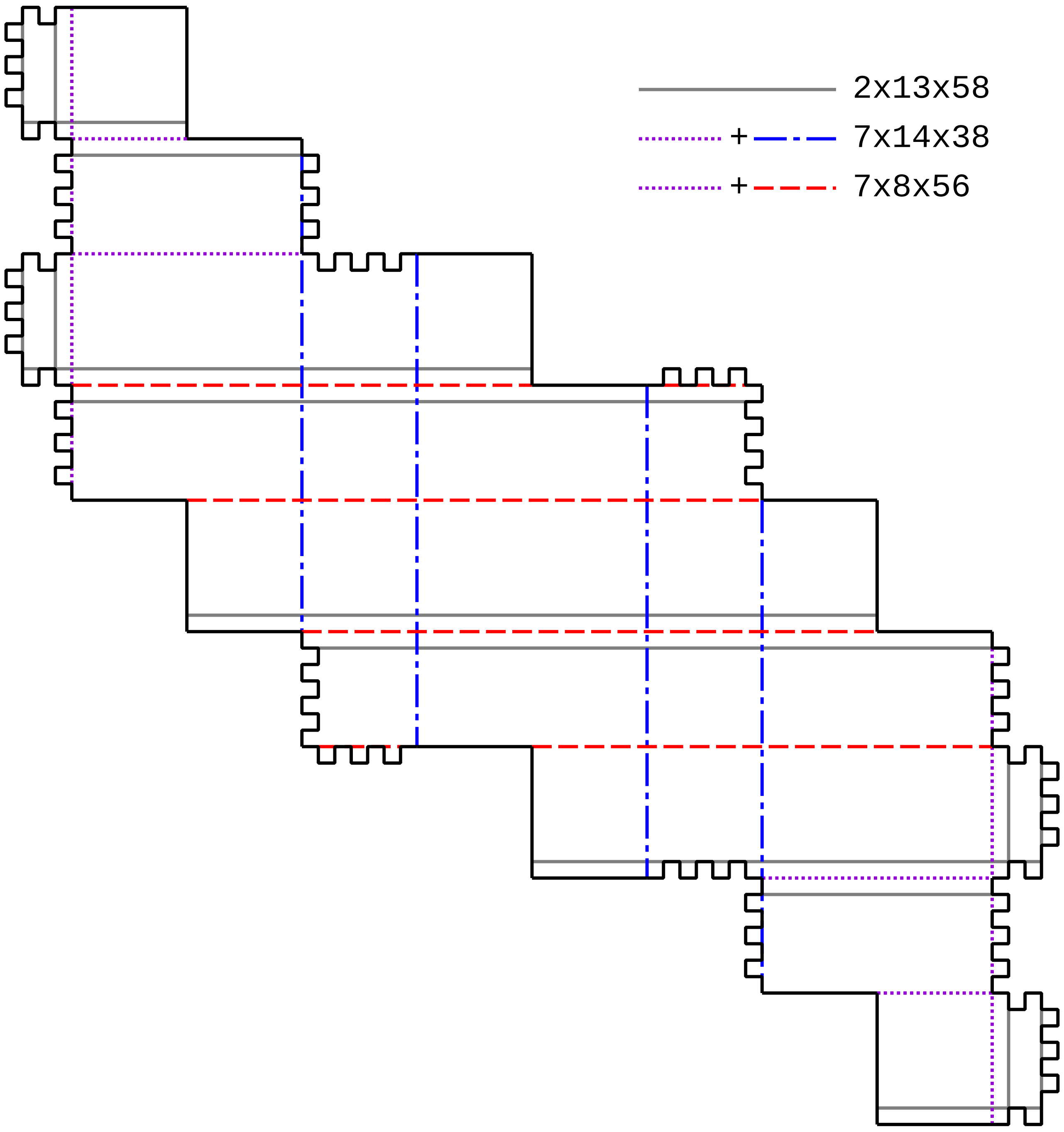


Figure 2: A common development of three different boxes of size $7 \times 8 \times 56$, $7 \times 14 \times 38$, and $2 \times 13 \times 58$.

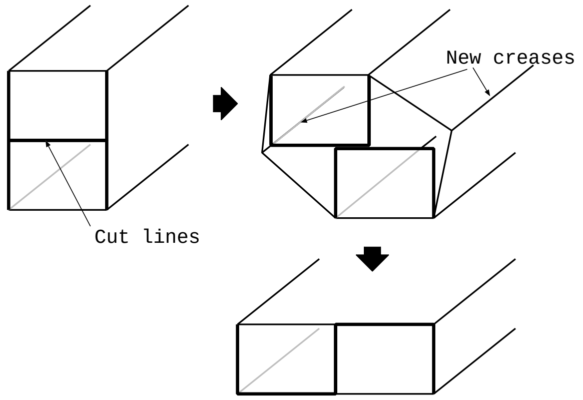


Figure 3: Basic idea: squash the box.

other rectangle of size 2×1, which are congruent.

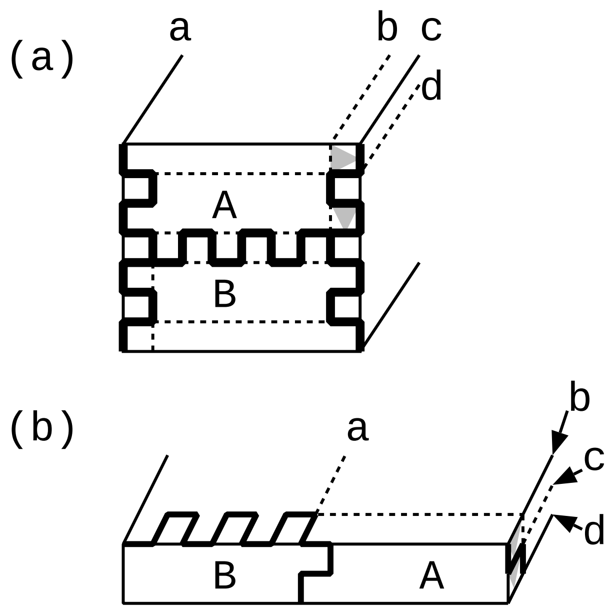


Figure 4: Squash the box: cut and fold.

The main trick to avoid this problem is to move pieces of the rectangles of size $a \times b$ of the box to the side rectangles of size $b \times c$ and $a \times c$. That is, after the squash operation above, the surface areas of the resultant top and bottom rectangles decrease, and the side rectangles grow a little. A specific example is given in Figure 4; in this example, the rectangle of size 8×7 is split into two congruent pieces by a mid zig-zag line; each piece in turn is divided into one central piece (labeled A, B in Figure 4). The result is a rectangle of size 13×2. (In Figure 4(a), the bold lines are cut lines, and dotted lines are folding lines to obtain (b). The lines a, b, c, and d are corresponding, and the gray triangles indicate how two squares are arranged by the operation.) Among the 56 squares, $56 - 26 = 30$ squares are moved to the four sides. We note that the perimeter of these two rectangles is not changed since $7 + 8 + 7 + 8 = 2 + 13 + 2 + 13 = 30$.

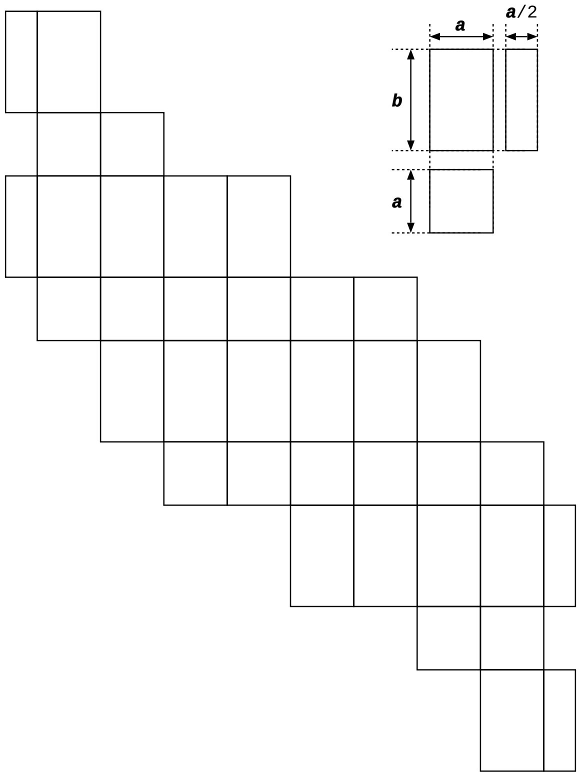


Figure 5: The base common development of two boxes of size $a \times b \times 8a$ and $a \times 2a \times (2a + 3b)$.

To apply this idea, we choose a common development of two boxes of size $a \times b \times 8a$ and $a \times 2a \times (2a + 3b)$ in Figure 5. This is a modification of the common development of two boxes of size $1 \times 1 \times 8$ and $1 \times 2 \times 5$ in [6, Figure 5]. To apply the idea, we cut each of the top and bottom rectangles of size $a \times b$ into two congruent rectangles of size $a/2 \times b$. For any integers a and b, the orthogonal polygon in Figure 5 is a common development of two boxes of size $a \times b \times 8a$ and $a \times 2a \times (2a + 3b)$ (the two folding ways are drawn in bold lines in Figure 6).

The development in Figure 5 has useful properties for applying the idea in Figure 3: (1) we can adjust the size of the top and bottom rectangles to an arbitrary size, and (2) two folding ways share several folding lines. Especially, in Figure 6, each of the two connected gray areas is folded in the same way in both folding ways. Thus we attach the gadget from Figure 3 at this area letting $a = 7$ and $b = 8$. That is, we replace the rectangles of size $a/2 \times b$ by the rectangles A and B surrounded by the zig-zag lines in Figure 4.

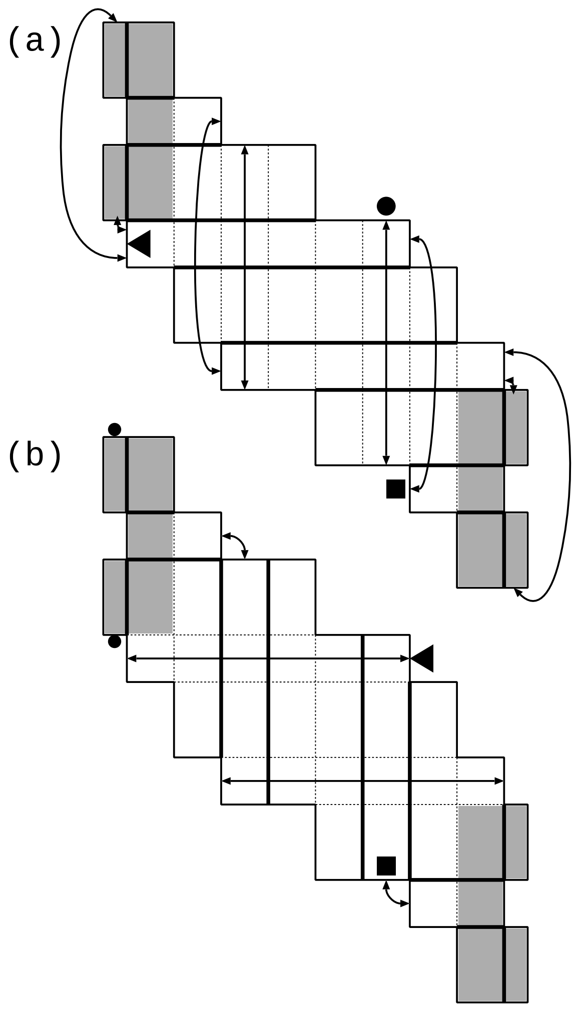


Figure 6: Some properties of the common development of two boxes of size $a \times b \times 8a$ and $a \times 2a \times (2a + 3b)$.

The only problem when applying the gadget is that the zig-zag lines propagate themselves according to the folding ways. That is, the zig-zag lines are glued to the different edges in some folding. For example, a zig-zag line at black triangle in Figure 6(a) is attached to the edge at black triangle in the folding way in Figure 6(b). Thus, these edges must consist of the same zig-zag pattern. On the other hand, this edge is attached to the edge at the black square in the folding way in Figure 6(a), which is attached to the black square in Figure 6(b). Thus, they also must have the same zig-zag pattern. Then the last edge is again attached to the edge with the black circle in Figure 6(a), and this is attached to the two edges with the smaller black circles in Figure 6(b). Then the loop of the propagation is closed, and we obtain the set of the edges that have to be represented by the zig-zag pattern.

Checking all the propagations, we finally obtain a common development of three different boxes of size $7 \times 8 \times 56$, $7 \times 14 \times 38$, and $2 \times 13 \times 58$ in Figure 2.

3 Generalization

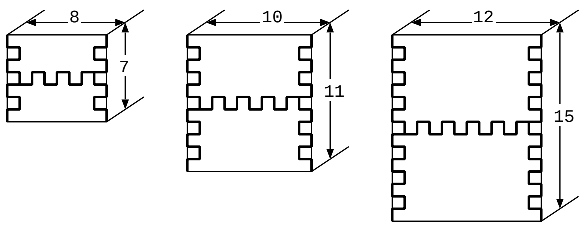


Figure 7: Generalization of the zig-zag cut.

In Section 2, we set $a = 7$ and $b = 8$, and change the rectangle of size 7×8 into 2×13. It is straightforward to generalize this method. For example, setting $a = 11$ and $b = 10$, we can change the rectangle of size 11×10 into 4×17 (see Figure 7). In general, for each integer $k = 0, 1, 2, \ldots$, setting $a = 4k+7$ and $b = 2(k+4)$, we can change the rectangle of size $a \times b$ to $2(k+1) \times (4k+13)$ in the same way as in Figure 4. The difference here from Figure 2 is in the number of turns of the zig-zags. Therefore, we have the following theorem immediately:

Theorem 1 *For each integer $k = 0, 1, 2, \ldots$, there is a common development that can fold into three different boxes of size $(4k+7) \times 2(k+4) \times 8(4k+7)$, $(4k+7) \times 2(4k+7) \times 2(7k+19)$, and $2(k+1) \times (4k+13) \times 2(16k+29)$.*

That is, there exists an infinite number of orthogonal polygons that can fold into three incongruent orthogonal boxes.

4 Concluding remarks

It is an open question if a polygon exists that can fold into four or more orthogonal boxes such that all of them have positive volume.

When two boxes of size $a \times b \times c$ and $a' \times b' \times c'$ share a common development, they satisfy a simple necessary condition $ab + bc + ca = a'b' + b'c' + c'a'$ since they have the same surface area. According to the experiments in [6], this necessary condition seems also sufficient for two boxes: for each pair of 3-tuples of integers satisfying the condition, there exist many common developments of two boxes of these size [6]. In this sense, the smallest possible surface area that can fold into three different boxes is 46; the area can produce three boxes of size $(1, 1, 11)$, $(1, 2, 7)$, and $(1, 3, 5)$. On the other hand, our construction produces a polygon of large surface area. The polygon in Figure 2 has area 1792. Applying the same idea to the different common development in [6], we also construct another smaller development of area

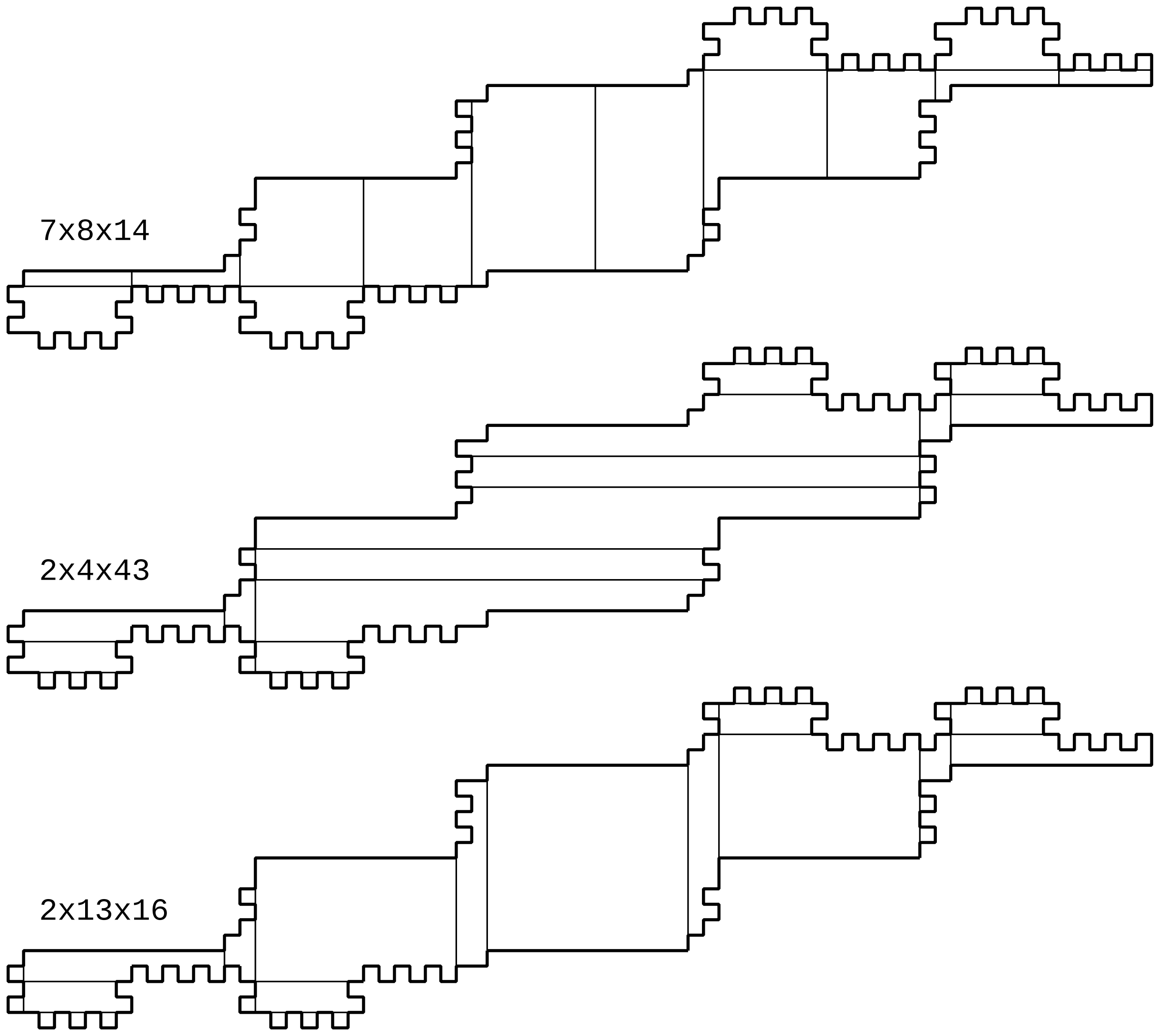


Figure 8: Another polygon that can fold into three boxes of size $7 \times 8 \times 14$, $2 \times 4 \times 43$, and $2 \times 13 \times 16$.

532 (Figure 8). Comparing to the results for two boxes, finding much smaller polygons would be a future work. Especially, is there a common development of area 46 that can fold into three boxes of size $(1, 1, 11)$, $(1, 2, 7)$, and $(1, 3, 5)$?

References

[1] Z. Abel, E. Demaine, M. Demaine, H. Matsui, G. Rote, and R. Uehara. Common Development of Several Different Orthogonal Boxes. In *23rd Canadian Conference on Computational Geometry (CCCG 2011)*, pages 77–82, 2011.

[2] J. Akiyama. Tile-Makers and Semi-Tile-Makers. *The Mathematical Association of Amerika*, Monthly 114:602–609, August-September 2007.

[3] T. Biedl, T. Chan, E. Demaine, M. Demaine, A. Lubiw, J. I. Munro, and J. Shallit. Notes from the University of Waterloo Algorithmic Problem Session. September 8 1999.

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

[5] A. Lubiw and J. O'Rourke. When Can a Polygon Fold to a Polytope? 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 *Canadian Conference on Computational Geometry (CCCG 2008)*, pages 39–42, 2008.

Unfolding Rectangle-Faced Orthostacks

Erin W. Chambers* Kyle A. Sykes† Cynthia M. Traub‡

Abstract

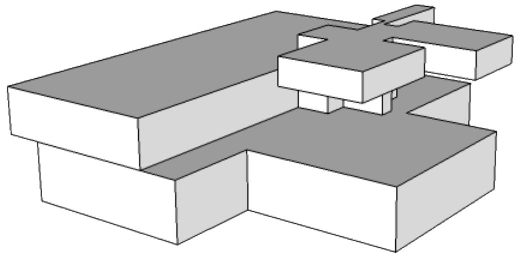
We prove that rectangle-faced orthostacks, a restricted class of orthostacks, can be grid-edge unfolded without additional refinement. We prove several lemmas applicable to larger classes of orthostacks, and construct an example to illustrate that our algorithm does not directly extend to more general classes of orthostacks.

1 Introduction

An *unfolding* of a polyhedron is a cutting of the surface of the polyhedron such that the surface may be unfolded into the plane as a simple polygon where the interior of any two faces does not overlap. An *edge unfolding* considers only cuts made along edges, while *general unfoldings* allow cuts anywhere on the surface.

There are many open questions relating to polyhedral unfoldings. For example, while it is known that not every nonconvex polyhedron has an edge-unfolding, it is still open whether every polyhedron has a general unfolding. In general, progress has been made on this problem by considering restricted classes of polyhedra [2, 3, 10, 5] or by varying the type of cuts that are allowed, such as vertex unfoldings [7, 6, 4] or star unfoldings [1, 9]. See [8, 11] for surveys of this area.

We will consider an unrefined grid-edge unfolding of a class of axis-orthogonal polyhedra known as orthostacks (formally defined in Section 2). An unrefined *grid-edge* unfolding creates new edges on the surface of a polyhedron by intersecting the surface with planes parallel to the $x, y, z-$axes through every vertex of the polyhedron. Any of the edges from the original polyhedral surface as well as these new edges are now available for cutting. The technique of grid-edge refinement can be generalized by further dividing every rectangle of the surface into $k \times l$ rectangles. An unrefined grid-edge unfolding is thus a 1×1 refinement. It is known that every orthostack can be grid-edge unfolded with a 1×2 refinement [2]. In this paper we prove that a certain class of orthostacks (which we call "rectangle-faced orthostacks") can be grid-edge unfolded without further refinement of the surface.



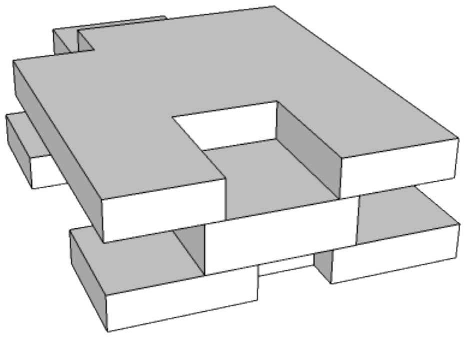


Figure 1: (a) An example of an orthostack. (b) A rectangle-faced orthostack.

Our algorithm is a natural one given the structure of orthostacks, where we unfold each layer of the orthostack and connect them via "bridges" between the layers. It has a similar setup to the one for the 1×2 refinement [2], although theirs cannot choose bridges in the same fashion; they instead cut each band vertically in half to "shift" the bridge-rectangle to the top position. Unfortunately, our algorithm will not extend to general orthostacks; in section 5 we present a (non-rectangular-faced) orthostack which our algorithm fails to unfold. We conclude with a discussion of how our structural results may be useful for computing 1×1 unfoldings of more general classes of orthostacks.

2 Definitions

An orthostack P is a genus-zero axis-orthogonal polyhedron with the property that that in at least one dimension, each distinct cross section of P is an orthog-

*Department of Mathematics and Computer Science, Saint Louis University, echambe5@slu.edu. Research supported in part by the NSF under Grant No. CCF 1054779.

†Department of Mathematics and Computer Science, Saint Louis University, ksykes2@slu.edu

‡Department of Mathematics and Statistics, Southern Illinois University Edwardsville, cytraub@siue.edu

onal polygon that is both connected and simply connected (containing no holes). Without loss of generality, assume this dimension is the $z-$dimension. Following the terminology used in [2], we name the faces of the orthostack according to the axis to which they are orthogonal, thus giving rise to $x-$, $y-$ and $z-$faces. Changes in the $z-$cross sections of an orthostack occur at finitely many $z-$coordinates $z_0, z_1, \ldots, z_k$. We call the *i-th band* B_i the collection of $x-$ and $y-$faces that form the boundary of the orthostack for $z_{i-1} \leq z \leq z_i$, where i ranges from 1 to k. So the faces of the orthostack P are therefore partitioned into bands B_i ($1 \leq i \leq k$) and the $z-$faces that occur in the planes $z = z_0, z = z_1, \ldots, z = z_k$. Note that the $z-$faces occur at the "top" and "bottom" of the orthostack as well as in the layers forming transitions between bands. We will also let *layer* L_i be the subset of the orthostack with $z_{i-1} \leq z \leq z_i$, which is the 3-dimensional solid bounded by $B_i, z = z_{i-1}$, and $z = z_i$.

We call an orthostack *rectangle-faced* if every $z-$face is a rectangle, excluding the $z-$faces on the top of bottom of the orthostack, and edges of these z-faces are entirely along one band or another (with no edge belonging to both adjacent bands). Examples of orthostacks with and without the rectangle-faced property are shown in Figure 1.

3 Structural results

We begin with several structural lemmas regarding orthostacks. Note that these results apply to *any* orthostack, not just rectangle-faced ones, and may be of use for more general classes of orthostacks.

Lemma 1 *Any z-face at height $z = z_i$, $1 \leq i \leq k-1$, must be incident to both B_i and B_{i+1}.*

Proof. Suppose to the contrary that some z-face, R, at height $z = z_i$ has edges only incident to one band, which we assume without loss of generality is B_i. The face R does not occur at $z = z_0$ or $z = z_k$ due to our initial assumption on i, so there must exist a subset of $B_i \cap (z = z_i)$ that is not incident to R. (Else, the orthostack will not continue past the face R, a contradiction.) The intersection produced by slicing the orthostack with a plane $z = z_i - \epsilon$ for a sufficiently small value of ϵ will either be disconnected or a degenerate polygon consisting of (at least) two polygons attached at a single vertex. Both situations contradict the definition of an orthostack, since each z-slice must be a simply connected polygon. Therefore, R must have edges incident to both B_i and B_{i+1}. □

Lemma 2 *The perimeter of any z-face at height $z = z_i$, $1 \leq i \leq k-1$, is partitioned into two contiguous components, one incident to band B_i and the other incident to B_{i+1}. Moreover, some pair of opposite edges a and b of the face will have edge e_1 containing a segment incident to B_i and edge e_2 containing a segment incident to B_{i+1}.*

Proof. Assume that the boundary of a z-face R is partitioned into more than two contiguous components from bands B_i and B_{i+1}; see Figure 2. At this z_i layer, the rectangle must be visible; this happens due to a change in the layers of the orthostack. Namely, the cross-sections above and below $z = z_i$ are distinguished by which cross-section includes R. Thinking of R in terms of x and y coordinates, we assume without loss of generality that, for sufficiently small ϵ, $R \times [z_i - \epsilon, z_i]$ is contained in layer L_i and $R \times [z_i, z_i + \epsilon]$ is exterior to layer L_{i+1}.

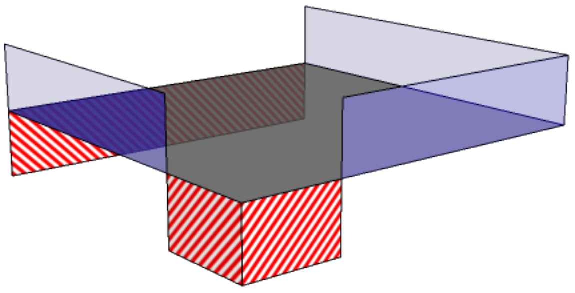


Figure 2: A 3-dimensional view of how rectangle R (shaded darker) appears in the orthostack; the adjacent portions of B_i and B_{i+1} that border R are shown red (striped) and blue (solid).

Since each intersection of the orthostack with a z-plane is one simply connected polygon, the two or more connected components of $R \cap B_{i+1}$ must be pathwise connected to one another via $B_{i+1} \cap (z = z_i)$, a curve we color solid blue in Figure 3. Moreover, since the blue curve is the boundary of a simply connected polygon in the $z = z_i$ plane, it will not self intersect.

Two possible orientations of these paths are given in Figure 3, where the rectangle boundaries shaded in red come from band B_i and those from B_{i+1} are shaded in blue. In case 1 (Figure 3, left), there is a region (illustrated near the upper right corner of the rectangle) exterior to the cross section of the rectangular face but completely surrounded by the union of the rectangle with the area bounded by the solid blue curve. This forms a cross-section at $z = z_i$ in the orthostack which fails to be simply connected, contradicting the definition of orthostack. In case 2 (Figure 3, right), the blue curve represents an "inner" boundary of B_{i+1}, which means that the cross section at $z = z_i$ will also fail to be simply connected (since there is a gap between the rectangle and the blue curve marking the inner boundary

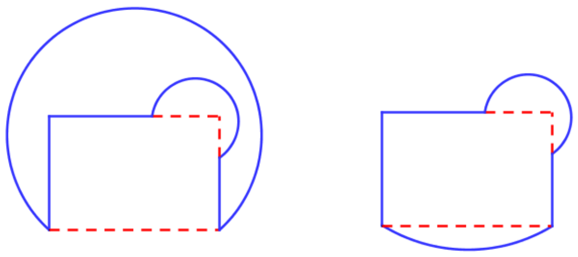


Figure 3: Two possible visualizations for our face R, where the red (dashed) component of the boundary is adjacent to band B_i and the blue (solid) is adjacent to band B_{i+1}.

of B_{i+1}). In either case, we contradict the assumption that the original object was an orthostack and therefore had simply connected orthogonal cross sections.

Note that in either of the two cases, the contradiction appears between two components of the boundary of R induced by B_{i+1} that appear sequentially around the boundary of R. While our image only shows 2 connected components total, the same contradiction is present even if more than two connected components are present along the boundary of R. □

We will use lemma 2 in the next section to set up "bridges" between the unfolded bands in our algorithm, but we also need to characterize the order in which the z-faces are encountered before proceeding with our algorithm.

Lemma 3 *For any orthostack, the cyclic ordering of the rectangles at height $z = z_i$, given by tracing around the band B_i counterclockwise and numbering z-faces by the order in which they are encountered, is the same as the cyclic ordering given by tracing the band B_{i+1} counterclockwise.*

Proof. Begin by fixing $1 \le i \le k-1$. First suppose that the bands B_i and B_{i+1} do not share any common point. If there are any z-faces, then using the previous lemma, we know that the z-face must form an annulus bounded by B_i on one side and B_{i+1} on the other, since any other configuration will either result in a common point or non-contiguous components adjacent to B_i or B_{i+1}. Moreover, this must be the only z-face at this level, since otherwise the band must stop tracing the boundary of the z-face and later re-enter it after tracing around another z-face, which results in that layer not being simply connected. Since there is only one z-face, the statement of the lemma holds trivially.

Now if B_i and B_{i+1} have some common point, pick any one of them as a start point. Proceed counterclockwise on a path along the boundary at cross section z_i shared by both bands until the bands diverge, which must happen if there is any z-face at height z_i. When the path diverges, you have met a z-face α at a boundary point where B_i and B_{i+1} meet. One path will trace the boundary of α shared with B_i and the other will trace the boundary of α shared with B_{i+1}. Note that neither band can move away from α and then return, since by Lemma 2 we know that each band stays adjacent to the face in a single connected component along the boundary of the α. Label this z-face as α_1. The paths continue tracing their respective boundaries until they intersect at a point shared by both boundaries, which again follows from Lemma 2. At this point, the paths merge into a single path again and continue tracing counterclockwise along a portion adjacent to both bands (which possibly consists of only a single point, if we meet the next z-face immediate). We can continue following the bands counterclockwise along the shared boundary at height z_i, with the bands diverging only when they meet the same new z-face. We will label these z-faces in the order we meet them. Proceeding in this manner around the perimeter of both bands gives a unique labeling of all the z-faces as $\alpha_1, \alpha_2, \ldots, \alpha_k$ which is common to both bands, so the clockwise ordering is identical. □

4 Algorithm

In this section, we restrict our attention to *rectangle-faced orthostacks*, where the z-faces are rectangles whose edges entirely belong to the boundary of B_i or B_{i+1}, but not both.

Our unfolding algorithm proceeds as follows. We will unfold into an xz-plane, in order of increasing z-coordinate. References to x-coordinates will refer to the unfolded shape in this plane. Start by unfolding band B_1 arbitrarily. Attach the z_0-face arbitrarily below band B_1. Loop through the following steps for $i = 1$ to $k-1$.

1. Consider all the rectangles from layer $z = z_i$. We know these must attach to both B_i and B_{i+1} on some opposite pair of edges by Lemma 2. We consider only these opposite edges as possible attachments for these z-faces, and attach them arbitrarily to the band B_i.

2. Now among the attached rectangles, choose the one which has the highest x-coordinate; this is our "bridge" between B_i and B_{i+1}. We then attach band B_{i+1} to the bridge rectangle and unfold B_{i+1} "upward" into the increasing x direction.

Lastly, we can glue z_k-face to the final band arbitrarily, since there is no "next" band to conflict with any possible attachment point.

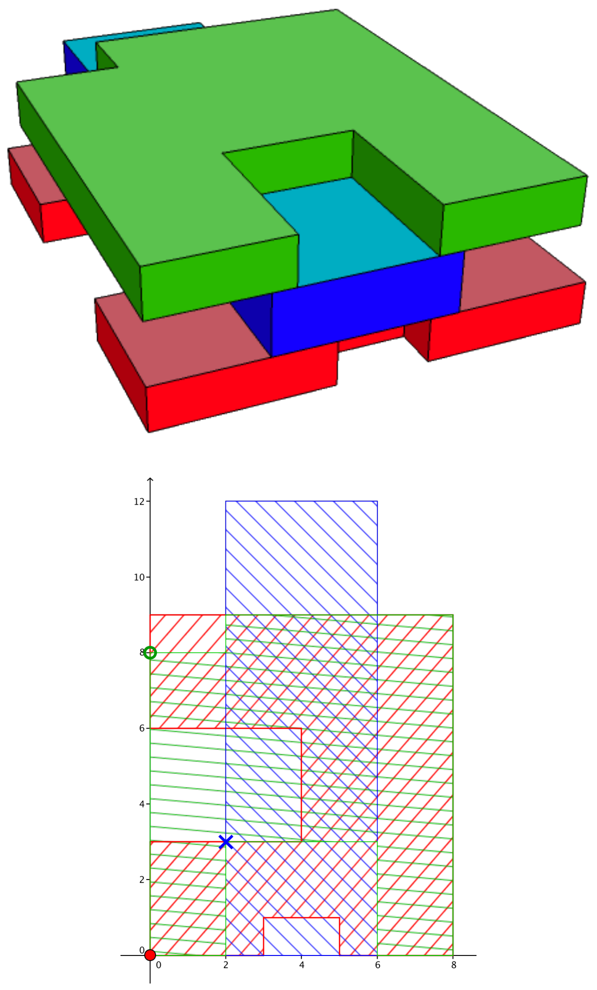


Figure 4: Top: An example of a rectangle-faced orthostack. Bottom: The same orthostack when viewed looking towards the $-z$ direction. A solid red dot indicates where we choose our initial cut for band B_1 (Red, bottom layer in top picture), a blue X indicates where the band B_2 (Blue, middle layer) is cut, and the hollow green point indicates where the band B_3 (Green, top layer) is cut.

5 Conclusion & Further Work

It remains to be shown whether every orthostack can be grid-unfolded with a 1x1 refinement. Recall that the structural lemmas in Section 3 extend to orthostacks in general, and might lend insight to the more general problem. In particular, even with rectangles that are not rectangle-faced, if the faces are rectangular then a version of Lemma 2 applies, and there must be a pair of edges which at least partially border the two adjacent bands.

The obvious extension of our algorithm to orthostacks

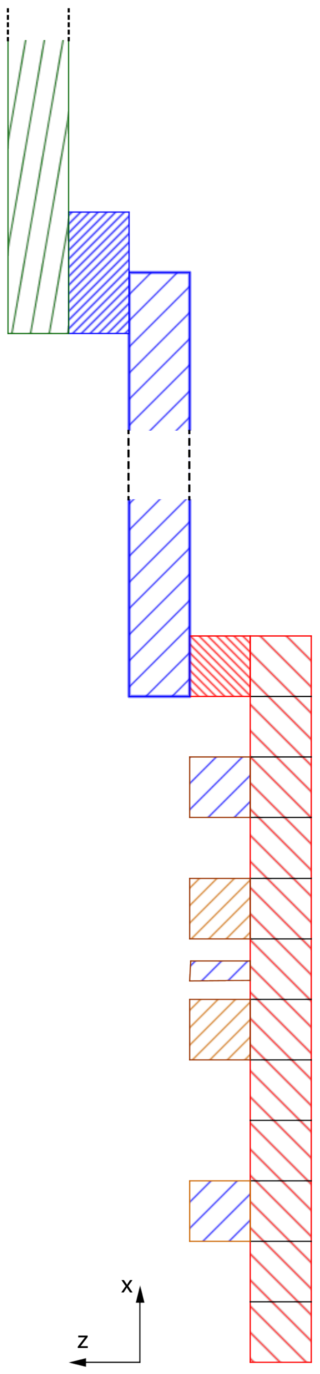


Figure 5: An partial unfolding of the orthostack in Figure 1(b). The bands are not to scale, and only the $z-$faces where $z = z_1$ between B_1 and B_2 are shown attached to B_1.

with only rectangular faces between the bands would be to again choose the bridge rectangle which has the highest x-coordinate, and unfold the adjacent band in the increasing x direction. However, in this case, since the bridge may not have an entire edge which is adjacent to

B_{i+1}, we lose the fact that we can arbitrarily glue rectangles as in step 1. This insight leads to an example of a rectangular orthostack (where all z-faces are rectangles but may have edges adjacent to each band) on which our algorithm will fail; see Figure 6. Note that in this example, the faces shaded red will overlap when unfolded via our algorithm.

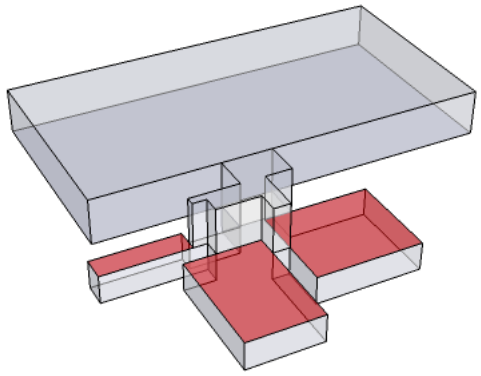


Figure 6: A rectangular orthostack (that is not rectangle-faced) where our algorithm fails to unfold into a planar polygon.

As an alternative, we propose the following algorithm. Instead of choosing the rectangle with the largest x-coordinate as our "bridge" between B_i and B_{i+1} in step 2, we could instead choose the rectangle which has the greatest width (measured so that this width has a component of B_i along one side and B_{i+1} along the opposite side, so that it could serve as a bridge). Intuitively, this rectangle separates the bands as much as possible, so that every other rectangle would have some point of attachment along the bands where it would fit without overlapping the neighboring band. The problem which remains is to show that none of the rectangles would overlap *each other*, since these rectangles are not rectangle-faced. This reduces to almost a type of matching argument; each rectangle has several possible attachment points, and we must find a selection so that no two overlap. It seems likely that Lemma 3 may prove useful here, since it provides a strong ordering on where the faces can be attached.

Extending this type of algorithm to non-rectangular orthostacks seems more difficult, since the notion of a good bridge would necessarily be more complex when z-faces are not simple rectangles. Choosing such a bridge would involve search for all possible ways that the z-faces could attach to the bands, and somehow finding the best (either "highest" or "widest") such bridge, as well as dealing with more complex overlap between z-faces when attached to the bands.

References

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

[2] T. Biedl, E. D. Demaine, M. L. Demaine, A. Lubiw, J. O'Rourke, M. Overmars, S. Robbins, and S. Whitesides. Unfolding some classes of orthogonal polyhedra. In *Proc. 10th Canad. Conf. Comput. Geom.*, pages 70–71, 1998.

[3] M. Damian, R. Flatland, H. Meijer, and J. O'Rourke. Unfolding well-separated orthotrees. In *15th Annu. Fall Workshop Comput. Geom.*, pages 23–25, Nov. 2005.

[4] M. Damian, R. Flatland, and J. O'Rourke. Grid vertex-unfolding orthogonal polyhedra. In *Proc. 23rd Sympos. Theoret. Aspects Comput. Sci.*, volume 3884 of *Lecture Notes Comput. Sci.*, pages 264–276. Springer-Verlag, 2006.

[5] M. Damian, R. Flatland, and J. O'Rourke. Epsilon-unfolding orthogonal polyhedra. *Graphs and Combinatorics*, 23[Suppl]:179–194, 2007. Akiyama-Chvátal Festschrift.

[6] E. Demaine, J. Iacono, and S. Langerman. Grid vertex-unfolding orthostacks. In J. Akiyama, M. Kano, and X. Tan, editors, *Discrete and Computational Geometry*, volume 3742 of *Lecture Notes in Computer Science*, pages 76–82. Springer Berlin / Heidelberg, 2005.

[7] E. D. Demaine, D. Eppstein, J. Erickson, G. W. Hart, and J. O'Rourke. Vertex-unfoldings of simplicial manifolds. In A. Bezdek, editor, *Discrete Geometry*, pages 215–228. Marcel Dekker, 2003.

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

[9] J. Itoh, J. O'Rourke, and C. Vîlcu. Star unfolding convex polyhedra via quasigeodesic loops. *Discrete Comput. Geom.*, 44:35–54, 2010.

[10] J. O'Rourke. Unfolding orthogonal terrains. Technical Report 084, Smith College, July 2007. arXiv:0707.0610v4 [cs.CG].

[11] J. O'Rourke. Unfolding orthogonal polyhedra. In J. Goodman, J. Pach, and R. Pollack, editors, *Proc. Snowbird Conference Discrete and Computational Geometry: Twenty Years Later*, pages 307–317. American Mathematical Society, 2008.

A Data Structure Supporting Exclusion Persistence Range Search

Stuart A. MacGillivray[*]   Bradford G. Nickerson[†]

Abstract

We present a new type of query on spatiotemporal data, termed "exclusion persistence". When bulk updates are made to stored spatial data, all previous versions remain accessible. A query returns the most recently observed data intersecting the query region, while permitting the exclusion of any subset of previous versions in areas where data overlaps. We propose several solutions to this new problem defined on a set of N points. For an axis-aligned rectangular exclusion persistence query, we give a 2-dimensional linear-space data structure that, after m updates, answers the query in $O(\sqrt{\frac{mN}{B}} + m^2 + \frac{K}{B})$ I/Os, where B is the number of points fitting in one disk block, and K is the number of points in range.

1 Introduction

In most large scale earth observation systems, data is gathered in short duration surveys of significantly sized regions. To maintain up-to-date representations of the covered area, successive surveys are performed. Surveys normally cover different areas, resulting in overlap as shown in Figure 1. These types of surveys usually observe a massive number of data points for each update. Queries are expected to return data intersecting the query region from the most recent survey where overlap occurs. The survey areas can be represented as overlapping polygonal regions, where the newest data in overlapping regions replaces older data.

When searching for up-to-date information, various search options are possible. If the data is obtained from multiple sources, we may wish to exclude all information obtained from an unreliable source, or restrict search only to those data sources we trust. A search that excludes all data in a specific time period is useful if uncorrectable errors are known to occur in data observed during that period. Lastly, we may simply wish to examine the region as it was in some period before the present time.

Given point data that maps to d-dimensional space, we consider search on N data points added over the course of m updates. Normally, $N \gg m$ and we can assume that $m < \log N$. This is a reasonable assumption under all known earth observation applications; for instance, LIDAR surveys can contain tens of millions of points per update [5].

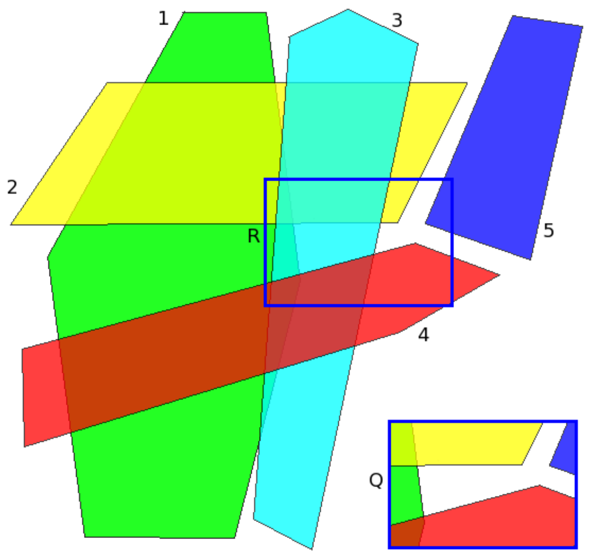


Figure 1: Overlapping updates and a query rectangle R, showing $Q = (R, 5, \{3\})$ where the query time $t_q = t_5$, the excluded points index set $T_e = \{3\}$, and $d = 2$. Areas of overlap have multiple distinct accessible versions.

2 Exclusion Persistence

These problems do not map cleanly to any existing model of persistence as described by Tarjan et al [4]. To resolve the issue, we define the idea of "exclusion persistence". If we assume that new data in a spatial region replaces older data in the same region, we can apply exclusion persistence to support the types of searches described above.

A data structure supporting exclusion persistence maintains updates independent of one another, and searches performed on the structure can omit any subset of past updates from consideration. Formally, we have m sets of d-dimensional data points $S_1 ... S_m$, contained in bounding regions $B_1 ... B_m$ and added to the structure at times $t_1 ... t_m$, respectively.

[*]Faculty of Computer Science, University of New Brunswick, t172q@unb.ca

[†]Faculty of Computer Science, University of New Brunswick, bgn@unb.ca

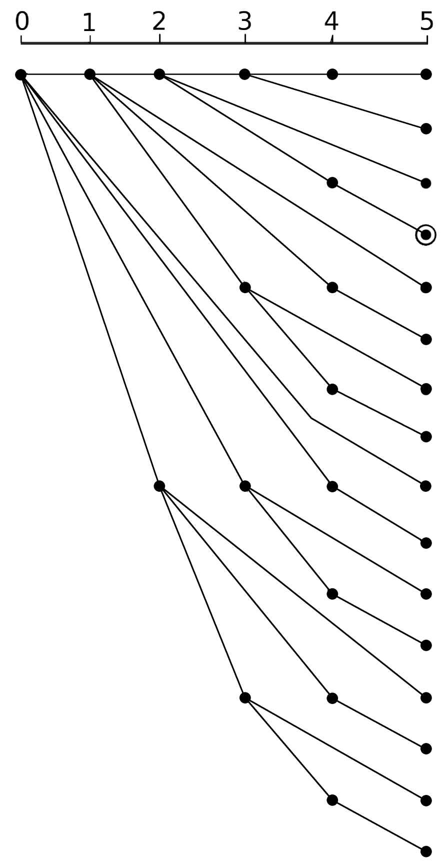


Figure 2: The 2^m possible versions in exclusion persistence with $m = 5$ updates, with the circled version matching the example query from Figure 1.

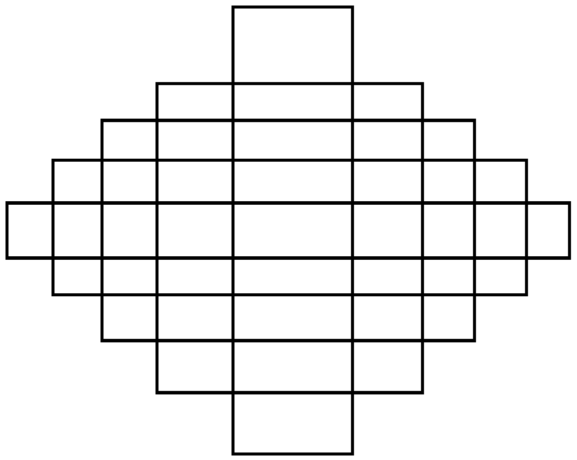


Figure 3: Subregions for rectangular regions grow quadratically in the worst case, as shown with $m = 5$.

We make the simplifying assumption that data is observed at a single time epoch, representing the time period the update covers. Given a convex query region R to search, and a set T_e of time epoch indices whose matching data sets are excluded from consideration, we define an exclusion persistence range search as follows.

An exclusion persistence range query is defined as $Q = (R, q, T_e)$, which asks to find all points intersecting R whose time epoch $t_i \le t_q$, and whose time epoch index $i \notin T_e$. The result of such a query is the union of data contained in queried regions C_i over all non-excluded updates i, i.e. $\bigcup_{\substack{i=1\\ i\notin T_e}}^{q} C_i \cap S_i$, where

$$C_i = (B_i \cap R) \setminus (B_i \cap (\bigcup_{\substack{j=i+1\\ j\notin T_e}}^{q} C_j).$$

This search will return all data in the valid sets added on or before t_q intersected by R, returning only the newest data (whose time epoch $t_i \le t_q$, and whose time epoch index $i \notin T_e$) in areas where bounding regions overlap. As any set may be excluded for a given query, however, older data sets are still accessible and must be maintained.

Informally, the problem can be summarized as follows. We have m updates, each of which is a spatial region B_i containing a set of data points S_i added to our structure at a time t_i. We assume that new data takes precedence over old data, meaning that in areas where multiple regions overlap, we only return data from the newest set. A structure not supporting any form of persistence could thus simply delete all data falling within a new region B_i before adding S_i to the structure. While partial persistence allows us to ignore all data newer than the query time, an exclusion persistence search has the ability to ignore any of the updates at the searcher's discretion.

While storing B points per disk block in the I/O model [7], is there a linear space data structure using $O((\frac{N}{B})^{\frac{d-1}{d}} + \frac{K}{B})$ I/Os to answer an exclusion persistence range search returning K points? If not, what tradeoffs are possible? Henceforth, we restrict the query region R to an axis-aligned rectangle.

3 Naïve Solutions

Two naïve solutions are possible. The first is to simply store each update completely independently, search all appropriate data sets independently, and remove excluded data after the fact. In such a scenario, efficiency can be obtained by storing each update in an optimal linear-space structure such as the Priority R-Tree [3]. While this means that only linear storage space is required overall, searches will be highly redundant. As we store no information about where overlap occurs, we return data from old updates even if they are completely covered by newer ones. In the worst case, each update region completely covers the previous, leaving us with a worst case of $O(m(\frac{N}{B})^{\frac{d-1}{d}} + \frac{mK}{B})$ search I/Os.

The second naïve approach is to use full persistence to store every possible combination of sets as a different version, and simply search the appropriate one. This gives the desired search I/Os, but to cover all possible combinations the storage requirement increases by a factor of 2^{m-1}. This is evident by virtue of the fact that m updates can be combined to produce 2^m possible versions, as shown in Figure 2, and that any given data point will be part of half those combinations. For N points, this requires $O(2^{m-1}\frac{N}{B})$ disk blocks of storage, as each data point is stored in 2^{m-1} versions.

4 Convex Regions Approach

Neither of the naïve solutions is acceptable; naïve linear storage has highly redundant searches, and full persistence is wasteful of space no matter what the value of m is. An improved solution is to store the updates or sets of points independently, and split data into subsets based on spatial overlap of the updates. We do this to avoid the problem of naïve linear storage, splitting old updates according to how newer ones cover them. We first consider the restricted case when all updates have a bounding region consisting of a convex polygon; our results are dependent on an upper bound f on the number of faces per polygon.

Let a 2-d structure solving this problem be defined as follows. In memory, we store an index of the m regions covering the m data sets. We also store, for each polygonal subregion created from the intersection of these m regions, a stack of pointers to data structures on disk. Each structure is guaranteed to contain data covering the subregion, meaning that only the topmost non-excluded structure of the stack must be searched. The structures on disk are any linear-space data structure supporting range search in $O((\frac{N}{B})^{\frac{1}{2}} + \frac{K}{B})$ I/Os (e.g. a Priority R-Tree [3]). A range search determines (in main memory) which sub-regions are intersected, follows the pointer from the stack to the most recent non-excluded data structure in each intersected sub-region, and performs an I/O-efficient range search independently on each structure.

As the sub-regions are distinct, there is no redundancy in this range search. The overall search cost is reasonable for small m; an exclusion range search requires $O((\frac{mN}{B})^{\frac{1}{2}} + m^2 + \frac{K}{B})$ I/Os in the worst case, as shown in Theorem 2. The m^2 term will not dominate unless $m > \sqrt[3]{\frac{N}{B}}$. Our initial proof in Theorem 2 requires that the regions be axis-aligned rectangles, but Theorem 6 extends the result to f-sided convex polygonal regions.

Lemma 1 *Given $x_i \in \mathbb{R}_{>0}$ such that $\sum_{i=1}^{m} x_i = N$, then $\sum_{i=1}^{m} \sqrt{x_i}$ has a maximum value of $\sqrt{mN}$ when $x_i = \frac{N}{m}\forall i$.*

Proof. Define a function $C = \sum_{i=1}^{m} \sqrt{x_i}$. As $x_i \in \mathbb{R}_{>0}$ and $\sum_{i=1}^{m} x_i = N$, this can be written as $C = \sum_{i=1}^{m-1} \sqrt{x_i} + \sqrt{N - \sum_{i=1}^{m-1} x_i}$. For all x_i where $1 \le i \le m-1$, we take the partial derivative $\frac{\partial C}{\partial x_i} = \frac{1}{2\sqrt{x_i}} - \frac{1}{2\sqrt{N - \sum_{i=1}^{m-1} x_i}}$. We find the critical values of C by setting each partial first derivative to zero, resulting in simultaneous equations $x_i = (N - \sum_{i=1}^{m-1} x_i)\forall i$. A unique solution exists where $x_i = \frac{N}{m}\forall i$. The second derivatives are $\frac{\partial^2 C}{\partial x_i \partial x_j} = -\frac{1}{4\sqrt{N - \sum_{i=1}^{m-1} x_i}^3}\forall i \neq j$, and $\frac{\partial^2 C}{\partial x_i^2} = \frac{-1}{4\sqrt{x_i}^3} - \frac{1}{4\sqrt{N - \sum_{i=1}^{m-1} x_i}^3}$. As these second derivatives are negative for all values of x_i, this critical value is a global maximum. In short, C is maximized when $x_i = \frac{N}{m}\forall i$, and has a value of $\sum_{i=1}^{m} \sqrt{\frac{N}{m}} = m\sqrt{\frac{N}{m}} = \sqrt{mN}$. □

Theorem 2 *Assuming we have N 2-dimensional data points from m updates, where each update is covered by an axis-aligned rectangle, there exists a data structure that can perform a rectangular exclusion persistence range search in $O((\frac{mN}{B})^{\frac{1}{2}} + m^2 + \frac{K}{B})$ I/Os.*

Proof. With m overlapping rectangles, the maximum number of intersections between their subregions and a horizontal or vertical line is $2m-1$; the number of subregions intersecting each side of the query rectangle R is, therefore, linear in m. Figure 3 shows that in the worst case the number of subregions is quadratic in m, meaning that at most $O(m^2)$ subregions can be covered by R. For each of these $O(m^2)$ subregions, we store a stack of pointers to linear space optimal I/O-efficient structures on disk storing data covering that subregion. At most one of these structures is searched for each subregion intersecting R. The geometric index structure describing each of the $O(m^2)$ subregions is assumed to be in main memory, so the I/O cost to find the intersected subregions is zero.

Our range search is an aggregate of range searches over the intersected subregions, with each subregion i containing x_i points. The subregions intersecting the sides require $O(\sqrt{\frac{x_i}{B}} + 1 + \lfloor\frac{K_i}{B}\rfloor)$ I/Os to return K_i points. Subregions entirely contained by the search require $O(1 + \lfloor\frac{K_i}{B}\rfloor)$ I/Os. The worst case occurs when all N points are distributed among the n_1 subregions intersected by the sides of R and none are in the n_2 subregions contained by R (see the illustration in Figure 4). This gives a total cost of $O(\sum_{i=1}^{n_1} \sqrt{\frac{x_i}{B}} + n_1 + n_2 + \frac{K}{B})$ I/Os to return K points, and $\sum_{i=1}^{n_1} x_i = N$. Lemma 1 shows that $\sum_{i=1}^{n_1} \sqrt{\frac{x_i}{B}}$ has a maximum value of $\sqrt{\frac{n_1 N}{B}}$, as in the worst case, $\sum_{i=1}^{n_1} x_i = N$. As n_1 is $O(m)$ and n_2 is $O(m^2)$, the total search cost is $O((\frac{mN}{B})^{\frac{1}{2}} + m^2 + \frac{K}{B})$ I/Os. □

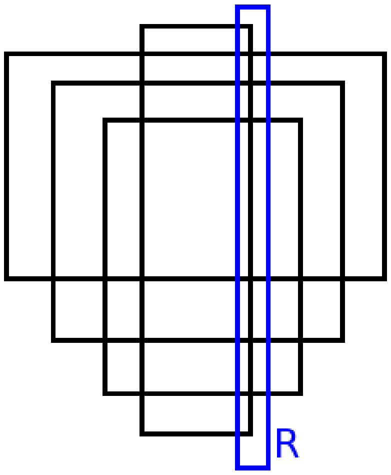


Figure 4: A worst case query on $m = 4$ subregions intersecting $O(m)$ subregions with the potential to be empty. Here, $n_1 = 12$ and $n_2 = 0$.

While the rectangular case is useful, we can generalize. We still require that the query region R be an axis-aligned rectangle, but the regions defining the data sets can be any convex polygon of at most f sides.

Theorem 3 *The intersection of m overlapping convex polygons with at most f sides will produce at most fm^2 subregions.*

Proof. By the definition of convexity, a line intersecting a convex polygon can intersect at most two of its sides. When adding a new f-sided polygon to a set of $i-1$ polygons, each side of the ith polygon can intersect at most two sides from each previous polygon. These intersections partition each new side into $2(i-1)+1 = 2i-1$ line segments. Each of these line segments can belong to at most two subregions, and the addition of the ith polygon can create at most one new subregion for each segment. The addition of the ith polygon to the set therefore creates at most $f(2i-1) = 2if - f$ subregions. A set of m polygons therefore has an upper bound of $\sum_{i=1}^{m} 2if - f = 2f(\frac{m(m+1)}{2}) - mf = fm^2 + mf - mf = fm^2$ subregions. □

Lemma 4 *A straight line passing through a set of m convex polygons can intersect at most $2m-1$ subregions.*

Proof. A straight line passing through a convex polygon begins intersecting that polygon at one point, and stops intersecting it at another. A straight line passing through a set of m polygons therefore has at most $2m$ intersections where it begins or stops passing through a polygon. The straight line intersects a new subregion if and only if one of those $2m$ intersections occurs, and the last such intersection denotes where the line stops intersecting any of the m polygons; as such, it can intersect at most $2m-1$ subregions. □

Lemma 5 *The sides of a rectangular query region R intersect $O(m)$ subregions of a set of m convex polygons.*

Proof. Lemma 4 shows that a straight line can intersect at most $2m-1$ subregions from a set of m convex polygons. Each side of R is a straight line, and as such the sides of R can only intersect at most $8m-4$ subregions. □

Theorem 6 *Assuming we have N 2-dimensional data points from m updates, where each update is covered by a convex region with at most f sides, there exists a data structure that can perform a rectangular exclusion persistence range search in $O((\frac{mN}{B})^{\frac{1}{2}} + fm^2 + \frac{K}{B})$ I/Os.*

Proof. Lemma 5 shows that $O(m)$ subregions will intersect the sides of the query region R, and Theorem 3 shows that $O(fm^2)$ subregions can be contained by a query. The proof of Theorem 2 therefore applies to the general case, giving the desired I/O bound. □

5 Algorithms

Our convex regions solution to exclusion persistence range search consists of a spatial partitioning of 2-dimensional space into subregions, with each subregion having a stack of pointers to spatial data structures on disk. Each pointer is given a time stamp denoting what update added its data. Space not covered by any update is treated as a subregion with an empty stack. Figure 5 illustrates the structure, demonstrating a simple insertion.

Algorithm 2 shows an implementation of exclusion persistence range search. The repeat-until loop finds the top non-excluded time epoch on the stack for each subregion. Lines 9-14 of Algorithm 2 are invoked at most once per subregion. We apply the entire query region R to each intersected subregion at line 13 for simplicity, as each structure only contains data within its subregion. The subregion shape could force a range search on highly clustered data, which is still linear space optimal. While our analysis of worst case range search requires axis-aligned rectangular query regions, Algorithm 2 can also be used with general polygonal queries. Algorithm 2 provides the correct exclusion persistence range search result by specifying T_e such that $i \in T_e \forall t_i > t_q$.

```
Algorithm 1: Insert(P,S,R,t,j)
Input : A convex regions data structure $P$, a set
        $S$ of new data points to be inserted, a
        convex shape $R$ that contains $S$, a time
        epoch $t$ associated with $S$, the index $j$ for
        time epoch $t$
Output: An updated convex regions data
        structure $P$ that includes $S$
1  begin
2      Use polygonal differences to find all subregions
       $P_i$ of $P$ that intersect $R$;
3      for each subregion $P_i \in P$ contained by $R$ do
4          Search $S$ for the points $S_i$ that fall within
           $P_i$;
5          Keeping $S_i$ in memory, delete the points in
           $S_i$ from $S$;
6          Bulk-load $S_i$ into a new linear-space data
           structure $D_i$ stored on disk;
7          Push a pointer to $D_i$ stamped with time
           $(t, j)$ onto the stack for $P_i$;
8      for each subregion $P_i \in P$ intersecting the edges
       of $R$ do
9          Update $P$ with new subregions $P_k$ formed
           by the intersections of $P_i$ and $R$;
10         for each pointer p in the stack for $P_i$, from
           the bottom to the top do
11             Follow the pointer $p$ to its data
               structure $D_i$, storing the time stamp
               $(t_i, \ell)$ in memory;
12             for each new subregion $P_k$ do
13                 Search $D_i$ for the set of points $S_k$
                   that fall within $P_k$; Bulk-load $S_k$
                   into a new linear-space data
                   structure $D_k$ stored on disk;
14                 Push a pointer to $D_k$ stamped with
                   time $(t_i, \ell)$ onto the stack for $P_k$;
15             Delete $D_i$;
16         for each new subregion $P_k$ do
17             Search $S$ for the points $S_k$ that fall
               within $P_k$;
18             Keeping $S_k$ in memory, delete the points
               in $S_k$ from $S$;
19             Bulk-load $S_k$ into a new linear-space
               data structure $D_k$ stored on disk;
20             Push a pointer to $D_k$ stamped with time
               $(t, j)$ onto the stack for $P_k$;
```

Our data structure has an update cost dependent on the data structures used for the subregions. Our analysis requires a linear-space spatial data structure supporting range search in $O(\sqrt{\frac{N}{B}} + \frac{K}{B})$ I/Os, such as the

```
Algorithm 2: Search(P,R,T_e)
Input : A convex regions data structure $P$, a
        query region $R$, a set $T_e$ of time epoch
        indices to be excluded
Output: A set $K$ of points found by the exclusion
        persistence query
1  begin
2      Use polygonal differences to find all subregions
       $P_i$ of $P$ that intersect $R$;
3      for each subregion $P_i \in P$ intersecting $R$ do
4          repeat
5              Pop the top pointer $p_k$ from the stack
               for $P_i$;
6              Let $(t_k, j)$ = the time stamp for $p_k$;
7          until $j \notin T_e$ or the stack for $P_i$ is empty;
8          if $j \notin T_e$ then
9              Follow $p_k$ to its data structure $D_k$;
10             if $P_i$ is contained by $R$ then
11                 Add all points in $D_k$ to $K$;
12             else
13                 $S_i$ = a range search on $D_k$ over $R$;
14                 Add the points in $S_i$ to $K$;
15         Push all popped pointers $p_k$ back onto the
           stack for $P_i$ with their respective time
           stamps $(t_k, j)$;
```

bkd-tree [6] or Priority R-Tree [3]. From Theorem 2.4 of [3] we know that such a structure can be bulk-loaded in $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ I/Os. This leads to the following theorem:

Theorem 7 *Using Algorithm 1, the total insertion cost of m updates into a convex regions data structure requires $O(\frac{mN}{B} \log_{M/B} \frac{N}{B})$ I/Os, where N is the total number of points after all updates.*

Proof. In the worst case, an update S that is inserted into a structure that previously contained $N - |S|$ points intersects subregions containing $O(N)$ of those points. Each of the intersected subregions must be split, requiring new data structures to be created. Loading the affected structures will require $O(N)$ I/Os. Structures must also be created for the $|S|$ points from the new update, for a total of j structures. Each new structure D_i, containing x_i points, can be bulk-loaded in $O(\frac{x_i}{B} \log_{M/B} \frac{x_i}{B})$ I/Os. As $\sum_{i=1}^{j} x_i \leq N$, creating j structures containing a total of $O(N)$ points requires $O(\frac{N}{B} \log_{M/B} \frac{N}{B})$ I/Os, which dominates the cost. □

Figure 5 illustrates the result of the insertion process described in Algorithm 1. A new update is added to the set, intersecting one previously existing subregion and the uncovered space. This results in data from the first update being repartitioned, and part of that data covered by a portion of the new update. Some of the resulting subregions are non-convex, but the previously proven search bounds apply regardless.

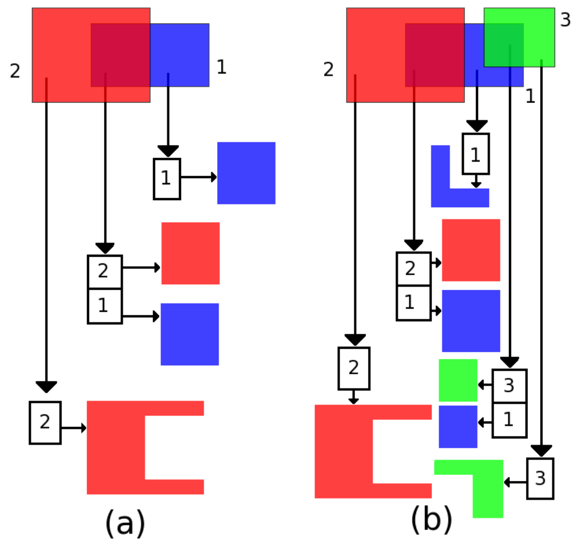


Figure 5: Subregions arising from $m = 2$ and $m = 3$ intersecting regions, with the data stored in each version of each subregion.

6 Conclusion

We have presented the exclusion persistence problem in spatiotemporal queries, along with a 2-dimensional solution. For a set of N data points collected over a series of m updates, where each update is bounded by a convex region of at most f sides, our linear space solution requires $O((\frac{mN}{B})^{\frac{1}{2}} + fm^2 + \frac{K}{B})$ I/Os in the worst case to perform an exclusion persistence range search, with K points reported in range. While relaxing the space requirement on the subregion structures could improve search cost as shown by Afshani et al [1][2], this would lead to the overall storage requirement becoming non-linear. Experimental validation of the data structure remains to be done.

Several interesting open problems remain. Is a linear space data structure supporting exclusion persistence range search in $O((\frac{N}{B})^{\frac{1}{2}} + \frac{K}{B})$ I/Os possible? What worst case search complexity (in the I/O model) is possible for general convex query regions R in place of rectangles? Is there a non-trivial linear space data structure storing d-dimensional points that can efficiently answer exclusion persistence search queries? What I/O search complexity arises when data updates are described by non-convex boundaries?

References

[1] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting in three and higher dimensions. In *FOCS*, pages 149–158. IEEE Computer Society, 2009.

[2] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting: query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In J. Snoeyink, M. de Berg, J. S. B. Mitchell, G. Rote, and M. Teillaud, editors, *Symposium on Computational Geometry*, pages 240–246. ACM, 2010.

[3] L. Arge, M. de Berg, H. Haverkort, and K. Yi. The priority R-tree: A practically efficient and worst-case optimal R-tree. *ACM Trans. Algorithms*, 4(1):1–30, 2008.

[4] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. E. Tarjan. Making data structures persistent. *J. Comput. Syst. Sci.*, 38(1):86–124, 1989.

[5] D. Latypov. Estimating relative lidar accuracy information from overlapping flight lines. *ISPRS Journal of Photogrammetry and Remote Sensing*, 56(4):236 – 245, 2002.

[6] O. Procopiuc, P. Agarwal, L. Arge, and J. Vitter. Bkd-tree: A dynamic scalable kd-tree. In T. Hadzilacos, Y. Manolopoulos, J. Roddick, and Y. Theodoridis, editors, *Advances in Spatial and Temporal Databases*, volume 2750 of *Lecture Notes in Computer Science*, pages 46–65. Springer Berlin / Heidelberg, 2003.

[7] J. S. Vitter. External memory algorithms and data structures. *ACM Comput. Surv.*, 33(2):209–271, 2001.

xy-Monotone Path Existence Queries in a Rectilinear Environment[*]

Gregory Bint[†] Anil Maheshwari[†] Michiel Smid[†]

Abstract

Given a planar environment consisting of n disjoint axis-aligned rectangles, we want to query on any two points and find whether there is a north-east monotone path between them. We present preprocessing and query algorithms which translate the geometric problem into a tree traversal problem and present a corresponding tree structure that gives us $O(n \log n)$ construction time, $O(n)$ space, and $O(\log n)$ query time.

1 Introduction

Consider a closed planar environment which consists of n disjoint axis-aligned rectangular obstacles. We want to query this environment on any two points s and t and determine whether a rectilinear north-east monotone path exists between them. In this paper, we mean rectilinear in the sense that all segments of a path are axis-aligned and that adjacent segments of a path meet at right angles. By north-east monotone, we mean that each path segment extends either above or to the right of the end of the previous segment (Figure 1).

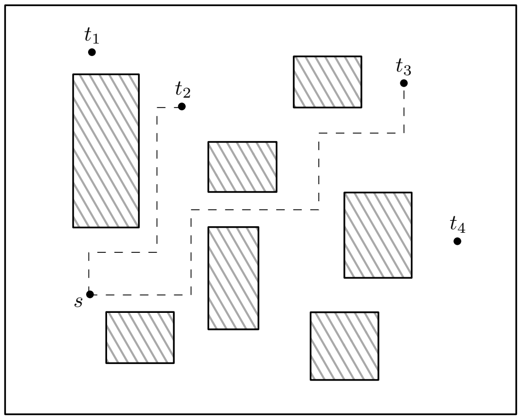


Figure 1: An environment with 7 obstacles. From s, there are north-east monotone paths to t_2 and t_3, but not to t_1 or t_4.

This work is inspired by a single point query algorithm for finding shortest paths in a similar environment by Rezende, Lee, and Wu [7]. Given a fixed point s, they preprocess the environment to allow for queries on any t. Our main contribution is to allow both s and t to be specified at query time, extracting only the information about north-east monotone path existence. We summarize our work with the following theorem.

Theorem 1 *Given a planar environment consisting of n disjoint axis-aligned rectangular obstacles, we can construct, in $O(n \log n)$ time, a data structure with size $O(n)$ with which we can query for the existence of a north-east monotone path between any two query points in $O(\log n)$ time.*

The remainder of this paper details a data structure and query method to satisfy this theorem. Section 2 covers some preliminary terminology and path construction techniques. Section 3 contains the main contribution of this work, showing how we can precalculate so-called *shared paths* and use them to answer monotone path existence queries. As our overall query requires the ability to perform a planar point location query, Section 4 reviews a particularly suitable method. Section 5 brings together the complete query algorithm with some discussion on possible extensions. Finally, we conclude in Section 6.

2 Preliminaries

In this section, we will review some construction techniques and lemmas presented in Rezende, Lee, and Wu. The proofs appear in their original paper.

Definition 1 *Let a path π be defined by a sequence of points $p_1, p_2, ...p_k$, then π is an xy-*path *if, for every adjacent pair of points $p_i, p_{(i+1)}$, either $p_{(i+1)}$ is directly above p_i (i.e. ${p_{(i+1)}}_x = p_{i_x}$ and ${p_{(i+1)}}_y > p_{i_y}$) or $p_{(i+1)}$ is directly to the right of p_i (i.e. ${p_{(i+1)}}_x > p_{i_x}$ and ${p_{(i+1)}}_y = p_{i_y}$). Following similar rules, we can define $x(-y)$-paths, $(-x)y$-paths, and $(-x)(-y)$-paths. Observe that any such path is rectilinear.*

Definition 2 *A rectilinear north-east monotone path is an xy-path.*

If we assume that no two adjacent segments of a path are co-linear, we observe that any vertical (horizontal)

[*]This research was supported in part by the NSERC-USRA program

[†]School of Computer Science, Carleton University {gbint, anil, michiel}@scs.carleton.ca

line can intersect such a path through at most a single point of one horizontal (vertical) segment or lie colinearly with at most one vertical (horizontal) segment.

When constructing xy-paths, we can *prefer* a direction of path extension by always travelling in a particular direction unless we need to route around an obstacle.

Definition 3 *An* x-preferred xy-path*,* π*, is an* xy*-path which extends east, in the* $+x$*, direction whenever possible. Let* o *be an obstacle with sides* $left(o)$*,* $top(o)$*,* $right(o)$ *and* $bottom(o)$*. If* π *encounters* $left(o)$*, a vertical segment is added which continues north (in the* $+y$ *direction) to the vertex located at the incidence of* $left(o)$ *and* $top(o)$*. From there, a horizontal segment is added, which resumes travelling east along* $top(o)$*, and beyond, until the next obstacle is encountered (see Figure 2).*

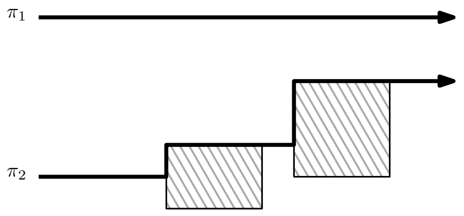


Figure 2: Two x-preferred xy-paths, one without, one with obstacles.

In a similar way, π may be a y-preferred xy-path, which travels north whenever possible, only travelling east when moving around an obstacle. $x(-y)$-paths, $(-x)y$-paths, and $(-x)(-y)$-paths can also be constructed with a preference for either of their two component directions. For example, an $x(-y)$-path can be x-preferred or $(-y)$-preferred.

Definition 4 *Given a point* s*, if we extend both an* x*-preferred* xy*-path and a* $(-x)$*-preferred* $(-x)y$*-path from* s*, the area above the union of these paths is called the* y-region *of* s*. In a similar fashion, the* x-region of s *is the area to the right of the union of the* y*-preferred* xy*-path and the* $(-y)$*-preferred* $x(-y)$*-path rooted at* s*.*

Definition 5 *The* xy-region of s *is the intersection of the* x*-region of* s *and the* y*-region of* s *(Figure 3).*

Using these definitions, Rezende, Lee, and Wu give the following lemma which is of particular interest.

Lemma 2 *There is a rectilinear north-east monotone path from* s *to* t *if and only if* t *lies within the* xy*-region of* s*.*

From their lemma, and from the construction of the xy-region of s, we derive the following additional lemma.

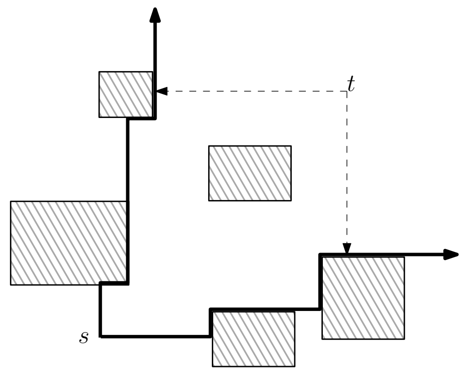


Figure 3: The complete xy-region of s with point t contained within it.

Lemma 3 *If* t *is in the* xy*-region of* s*, then, disregarding all obstacles, a vertical line through* t *must intersect the* x*-preferred* xy*-path rooted at* s *somewhere below* t*, and a horizontal line through* t *must intersect the* y*-preferred* xy*-path rooted at* s *somewhere left of* t*.*

Rezende, Lee, and Wu's algorithm constructs the xy-region of s, which is then further refined into a rectangular subdivision. When a query point t is given, they perform a point location on t within the rectangular subdivision. Assuming t lies within the xy-region of s, their subdivision stores enough information to allow them to construct a rectilinear north-east monotone path between the two points. By Lemma 2, we know that such a path is possible. Such a path is also a shortest path, which was the goal of their work, however we are only interested in the existence of it.

3 Shared Paths

Having seen how xy-paths are constructed around obstacles, we now turn our attention to how multiple paths will route around the same obstacle.

Lemma 4 *Given an* x*-preferred* xy*-path* π *which meets* $left(o)$ *for some obstacle* o*, any other* x*-preferred* xy*-path* π' *which meets* $left(o)$ *will have the same structure as* π *from the corner of* $left(o)$ *and* $top(o)$ *and beyond.*

The proof is apparent directly from construction. It should be clear that this lemma holds for other $(\pm x)(\pm y)$-paths, with either component direction as the preferred direction.

3.1 Shared Path Tree

We can use these *shared paths* to aid us in identifying the xy-region of any s during query time. As the construction of x-regions and y-regions are similar, we will consider only the y-region case. To do so, we will pre-calculate paths from the top-left vertex of each obstacle. Then, during query time, all that remains is to find the first obstacle that an x-preferred xy-path from s would hit. From that obstacle, we can follow the pre-calculated xy-path from its top-left vertex. Specifically, these shared paths will be stored in a tree, which is constructed in the following way.

Imagine a bounding box that contains all of the obstacles and all of the valid query range for s and t. Along the right-hand side of this box is a vertical line segment obstacle, labeled o_0, whose top-left vertex is v_0. All x-preferred xy-paths must eventually meet this obstacle.

We sort the remaining obstacles from right to left, top to bottom, according to their top-left vertices so that we have a sequence of obstacles $o_1, ..., o_n$ with corresponding top-left vertices $v_1, ..., v_n$. Notice that o_n is the leftmost obstacle, and that v_1 is the rightmost vertex to the left of v_0.

Our tree, $\mathcal{T}$, will store $v_0, ..., v_n$, with v_0 at the root. Then, processing $v_1, ..., v_n$ in order starting with v_1, we process each v_i in the following way. Let $seg(v_i)$ be a horizontal line segment starting at v_i, traveling east in the $+x$-direction, and let o_j be the obstacle impacted on the left side by $seg(v_i)$. Note that $j < i$. From o_j, we obtain a pointer to v_j, which must already exist in $\mathcal{T}$. We insert v_i into $\mathcal{T}$ as a child of v_j. Notice that $seg(v_i)$ is the line segment $(v_{i_x}, v_{i_y}) - (v_{j_x}, v_{i_y})$ and that $v_{i_y} \leq v_{j_y}$. As a result of the insertion method, any path of vertices in the tree will be ordered with respect to their x components. See Figure 4 for an example environment and corresponding $\mathcal{T}$.

Reconstructing an xy-path based on $\mathcal{T}$ is simple: given a pointer to a particular vertex v_{p_1} in the tree, if $v_{p_1}, v_{p_2}, ..., v_{p_k}$ is the sequence of vertices in the path from v_{p_1} to the root of the tree, then $seg(v_{p_1}), seg(v_{p_2}), ..., seg(v_{p_k})$ is the sequence of horizontal line segments that make up the xy-path of v_{p_1}[1].

Construction of $\mathcal{T}$ takes $O(n \log n)$ time. We first sort the vertices in the order given above. Next, for each of the n obstacles, we maintain a line segment intersection sweepline to find, in $O(\log n)$ time, the obstacle which will be hit by a horizontal line segment leaving the top-left vertex. Insertion into $\mathcal{T}$ takes only $O(1)$ as we acquire the parent pointer directly from the sweepline structure. $\mathcal{T}$ has size $O(n)$ as each top-left obstacle vertex is inserted exactly once.

3.2 Querying the Shared Path Tree

From Lemma 3, we see that it is sufficient to show that t is in the y-region of s by testing that t is above the x-preferred xy-path rooted at s. Here, we refer only to that portion of the y-region which is to the east of s, since no other part of the y-region can contribute to the xy-region of s. To that end, we will further assume that t is also east of s.

The first step towards identifying the y-region of s is to identify the obstacle which a horizontal ray leaving s in the $+x$ direction would hit. We label that obstacle as o_s. We can use a point location data structure to find this obstacle and return the pointer v_s, corresponding to an entry in $\mathcal{T}$. Section 4 explores this in more detail, but for now it suffices to assume that we can acquire v_s.

With v_s, we know that the first segment of the x-preferred xy-path rooted at s is the horizontal segment defined by $(s_x, s_y) - (v_{s_x}, s_y)$. The remaining segments of the path are already stored in the tree, and so a simple query method would be as follows.

Assume that t is north-east of s, otherwise the query result is 'no'. Let $v(t)$ be the vertical line through t. If $v(t)_x$ is within the x-interval of the first horizontal line segment, then we test and return whether t is above it and we are done.

Otherwise, let $v_{s_1}, v_{s_2}, ..., v_{s_k}$ be the path through $\mathcal{T}$ where $v_{s_1} = v_s$ and v_{s_k} is the root of $\mathcal{T}$. We test $v(t)$ against each $seg(v_{s_i})$ in order from 1 to k. If $v(t)_x$ is not within the x-interval of a line segment, we advance to the next segment by following the parent pointer of v_{s_i}. If it is, we test and return whether t is above that segment, and we are done. Since we construct our environment such that the root node of $\mathcal{T}$ is farther right than any other input, there must be some segment which $v(t)$ intersects. See Figure 4 for an example.

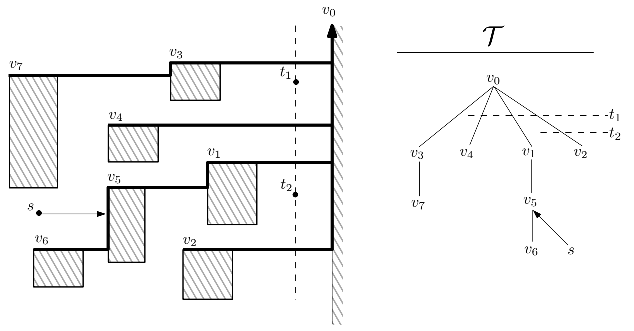


Figure 4: An environment and corresponding shared path tree showing how s, t_1, and t_2 interact. Notice that there is a north-east monotone path from s to t_1, but not from s to t_2.

Performance of this query method is dependent on the height of $\mathcal{T}$. If we consider the case where all obstacles are arranged in an ascending staircase such that a path starting with the leftmost obstacle hits every remaining obstacle on its way east, we see that $\mathcal{T}$ has a height and query time of $O(n)$, which is not sufficient for our theorem.

[1]We assume that the vertical segments of an x-preferred xy-path have no width, and as a result, any vertical line through such a path must impact some horizontal segment. As we will see, we are only interested in performing vertical line tests on these paths, so we can disregard the vertical path segments.

3.3 Augmenting the Shared Path Tree

In order to achieve $O(\log n)$ query time on $\mathcal{T}$, we will need to augment it. The augmentation we will use is a method first discussed by Cole and Vishkin [3]. This method was later illustrated by Narasimhan and Smid [6] in a manner very similar to our own use.

In brief, the vertices in the tree are processed into *groups* which have the property that we can follow a path from any vertex to the root by looking at only $O(\log n)$ groups.

The augmentation adds the following information to $\mathcal{T}$. For every vertex v, define m to be the number of vertices in the subtree of v. We define *l-value(v)* to be $\lfloor \log m \rfloor$. We define a *group* to be a path of vertices that share the same *l-value*. The head of this group is the vertex closest to the root and is called the *group parent* for all vertices in the group, a pointer to which is stored at every v as *gpar(v)*.

From the definitions given, observe the following properties about groups. Every leaf will have an *l-value* of 0 and will belong to a group consisting only of itself, as its parent's subtree size and thus its parent's *l-value* must be ≥ 2 and ≥ 1, respectively. We also see that the root of the tree will have an *l-value* of $\lfloor \log n \rfloor$. An example is given in Figure 5.

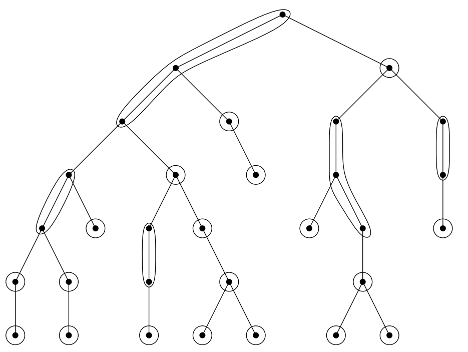


Figure 5: A binary tree with grouped nodes indicated.

Note that *l-values* can only increase as we consider vertices closer to the root, and that groups must be paths. Thus, we can traverse from any v to the root of the tree by following $O(\log n)$ *gpar* pointers.

With the groups configured, we will further augment $\mathcal{T}$ at each group parent by creating an ordered *group array* containing all the vertices of its corresponding group. This array permits us to perform a binary search on the group while examining the group parent. By copying the children pointers in the appropriate order, according to their orientation along the group path, we can create this array in $O(n)$ time over all groups.

The group number and group parent pointers of each vertex, and the group arrays associated with each group parent, can be calculated with a simple post-order traversal of the tree in $O(n)$ time and requires only $O(1)$ extra space per vertex. Note that since each vertex appears in exactly one group array, the total size of all group arrays is $O(n)$.

3.4 Querying the Augmented Shared Path Tree

Querying the augmented tree has the same goal as in Section 3.2: to identify the eastern y-region of s and determine if t is within it.

The initial parts of the query are performed identically. Again we assume that we have a point location data structure that allows us to identify the first obstacle to the right of s, labeled o_s, and which gives us the corresponding pointer v_s into $\mathcal{T}$. From this, we can define and test the first horizontal line segment of the x-preferred xy-path at s.

Recall that $v(t)$ is the vertical line through t. If $v(t)$ does not intersect that first horizontal segment, then we follow v_s into $\mathcal{T}$. We immediately jump to the group parent of v_s, labeled $gpar(v_s)$. If $v(t)$ is to the left of (or at) $gpar(v_s)$, then we perform a binary search on the array of vertices stored there, each of which correspond to a horizontal line segment in the environment. $v(t)$ must intersect one of these segments, and we test and return whether $v(t)$ is above that segment.

If $v(t)$ is to the right of $gpar(v_s)$, then we follow $gpar(v_s)$'s parent pointer, which brings us to some vertex in the next group towards the root of $\mathcal{T}$ and repeat the same procedure, jumping to the group parent, testing the array contents there, and so on. Since we construct our environment such that the root vertex of $\mathcal{T}$ must be farther right than any other input, there must be a group such that $v(t)$ intersects one of its constituent vertices.

In performing this query, we need to test at most $O(\log n)$ group parent pointers. In one group, we will also need to perform a binary search on the array stored there, for a total query time of $O(\log n)$, as required by our theorem.

4 Finding s

Before we can use $\mathcal{T}$ to identify the y-region of s, we need to identify the first obstacle that an x-preferred xy-path leaving s will impact.

4.1 Planar Subdivision

Step 1 is to create a horizontal subdivision of the environment into planar rectangles. Conceptually, this is accomplished by extending a horizontal ray from each obstacle vertex away from the obstacle until it strikes another obstacle or the environment boundary. Every such ray bisects the space it travels through into two regions resulting in a subdivision of size $O(n)$ (Figure

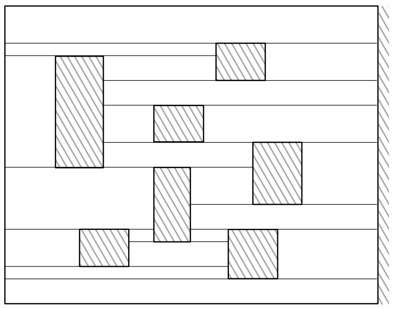


Figure 6: example of a planar rectangular subdivision of a set of obstacles in enclosing environment. note the imaginary obstacle along the right-hand side.

6). We call each non-obstacle face of the subdivision a *cell*.

Lemma 5 *The right-hand boundary of a cell is defined by a single obstacle.*

The subdivision can be constructed using the same horizontal sweepline used to build $\mathcal{T}$. With each cell, we store the appropriate pointer into $\mathcal{T}$ based on the obstacle that the cell sees to its right.

4.2 Point Location Query

To find the cell that a particular point falls into, we will use a *Skewer Tree* [4], a data structure by Edelsbrunner, Haring, and Hilbert specifically for point location in a collection of axis-aligned, non-overlapping rectangles.

Construction of a skewer tree follows a divide and conquer approach. For a set, S, of rectangles, we place a vertical line l through the median x-value. We divide S into three subsets: S_1 is the set of rectangles that lie strictly to the left of l, S_2 is the set of rectangles intersected by l, and S_3 is the set of rectangles that lie strictly to the right of l, so that $|S_1| + |S_2| + |S_3| = |S|$. We create a node n_S in the skewer tree which contains the definition of l, the size of S_2, and a balanced tree containing the rectangles of S_2, sorted by y-value.

If S_1 is non-empty, we recurse on S_1, attaching the resulting subtree as the left child of n_S. Similarly, if S_3 is non-empty, we recurse on S_3, attaching the resulting subtree as the right child of n_S.

Every rectangle appears exactly once in the skewer tree, as it is attributed only to the first node whose l intersected it and not passed down to deeper levels of recursion. The skewer tree requires $O(n)$ space and $O(n \log n)$ construction time.

The query time is a bit more interesting. We start at the root node and check if our query point s is contained within one of the rectangles stored there, which takes $O(\log n)$ time. If it is not, we compare l with s_x and decide whether we will next check the left or right subtree. We repeat these steps at every level of the tree until the rectangle containing s is found. The height of the skewer tree is $O(\log n)$, so we need to make at most that many queries for a total query time of $O(\log^2 n)$ time.

We can improve the query time to $O(\log n)$ with *Fractional Cascading* [1, 2, 5, 8]. To help illustrate how, we will refer to the outer nodes of the Skewer tree as the *line tree*, and the trees of rectangles attached to each node of the line tree as *rectangle trees*.

Every path through the line tree represents a sequence of arrays of sorted values. Each array is queried over the same range of keys, defined by the interval of y-values covered by the bounding box of the environment.

Considering a single path through the line tree for now, we will store extra pointers in each of the rectangle trees so that a query on one tree can return not just the successor to the search key value in that tree, but in the next tree in the path as well. If this mechanism is implemented in every rectangle tree, then we can answer our query in every tree of the path by performing a standard binary tree search on the root rectangle tree in $O(\log n)$ time, and then continuing by walking along $O(\log n)$ pointers through the path, each taking $O(1)$ time.

Because the line tree is a binary tree, we need to store two sets of additional pointers in each rectangle tree: one to use if we follow a line tree node's left child, and one to use if we follow its right child. See Chazelle and Guibas [1, 2] for details on how these extra pointers can be developed in linear time.

5 The Complete Algorithm

We now have all the tools we need to solve our query problem.

5.1 Construction

The construction phase of the algorithm involves building both the shared path data structure and a point location data structure.

When discussing shared path data structures, we have primarily considered obstacles to the east of our query point, and the resulting x-preferred xy-paths. We will relabel that data structure as $\mathcal{T}_E$. We also need to consider obstacles to the north and the associated y-preferred xy-paths, which we do by creating a second data structure labeled $\mathcal{T}_N$. The algorithms as written in Section 3.1 for $\mathcal{T}_E$ can easily be adapted for $\mathcal{T}_N$.

While performing the plane-sweeps needed to construct $\mathcal{T}_E$ and $\mathcal{T}_N$, we can also produce the rectangular planar subdivisions used by the skewer trees.

Total construction time and space for $\mathcal{T}_E$, $\mathcal{T}_N$, and the skewer trees, is $O(n \log n)$ and $O(n)$, respectively, as required by our theorem.

5.2 Query

When a query is made, we are given s and t. Assume that t is actually north-east of s, otherwise the query result is 'no'.

We first perform a point location query on s, identifying the obstacle to its right. We then follow the procedure in Section 3.4 to determine whether t is in the y-region of s.

Using the obstacle above s, we identify the x-region of s, and again follow the procedure in Section 3.4 to determine if it contains t. If both return true, then we know that t is in the xy-region of s, and so by Lemma 2, there is a north-east monotone path from s to t. The total query time is $O(\log n)$, as required by our theorem.

6 Conclusion

In this paper we have discussed a method for deciding whether there exists a north-east monotone path between any two query points in the plane. We developed the concept of shared paths and showed a method for storing them in a tree, and for augmenting that tree to allow for quick query time. We also reviewed a suitable point location query method. Together, these methods require $O(n \log n)$ preprocessing time, $O(n)$ space, and $O(\log n)$ query time.

One problem which remains open is the following. In the event that a path is found to be possible, we would like to report one such path in $O(\log n + k)$ time, where k is the number of segments in the reported path, and is within a constant factor of the minimum number of segments among all such paths.

References

[1] B. Chazelle and L. Guibas. Fractional cascading: I. a data structuring technique. *Algorithmica*, 1:133–162, 1986.

[2] B. Chazelle and L. Guibas. Fractional cascading: II. applications. *Algorithmica*, 1:163–191, 1986.

[3] R. Cole and U. Vishkin. The accelerated centroid decomposition technique for optimal parallel tree evaluation in logarithmic time. *Algorithmica*, 3:329–346, 1988.

[4] H. Edelsbrunner, G. Haring, and D. Hilbert. Rectangular point location in d dimensions with applications. *The Computer Journal*, 29(1):76–82, 1986.

[5] K. Mehlhorn and S. Nher. Dynamic fractional cascading. *Algorithmica*, 5:215–241, 1990.

[6] G. Narasimhan and M. Smid. *Geometric Spanner Networks*. Cambridge University Press, New York, NY, USA, 2007.

[7] P. Rezende, D. Lee, and Y. Wu. Rectilinear shortest paths in the presence of rectangular barriers. *Discrete & Computational Geometry*, 4:41–53, 1989.

[8] M. Smid. Rectangular point location and the dynamic closest pair problem. In W.-L. Hsu and R. Lee, editors, *ISA'91 Algorithms*, volume 557 of *Lecture Notes in Computer Science*, pages 364–374. Springer Berlin / Heidelberg, 1991.

Covering Points with Disjoint Unit Disks

Greg Aloupis* Robert A. Hearn† Hirokazu Iwasawa‡ Ryuhei Uehara§

Abstract

We consider the following problem. How many points must be placed in the plane so that no collection of disjoint unit disks can cover them? The answer, k, is already known to satisfy $11 \le k \le 53$. Here, we improve the lower bound to 13 and the upper bound to 50. We also provide a set of 45 points that apparently cannot be covered, although this has been determined via computer search.

1 Introduction

In 2008, Japanese puzzle designer Naoki Inaba proposed and solved an interesting question [3, 4], which was to determine if every given configuration of 10 points can be covered by identical coins. Any number of coins can be used, but they cannot overlap. That is, Inaba proved the following lower bound.

Theorem 1 **[Inaba]** *Any configuration of 10 points in the plane can be covered by disjoint unit disks.*

Inaba gave an interesting proof based on the probabilistic method (see [6, 8]), and asked the natural extension: *How many points do we need to use, so that their appropriate arrangement cannot be covered by disjoint unit disks?*

Let k be the size of the smallest point set that is not coverable. Inaba's theorem states that $11 \le k$, and trivially k is finite; if we place sufficiently many points on a fine lattice, disjoint disks cannot cover them all (see Figure 1). This problem gained popularity within the puzzle society in 2010 (at the 9th *Gathering 4 Gardner*). Winkler [7] proposed a configuration of 60 points that cannot be covered by disjoint disks. Winkler also suggested how to improve the lower bound in [8], but this has not been settled[1]. Elser [1] improved the upper bound to 55, and Okayama et. al [6] further improved this to 53.

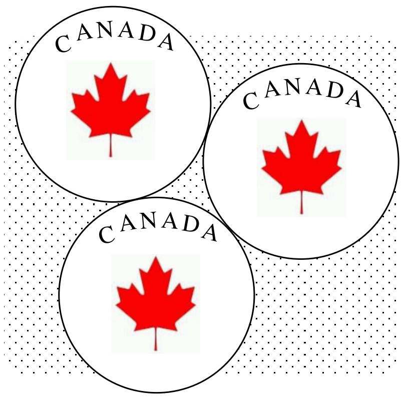


Figure 1: A dense point set that cannot be covered by disjoint unit disks.

In this paper, we improve the known bounds as follows.

Theorem 2 *Let k be the size of the smallest point set that is not coverable by disjoint unit disks. Then $13 \le k \le 45$.*

That is, we improve the lower bound from 11 to 13, and the upper bound from 53 to 45.

For the lower bound, we give a refinement of Inaba's proof based on the probabilistic method. For the upper bound, we have used two different approaches. First we give a configuration of 50 points on a lattice, for which an analytical proof exists. This is an improvement over the solution in [6], from which we remove three points. We also state a better upper bound of 45. The validity of this configuration has been determined via an exhaustive computer search, however we note that a mathematically rigorous proof remains to be shown. Finally we mention that it is NP-complete to decide if a given set of n points can be covered [2].

2 Preliminaries

We say that a disk C is *placed at* (x, y) if its center is placed at the point (x, y). This is sometimes denoted by $C(x, y)$. To simplify our arguments, let each unit disk be open, so that it does not cover points on its boundary. Using a perturbation technique, our results can be applied to closed disks as well. We denote by $|A|$ the area of a bounded region A in the plane. A

*Départment d'Informatique, Université Libre de Bruxelles, aloupis.greg@gmail.com
†H3 Labs, Portland, OR 97205, USA. bob@hearn.to
‡Kobo Iwahiro, iwahiro@bb.mbn.or.jp
§School of Information Science, Japan Advanced Institute of Science and Technology, Ishikawa 923-1292, Japan. uehara@jaist.ac.jp
[1]Personal communication with Peter Winkler.

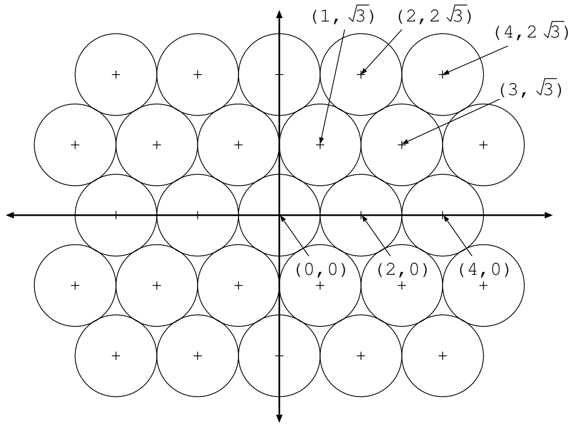


Figure 2: An infinite configuration $S(0,0)$ of unit disks.

region S in the plane is *periodic* if there is a bounded region $A \subset \mathbb{R}^2$ such that for any vector $(x,y) \in \mathbb{R}^2$ there is a vector $(a,b) \in A$ with $S+(x,y) = S+(a,b)$. A measurable minimum-area set A with this property is then said to be a *fundamental region* for S. The *density* $\rho(S)$ of S is defined by

$$\rho(S) = \frac{|S \cap A|}{|A|}$$

which is independent of the choice of A, if A is a fundamental region for S.

3 Lower bound

In this section, we show that $13 \le k$. That is, any set of 12 points can be covered by disjoint unit disks. Let S be the union of unit disks placed at $(2i+(j \bmod 2), j\sqrt{3})$, for $i,j \in \mathbb{Z}$. (See Figure 2.) S is then periodic. Its fundamental set is the regular hexagon H with vertices at $(\pm 1, \pm\sqrt{3}/3)$ and $(0, \pm 2\sqrt{3}/3)$. Its density is thus $\rho(S) = |S \cap H|/|H| = \pi/\sqrt{12} \sim 0.9069$. We will show that any 12 points in the plane are covered by $S+(x,y)$ for some $(x,y) \in H$.

Inaba's proof for 10 points notes that if (x,y) is chosen uniformly from H then any fixed point in the plane is covered by $S+(x,y)$ with probability $\rho(S) > 9/10$. Thus, of any 10 points, the expected number covered by $S+(x,y)$ is greater than 9; it follows that with positive probability $S+(x,y)$ will cover all ten points. Of course, only at most 10 of the disks in $S+(x,y)$ are actually needed to cover the points.

We refine Inaba's method to show:

Lemma 3 *Any configuration of 12 points can be covered by $S+(x,y)$, for appropriate values of x and y.*

Proof. (Outline) We denote by $\overline{S+(x,y)}$ the set of points that are not covered by $S+(x,y)$. Since $(a,b) \in S+(x,y)$ if and only if $(x,y) \in S-(a,b)$, we have:

Observation 1 *Let X be a set of m points $p_1 = (x_1,y_1), p_2 = (x_2,y_2), \ldots, p_m = (x_m,y_m)$. Then the following statements are equivalent.*

1. *For all $(x,y) \in H$, $S+(x,y)$ fails to cover X.*
2. *$\cup_{i=1,\ldots,m}\overline{S-(x_i,y_i)}$ covers the plane;*
3. *$\cup_{i=1,\ldots,m}\overline{S-(x_i,y_i)}$ covers H.*

For $x \in [-1,1]$, let $P(x)$ be the vertical line segment consisting of all points in H of x-coordinate x. Then the ratio $\phi(x) = \left|\overline{S+(x,y)} \cap P(x)\right|/|P(x)|$ (where the set-measure is now ordinary one-dimensional length) is given by

$$\phi(x) = \left(\sqrt{3} - \sqrt{1-x^2} - \sqrt{2|x|-x^2}\right)/\sqrt{3}.$$

Thus, by Observation 1, to prove Lemma 3 it is sufficient to establish that

$$\min\{\phi(x)+\phi(x-d_1)+\phi(x-d_2)+\cdots+\phi(x-d_{11})\} < 1$$

for any real numbers $d_1, d_2, \ldots, d_{11}$. Let $\psi(x; d_1, d_2, \ldots, d_{11}) = \phi(x) + \phi(x-d_1) + \phi(x-d_2) + \cdots + \phi(x-d_{11})$. We will prove that $\max\min\{\psi(x; d_1, \ldots, d_{11}) \mid 0 \le x \le 1\}$ is given when $d_i = i/12$ for $i = 1, \ldots, 11$, and then $\max\min\{\psi(x; d_1, \ldots, d_{11}) \mid 0 \le x \le 1\} \approx 0.942809 < 1$. We omit the details of this calculation, which will be given in the electronic proceedings. □

4 Upper bounds

In this section, we state two upper bounds, by providing configurations of point sets that cannot be covered. Our first set is simply a subset of 50 points taken from the pattern in [6]. In fact this configuration is constrained to a triangular lattice, which permits a concise proof. The second set contains only 45 points and was checked by a computer program that is based on a non-trivial exhaustive search. Although disk placement is not a finite process, we explain how this problem can be solved in a discrete way.

4.1 50-point configuration on a triangular lattice

The configuration is given in Figure 3. This is based on a triangular lattice; the smallest equilateral triangle is on a circle of radius $2\sqrt{3}/3 - 1$. Unless mentioned otherwise, when we use the term "triangle", we will be referring to three points that are mutual neighbors on the lattice, i.e., forming the equilateral triangle mentioned. The radius $2\sqrt{3}/3 - 1$ was chosen to be the largest value satisfying the following property:

Lemma 4 ([6]) *The three points forming a triangle cannot be covered by three disjoint unit disks.*

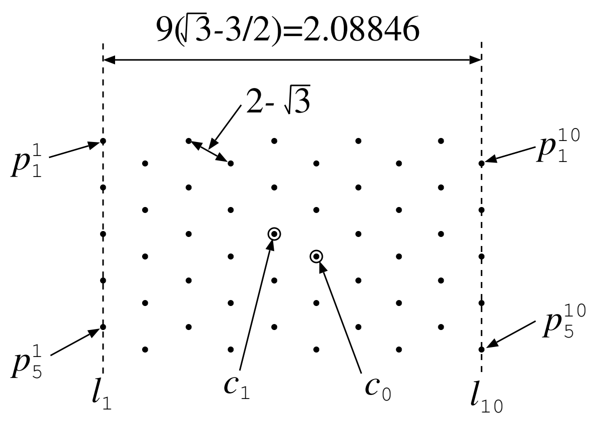


Figure 3: A 50-point configuration that cannot be covered by disjoint unit disks.

Thus, if our set of 50 points is to be covered by disjoint unit disks, each triangle must be covered by either one or two disks. We say that a disk *partially covers* a triangle if it only covers one or two of its points. A *chain* is a sequence of triangles $t_1, t_2, \ldots, t_h$, with consecutive triangles sharing a common edge.

Lemma 5 *Suppose that the chain $t_1, t_2, \ldots, t_h$ is covered, and every disk involved only covers triangles partially. Let C and C' be two disks that cover t_1. Then the entire chain is covered only by C and C'.*

Proof. By Lemma 4, without loss of generality assume that two points p_1, p_2 of t_1 are covered by C, and the remaining point p_3 is covered by C'. Consider the first two triangles ($h = 2$). By definition, t_1 and t_2 share two points. It is not possible for t_2 to share p_1 and p_2, since C cannot cover both points without covering either p_3 or the third point of t_2, which would contradict the assumption about partial coverage by every disk. Thus we can assume that the two triangles share p_1 and p_3, which as mentioned are covered by C and C' respectively. By Lemma 4, the third point of t_2 must be covered by C or C'. Our claim follows by iterating through adjacent pairs of triangles in the chain. Every such pair must share two points that are not covered by the same disk. $\square$

Now we are ready to show the upper bound:

Theorem 6 *The 50-point configuration in Figure 3 cannot be covered by disjoint unit disks.*

Proof. In the configuration, there are 10 vertical columns, each containing 5 points. The columns are labeled ℓ_1 to ℓ_{10} from left to right. Let $p_1^j, p_2^j, p_3^j, p_4^j, p_5^j$ be the five points on ℓ_j from top to bottom. Notice that by rotating a half-turn, the same configuration is obtained.

For the sake of contradiction, suppose that the configuration is covered. Let the *centers* of the configuration be the points $c_0 = p_3^6$ and $c_1 = p_3^5$, as shown in Figure 3. Let C_0 be a unit disk that covers c_0 or c_1. (Choose arbitrarily, if the two centers are covered by two different disks.) It is easy to see that C_0 cannot cover points both in ℓ_1 and in ℓ_{10} since the distance between ℓ_1 and ℓ_{10} is around 2.08846. Without loss of generality, assume that no point in ℓ_1 is covered by C_0. More precisely, suppose that the first r columns are not covered by C_0. Then we have $1 \leq r \leq 5$.

Suppose that p_1^{r+1} and p_5^{r+1} are not covered by C_0. In other words, C_0 covers all points $p_{k_1}^{r+1}, \ldots, p_{k_2}^{r+1}$ for some $2 \leq k_1 \leq k_2 \leq 4$. Then, by Lemma 5, $p_{k_1-1}^{r+1}$ and $p_{k_2+1}^{r+1}$ must be covered by one disk, since a suitable chain connecting the two always exists. However, by convexity, this is impossible.

Therefore, C_0 covers p_1^{r+1} or p_5^{r+1}. We first consider the case that C_0 covers p_5^{r+1}. We distinguish between two subcases, depending on the parity of r as shown in Figure 4 or Figure 5.

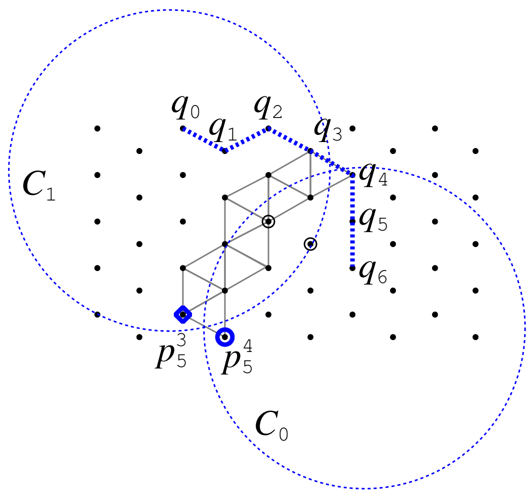


Figure 4: Contradiction for $r = 3$.

Subcase $r = 1, 3, 5$**:** We consider a polyline $L = (q_0, q_1, q_2, q_3, q_4, q_5, q_6)$ defined by $q_0 = p_1^r$, $q_1 = p_1^{r+1}$, $q_2 = p_1^{r+2}$, $q_3 = p_1^{r+3}$, $q_4 = p_2^{r+4}$, $q_5 = p_3^{r+4}$, and $q_6 = p_4^{r+4}$. Figure 4 illustrates the case for $r = 3$. Recall that C_0 covers p_5^{r+1} and at least one of the two centers c_0 and c_1, and does not cover any point in ℓ_r. Given these restrictions, we claim that the boundary of C_0 must cross L. It suffices to show that some vertex of L is contained in C_0, since q_0 is not. For $r = 1$, q_6 lies on the segment joining p_5^{r+1} and c_0. Also, $q_5 = c_1$. Therefore regardless of which center is in C_0, a vertex of L is also in C_0. For $r = 3$ and $r = 5$, C_0 must cover even more of L. Specifically, C_0 cannot reach to cover c_0 or c_1 while containing p_5^{r+1} and excluding q_6.

Let z be the smallest index such that q_z is contained in

C_0. Given the convexity of C_0 as well, we can determine that there must exist a chain of triangles such that every triangle is partially covered by C_0. Furthermore this chain starts with a triangle that has p_5^r, p_5^{r+1} as an edge, and ends with a triangle that has q_{z-1}, q_z as an edge. By Lemma 5, q_{z-1} is covered by the same disk as p_5^r.

An example of a chain is illustrated in Figure 4. Each triangle contains either one or two points covered by C_0. For any given placement of C_0, the chain is easy to determine. Regardless of the value of z, the coverage requirement is geometrically impossible. For example, in Figure 4, to achieve the smallest overlap, C_1 passes through p_5^3 and q_3, and C_0 passes through p_5^4 and q_4 (precisely, C_0 and C_1 are closer since they are open disks). In the case, the distance between the centers of C_0 and C_1 is $\sqrt{3.57198} = 1.88997 < 2$. (Letting $p_5^3 = (0,0), p_5^4 = (2-\sqrt{3}, 0), q_3 = ((2-\sqrt{3})/2, 5(\sqrt{3}-3/2))$, and $q_4 = (3(2-\sqrt{3})/2, 5(\sqrt{3}-3/2))$, we solve $x_1^2+y_1^2 = 1$ and $(x_1 - ((2-\sqrt{3})/2))^2 + (y_1 - (5(\sqrt{3}-3/2)))^2 = 1$ for $C_1(x_1, y_1)$, and $(x_0 - (2-\sqrt{3}))^2 + y_0^2 = 1$ and $(x_0 - (3(2-\sqrt{3})/2))^2 + (y_0 - (5(\sqrt{3}-3/2)))^2 = 1$ for $C_0(x_0, y_0)$. Then we have $(x_0, y_0, x_1, y_1) = (1.14135, 0.487011, -0.739421, 0.673243)$ and $(x_1 - x_0)^2 + (y_1 - y_0)^2 = 3.57198$.) Therefore, C_0 and C_1 overlap in this case. All cases are summarized in Table 1. In each case, the distance is less than 2, and hence the disks C_0 and C_1 overlap.

Case	$q_0 \in C_1$, $q_1 \in C_0$	$q_1 \in C_1$, $q_2 \in C_0$	$q_2 \in C_1$, $q_3 \in C_0$
Distance	1.92528	1.89161	1.88996
Case	$q_3 \in C_1$, $q_4 \in C_0$	$q_4 \in C_1$, $q_5 \in C_0$	$q_5 \in C_1$, $q_6 \in C_0$
Distance	1.88997	1.89160	1.88588

Table 1: Distance in each case (r:odd)

Subcase $r = 2, 4$: In this case, we just change the definition of the polyline $L = (q_0, q_1, q_2, q_3, q_4, q_5)$ as shown in Figure 5 (for $r = 2$). The distances between the two disk centers are summarized in Table 2. In each case, the distance is less than 2. Thus the disks C_0 and C_1 overlap.

Case	$q_0 \in C_1$, $q_1 \in C_0$	$q_1 \in C_1$, $q_2 \in C_0$	$q_2 \in C_1$, $q_3 \in C_0$
Distance	1.92528	1.89161	1.90467
Case	$q_3 \in C_1$, $q_4 \in C_0$	$q_4 \in C_1$, $q_5 \in C_0$	
Distance	1.86166	1.81971	

Table 2: Distance in each case (r:even)

The last case is that C_0 covers p_1^{r+1} (We also know

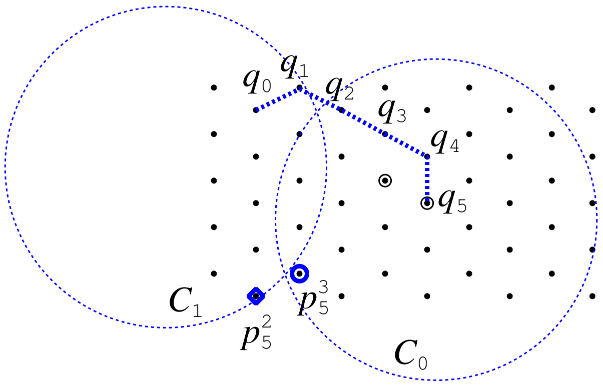


Figure 5: Contradiction for $r = 2$.

that it does not cover p_5^{r+1}, although this does not affect our analysis.) In this case we flip L as shown in Figure 6. We also change how we handle the parity of r, since p_1^{r+1} is on the convex hull if and only if p_5^{r+1} is not.

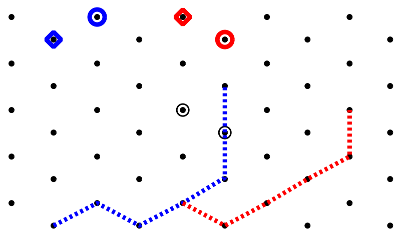


Figure 6: Polylines for $r = 2$ and $r = 5$.

We conclude that in all cases it is impossible to cover all points in Figure 3 with disjoint unit disks. □

4.2 45-point configuration

The configuration given in Figure 7 consists of 45 points equally spaced on three concentric circles: 3 points on the circle of radius 0.1, 21 points on the circle of radius 0.721, and 21 points on the circle of radius 1.0001. By computer search, we have determined that this set can-

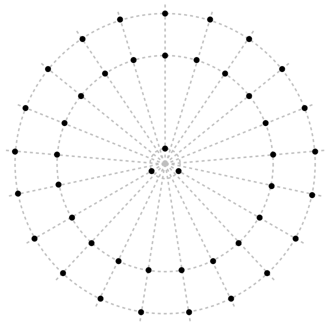


Figure 7: A 45-point configuration that cannot be covered by disjoint unit disks.

not be covered. This set was found heuristically, using the search program interactively.[2]

The search program exhaustively considers all possible ways of covering the given points with disks. This is not obviously a discrete combinatorial search problem — there are an infinite number of possible disk placements. Therefore, we describe here how the search algorithm works.

The algorithm considers (in principle) all possible ways of assigning points to disks that must cover them. This is a discrete search. For each such partition of the points, the algorithm proves conservative bounds for the location of the center of each disk, represented as a rectangle that must contain the disk center. For example, when a disk C is assigned its first point, its rectangle is set to a square of size 2 centered on the point. As more points are assigned to C, the rectangle can be shrunk: no rectangle edge can be farther than 1 from any point p that must be covered. Similar rectangle restrictions may be applied based on the requirement that the disks do not overlap. Sometimes a rectangle may be shrunk to nonexistence, ruling out the current point assignment. This pruning makes the search over all point partitions tractable; most partitions are ruled out without ever being considered explicitly.

If a point assignment survives this first stage of analysis, we are left with a "candidate solution": an assignment of points to disks, and for each disk, a corresponding rectangle. The problem then is to find a solution within this space, or prove that none exists.[3] To do this, we subdivide the largest rectangle, and recursively consider each candidate solution. Eventually, all rectangles are shrunk to the point where either a solution is easy to find (by testing, for example, the rectangle centers), or we can prove impossibility, via the same geometric rectangle restrictions used in the initial part of the search. For example, if we have two disk rectangles that can together be contained in a circle of radius < 2, we can rule out this candidate, because any disk placement respecting these bounds will have overlapping disks (see Figure 8).

Finally, we must mention numerical issues. Our program uses IEEE double-precision floating point numbers. We must ensure that roundoff problems do not cause us to miss a solution. The program uses an adjustable numerical tolerance ϵ for all of its geometrical restrictions — all operations are performed conservatively to this tolerance. (For example, if two points are $< 2 + \epsilon$ apart, the program will not rule out the possibility of coverage by a single disk.) This means that in principle, the program could be unable to either find a solution or prove that none exists. However, this has not been a problem for the configurations we have searched. IEEE floating point is accurate to 15 decimal places, and we have set $\epsilon = 10^{-5}$, giving us a high confidence that our results are correct.

[2]This general family of configurations was suggested by Bram Cohen.

[3]At this stage, the problem could also be treated as a quadratically constrained quadratic program, for which solvers exist (e.g., [5]). Our solution is optimized for this particular application.

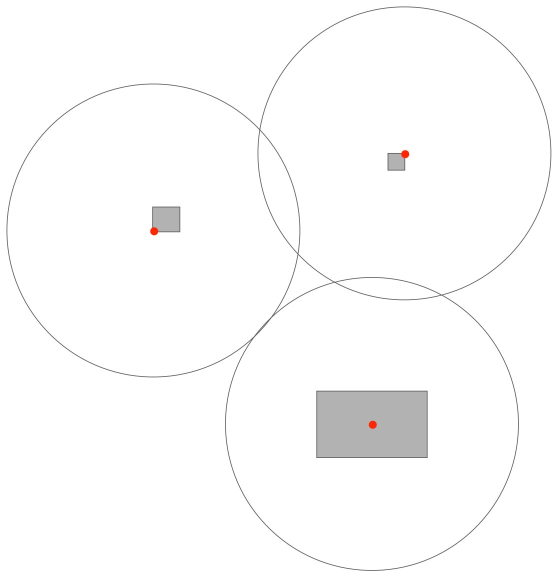


Figure 8: A candidate solution that can be ruled out: any placement will have overlapping disks.

5 Concluding remarks

We provide lower and upper bounds for the size k of the smallest point set that cannot be covered by disjoint unit disks. Our conclusion is that $13 \leq k \leq 45$. We conjecture that the true value lies closer to 45. For the lower bound, we have restricted to considering only the fixed configuration in Figure 2 and its translation. By considering rotations and other arrangements of unit disks, the bound might be improved. Moreover, since the bound is (essentially) obtained via the probabilistic method, it is not likely to be tight. We are currently working on a concise mathematical proof for the configuration of 45 points in Figure 7. Small perturbations are not likely to yield improvements. For instance, if the second radius is reduced to 0.720 from 0.721, our program finds a covering. Also, our program has determined that removing any points from the 50-point configuration always yields a covering.

Acknowledgments

The authors are grateful to Peter Winkler for fruitful discussions on this topic.

References

[1] V. Elser. Packing-constrained point coverings. Geombinatorics, to appear.

[2] R. Hearn. Complexity of Inaba's Coin-Covering Problem. Manuscript in preparation, 2012.

[3] N. Inaba. `http://inabapuzzle.com/hirameki/suuri_4.html`. (in Japanese), 2008.

[4] N. Inaba. `http://inabapuzzle.com/hirameki/suuri_ans4.html`. (in Japanese), 2008.

[5] N. Lamba, M. Dietz, D. P. Johnson, and M. S. Boddy. A method for global optimization of large systems of quadratic constraints. In *Proceedings of the Second international conference on Global Optimization and Constraint Satisfaction*, COCOS'03, pages 61–70, Berlin, Heidelberg, 2005. Springer-Verlag.

[6] Y. Okayama, M. Kiyomi, and R. Uehara. On covering of any point configuration by disjoint unit disks. In *23rd Canadian Conference on Computational Geometry (CCCG)*, pages 393–397, 2011. (Accepted to Geombinatorics).

[7] P. Winkler. Puzzled: Figures on a Plane. *Communications of the ACM*, 53(8):128, August 2010.

[8] P. Winkler. Puzzled: Solutions and Sources. *Communications of the ACM*, 53(9):110, September 2010.

The Approximability and Integrality Gap of Interval Stabbing and Independence Problems

Shalev Ben-David[*] Elyot Grant[†] Will Ma[‡] Malcolm Sharpe[§]

Abstract

Motivated by problems such as rectangle stabbing in the plane, we study the *minimum hitting set* and *maximum independent set* problems for families of *d-intervals* and *d-union-intervals*. We obtain the following: (1) constructions yielding asymptotically tight lower bounds on the integrality gaps of the associated natural linear programming relaxations; (2) an LP-relative d-approximation for the hitting set problem on d-intervals; (3) a proof that the approximation ratios for independent set on families of 2-intervals and 2-union-intervals can be improved to match tight duality gap lower bounds obtained via topological arguments, if one has access to an oracle for a PPAD-complete problem related to finding Borsuk-Ulam fixed-points.

1 Introduction

In this work, we examine a family of NP-hard packing and covering problems. Our study is motivated by the *minimum rectangle stabbing* problem, in which we are given a family $\mathcal{H}$ of axis-aligned rectangles in the plane, and the goal is to find a minimum-cardinality family of horizontal and vertical lines that intersect (or 'stab') each rectangle in $\mathcal{H}$ at least once. Viewing this as a geometric covering problem, we also consider the related 'dual' geometric packing problem of finding a *maximum conflict-free subset*, where the goal is to find a maximum subset of $\mathcal{H}$ containing no pair of rectangles that can be stabbed by a single horizontal or vertical line.

The rectangle stabbing and conflict-free subset problems have many applications. The rectangles themselves can be the bounding boxes of arbitrary connected objects in the plane, so applications need not be limited to problems involving rectangles. The rectangle stabbing problem can directly encode the problem of optimally subdividing the plane into a grid of axis-aligned cells so as to separate a given family of points, with applications to fault-tolerant sensor networks [3] and resource allocation in parallel processing systems [7]. The maximum conflict-free subset problem, and its higher dimensional analogues, are relevant to areas such as resource allocation, scheduling, and computational biology [2]. The properties of certain rectangle stabbing instances are also of theoretical interest in combinatorics [19].

Given a family $\mathcal{H}$ of axis-aligned rectangles, we write $\rho(\mathcal{H})$ for the minimum cardinality of a family of lines stabbing it, and $\alpha(\mathcal{H})$ for the maximum size of a conflict-free subset. It is clear that $\alpha(\mathcal{H}) \leq \rho(\mathcal{H})$. As in many geometric packing-covering dual problems, there is a bound in the other direction. In 1994, Tardos proved that $\rho(\mathcal{H}) \leq 2\alpha(\mathcal{H})$, which is easily seen to be tight [18]. However, all known proofs of Tardos's result rely on topological fixed-point theorems, and consequently do not seem to lead to polynomial-time approximations. In fact, only a 4-approximation is known for the maximum conflict-free subset problem [2], despite the fact that we can establish the optimal objective value to within a factor of 2 by solving a linear program. Improving upon this remains an important open problem.

In this paper, we obtain results for generalized versions of the stabbing and conflict-free subset problems. We examine the standard linear programming relaxations for hitting set and independent set problems involving *d-intervals*, and establish asymptotically tight upper and lower bounds on their integrality gaps. These bounds imply that no LP-relative approximation algorithm can obtain a factor below 2 for either the rectangle stabbing or the maximum conflict-free subset problems. Additionally, we establish some interesting theoretical consequences of topological methods such as Tardos's. For example, we show that the maximum conflict-free subset problem admits a 2-approximation if one has access to an oracle for a PPAD-complete problem.

This article proceeds as follows: in the current section, we define the generalized problems that we study, explain the current state of the art, and describe our contribution. Section 2 contains our integrality gap upper and lower bounds, and Section 3 contains algorithmic results that depend on PPAD oracles.

1.1 Preliminaries

We begin by defining generalized versions of the rectangle stabbing and conflict-free subset problems. For $d \in \mathbb{N}$, a *d-interval* I is a union of d non-empty com-

[*]Comp. Sci. and A.I. Lab, MIT, shalev@mit.edu

[†]Comp. Sci. and A.I. Lab, MIT, elyot@mit.edu

[‡]Operations Research Center, MIT, willma@mit.edu Supported in part by ONR Grant N000141110056.

[§]Department of Combinatorics and Optimization, University of Waterloo, sharpe.malcolm@gmail.com pact intervals $I^1, \dots, I^d \subset \mathbb{R}$. The input to all problems we consider will be a finite collection $\mathcal{H}$ of d-intervals, represented explicitly. We say that a subset of $\mathcal{H}$ is *independent* if its members are pairwise disjoint, and a set $X \subseteq \mathbb{R}$ is a *hitting set* for $\mathcal{H}$ if it intersects every member of $\mathcal{H}$. We define the hypergraph $G_{\mathcal{H}} = (V, E)$ where $V = \mathcal{H}$ and E consists of all subsets $\mathcal{I} \subseteq \mathcal{H}$ such that there is a point $p \in \mathbb{R}$ that hits *exactly* the intervals in $\mathcal{I}$. Such a point p shall be called a *representative* of the hyperedge $\mathcal{I} \in E$, and we let $P(\mathcal{H})$ denote a set containing an arbitrary representative for each distinct edge in $G_{\mathcal{H}}$. Note that $|P(\mathcal{H})| \leq 2d|\mathcal{H}|$ as there are at most $2d|\mathcal{H}|$ interval endpoints. We call $G_{\mathcal{H}}$ a *d-interval hypergraph*, and observe that d-interval hypergraphs generalize d-regular hypergraphs (which are obtained when each d-interval in $\mathcal{H}$ is simply d points). We denote by $\alpha(\mathcal{H})$ the maximum size of an independent set in $\mathcal{H}$, and denote by $\rho(\mathcal{H})$ the minimum size of a hitting set for $\mathcal{H}$, in analogy with the usual notation of $\alpha(G)$ and $\rho(G)$ for the maximum independent set size and minimum edge cover size of a hypergraph G.

Special cases such as the rectangle stabbing problem arise when we impose structural restrictions on $\mathcal{H}$. If $\{J^i\}_{i=1}^d$ is a family of disjoint intervals and each d-interval $I = \cup_{i=1}^d I^i$ in $\mathcal{H}$ satisfies $I^i \subseteq J^i$ for all i, then $\mathcal{H}$ is known as a collection of *d-union-intervals*, and $G_{\mathcal{H}}$ is known as a *d-union hypergraph.* The term *d-track-interval* is sometimes used for the same concept, with the idea that each d-interval contains a piece from one of d different 'tracks', each of which is a disjoint copy of $\mathbb{R}$. The rectangle stabbing and minimum conflict-free subset problems correspond precisely to the minimum hitting set and maximum independent set problems on 2-union-intervals, but with each 'track' mapped onto a separate Euclidean dimension. In general, one can think of the hitting set problem for d-union-intervals as the problem of hitting a family of d-dimensional 'boxes' using a minimum number of 'walls', each of which is orthogonal to one of the coordinate axes.

For 1-interval hypergraphs (which are the same as 1-union hypergraphs), the independent set and edge cover problems can both be solved in polynomial time via simple greedy algorithms that perform a left-to-right sweep across the intervals. However, even for 2-union hypergraphs, the independent set and edge cover problems are both APX-hard. Nagashima and Yamazaki, and independently Bar-Yehuda et al., have shown the conflict-free subset problem to be APX-hard [2, 14], even when the rectangles are all unit squares with integer vertices. Kovaleva and Spieksma show that the rectangle stabbing problem is APX-hard even when each rectangle is of the form $[x, x+1] \times [y, y]$ for integers x and y [11].

For a hypergraph G, the relations $\alpha(G) \leq \rho(G)$ and $\rho(G) \leq O(\log|V|) \cdot \alpha(G)$ are well known, with the latter being tight for general hypergraphs. However, using methods of topological combinatorics, Kaiser proves that $\frac{\rho(G_{\mathcal{H}})}{\alpha(G_{\mathcal{H}})}$ is upper bounded by $d^2 - d + 1$ for d-interval hypergraphs and $d^2 - d$ for d-union hypergraphs (for $d \geq 2$), a bound independent of $|V|$ [10]. His result improves upon that of Tardos, who originally established a tight upper bound for the $d = 2$ case [18]. In a one-page paper, Alon shows that an upper bound of $2d^2$ can be established without topological methods by applying Turán's theorem [1]. The best known lower bounds for large d are $\Omega(\frac{d^2}{\log d})$ and $\Omega(\frac{d^2}{\log^2 d})$ for d-interval and d-union hypergraphs respectively [13].

1.2 Overview of Results

We use the term *duality gap* to denote the quantity $\sup_{\mathcal{H}} \frac{\rho(\mathcal{H})}{\alpha(\mathcal{H})}$, where $\mathcal{H}$ ranges over a collection of d-intervals. In an effort to study various duality gaps, we examine standard linear programming relaxations for the hitting set and independent set problems. The standard LP relaxation for the maximum independent set problem corresponds to the *maximum fractional independent set problem*, and can be written as follows:

$$\begin{aligned} \max \quad & \sum_{I \in \mathcal{I}} x_I \\ \text{s.t.} \quad & \sum_{I \ni p} x_I \leq 1 \quad \forall\, p \in P(\mathcal{H}) \\ & x_I \geq 0 \qquad\quad \forall\, I \in \mathcal{H} \end{aligned} \tag{1}$$

A corresponding dual linear program for the *minimum fractional hitting set problem* is as follows:

$$\begin{aligned} \min \quad & \sum_{p \in P(\mathcal{H})} y_p \\ \text{s.t.} \quad & \sum_{p \in I} y_p \geq 1 \quad \forall\, I \in \mathcal{H} \\ & y_p \geq 0 \qquad\quad \forall\, p \in P(\mathcal{H}) \end{aligned} \tag{2}$$

If $\alpha^*(\mathcal{H})$ is the optimal objective value for (1) and $\rho^*(\mathcal{H})$ is the optimal objective value for (2), then we have $\alpha(\mathcal{H}) \leq \alpha^*(\mathcal{H}) = \rho^*(\mathcal{H}) \leq \rho(\mathcal{H})$ and can write

$$\sup_{\mathcal{H}} \frac{\rho(\mathcal{H})}{\alpha(\mathcal{H})} \leq \sup_{\mathcal{H}} \frac{\rho(\mathcal{H})}{\rho^*(\mathcal{H})} \cdot \sup_{\mathcal{H}} \frac{\alpha^*(\mathcal{H})}{\alpha(\mathcal{H})}.$$

The quantity $\sup_{\mathcal{H}} \frac{\rho(\mathcal{H})}{\rho^*(\mathcal{H})}$ is called the *integrality gap* of the minimum hitting set problem (for d-intervals or d-union-intervals). Similarly, $\sup_{\mathcal{H}} \frac{\alpha^*(\mathcal{H})}{\alpha(\mathcal{H})}$ is the integrality gap of the maximum independent set problem. Since $\frac{\rho(\mathcal{H})}{\rho^*(\mathcal{H})}$ and $\frac{\alpha^*(\mathcal{H})}{\alpha(\mathcal{H})}$ are always at least 1, both integrality gaps are a lower bound on the duality gap. For the case of 1-intervals, we actually have $\alpha(\mathcal{H}) = \rho(\mathcal{H})$; both linear programs have an integrality gap of 1 because the incidence matrix of $G_{\mathcal{H}}$ exhibits the consecutive ones property and is thus totally unimodular.

Often, upper bounds on integrality gaps for packing and covering problems come alongside LP-relative approximation algorithms. Bar-Yehuda et al. employ the *local ratio* technique to obtain a polynomial-time LP-relative $2d$-approximation algorithm for the d-interval maximum independent set problem, proving that the integrality gap of maximum independent set for d-intervals is at most $2d$. Their result carries over to the version in which each element in $\mathcal{H}$ has a positive weight and a maximum weight independent set is desired. In Section 2, we show that their bound is tight up to an additive constant by establishing the following:

Theorem 1 *For any $\epsilon > 0$, there exists a collection $\mathcal{H}$ of d-intervals (respectively, d-union-intervals) for which $\frac{\alpha^*(\mathcal{H})}{\alpha(\mathcal{H})} \geq 2d - 1 - \epsilon$ (respectively, $2d - 2 - \epsilon$).*

Our constructions generalize examples from [5] and [8] but employ a novel amplification trick.

For both the d-interval and d-union-interval hitting set problems, we are able to prove that the integrality gap is exactly d. We show the following:

Theorem 2 *There exists a polynomial-time LP-relative d-approximation for the d-interval hitting set problem. Accordingly, for any collection $\mathcal{H}$ of d-intervals, $\frac{\rho(\mathcal{H})}{\rho^*(\mathcal{H})} \leq d$.*

Theorem 3 *For any $\epsilon > 0$, there exists a collection $\mathcal{H}$ of d-union-intervals for which $\frac{\rho(\mathcal{H})}{\rho*(\mathcal{H})} \geq d - \epsilon$.*

Theorem 2 uses standard techniques to generalize a 2-approximation algorithm for rectangle stabbing due to [7], but Theorem 3 employs a novel construction.

The table below summarizes the known integrality and duality gap bounds for large d:

d-Interval	Lower Bound	Upper Bound
Duality Gap	$\Omega(\frac{d^2}{\log d})$ [13]	$d^2 - d + 1$ [10]
Max-IS Integ. Gap	$2d - 1$	$2d$ [2]
Min-HS Integ. Gap	d	d
d-Union		
Duality Gap	$\Omega(\frac{d^2}{\log^2 d})$ [13]	$d^2 - d$ [10]
Max-IS Integ. Gap	$2d - 2$	$2d$ [2]
Min-HS Integ. Gap	d	d

We note that for $d = 2$, Kaiser's topology-based duality gap upper bounds of $d^2 - d + 1$ and $d^2 - d$ match our independent set integrality gap lower bounds of $2d - 1$ and $2d - 2$, but are tighter than the constructive integrality gap upper bounds of $2d$ due to Bar-Yehuda et al. Hence, despite knowing that $\frac{\rho(\mathcal{H})}{\alpha(\mathcal{H})}$ is bounded above by 3 and 2 for families of 2-intervals and 2-union-intervals respectively, no polynomial-time approximation factor below 4 is known for the maximum independent set problem on 2-union-intervals. We observe, however, that Kaiser's proof can be turned into a 2-approximation if one has access to an oracle to solve the topological subproblems that arise. The particular topological problems in question are closely related to finding Borsuk-Ulam fixed-points. Unfortunately, the problem of finding Borsuk-Ulam fixed-points is PPAD-complete [15] and thus seems unlikely to admit polynomial algorithms unless a major breakthrough occurs. Nevertheless, we establish the following in Section 3:

Theorem 4 *There exists an algorithm for the maximum independent set problem on 2-intervals (respectively, 2-union-intervals) returning a solution of size at least $\frac{\alpha(\mathcal{H})}{3}$ (respectively $\frac{\alpha(\mathcal{H})}{2}$), requiring $O(\log(\alpha(\mathcal{H})))$ calls to an oracle for a PPAD-complete fixed-point problem, and polynomial time for all other computations.*

Despite the fact that Theorem 4 is likely not of practical value, we find it interesting because it implies that the 2-dimensional maximum conflict-free subset problem is a natural APX-hard geometric optimization problem whose best known approximability appears to improve in the presence of a PPAD oracle. It remains an open problem to find an alternative method of achieving the approximation ratios of Theorem 4 while bypassing the need for a PPAD oracle (of course, this may very well be impossible, but proving so would separate P from PPAD, resolving a longstanding open problem).

1.3 Related Work

Many variations and special cases of d-interval stabbing and independence problems have been studied in a variety of contexts. Kovalena and Spieksma have examined the special case of the rectangle stabbing problem in which each rectangle is a horizontal line segment [11, 12]. They obtain an LP-relative $\frac{e}{e-1}$-approximation for this case, alongside an example showing that the integrality gap is precisely $\frac{e}{e-1}$.

Even et al. explore weighted and capacitated variations of d-union-interval hitting set [6]. Their results include a $3d$-approximation for a variant in which each point may only be used to hit a specified number of d-intervals, but may be purchased multiple times.

Spieksma considers the version of maximum d-interval independent set where the goal is to select a *single* interval from each d-interval such that none intersect [17]. It is shown that a straightforward greedy procedure yields a 2-approximation.

Some additional hardness results are also known. Even et al. show that there is a constant $c > 0$ such that it is NP-hard to approximate the d-interval hitting set problem to within $c \log d$ [6]. Dom et al. show that rectangle stabbing is $W[1]$-hard, even when the input consists of squares of the same size, implying that the problem is unlikely to be fixed-parameter tractable in the optimal objective value $\rho(\mathcal{H})$ [4].

2 Integrality Gap Bounds

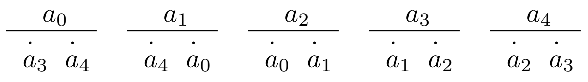
To prove Theorem 1 and establish tight lower bounds on the integrality gap of the independent set problem, we rely on an amplification lemma. We shall refer to the d individual intervals composing a d-interval as its *pieces*, and call a piece *inert* if it is a point. We shall call $\mathcal{H}$ a *clique* if $\alpha(\mathcal{H}) = 1$, and write $r(\mathcal{H})$ for the *rank* of $G_{\mathcal{H}}$—the maximum number of d-intervals intersected by any point in $\mathbb{R}$. We observe that if $\mathcal{H}$ is a clique and $r(\mathcal{H}) = p$, then a fractional independent set of value $\frac{|\mathcal{H}|}{p}$ can be obtained by simply putting weight $\frac{1}{p}$ on each d-interval in $\mathcal{H}$. In some situations, we can do better:

Lemma 5 *Suppose that $\mathcal{H}$ is a clique, and that $\mathcal{H}$ contains no two inert pieces that intersect and no d-intervals consisting entirely of inert pieces. Furthermore, suppose that $r(\mathcal{H}^*) = q$, where $\mathcal{H}^*$ is a modified version of $\mathcal{H}^*$ obtained by deleting all inert pieces. Then for any $N \in \mathbb{N}$, there is a clique $\mathcal{H}'$ of d-intervals admitting a fractional independent set of value $\frac{N|\mathcal{H}|}{Nq+1}$.*

Proof. We construct $\mathcal{H}'$ by making N copies of each d-interval in $\mathcal{H}$, and then perturbing all inert pieces in the resulting family of d-intervals such that no two inert pieces intersect, while preserving intersections of inert pieces with non-inert pieces. It is immediate that $\mathcal{H}'$ is still a clique; note that copies of the same d-interval in $\mathcal{H}$ must intersect in $\mathcal{H}'$ because no d-interval consists entirely of inert pieces. Moreover, $r(\mathcal{H}') \leq Nq + 1$, so we can place a weight of $\frac{1}{Nq+1}$ on each d-interval in $\mathcal{H}'$, yielding a fractional independent set of value $\frac{N|\mathcal{H}|}{Nq+1}$. □

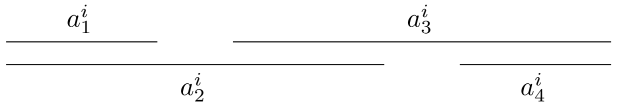
By taking the limit as $N \to \infty$, Lemma 5 yields an integrality gap lower bound of $\frac{|\mathcal{H}|}{q}$ given a d-interval graph satisfying the necessary requirements. We note that the amplification in Lemma 5 also works for d-union-intervals. We proceed with the proof of Theorem 1:

Proof of Theorem 1. For the case of d-interval graphs, we exhibit a clique $\mathcal{H}$ satisfying the conditions of Lemma 5 with $|\mathcal{H}| = 2d - 1$ and $q = 1$. We label the d-intervals $\{a_0, a_1, \ldots, a_{2d-2}\}$. Each d-interval will have exactly one non-inert piece (a closed interval in $\mathbb{R}$) and $d - 1$ inert pieces. We position the non-inert pieces such that no two intersect, which ensures that $q = 1$. Then for all $0 \leq i \leq 2d - 2$, we position the remaining $d - 1$ inert pieces of a_i (each of which is a point) on the non-inert pieces of d-intervals $\{a_{i+1}, \ldots, a_{i+(d-1)}\}$, where the addition is modulo $2d - 1$. This ensures that an inert piece of a_i intersects non-inert pieces in $\{a_{i+1}, \ldots, a_{i+(d-1)}\}$, hence ensuring that inert pieces of $\{a_{i-1}, \ldots, a_{i-(d-1)}\}$ all intersect the non-inert piece of interval a_i. This proves that the construction yields a clique, from which it follows that the independent set problem in d-interval graphs has an integrality gap of $\frac{|\mathcal{H}|}{q} = 2d - 1$. An example of the construction for $d = 3$ is shown (intervals are vertically separated for clarity):



For the case of d-union-intervals, we exhibit a clique $\mathcal{H}$ of size $4d - 4$ satisfying the conditions of Lemma 5 with $q = 2$. This yields an integrality gap $\frac{|\mathcal{H}|}{q} = 2d - 2$.

We label the $4d - 4$ d-union-intervals by $\{a^i_1, a^i_2, a^i_3, a^i_4\}_{i=1}^{d-1}$. We shall say that each interval has its k^{th} piece in the k^{th} *track*, where each track is a copy of $\mathbb{R}$. We first explain what happens in tracks 1 through $d - 1$, and then explain what happens in the final track, which is treated differently. For $1 \leq i \leq d - 1$, all of the pieces in track i are inert (single points) except for the i^{th} pieces of a^i_1, a^i_2, a^i_3, and a^i_4, which are arranged as follows:



Now, for all $j \neq i$, the i^{th} pieces of $\{a^j_1, a^j_2, a^j_3, a^j_4\}$ are positioned according to the following rules:

- If $j < i$, put a^j_1, a^j_2 in $a^i_1 \cap a^i_2$; put a^j_3, a^j_4 in $a^i_3 \cap a^i_4$
- If $j > i$, put a^j_3, a^j_4 in $a^i_1 \cap a^i_2$; put a^j_1, a^j_2 in $a^i_3 \cap a^i_4$

In the last track d, none of the intervals need to be inert. For all $1 \leq i \leq d - 1$, the d^{th} pieces of $\{a^i_1, a^i_2, a^i_3, a^i_4\}$ are positioned similarly to the diagram above, but are permuted to induce the remaining three dependencies among the d-intervals. Figure 1 illustrates this and provides an example of the entire construction for $d = 4$.

Observe that any two d-union-intervals with the same superscript i must be adjacent in either track i or track d. For $1 \leq i < j \leq d - 1$, we check that all 16 dependencies between $a^i_1, a^i_2, a^i_3, a^i_4$ and $a^j_1, a^j_2, a^j_3, a^j_4$ are accounted for: In track i, a^j_3 and a^j_4 intersect $a^i_1 \cap a^i_2$; a^j_1 and a^j_2 intersect $a^i_3 \cap a^i_4$. In track j, a^i_1 and a^i_2 intersect $a^j_1 \cap a^j_2$; a^i_3 and a^i_4 intersect $a^j_3 \cap a^j_4$. Thus $\mathcal{H}$ is a clique. It is easy to verify that the other conditions of Lemma 5 are satisfied with $q = 2$, so the proof is complete. □

Next, we provide a polynomial algorithm yielding an upper bound of d for the integrality gap of the general d-interval hitting set problem:

Proof of Theorem 2. Let $\mathcal{H}$ be a collection of d-intervals, and let $\{y^*_p : p \in P(\mathcal{H})\}$ be an optimal fractional hitting set of weight $\rho^*(\mathcal{H})$ obtained by solving linear program (2). We demonstrate how to round $\{y^*_p\}$ to an integral solution of weight at most $d \cdot \rho^*(\mathcal{H})$. For a d-interval I in $\mathcal{H}$, let $I^* \subseteq I$ be any piece of I that is hit by weight at least $\frac{1}{d}$ under $\{y^*_p\}$ (one must exist by the

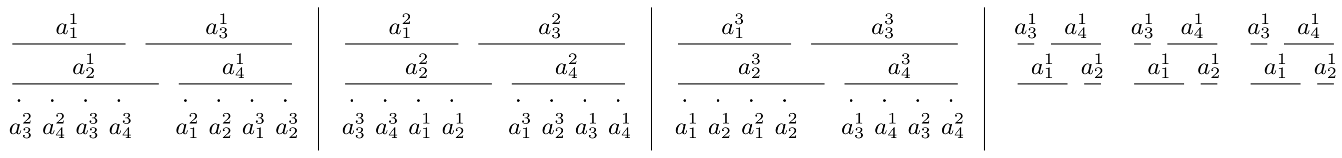

a_1^1 a_3^1 a_2^1 a_4^1
$a_3^2\ a_4^2\ a_3^3\ a_4^3$ $a_1^2\ a_2^2\ a_1^3\ a_2^3$
a_1^2 a_3^2 a_2^2 a_4^2
$a_3^3\ a_4^3\ a_1^1\ a_2^1$ $a_1^3\ a_2^3\ a_3^1\ a_4^1$
a_1^3 a_3^3 a_2^3 a_4^3
$a_1^1\ a_2^1\ a_1^2\ a_2^2$ $a_3^1\ a_4^1\ a_3^2\ a_4^2$
a_3^1 a_4^1 a_1^1 a_2^1

Figure 1: A clique of 12 4-union-intervals satisfying the conditions of Lemma 5 with $q = 2$

pigeonhole principle). Then the set $\mathcal{C} = \{I^* : I \in \mathcal{H}\}$ is a set of intervals in $\mathbb{R}$ that are each hit by weight at least $\frac{1}{d}$ under $\{y_p^*\}$.

By multiplying solution $\{y_p^*\}$ by d, we obtain a new fractional hitting set $\{dy_p^*\}$ of weight $d\rho^*(\mathcal{H})$ that hits, with weight at least 1, all elements of $\mathcal{C}$. However, the incidence matrix for the hitting set problem on 1-intervals is totally unimodular, so there must exist an integral hitting set Q of weight at most $d\rho^*(\mathcal{H})$ that hits all of $\mathcal{C}$—one can be found by simply solving linear program (2) again for $\mathcal{C}$ instead of $\mathcal{H}$. Of course, Q is also a hitting set for $\mathcal{H}$, from which it follows that $\rho(\mathcal{H}) \le d \cdot \rho^*(\mathcal{H})$. By simply returning Q, we obtain a polynomial-time LP-relative d-approximation for the d-interval hitting set problem, completing the proof. $\square$

We note that the above algorithm also works for the *weighted* variant of the minimum hitting set problem, in which each point $p \in P(\mathcal{H})$ is given a positive cost, and the goal is to compute a minimum cost hitting set.

Finally, we establish Theorem 3 by giving a set of d-union-intervals with a hitting set integrality gap of $d-\epsilon$:

Proof of Theorem 3. Fix $\epsilon > 0$. Choose any integer $t \ge \frac{2d^2}{\epsilon}$ and any integer $n \ge \frac{2t}{\epsilon}$. Fix some small δ, say $\delta = 0.1$. Here, we regard the d tracks $\{J^1, \dots, J^d\}$ as disjoint copies of $\mathbb{R}$. A d-union-interval I is called *aligned* if, for all $1 \le k \le d$, the piece of I in J^k has the form $[i_k + \delta, j_k - \delta]$ for some integers $0 \le i_k < j_k \le n$. In other words, a d-union-interval is aligned if the endpoints of all of its pieces each barely miss an integer point between 0 and n. Let $\mathcal{H}$ be the collection of all aligned d-union-intervals I such that the total length of all pieces in I is exactly $t - 2d\delta$. Note that $|\mathcal{H}|$ is finite.

Let P contain all points of the form $i + 0.5$ for $i \in \{0, 1, \dots, n-1\}$ in each of the d tracks, for a total of dn points. Each d-union-interval in $\mathcal{H}$ must contain at least t points in P, so we can obtain a fractional hitting set of total weight $\frac{dn}{t}$ by placing a value of $\frac{1}{t}$ at each point in P. This shows that $\rho^*(\mathcal{H}) \le \frac{dn}{t}$.

Let Q be any feasible integral hitting set for $\mathcal{H}$. We wish to show that $\frac{|Q|}{\rho^*(\mathcal{H})} \ge d - \epsilon$, so we may assume that $|Q| < n$. Let b_i be the number of points of Q in J^i. By the pigeonhole principle, there must exist an open interval $K^i \subseteq [0, n]$ in track J^i that has integer endpoints, has length at least $\frac{n-b_i}{b_i+1}$, and contains no points in Q. Consequently, there is a d-union-interval $K = \cup_{i=1}^d K^i$ having total length $\sum_{i=1}^d \frac{n-b_i}{b_i+1}$ that has integer endpoints and is missed by Q in all tracks. However, Q hits all aligned d-union-intervals having total length $t - 2d\delta$, and K is missed by Q, so we must have

$$\sum_{i=1}^d \frac{n-b_i}{b_i+1} < t.$$

By rearranging this, we obtain

$$d\left(\sum_{i=1}^d \frac{1}{b_i+1}\right)^{-1} > \frac{d(n+1)}{t+d}.$$

The left side of the above equation is a harmonic mean. Since an arithmetic mean is always greater than or equal to the corresponding harmonic mean, we get

$$\frac{1}{d}\sum_{i=1}^d (b_i + 1) > \frac{d(n+1)}{t+d}$$

and hence $\rho(\mathcal{H}) \ge |Q| = \sum_{i=1}^d b_i > \frac{d^2(n+1)}{t+d} - d$. Dividing by the upper bound we had for $\rho^*(\mathcal{H})$ gives

$$\frac{\rho(\mathcal{H})}{\rho^*(\mathcal{H})} > \frac{td(n+1)}{n(t+d)} - \frac{t}{n} > \left(1 - \frac{d}{t+d}\right)d - \frac{t}{n},$$

where the last inequality is due to $\frac{n+1}{n} > 1$. Since we chose t and n such that $t \ge \frac{2d^2}{\epsilon}$ and $n \ge \frac{2t}{\epsilon}$, we get

$$\frac{\rho(\mathcal{H})}{\rho^*(\mathcal{H})} > \left(1 - \frac{d}{\frac{2d^2}{\epsilon} + d}\right)d - \frac{t}{\frac{2t}{\epsilon}} = d - \frac{\epsilon}{2 + \frac{\epsilon}{d}} - \frac{\epsilon}{2} > d - \epsilon,$$

completing the proof of Theorem 2. $\square$

3 Topology-based algorithms

In this section, we sketch a proof of Theorem 4, illustrating how to obtain a 3-approximation (respectively, a 2-approximation) for the independent set problem on 2-intervals (respectively, 2-union-intervals), supposing one has access to oracles for PPAD-complete topological subproblems. We assume familiarity with the complexity class PPAD and its connection to topological fixed-point theorems; see [15] for background information.

Our approach follows Kaiser's duality gap upper bound proof [10], which we outline here. We first consider the case of 2-union-intervals. For concreteness, we consider a family $\mathcal{H}$ of axis-aligned rectangles in the plane. Let n be an arbitrary positive integer. Kaiser considers the space $S^n \times S^n$ (where S^n is the boundary of an $(n+1)$-dimensional unit ball), and associates each point $x \in S^n \times S^n$ to a set of n horizontal lines and n vertical lines in the plane. Kaiser then constructs a family of $2n+2$ real-valued functions $h_1^{\mathcal{H}}, \ldots, h_{2n+2}^{\mathcal{H}}$ on $S^n \times S^n$ having the following properties:

1. If $h_i^{\mathcal{H}}(x) = 0$ for all i, then x corresponds to a set of lines that intersect all rectangles in $\mathcal{H}$.
2. If $h_1^{\mathcal{H}}(x) = h_2^{\mathcal{H}}(x) = \ldots = h_{2n+2}^{\mathcal{H}}(x) \neq 0$ for all i, then x corresponds to a set of n horizontal lines and n vertical lines defining a grid from which we can easily find a conflict-free set of rectangles of size $n+1$ in polynomial time.

Kaiser then establishes that, for topological reasons, there must exist a point $x \in S^n \times S^n$ such that $h_1^{\mathcal{H}}(x) = h_2^{\mathcal{H}}(x) = \ldots = h_{2n+2}^{\mathcal{H}}(x)$ and thus a point x must exist satisfying item 1 or item 2 above. Tardos's result that $\rho(\mathcal{H}) \leq 2\alpha(\mathcal{H})$ follows immediately by setting $n = \alpha(\mathcal{H})$, since then a point where $h_i^{\mathcal{H}}(x) \neq 0$ for all i cannot exist, and thus a stabbing consisting of $\alpha(\mathcal{H})$ horizontal lines and $\alpha(\mathcal{H})$ vertical lines must exist.

Kaiser's proof can easily be adapted to yield a procedure $\mathcal{P}$ that, given an integer n, finds either a stabbing of size $2n$ or a conflict-free subset of size $n+1$, using polynomial time plus a single call to an oracle for a topological fixed-point problem. To obtain a 2-approximation for the maximum conflict-free subset problem, it then suffices to find a *cutoff point* $t \in \mathbb{N}$ such that $\mathcal{P}$ returns a conflict-free subset $\mathcal{S}$ of size t when run with $n = t-1$, but returns a stabbing of size $2t$ when run with $n = t$ (if this happens, we have $|\mathcal{S}| \geq \frac{\rho(\mathcal{H})}{2} \geq \frac{\alpha(\mathcal{H})}{2}$). Although there may be many cutoff points t, one must exist in the interval $[\frac{\rho(\mathcal{H})}{2}, \alpha(\mathcal{H})]$. One can be found using only $O(\log(\alpha(\mathcal{H})))$ calls to $\mathcal{P}$ by running a galloping binary search that first tries $t = 1, t = 2, t = 4, \ldots$ until a stabbing of size $2t$ is returned, and then binary searches between $\frac{t}{2}$ and t to find a cutoff point.

Kaiser's topological argument employs a result of Ramos that generalizes the Borsuk-Ulam theorem to cross products of spheres [16], so procedure $\mathcal{P}$ must invoke calls to an oracle for Ramos-style fixed-points. Ramos invokes a parity argument that can be adapted, in a straightforward manner, to show that an appropriate computational version of the Ramos fixed-point problem lies in the complexity class PPAD. Indeed, Ramos provides a searching algorithm to locate such fixed-points, although it may be exponential in the worst case. We also note that, by the discreteness of our problem, the particular instances that we must solve can be efficiently represented and have rational solutions. This establishes Theorem 4 for 2-union-intervals.

For the case of general 2-intervals, Kaiser provides a related argument that can be adapted in the same manner to yield a binary search algorithm. In this case, only an oracle for standard Borsuk-Ulam fixed-points is required. However, due to changes in how the functions $h_i^{\mathcal{H}}$ must be formulated, only a 3-approximation can be obtained. Still, the integrality gap bounds imply that this is optimal among all LP-relative approximations.

References

[1] N. Alon. Piercing d-intervals. *Disc. Comp. Geom.*, 19(3, Special Issue):333–334, 1998.

[2] R. Bar-Yehuda, M. Halldórsson, J. Naor, H. Shachnai, and I. Shapira. Scheduling split intervals. *SIAM J. Comput.*, 36(1):1–15 (electronic), 2006.

[3] G. Călinescu, A. Dumitrescu, H. Karloff, and P. Wan. Separating points by axis-parallel lines. *Internat. J. Comp. Geom. Appl.*, 15(6):575–590, 2005.

[4] M. Dom, M. Fellows, and F. Rosamond. Parameterized complexity of stabbing rectangles and squares in the plane. In Proc. *WALCOM*, LNCS 5341:298–309, 2009.

[5] A. Dumitrescu. On two lower bound constructions. In *Proc. 11th Canadian Conf. Comp. Geom.*, 1999.

[6] G. Even, R. Levi, D. Rawitz, S. Baruch, S. Shahar, and M. Sviridenko. Algorithms for capacitated rectangle stabbing and lot sizing with joint set-up costs. *ACM Trans. Algorithms*, 4(3), 2008.

[7] D. Gaur, T. Ibaraki, and R. Krishnamurti. Constant ratio approximation algorithms for the rectangle stabbing problem and the rectilinear partitioning problem. *J. Algorithms*, 43(1): 138–152, 2002.

[8] A. Gyárfás and J. Lehel. A Helly-type problem in trees. *Comb. Theory and its Appl., II*, 571–584, 1970.

[9] P.J-J. Herings and R. Peeters. Homotopy methods to compute equilibria in game theory. *Economic Theory*, 42(1):119–156, 2010.

[10] T. Kaiser. Transversals of d-intervals. *Disc. Comp. Geom.*, 18(2):195–203, 1997.

[11] S. Kovaleva and F. Spieksma. Primal-dual approximation algorithms for a packing-covering pair of problems. *RAIRO Oper. Res.*, 36(1):53–71, 2002.

[12] S. Kovaleva and F. Spieksma. Approximation algorithms for rectangle stabbing and interval stabbing problems. *SIAM J. Disc. Math.*, 20(3):748–768, 2006.

[13] J. Matoušek. Lower bounds on the transversal numbers of d-intervals. *Disc. Comp. Geom.*, 26(3):283–287, 2001.

[14] H. Nagashima and K. Yamazaki. Hardness of approximation for non-overlapping local alignments. *Disc. Appl. Math.*, 137(3):293–309, 2004.

[15] C. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. System Sci.*, 48(3):498–532, 1994.

[16] E. Ramos. Equipartition of mass distributions by hyperplanes. *Disc. Comp. Geom.*, 15(2):147–167, 1996.

[17] F. Spieksma. On the approximability of an interval scheduling problem. *J. Sched.*, 2(5):215–227, 1999.

[18] G. Tardos. Transversals of 2-intervals, a topological approach. *Combinatorica*, 15(1):123–134, 1995.

[19] V. Vatter. Small permutation classes. In *Proc. Lond. Math. Soc. (3)*, 103(5):879-921, 2011.

The Within-Strip Discrete Unit Disk Cover Problem

Robert Fraser [*] Alejandro López-Ortiz [†]

Abstract

We investigate the Within-Strip Discrete Unit Disk Cover problem (WSDUDC), where one wishes to find a minimal set of unit disks from an input set $\mathcal{D}$ so that a set of points $\mathcal{P}$ is covered. Furthermore, all points and disk centres are located in a strip of height h, defined by a pair of parallel lines. We give a general approximation algorithm which finds a $3\lceil 1/\sqrt{1-h^2}\rceil$-factor approximation to the optimal solution. We also provide a 4-approximate solution given a strip where $h \leq 2\sqrt{2}/3$, and a 3-approximation in a strip if $h \leq 4/5$, improving over the 6-approximation for such strips using the general scheme. Finally, we show that WSDUDC is NP-complete for a strip with any height $h > 0$.

1 Introduction

In the *Within-Strip Discrete Unit Disk Cover* (WSDUDC) problem, the input consists of a set of m unit disks $\mathcal{D}$ with centre points $\mathcal{Q}$, and a set of n points $\mathcal{P}$, all of which lie in the Euclidean plane. We define the *strip* s of height h as the region of the plane between two parallel lines ℓ_1 and ℓ_2, where $\mathcal{Q} \cap s = \mathcal{Q}$ and $\mathcal{P} \cap s = \mathcal{P}$. We assume that we are provided with the lines ℓ_1 and ℓ_2; alternatively, a minimum width strip may be computed. We wish to determine the minimum cardinality set of disks $\mathcal{D}^\star \subseteq \mathcal{D}$ such that $\mathcal{P} \cap \mathcal{D}^\star = \mathcal{P}$. This is a seemingly simpler context than the general Discrete Unit Disk Cover (DUDC) problem, which has no strip confining the positions of the points and disks. The DUDC problem is NP-complete [10], and has received attention due to applications in wireless networking and related optimization problems [14].

This paper addresses an open question regarding the hardness of the general DUDC problem. An implication of a polynomial time algorithm for WSDUDC for strips of any fixed width would be a simple PTAS for DUDC, using the shifting techniques of Hochbaum and Maass [9]. The recent PTAS for DUDC [12], as discussed shortly, uses fundamentally different techniques.

The notion of decomposing a problem into strip-based subproblems is natural, since an exact algorithm or PTAS for the subproblem can potentially be used to derive a general PTAS using the "shifting strategy" [9]. For example, the PTAS for the geometric unit disk cover problem (like DUDC except the centres of the disks are unrestricted) operates by dividing the problem into strips [9]. The maximum independent set of a unit disk graph may be found in polynomial time if the setting is confined to a strip of fixed height [11]. Geometric set cover on unit squares (precisely WSDUDC, except the disks are replaced with axis-aligned unit squares) may be solved optimally in $n^{O(k)}$ time when confined to strips of height k [7]. Considering these results, the hardness of WSDUDC is somewhat surprising.

The WSDUDC problem was formally introduced by Das et al. [6], as a subroutine for their DUDC approximation algorithm. In that work, it was demonstrated that points in a strip of height $1/\sqrt{2}$ (≈ 0.707) may be covered in $O(mn + n\log n)$ time using a fixed partitioning technique to obtain a 6-approximate algorithm.

The Strip-Separated Discrete Unit Disk Cover (SSDUDC) problem was first addressed by Ambühl et al. [1, Lemma 1]. The input consists of a set of points $\mathcal{P}$ located in a strip in the plane, like WSDUDC, but the set of unit disk centres $\mathcal{Q}$ lies strictly outside of the strip rather than in the strip. In the electronic version of this paper, we outline an $O(m^2n + n\log n)$ time exact algorithm for SSDUDC based on [1], which we use as a subroutine in our work. The Line-Separated Discrete Unit Disk Cover (LSDUDC) problem has a single line separating $\mathcal{P}$ from $\mathcal{Q}$. A version of LSDUDC was first discussed by [5], where a 2-approximate solution was given; an exact algorithm for LSDUDC was presented in [4]. Another generalization of this problem is the Double-Sided Disk Cover (DSDC) problem, where disks centred in a strip are used to cover points outside of the strip. This also has an exact dynamic programming solution [13].

Many papers have addressed DUDC using a variety of techniques, e.g. [3, 5]; a summary of such results is presented in [6]. Brönnimann and Goodrich [2] established the first constant factor approximation algorithm based on epsilon nets. Mustafa and Ray [12] described a PTAS for a more general version of DUDC based on local search. Interest in research on approximation algorithms for DUDC and related problems has remained high because of the large running time associated with the PTAS ($O(m^{65}n)$ for a 3-approximation, $O(m^{O(1/\varepsilon)^2}n)$ in general for $0 < \varepsilon \leq 2$). The best tractable result for DUDC is that of [6], which

[*]University of Waterloo, Canada, r3fraser@uwaterloo.ca
[†]University of Waterloo, Canada, alopez-o@uwaterloo.ca

describes a 18-approximate algorithm which runs in $O(mn + n \log n)$ time.

1.1 Our Results

We provide a general $3\lceil 1/\sqrt{1-h^2}\rceil$-approximate algorithm for solving the Within-Strip Discrete Unit Disk Cover (WSDUDC) problem on strips of height $h < 1$, which runs in $O(m^2 n + n \log n)$ time. Given a strip of height at most $2\sqrt{2}/3$ (≈ 0.94), a 4-approximate solution is given which refines the general algorithm by checking for simple redundancy while still running in $O(m^2 n + n \log n)$ time. For a strip of height at most 4/5, an $O(m^6 n)$ time 3-approximate solution is provided which uses dynamic programming to solve all subproblems optimally (using the general $3\lceil 1/\sqrt{1-h^2}\rceil$-approximate algorithm on strips of height $2\sqrt{2}/3$ or 4/5 would produce a 6-approximation). To conclude, we show that WSDUDC is NP-complete.

2 Approximation Algorithms for WSDUDC

In this section, we present algorithms for approximating the optimal WSDUDC solution. We begin with a general technique, followed by refinements which achieve better approximation factors in narrower strips.

Theorem 1 *Given a strip of height $h < 1$, we may find a $3\lceil 1/\sqrt{1-h^2}\rceil$-approximation to the WSDUDC problem in $O(m^2 n + n \log n)$ time. If $h \le 2\sqrt{2}/3$, we can improve the approximation factor to 4 in $O(m^2 n + n \log n)$ time. Given a strip of height $h \le 4/5$, a 3-approximate solution may be found in $O(m^6 n)$ time.*

We define the set of rectangles $\mathcal{R}^\circ$, where R_i° is the largest rectangle of height $2h$ which may be covered by $D_i \in \mathcal{D}$, where the strip s has height h and is assumed to be horizontal. Further, we use a set of rectangles $\mathcal{R}$ of height h, defined as $R_i = R_i^\circ \cap s, \forall R_i^\circ \in \mathcal{R}^\circ$.

Observation 1 *Suppose we are given a strip of height $h < 1$ and a unit disk D whose centre lies in the strip. R° is defined as the rectangle of height $2h$ and width $k = 2\sqrt{1-h^2}$ which is circumscribed by D. If a point q is covered by R°, then D also covers q. Furthermore, R° covers the entire height of the strip.*

We divide the set of points $\mathcal{P}$ into two sets $\mathcal{P} = \mathcal{P}_{\mathcal{R}} \cup \mathcal{P}_{\overline{\mathcal{R}}}$, where $\mathcal{P}_{\mathcal{R}}$ is the set of points covered by the set of rectangles $\mathcal{R}$, and $\mathcal{P}_{\overline{\mathcal{R}}} = \mathcal{P} \setminus \mathcal{P}_{\mathcal{R}}$, i.e. those points covered by $\mathcal{D}$ but not $\mathcal{R}$. The approximation algorithms proceed in two stages to compute the cover: first the points in $\mathcal{P}_{\overline{\mathcal{R}}}$ are covered, and then the remaining uncovered points in $\mathcal{P}_{\mathcal{R}}$ are covered. We refer to the points in $\mathcal{P}_{\overline{\mathcal{R}}}$ as occurring in the *gaps* of the strip, and the points in $\mathcal{P}_{\mathcal{R}}$ are in the *intervals* (see Figure 1). In our discussion, we assume that $h > 0$, so that $k = 2\sqrt{1-h^2} < 2$. [1]

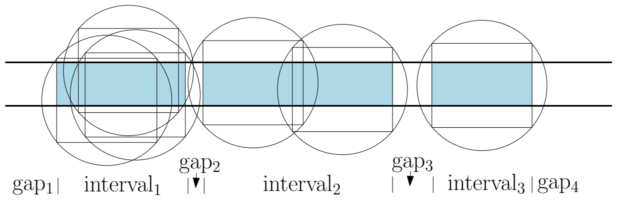


Figure 1: Intervals are continuous segments of the strip covered by the rectangles in $\mathcal{R}$, and gaps are the segments of the strip outside of the intervals.

Algorithm 1 GREEDY-RECTANGLES($\mathcal{R}, \mathcal{P}_{\mathcal{R}}$)

```
$\mathcal{R}' \leftarrow \emptyset$, sort $\mathcal{R}$ by x-coordinate, sort $\mathcal{P}_{\mathcal{R}}$ by left boundary
while $\mathcal{P}_{\mathcal{R}} \neq \emptyset$ do
    $p_\ell \leftarrow$ left-most point in $\mathcal{P}_{\mathcal{R}}$
    $R_r \leftarrow$ right-most rectangle in $\mathcal{R}$ covering $p_\ell$
    $\mathcal{R}' = \mathcal{R}' \cup R_r$
    $\mathcal{P}_{\mathcal{R}} = \mathcal{P}_{\mathcal{R}} \setminus (R_r \cap \mathcal{P}_{\mathcal{R}})$
return $\mathcal{R}'$
```

2.1 Covering $\mathcal{P}_{\overline{\mathcal{R}}}$

The centres of all disks are separated from the points in $\mathcal{P}_{\overline{\mathcal{R}}}$ by vertical lines (those of the gap boundaries). For each gap of the strip, the points are covered optimally with the $O(m^2 n + n \log n)$ time algorithm for SSDUDC. While points in each gap are covered optimally, we may lose optimality when we combine these solutions[2]. Recall that rectangles have width $k = 2\sqrt{1-h^2}$. There is a rectangle for each disk, and so no disk centre lies within a distance of $k/2$ of any gap. By interleaving rectangles with gaps of width ε, a disk may cover points in $2\lceil 1/k - 1/2\rceil$ gaps as $\varepsilon \to 0$. To see this, consider the right side of a disk D_i, where R_i defines an interval of width $k/2$ on this right side. Since D_i has unit radius, $\lceil (1-k/2)/k\rceil$ additional intervals (and gaps, one to the left of each interval) may be at least partially covered by the right half of D_i. Thus, the union of the solutions for each gap has an approximation factor of $2\lceil 1/k - 1/2\rceil$ for covering $\mathcal{P}_{\overline{\mathcal{R}}}$.

2.2 Covering $\mathcal{P}_{\mathcal{R}}$

To cover the points remaining after the previous step, we iteratively add the right-most rectangle that covers the left-most remaining point to the solution, as detailed in Algorithm 1 (GREEDY-RECTANGLES).

[1] If $h = 0$, all points and disk centres are collinear, and $\mathcal{P}_{\overline{\mathcal{R}}}$ is empty. This setting is solved optimally by the GREEDY-RECTANGLES algorithm detailed in Section 2.2.

[2] Covering the points in the union of the gaps cannot be covered optimally in general, as the hardness proof for WSDUDC (Section 3) only has points in gaps.

Lemma 2 *A rectangle R'_i selected by* Greedy-Rectangles *may overlap another rectangle R'_{i-1} (the previous rectangle chosen) by $k - \varepsilon$, for any $\varepsilon > 0$.* [3]

Lemma 3 *Let $\mathcal{R}' = \{R'_1, \ldots, R'_{|\mathcal{R}'|}\}$ be the set of rectangles found by* Greedy-Rectangles*, indexed from left to right so that $\forall i, j, i < j \leftrightarrow left(R'_i, R'_j)$ where $left(R'_i, R'_j)$ indicates that R'_i is left of R'_j. Then $\forall i, j, j > i + 1 \rightarrow R'_i \cap R'_j = \emptyset$.* [3]

Lemma 4 Greedy-Rectangles *computes a cover of $\mathcal{P}_{\mathcal{R}}$ with an approximation factor of $3\lceil 1/k - 1/2\rceil$ times the optimal solution.*

Proof. Consider the maximum number of rectangles in the Greedy-Rectangles solution that may be replaced by a single disk D_i in the strip. One of the rectangles available to the algorithm is $R_i \subset R_i^\circ$, where R_i° is circumscribed by D_i. By Lemma 2, there may be another rectangle ε to the left or right of R_i which will be selected by the algorithm, and so the approximation factor is at least 2. It may be possible to pack additional pairs of nearly overlapping rectangles as densely as permitted by Lemma 3 so that the points covered by these rectangles are also covered by D_i. Since all disks have unit radius and R_i° is circumscribed, each side of D_i can potentially cover all points covered by at most $2\lceil(1-k/2)/k\rceil - 1$ additional rectangles. This analysis is similar to Section 2.1, but now all rectangles are paired except for the right-most one (in a right-most pair, the region covered only by the right rectangle cannot be covered at all by D_i since we consider the *pairs* to have width k, i.e. $\varepsilon = 0$). Thus, the total approximation factor is $4\lceil 1/k - 1/2\rceil$. □

Greedy-Rectangles requires both the set of rectangles $\mathcal{R}$ and the set of points $\mathcal{P}_{\mathcal{R}}$ to be sorted in left to right order. The sorted lists are each walked through once, so the running time is $O(m \log m + n \log n)$.

2-approximation when $k \geq 2/3$ ($h \leq 2\sqrt{2}/3$). The general algorithm for covering $\mathcal{P}_{\mathcal{R}}$ presented above has an approximation factor of 4 when $k \geq 2/3$. For each pair of consecutive rectangles R'_{i-1} and R'_i found by Greedy-Rectangles, we determine whether there exists a disk D_j such that $(R'_{i-1} \cup R'_i) \cap \mathcal{P} \subseteq D_j \cap \mathcal{P}$. To do so, we run through $\mathcal{R}'$ in order, and check whether the current pair may be replaced by any disk in $\mathcal{D}$.

Consider a disk $D_i \in \mathcal{D}^\star$, which may or may not be a member of our refined solution set. D_i may intersect at most four rectangles in $\mathcal{R}'$. Every consecutive pair of rectangles in $\mathcal{R}'$ now requires at least two disks, so at least two disks are required to cover any four consecutive rectangles. Therefore, the overall approximation factor is two. This operation will scan m disks for every possible disk to remove from the solution, so the operation takes $O(m^2 n + n \log n)$ time.

Optimal solution when $k \geq 6/5$ ($h \leq 4/5$). In this case[4], the $\mathcal{P}_{\mathcal{R}}$ sub-problem may be solved optimally using dynamic programming. We define a set of disks D_s as *mutually spanning* if each disk in D_s covers a non-empty set of points which lies to the left of all other disks in D_s, as well as a non-empty set of points lying to the right of all other disks in D_s.

Lemma 5 *If $h \leq 4/5$, an optimal solution to $\mathcal{P}_{\mathcal{R}}$ requires mutually spanning sets of size at most 3.* [3]

By Lemma 5, a dynamic program which add disks to the solution in a left-to-right fashion need only consider up to triples of disks to terminate sub-problems to ensure that the sub-problems are independent and optimal. Such a dynamic program is described in Algorithm 2. In the algorithm, $\mathcal{D}^2$ and $\mathcal{D}^3$ are the sets of mutually spanning doubles and triples of disks respectively, and $\mathfrak{D}$ is the set of all sets of disks under consideration. Given two sets $\mathcal{D}_i, \mathcal{D}_j \in \mathfrak{D}$, if $\mathcal{D}_i$ covers points left of $\mathcal{D}_j$, and $\mathcal{D}_j$ does not cover points left of $\mathcal{D}_i$, we write $\mathcal{D}_i <_c \mathcal{D}_j$ to indicate this relationship. Otherwise, we consider them incomparable under this operator. Hence, we may establish a partially ordered set over all of the sets in $\mathfrak{D}$ w.r.t. the $<_c$ operator. Note that directed cycles are impossible in this set, since the transitive property holds for the $<_c$ operator. We impose a topological sorting $\mathfrak{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_{|\mathfrak{D}|}\}$ so that for any two sets $\mathcal{D}_i, \mathcal{D}_j$ in this ordering, we have that $i < j \rightarrow \mathcal{D}_j \not<_c \mathcal{D}_i$.

The correctness of Optimal-$\mathcal{P}_{\mathcal{R}}$ follows from the fact that all points left of a set $\mathcal{D}_i$ are covered in a valid solution to a subproblem terminating with $\mathcal{D}_i$, and all mutually spanning sets up to size three are considered. Optimal-$\mathcal{P}_{\mathcal{R}}$ runs in $O(m^6 n)$ time: there are $O(m^3)$ possible combinations of disks that we consider in two nested for loops, and inside the nested loop we check the disks against the point set $\mathcal{P}$.

2.3 Combining solutions for $\mathcal{P}_{\overline{\mathcal{R}}}$ and $\mathcal{P}_{\mathcal{R}}$

Recall that the approximation factor for covering the entire set of $\mathcal{P}_{\overline{\mathcal{R}}}$ is $2\lceil 1/k - 1/2\rceil$ and $4\lceil 1/k - 1/2\rceil$ for covering $\mathcal{P}_{\mathcal{R}}$, where k is the width of the rectangles. We simply sum these factors to get an overall approximation factor of $6\lceil 1/k - 1/2\rceil < 3\lceil 1/\sqrt{1-h^2}\rceil$ for strips of arbitrary height $h < 1$. The running time is $O(m^2 n + n \log n)$, effectively dominated by the SSDUDC algorithm used to cover $\mathcal{P}_{\overline{\mathcal{R}}}$.

[3]Due to lack of space, the proof of this lemma is omitted. It is presented in the electronic version of this paper.

[4]A similar dynamic programming algorithm applies to larger strips, but the running time increases rapidly with h.

Algorithm 2 OPTIMAL-$\mathcal{P}_{\mathcal{R}}$ $(\mathcal{D}, \mathcal{P}_{\mathcal{R}})$ (Assumes $k \geq 6/5$)

```
$\mathfrak{D} \leftarrow \mathcal{D} \cup \mathcal{D}^2 \cup \mathcal{D}^3$, $m' \leftarrow |\mathfrak{D}|$
Topologically sort $\mathfrak{D}$ on the $<_c$ operator
$c[0] = 0, c[1 \ldots m'] = \infty$
for $i = 1 \ldots m'$ do
  for $j = 0 \ldots i-1$ do
    size $\leftarrow c[j] + |\mathcal{D}_i|$
    if size $< c[i]$ and no points lie between $\mathcal{D}_i$ and $\mathcal{D}_j$ then
      $c[i] \leftarrow$ size
Backtrack on $c$ to recover optimal cover $\mathcal{D}^\star$
return $\mathcal{D}^\star$
```

4-approximation when $k \geq 2/3$ ($h \leq 2\sqrt{2}/3$). We have a 2-approximate algorithm for $\mathcal{P}_{\mathcal{R}}$ when $k \geq 2/3$, and we may solve each gap of $\mathcal{P}_{\overline{\mathcal{R}}}$ optimally. For the purposes of counting, we may assume that the disks forming the cover for each gap are equally distributed amongst the neighbouring intervals for both the approximate solution and the optimal one. We are not interested in the worst-case approximation factor in any given interval; rather we are interested in the approximation factor over the strip as a whole. For each gap, only disks found in adjacent intervals may form part of the solution. Disk centres are located at least a distance 1/3 from the end of an interval, and so disk centres in non-adjacent intervals are more than unit distance away from the gap. Thus, for each interval of the strip, assume that n_ℓ (resp. n_r) disks are used for covering the gap to the left (resp. right), and n_s disks are used for covering the points in the interval. The minimum number of disks required is $\max\{n_\ell, n_s/2, n_r\}$, since both n_ℓ and n_r are optimal and n_s is a 2-approximation. We conclude that $n_\ell + n_s + n_r \leq 4 \cdot \max\{n_\ell, n_s/2, n_r\}$, and thus it is a 4-approximation algorithm. Again, the running time is $O(m^2 n + n \log n)$.

3-approximation when $k \geq 6/5$ ($h \leq 4/5$). We have optimal algorithms for computing the cover of each gap of $\mathcal{P}_{\overline{\mathcal{R}}}$ and each interval of $\mathcal{P}_{\mathcal{R}}$. Further, the disks covering a gap only come from the two adjacent intervals, and the disks covering an interval only come from the interval itself. Since the disks in each interval can contribute to only three problems, each of which is solved optimally, the worst-case is that three times the optimal number of disks is used. The running time of the algorithm is dominated by OPTIMAL-$\mathcal{P}_{\mathcal{R}}$, so the overall running time is $O(m^6 n)$.

Corollary 6 *There is a 15- (resp. 16-) approximate algorithm for DUDC, which runs in $O(m^6 n)$ (resp. $O(m^2 n + n \log n)$) time.*

3 Hardness of WSDUDC

We prove that WSDUDC is NP-complete by reducing from the minimum vertex cover problem (VERTEX-COVER) on planar graphs of maximum degree three, which is known to be NP-complete [8]. Recall the setting for VERTEX-COVER: We are given a graph $G = (V, E)$, and we seek a minimum cardinality subset $V^\star \subseteq V$ such that for all $e_{(i,j)} = (v_i, v_j) \in E$, either $v_i \in V^\star$ or $v_j \in V^\star$. In other words, the vertex cover is a minimum cardinality hitting set of all of the edges in the graph.

Theorem 7 *WSDUDC is NP-complete.*

WSDUDC is in NP, since a certificate may be provided as a set of disks that covers all of the points in $\mathcal{P}$, which is trivial to verify.

In the reduction, we create an instance of WSDUDC from a planar graph so that a solution $\mathcal{D}^\star$ to the WSDUDC problem provides a solution $V^\star$ to the VERTEX-COVER problem on the graph. For our reduction, it is easier to consider the dual (disk piercing) setting of WSDUDC. The Within-Strip Discrete Unit Disk Piercing problem (WSDUDP) accepts a set of points $\mathcal{Q}$, a set of unit disks $\mathcal{D}_{\mathcal{P}}$ with centre points $\mathcal{P}$, and a strip of height h as inputs, and computes the minimum number of points $\mathcal{Q}^\star \subseteq \mathcal{Q}$ such that each disk in $\mathcal{D}_{\mathcal{P}}$ contains at least one point from $\mathcal{Q}^\star$. Let WS(G) be the WSDUDP instance created from a graph G. Note that a solution $\mathcal{Q}^\star$ for WSDUDP is exactly the set of centre points to $\mathcal{D}^\star$, the optimal solution to the WSDUDC problem in the primal setting.

Assume that we have a planar embedding of the graph and a horizontal strip so that the terms *left*, *right*, *above* and *below* are all well defined. Let ℓ^v_{vert} be a vertical line through vertex v. For the reduction, we make use of *dummy vertices*, which are simply extra vertices that we may place on an edge of the graph G. A *dummy edge* is an edge which is incident upon at least one dummy vertex. Informally, the steps of the reduction are:

1. Obtain a planar embedding of G where each vertex has a distinct x-coordinate.

2. For any vertex v with degree three where all incident edges are left or right of ℓ^v_{vert}, 'bend' the lowest edge with a dummy vertex so that the edge becomes incident to v from the opposite side of ℓ^v_{vert}, call this new graph $G' = (V', E')$.

3. For each vertex $v \in V'$, add a dummy vertex at each point where $\ell^v_{\text{vert}} \cap e \neq \emptyset, \forall e \in E'$.

4. Identify each vertex v of degree one or two where all edges are incident on the same side of ℓ^v_{vert}, say w.l.o.g. the edges are incident from the right. Place a vertical line ℓ_{vert} between v and the next vertex

to the left, and add a dummy vertex at each point where $\ell_{\text{vert}} \cap e \neq \emptyset, \forall e \in E'$. This ensures that consecutive vertical arrays of vertices differ in cardinality by at most one.

5. For any pair of vertices $v_i, v_j \in V$, ensure that an even number of vertices occur any path from v_i to v_j in G', by adding additional dummy vertices.

6. Create the WSDUDP instance WS(G) from G' so that every edge in E' corresponds to a disk in $\mathcal{D}$ and every vertex in V' to a point in $\mathcal{Q}$. We then show that an optimal solution to WSDUDP provides an optimal cover for G', from which an optimal vertex cover for G may be found, as required.

Lemma 8 *Given an edge $e_{(i,j)}$ of the graph $G = (V, E)$, we can add a pair of adjacent dummy vertices $V_d = \{v_{i_1}, v_{i_2}\}$ along the edge $e_{(i,j)}$ to create the graph $G' = (V \cup V_d, E \cup \{e_{(i,i_1)}, e_{(i_1,i_2)}, e_{(i_2,j)}\} \setminus \{e_{(i,j)}\})$. The graph remains planar, and the size of the optimal solution to* Vertex-Cover *over G' is $|V^\star| + 1$, where $V^\star$ is the set of vertices in a minimum vertex cover of G.* [3]

Lemma 9 *Given any optimal solution $V^\star_{G'}$ to* Vertex-Cover *on G', we can find an optimal solution $V^\star_G$ to* Vertex-Cover *on G in polynomial time.* [3]

An example WSDUDP construction WS(G) is shown in Figure 2, to provide intuition for the gadgets used in the reduction. Each edge of the graph G' (actual or dummy) corresponds to a disk in WS(G), and each vertex (actual or dummy) corresponds to a point in $\mathcal{Q}$. A point in $\mathcal{Q}$ stabs two disks in WS(G) if the degree of the corresponding vertex in G' is two; the remaining points stab three disks and their corresponding vertices have degree three.

A *wire* w_i is a sequence of disks positioned so that consecutive centres are spaced d_{disk} units apart, not necessarily collinearly, where $2\sqrt{1 - h_\ell^2} < d_{\text{disk}} < \sqrt{2 + 2\sqrt{1 - (3h_\ell/4)^2}}$, so that there exists a small area of overlap between consecutive disks which contains a point in $\mathcal{Q}$.[5] Disk centres on adjacent wires are $d_{\text{vert}} = 3h_\ell/2$ units apart vertically, and we define a *stack* as a set of such vertically aligned disks. The centres of the disks in a stack are shifted within the strip by $d_{\text{vert}}/2$ relative to an adjacent stack when the number of disks in the two stacks differs, while the distance between consecutive centres in each wire remains d_{disk}.

Lemma 10 *There is a non-empty area of intersection between three disks in consecutive stacks when the centres of the stacks are shifted by $d_{vert}/2$ relative to each other, and $d_{disk} < \sqrt{2 + 2\sqrt{1 - (3h_\ell/4)^2}}$.* [3]

[5]Note that $2\sqrt{1 - h_\ell^2} < \sqrt{2 + 2\sqrt{1 - (3h_\ell/4)^2}}$ for $h_\ell > 0$.

3.1 Gadgets

In the graph, we may encounter vertices of degree one, two, or three. With each vertex, wires may begin, end, split, merge, or continue unchanged. For vertices of degree one, the incident edge will correspond to a terminal disk on a wire. For vertices of degree two, if one edge leaves to the left and the other to the right in the embedding, this is a *trans*-2 vertex, and we handle it by continuing all wires. If both edges go in the same direction (left or right), we call this a *cis*-2 vertex, and we have a gadget to merge the pair of wires corresponding to the edges. Analogously, we have gadgets for both the *trans*-3 and *cis*-3 degree three vertices. Finally, we build a gadget to increase the number of vertices on an edge. With each gadget, we apply the analogous modification to G' by adding dummy vertices to the respective edges. This ensures that an optimal solution to WS(G) corresponds exactly to an optimal vertex cover for G'.

***cis*-2 Gadget.** In this case, a pair of wires will terminate, and since the two terminal disks correspond to a pair of edges sharing a vertex, we place a vertex in the area covered by both disks and no others. An extra column of dummy nodes should be used to extend all other wires if the vertex is on an interior face of the planar embedding of the graph, since two wires are terminated simultaneously, and we may only shift wires by $d_{\text{vert}}/2$ with each column.

***trans*-3 Gadget.** Suppose we have an upper wire ending in disk D_u and a lower wire ending in disk D_l, and they merge into a single wire beginning with disk D_c. Therefore, we can place D_c at a point so that the distance between the centres of both D_c to D_u and D_c to D_l is d_{disk}, as described in Lemma 10. By placing a vertex in $D_c \cap D_u \cap D_l$, a single point stabs three disks, which corresponds to a vertex which can cover three edges in the graph.

***cis*-3 Gadget.** For this gadget, we combine the *trans*-3 and *cis*-2 gadgets to build a *cis*-3 configuration. In the planar graph embedding, this corresponds to introducing a bend in the lowest edge incident to the *cis*-3 vertex with a dummy vertex, so that it becomes a *trans*-3 vertex.

Card+ **Gadget.** If the total number of dummy vertices added to an edge of G is odd, we require a gadget which increases the number of disks between a pair of points on a wire by one. An extra disk whose centre is very close to the centre point of a disk on the wire allows points to be placed so that the wire remains independent from adjacent wires, while increasing the number of disks on the wire by one.

Now an instance of WSDUDP WS(G) may be constructed from any planar graph G with no vertex of degree greater than three. A solution $\mathcal{Q}^\star$ to WS(G) is also a solution $V^\star_{G'}$ to the Vertex-Cover problem on $G' = (V', E')$, where $v_i \in V'$ is mapped to $q_i \in \mathcal{Q}$ and

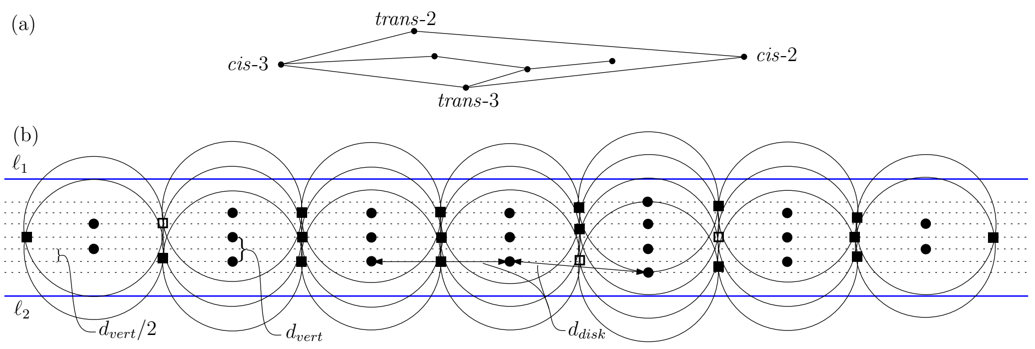


Figure 2: A sample WSDUDP construction WS(G) for the NP-hardness reduction. (a) Given a graph G, we compute a planar embedding (see Section 3.1 for vertex classes). (b) We construct a series of stacks of disks, where disks in adjacent stacks have slight overlap. The disk centres in each stack are aligned vertically and separated by a fixed distance d_{vert}. The number of disks in adjacent stacks may only vary by one. If two consecutive stacks have the same number of disks, the centres are aligned horizontally and separated by d_{disk}. If two consecutive stacks have differing numbers of disks, the centres are staggered vertically by $d_{\text{vert}}/2$, so that each disk centre is d_{disk} from two disk centres in the adjacent stack (thus, these stacks are distance $\sqrt{d_{\text{disk}}^2 - d_{\text{vert}}^2}$ apart). The points of $\mathcal{Q}$ are indicated by squares; those points stabbing three disks are empty. The centre points of the disks $\mathcal{P}$ are displayed as filled circles.

$q_i \in D_j \leftrightarrow v_i \in e_j \in E'$. By Lemma 9, we can find a minimum vertex cover $V_G^\star$ for G from $V_{G'}^\star$ in polynomial time. Therefore, there is a hitting set of size $c + (|\mathcal{D}| - |V|)/2$ for WS(G) if and only if there exists a vertex cover of size c for G (exactly half of the extra points added in the construction of WS(G) from G are required for a hitting set for $\mathcal{D}$). The number of disks stacked vertically in any column of WS(G) is in $O(m)$, where m is the number of edges and n is the number of vertices in the graph G. The number of such stacks is in $O(n)$, so the total number of disks and points in the WSDUDP construction is $O(mn)$. This completes the proof of Theorem 7.

4 Conclusions

We outlined several approximation algorithms for WS-DUDC and a proof of NP-completeness. The general $3\lceil 1/\sqrt{1-h^2}\rceil$-approximate algorithm and the 4-approximation for strips of height $\leq 2\sqrt{2}/3$ both run in $O(m^2 n + n \log n)$ time. The 3-approximate algorithm for strips of height $\leq 4/5$ runs in $O(m^6 n)$ time.

References

[1] C. Ambühl, T. Erlebach, M. Mihal'ák, and M. Nunkesser. Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs. In *APPROX*, pages 3–14, 2006.

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

[3] P. Carmi, M. Katz, and N. Lev-Tov. Covering points by unit disks of fixed location. In *ISAAC*, pages 644–655, 2007.

[4] F. Claude, G. Das, R. Dorrigiv, S. Durocher, R. Fraser, A. López-Ortiz, B. Nickerson, and A. Salinger. An improved line-separable algorithm for discrete unit disk cover. *Disc. Math. Alg. & Appl.*, 2(1):77–87, 2010.

[5] G. Călinescu, I. I. Măndoiu, P.-J. Wan, and A. Z. Zelikovsky. Selecting forwarding neighbors in wireless ad hoc networks. *Mob. Net. & Appl.*, 9(2):101–111, 2004.

[6] G. Das, R. Fraser, A. López-Ortiz, and B. Nickerson. On the discrete unit disk cover problem. In *WALCOM: Alg. & Comp.*, pages 146–157. 2011.

[7] T. Erlebach and E. van Leeuwen. PTAS for weighted set cover on unit squares. In *APPROX*, pages 166–177. 2010.

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

[9] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems inimage processing and VLSI. *J. ACM*, 32:130–136, 1985.

[10] D. Johnson. The NP-completeness column: An ongoing guide. *J. of Alg.*, 3(2):182–195, 1982.

[11] T. Matsui. Approximation algorithms for maximum independent set problems and fractional coloring problems on unit disk graphs. In *JCDCG*, pages 194–200. 2000.

[12] N. Mustafa and S. Ray. Improved results on geometric hitting set problems. *Disc. & Comp. Geom.*, 44:883–895, 2010.

[13] X. Xu and Z. Wang. Wireless coverage via dynamic programming. In *WASA*, pages 108–118. 2011.

[14] D. Yang, S. Misra, X. Fang, G. Xue, and J. Zhang. Two-tiered constrained relay node placement in wireless sensor networks: Efficient approximations. In *(SECON)*, pages 1 –9, 2010.

The Cover Contact Graph of Discs Touching a Line

Stephane Durocher* Saeed Mehrabi* Matthew Skala* Mohammad Abdul Wahid*

Abstract

We answer a question of Atienza et al. [4] by showing that the *circular CCG$^+$ problem* is $\mathcal{NP}$-complete. If we cover a set of objects on the plane with discs whose interiors are pairwise disjoint, then we can form a cover contact graph (CCG) that records which of the covering discs touch at their boundaries. When the input objects are themselves discs, and both input and covering discs are constrained to be touching and above the x-axis, then the circular CCG$^+$ problem is to decide the existence of a covering with a connected CCG. We also define an approximate version of this problem by allowing a small overlap between covering discs, and give an algorithm that in polynomial time finds an approximate solution for any yes-instance of the exact problem.

1 Introduction

Given a set S of n disjoint *input discs* in the plane, a covering C of S consists of n *covering discs* such that each covering disc covers exactly one input disc and no two covering discs intersect except on their boundaries. In general, the radii of the covering discs need not all be the same. The *cover contact graph* (CCG) induced by C is a graph $G = (V, E)$ such that each input disc corresponds to a vertex in V and two vertices are connected by an edge if and only if their corresponding covering discs are tangent. In other words, G is the intersection graph of a set of discs in the plane whose interiors are pairwise disjoint. Koebe's theorem [6] states that every planar graph can be represented as a *coin graph.* The coin graph of a set of discs in the plane is in fact the CCG of that set of discs. Problems related to these graphs arise in many application areas, such as wireless communication networks [5] and facility location [7].

Given a set of discs in the plane, the *circular CCG problem* asks if the given set admits a covering whose CCG is connected. Atienza et al. [4] show that the circular CCG problem is $\mathcal{NP}$-hard using a reduction from Planar3Sat, a constrained version of 3Sat in which the corresponding *variable-clause graph* must be planar. They also explore a variant in which the input discs are reduced to distinct points, with different kinds of connectivity required for the contact graph. They give algorithms with $O(n \log n)$ worst-case running time for 1-connectivity, and with $O(n^2 \log n)$ expected running time for 2-connectivity. They also study variants in which the covering discs are required to touch the x-axis. While they extensively examine the axis-touching case when the input is limited to distinct points, they leave open the case where the input is a set of discs and both the input and the covering discs are required to touch the x-axis. This is the circular CCG$^+$ problem, for which we show $\mathcal{NP}$-hardness in this paper. Our proof depends on very precise differences in the radii of the discs, and we show that such differences are essential to the hardness of the problem. We define an ϵ-approximate version of the circular CCG$^+$ problem and give a polynomial-time algorithm such that if the circular CCG$^+$ problem has an exact solution, then our algorithm produces an ϵ-approximate solution.

Many related problems are known. One is that of *realizability.* In addition to the set of input discs, we can be given an unlabeled planar graph G and the goal of deciding whether there exists a covering for the given input set whose CCG is G. Atienza et al. [4] show, again by reduction from Planar3Sat, that this realizability problem is also $\mathcal{NP}$-hard, even if the input is a set of points.

Notwithstanding Koebe's theorem that every planar graph is realizable as a coin graph (without constraining the centres of the discs), if we fix the coordinates of the vertices to make a *geometric* graph and require the discs to be centred on their respective vertices, then not every geometric graph can be so realized. Under the further constraint that the geometric graph is a tree, Abellanas and Moreno-Jiménez [3] present an $O(n \log n)$-time algorithm that decides if a given tree can be realized as a coin graph with coins centred at the vertices of the tree. For a graph that may not be a tree, they find a spanning tree and adapt their tree algorithm to solve the problem in polynomial time. Moreover, if the answer to this decision problem is affirmative, then the algorithm also computes all possible coin sets.

Abellanas et al. [1] show that given n points and n discs in the plane, it is $\mathcal{NP}$-complete to decide whether the discs can be placed in such a way that each disc is centred at one of the points and no two discs overlap. Abellanas et al. (with a different set of authors) show that the following problem is $\mathcal{NP}$-complete: Given a set of points in the plane, determine whether there are disjoint discs centred at the points such that the

*Department of Computer Science, University of Manitoba, {durocher, mehrabi, mskala, wahid}@cs.umanitoba.ca

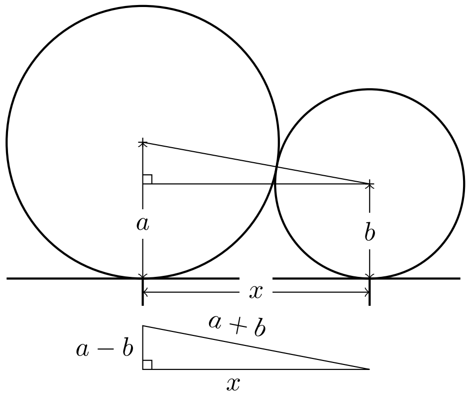


Figure 1: The constraint between two discs. See (1).

CCG of the discs is connected [2]. Note that if we relax the constraint that the discs must be centred at the given points, then the problem is polynomial-time tractable [4].

2 Proof of $\mathcal{NP}$-hardness

In this section, we show that the following problem is $\mathcal{NP}$-hard.

Given n distinct real numbers $x_1 < x_2 < \cdots < x_n$, and n nonnegative real numbers $r_1, r_2, \ldots, r_n$, the *circular CCG$^+$ problem* is to decide whether there exist real numbers $y_1, y_2, \ldots, y_n$ such that if C is the set of closed discs $C_1, C_2, \ldots, C_n$ where C_i is centred at (x_i, y_i) and has radius y_i (implying that C_i touches the x axis) with $y_i \geq r_i$, then the interiors of the discs in C are pairwise disjoint and the CCG induced by C is connected.

Consider two discs in the circular CCG$^+$ problem whose radii are a and b with the horizontal distance between their centres equal to x (see Figure 1). The constraint between the radii corresponds to the right triangle shown. We have:

$$\begin{aligned}(a+b)^2 &\leq (a-b)^2 + x^2 \\ \Leftrightarrow ab &\leq x^2/4\,.\end{aligned} \tag{1}$$

The constraint holds as an inequality for every pair of discs. It achieves equality if and only if the discs touch, corresponding to an edge in the CCG$^+$. Taking the logarithm, we have:

$$\log a + \log b \leq 2\log x - \log 4\,.$$

The logarithmic form of the constraint provides intuition for the hardness of the problem. Finding a circular CCG$^+$ means solving a linear program on the logarithms of the disc radii, subject to an additional constraint that the graph formed by the constraints that reach equality is a connected graph. It is intuitively reasonable that linear programming with a connectivity constraint should be $\mathcal{NP}$-hard, because if we could constrain linear programming variables to represent a connected graph, then we could constrain them to represent a 2-regular connected graph, which would be a Hamiltonian cycle.

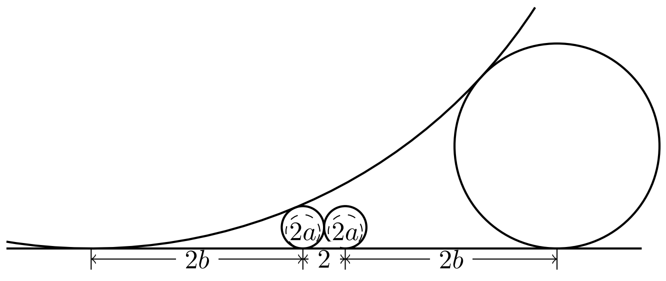


Figure 2: A non-convexity gadget.

Our proof, however, reduces from 3SAT. We will have gadgets and ways to combine them so that different values of the radius of the leftmost disc in the problem will correspond to different assignments of values to boolean variables; then we will manipulate the sets of radii that could satisfy the CCG$^+$ problem so that they correspond to exactly the assignments that satisfy the 3SAT problem.

The first step to create a hard instance of the circular CCG$^+$ problem is to create a non-convex solution set in the related linear programming problem.

2.1 A Non-convex Gadget

Figure 2 shows a gadget for creating non-convexity. The two discs in the middle are constrained to a minimum radius a slightly less than 1; their proximity to each other also gives them a maximum radius slightly greater than 1. Then in order to form a connected contact graph, the two outer discs must touch each other as shown; they cannot both touch the inner discs, but one must, and there is a choice as to which one that is. The radius y_1 of the leftmost disc is constrained to be in one of two non-overlapping intervals depending on that choice. The following lemma describes the behavior and existence of the non-convexity gadget.

Lemma 1 *We can choose the dimensions of a gadget like that shown in Figure 2 to constrain the radius y_1 of the leftmost disc such that the circular CCG$^+$ problem is satisfiable if and only if $(c \leq y_1 \leq d) \vee (e \leq y_1 \leq f)$ is true, for any positive $c, d, e,$ and f such that $1 \leq d/c = f/e < 16/9$ and $1 < e/c < (9/7)^2$.*

Proof. Consider the non-convex gadget shown in Figure 2 and let $C_1, C_2, C_3,$ and C_4 be the four discs, left to right. Their minimum radii are given by $r_1 = r_4 = 0$ and $r_2 = r_3 = a$ for some a slightly less than 1, which we will choose later. The horizontal distances are as shown: $2b$ from C_1 to C_2, 2 from C_2 to C_3, and $2b$ from C_3 to C_4, for some b we will choose later.

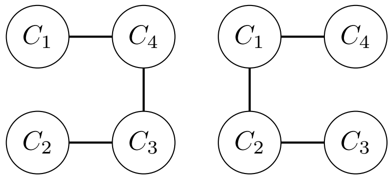


Figure 3: Allowable contact graphs for the non-convex gadget.

Each pair of discs in the gadget corresponds to an inequality constraint, which will achieve equality if and only if the discs touch.

$$y_1y_2 \le b^2 \quad \text{for } (C_1, C_2) \quad (2)$$
$$y_1y_3 \le (b+1)^2 \quad \text{for } (C_1, C_3) \quad (3)$$
$$y_1y_4 \le (2b+1)^2 \quad \text{for } (C_1, C_4) \quad (4)$$
$$y_2y_3 \le 1 \quad \text{for } (C_2, C_3) \quad (5)$$
$$y_2y_4 \le (b+1)^2 \quad \text{for } (C_2, C_4) \quad (6)$$
$$y_3y_4 \le b^2 \quad \text{for } (C_3, C_4) \quad (7)$$

We also have, as a consequence of (5) and the minimum radii $r_2 = r_3 = a \le 1$, the constraints $a \le y_2 \le 1/a$ and $a \le y_3 \le 1/a$. For the gadget to work as intended, we must ensure that the only connected contact graphs allowed are those shown in Figure 3. Requiring one of those graphs implies that (4), (5), and either of (2) or (7) can achieve equality, but (3) and (6) cannot, nor (2) and (7) simultaneously.

We can prevent C_1 and C_3 from touching by making C_2 big enough. We have $y_2 \ge a$. That gives $y_1 \le b^2/a$ from (2), and $y_3 \le 1/a$ from (5). Therefore $y_1y_3 \le b^2/a^2$. Substituting into (3), C_1 and C_3 will be prevented from touching if $a > b/(b+1)$. This relation will hold if we choose a large enough and b small enough; $a \ge 3/4$ and $b < 3$ are sufficient. These conditions also make (6) strict, by symmetry.

It remains to prevent (2) and (7) from both achieving equality at once. Suppose they did that. Then we would have $y_1y_2 = b^2$ and $y_3y_4 = b^2$, so $y_1y_2y_3y_4 = b^4$, but by (5), we can eliminate y_2 and y_3 and get $y_1y_4 \ge b^4$. That will contradict (4) if $b^4 > (2b+1)^2$, or $b^2 > 2b + 1$. Solving the quadratic, (2) and (7) cannot both be equalities if $b > 1 + \sqrt{2} = 2.414\ldots$. Therefore if $3/4 < a \le 1$ and $1+\sqrt{2} < b < 3$, the connected contact graphs that can be achieved are exactly the ones in Figure 3.

Assume we choose $3/4 < a \le 1$ and $1 + \sqrt{2} < b < 3$, and consider the possible values for y_1 in a solution to the problem. It must fall in one of two intervals, depending on whether C_1 touches C_2, or C_3 touches C_4. One and only one of those conditions must hold, as described above. If C_1 touches C_2, then because $a \le y_2 \le 1/a$, we have:

$$ab^2 \le y_1 \le b^2/a\,. \quad (8)$$

Symmetrically, if C_3 touches C_4, then $ab^2 \le y_4 \le b^2/a$, and then since C_1 and C_4 must touch each other, (4) is an equality and we have:

$$\frac{a(2b+1)^2}{b^2} \le y_1 \le \frac{(2b+1)^2}{ab^2}\,. \quad (9)$$

Note that in both (8) and (9), the ratio between the upper and lower limits is equal to $1/a^2$. By choosing an $a > 3/4$, we can choose any value less than $16/9 = 1.777\ldots$ for that ratio. Dividing the lower limits in (8) and (9) gives the ratio $b^4/(2b+1)^2$. Note that a cancels out and so the ratio between the lower ends of the intervals is independent of a. By choosing b between $1+\sqrt{2}$ and 3, we can choose this ratio anywhere between 1 and $(9/7)^2 = 1.653\ldots$. Also note that we can scale the entire gadget arbitrarily, with the effect of scaling all the interval bounds by the same factor. The result follows. □

2.2 Coupling Gadgets and Interval Duplication

The next step is to combine several such gadgets into a chain, by placing them side by side in such a way that they are forced to touch. Then the radius of the leftmost disc is constrained by all the constraints of all the gadgets; we have taken the intersection of the solution sets of the individual gadgets. Figure 4 illustrates the technique.

But linking gadgets side by side is not the only way to apply one gadget's constraints to another; we can also take one gadget, or a chain of them, scale it down to be arbitrarily small, and tuck it underneath another disc as shown in Figure 5. Making it arbitrarily small means we can prevent any other discs in the problem from interfering with the contact. Moving the small disc closer to the large disc cancels out the effect of scaling them smaller, so the effect on the large disc's radius is, if we so choose, no different from placing the gadgets side by side. The difference is that because it does not require access from the sides, only from the bottom arbitrarily close to the centre, we can apply this technique to the inner discs of the gadget from Figure 2; and whatever we do to one of those is done *twice* to the leftmost disc of the gadget. The set of allowed radii for the leftmost disc becomes the union of two copies of the set of allowed radii we apply to the inner disc, separated by an adjustable scaling factor. The following lemma states this property precisely.

Lemma 2 *Given a chain of discs that constrains its leftmost member's radius to be in a set $R \subseteq \mathbb{R}$ with $\sup R \le \sqrt{2} \inf R$, and positive reals c and d with*

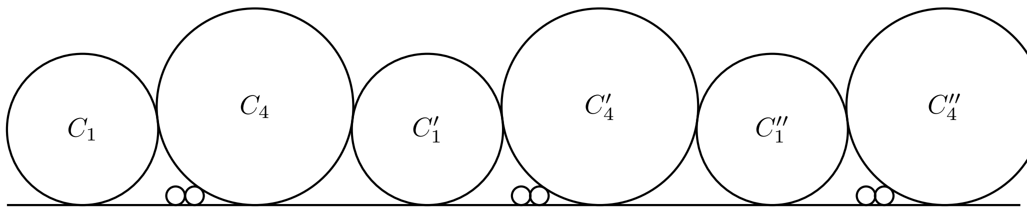


Figure 4: Gadgets coupled into a chain.

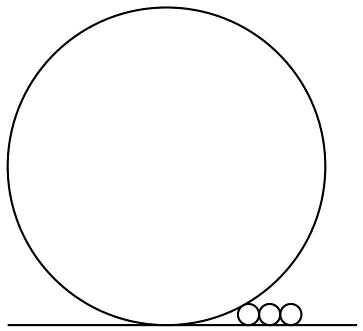


Figure 5: Hiding a chain of discs under a large disc (scale distorted for clarity).

$c < d < (9/7)^2c$, by adding five more discs we can construct a gadget in which the leftmost disc's radius is constrained to be an element of the set $\{cy \mid y \in R\} \cup \{dy \mid y \in R\}$.

That construction can be repeated a linear number of times to create an exponential number of disjoint intervals, giving the following corollary. The radius of the leftmost disc in the problem is constrained to be in one of the intervals, but so far unconstrained as to which one.

Corollary 3 *For any nonnegative integer k and positive reals c and d with $c > 1$, $d \geq 1$, and $c^{2^k}d \leq \sqrt{2}$, we can construct a gadget using a number of discs linear in k that constrains the radius of its leftmost disc to be in the set $\{y \mid c^i \leq y \leq dc^i$ for some $i \in \{0, 1, \ldots, 2^k - 1\}\}$.*

2.3 Encoding a 3Sat Instance

Choose any instance of 3Sat. We may add a polynomial number of extra variables to it for technical reasons to be described later, but suppose that after adding those it contains n boolean variables $v_1, v_2, \ldots, v_n$. There are 2^n ways to assign values to all the variables. We will associate those with 2^n intervals, disjoint but arbitrarily close to each other and proportionally equally sized and spaced. That is, the ratios between the upper and lower bound of each interval, and between the lower bound of each interval and the lower bound of the next, are the same for all intervals. All the intervals will be contained in $(1, \sqrt{2})$. The intervals are associated with variable

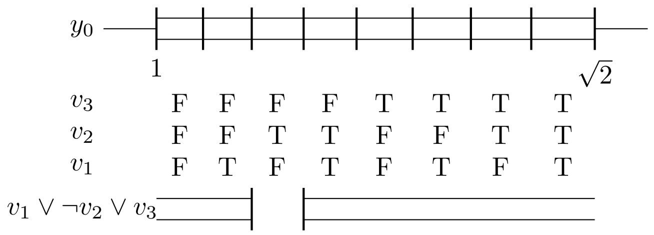


Figure 6: Satisfying a 3-clause.

assignments from all-false to all-true in binary counting order with v_1 as the least significant bit and v_n as the most significant bit. Figure 6 illustrates the encoding.

Figure 6 also illustrates how we can enforce a 3-literal disjunctive clause constraint on this encoding. Provided the clause only involves the three most significant variables, it corresponds to the negation of a single one of the eight intervals. For a clause of the form $(\neg a \vee \neg b \vee \neg c)$ or $(a \vee b \vee c)$, we just increase the minimum radius of the leftmost disc in the problem, or increase the minimum radius of the rightmost in order to have the effect of decreasing the maximum for the leftmost, and we can require the clause to be satisfied. For any other 3-clause over the three most significant variables, we can require satisfaction by intersecting with a union of two intervals that satisfy the numerical requirements of Lemma 1; so adding one more gadget to the right can have the effect of enforcing the clause as a constraint.

The clause constraint gadget only works for clauses in the three most significant variables, so we need all our clauses to be of that form when we apply it. Although other approaches more economical of variables might be possible, we will introduce three new variables for every clause, forcing them equal to the existing variables the clause is intended to constrain. This technique is illustrated in Figure 7. Here v_3 is the new variable being set equal to v_1. Observe that the set of intervals corresponding to $v_1 = v_3$ consists of two halves, and to equate v_i and v_j, $i < j$, each half is a set of 2^{j-i-1} intervals with equal proportional size and equal proportional spacing. Only the gap in the middle is different. We can create one of the halves with the gadget of Corollary 3, and then combine two copies of it with the appropriate spacing in the middle using the more general form of Lemma 2. The following lemma states precisely the ap-

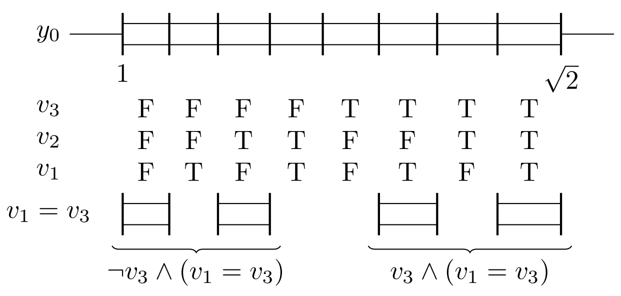


Figure 7: Duplicating a variable.

proach that we use to add a new variable and make it equal to an existing variable.

Lemma 4 *Given that n boolean variables $v_1, v_2, \ldots, v_n$ are encoded by the radius of a disc in a CCG$^+$ instance using a range from 1 to $r \leq \sqrt{2}$ according to our encoding scheme, for any integer $1 \leq i < n$ we can, with a number of discs linear in n, create a gadget that constrains the radius of its leftmost disc to enforce the constraint $v_i = v_n$.*

Proof. First we apply Corollary 3 with

$$\log c = \frac{\log r}{2^{n-i}},$$
$$\log d = \frac{\log r}{2^{n-i+1}}, \text{and}$$
$$k = n - i - 1$$

to create a gadget that enforces $v_i = v_n$ when v_n is false. Then we apply Lemma 2 to that gadget using

$$c = 1, \text{and}$$
$$\log d = \left[\frac{1}{2} + \frac{1}{2^{n-i+1}}\right] \log r\,.$$

□

The key observation is that the set of intervals corresponding to the statement $v_i = v_n$ may include an exponential number of intervals, but it is of a special form regardless of i and n: it is the union of two copies of a collection of proportionally equally spaced and sized intervals, and we can create it by applying Corollary 3 followed by Lemma 2.

Then we have all the pieces necessary to construct an instance of the circular CCG$^+$ problem equivalent to an instance of 3SAT. First, we calculate the number of variables we will add, which is equal to three times the number of 3-clauses. That gives us the size we need for the smallest intervals in our encoding. Note that this size comes from starting at a constant and scaling down by at most a constant amount, a linear number of times; we can represent the numbers involved in a polynomial number of bits.

We represent the variables from the original problem in a suitably narrow range of radii using Corollary 3. For each clause, we add three new variables with Lemma 4, doubling the number of variable assignments in the encoding with each one. We enforce the 3-clause. Then we proceed to the next. When we are done, we have a polynomial-sized instance of the circular CCG$^+$ problem whose leftmost disc is constrained to have a radius that represents a satisfying assignment for the original 3SAT instance; this is satisfiable if and only if the 3SAT instance was satisfiable. Therefore the following holds:

Theorem 5 *The circular CCG$^+$ problem is $\mathcal{NP}$-hard.*

3 An Approximation Algorithm

The $\mathcal{NP}$-hardness proof depends on high numeric precision. The constraints on disc radii create intervals that are exponentially small, although represented by a number of bits polynomial in the problem size. In this section we show that that precision is essential to the hardness of the problem: if we relax the problem definition in such a way as to permit imprecise solutions, then it becomes tractable.

First, discs cannot grow arbitrarily large or small. This follows from doing two rounds of simple constraint propagation (detailed proof omitted).

Lemma 6 *Any instance of the circular CCG$^+$ problem either is trivially satisfiable, or contains at least two discs with nonzero minimum size, and in the latter case we can in polynomial time compute a nonzero minimum and finite maximum size for every disc in the problem, which must be satisfied by any exact solution.*

Recall that the circular CCG$^+$ problem requires, for each pair of discs whose radii are a and b and whose horizontal distance is x, the constraint $ab \leq x^2/4$; and this holds as an equality if and only if the discs are touching each other and correspond to an edge in the contact graph. Let us relax the constraint to say that for any $\epsilon > 0$ two discs are *ϵ-approximately touching* if $x^2/4 \leq ab \leq (1+\epsilon)x^2/4$. Then other definitions arise by analogy: the *ϵ-approximate contact graph* is the graph with a vertex for each disc and an edge between any two discs that are ϵ-approximately touching, and the *ϵ-approximate CCG$^+$ problem* is like the CCG$^+$ problem but requires a choice of radii for the discs such that the $ab \leq (1+\epsilon)x^2/4$ constraint is obeyed by every pair of discs, instead of $ab \leq x^2/4$, and the ϵ-approximate contact graph is connected instead of the exact contact graph necessarily being connected.

Allowing ϵ-approximate contacts means that we can reduce the precision of all the disc radii. If we start from an exact solution and round up each disc radius to the next larger integer power of $\sqrt{1+\epsilon}$, we still have an ϵ-approximate solution, giving the following lemma.

Lemma 7 *If there exists an exact solution to an instance of the CCG$^+$ problem, then there exists a solution to the corresponding instance of the ϵ-approximate CCG$^+$ problem in which every disc radius is an integer power of $\sqrt{1+\epsilon}$.*

Therefore we can search for a solution with the radii limited to powers of $\sqrt{1+\epsilon}$, and still be assured of finding an ϵ-approximate solution if an exact solution exists. We can prove the following approximate result.

Theorem 8 *There exists an algorithm such that given a circular CCG$^+$ instance for which an exact solution exists, it finds an ϵ-approximate solution. If no exact solution exists, it may produce an ϵ-approximate solution or fail. Where R is the greatest ratio, for any disc in the problem, between the maximum and minimum radii of Lemma 6, the algorithm runs (unconditionally on solution existence) in time polynomial to the instance size and to $\log R / \log(1+\epsilon)$.*

Note that this is a one-sided test: when an exact solution exists, our algorithm guarantees to find an approximate solution, but if there exists an approximate solution and no exact solution, the algorithm does not guarantee finding the approximate solution.

The approximation algorithm performs dynamic programming on intervals of the left-to-right sequence of discs, using the following observation: if we have three discs left to right and we know that the one in the middle is the largest (possibly tying with either of the other two), then the one on the left and the one on the right cannot touch each other. Thus if we know which disc is largest in the entire problem and its radius, then we can split the problem into two smaller ones whose solutions are independent, which is the necessary structure for dynamic programming. We can try all possibilities for the largest disc, and recurse on each side, memoizing the answers to the recursive subproblems.

Each recursive subproblem corresponds to an interval of the sequence of discs, with specified sizes for the discs on either end, an assumption that no disc in between is larger than either of those, and a choice between a small constant number of cases for whether this subproblem is the leftmost or rightmost in the entire problem and whether or not the two endmost discs are already connected by larger discs outside the subproblem. Lemma 7's limitation on the number of radii we need to consider forces the number of subproblems, and thus the algorithm's time complexity, to be polynomial.

4 Conclusion

In this paper, we considered a circular cover contact graph problem defined by Atienza et al. [4]. We showed that when the input discs and the covering discs are all constrained to touch a line, then the problem of deciding whether the input set has a connected CCG is $\mathcal{NP}$-hard.

We also defined an approximate variation of the problem, where the covering discs are allowed to overlap by a small amount. We gave a polynomial-time algorithm such that if there exists an exact solution to the problem, then the algorithm returns an ϵ-approximate solution. However, if there is no exact solution, then the algorithm does not guarantee to return an approximate solution that might exist. Our algorithm provides an approximate answer to the decision problem of exact solution existence. The decision problem of approximate solution existence is a different problem, and the complexity of that problem remains open.

References

[1] M. Abellanas, S. Bereg, F. Hurtado, A. García Olaverri, D. Rappaport, and J. Tejel. Moving coins. *Computational Geometry: Theory and Applications (CGTA)*, 34:35–48, 2006.

[2] M. Abellanas, N. de Castro, G. Hernández, A. Máarquez, and C. Moreno-Jiménez. Gear system graphs. Manuscript, 2006.

[3] M. Abellanas and C. Moreno-Jiménez. Geometric graphs realization as coin graphs. In *International Conference on Computational Science and Its Applications*, volume 3045, pages 1–10, Assisi, Italy, 2004. Springer.

[4] N. Atienza, N. de Castro, C. Cortés, M. A. Garrido, C. I. Grima, G. Hernández, A. Márquez, A. Moreno-González, M. Nöllenburg, J. R. Portillo, P. Reyes, J. Valenzuela, M. T. Villar, and A. Wolff. Cover contact graphs. In S.-H. Hong, T. Nishizeki, and W. Quan, editors, *Graph Drawing*, volume 4875 of *Lecture Notes in Computer Science*, pages 171–182. Springer, 2007.

[5] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86:165–177, 1990.

[6] P. Koebe. Kontaktprobleme der konformen abbildung. *Ber. Sachs. Akad. Wiss. Leipzig Math.-Phys. Kl.*, 88:141–164, 1936.

[7] J.-M. Robert and G. T. Toussaint. Computational geometry and facility location. In *International Conference on Operations Research and Management Sciences*, pages B–1–B–19, Manila, Philippines, 1990.

On Piercing (Pseudo)Lines and Boxes

Subramanya Bharadwaj B. V* Chintan H. Rao* Pradeesha Ashok* Sathish Govindarajan *

Abstract

We say a family of geometric objects C has the (l, k)-property if every subfamily $C' \subseteq C$ of cardinality at most l is k-piercable. In this paper we investigate the existence of $g(k, d)$ such that if any family of objects C in $\mathbb{R}^d$ has the $(g(k, d), k)$-property, then C is k-piercable. Danzer and Grünbaum showed that $g(k, d)$ is infinite for families of boxes and translates of centrally symmetric convex hexagons. In this paper we show that any family of pseudolines with the $(k^2 + k + 1, k)$-property is k-piercable and extend this result to certain families of objects with discrete intersections. This is the first positive result for arbitrary k for a general family of objects. We also pose a relaxed version of the above question and show that any family of boxes in $\mathbb{R}^d$ with the (k^{2d}, k)-property is $2^d k$-piercable.

1 Introduction

A family of geometric objects C in $\mathbb{R}^d$ is said to be k-piercable if there exists a set of points $P \subset \mathbb{R}^d$ of cardinality k such that every object in C contains (is pierced by) at least one of the points of P.

Definition 1 *We say a family of geometric objects C has the (l, k)-property if every subfamily $C' \subseteq C$ of cardinality at most l is k-piercable.*

The classical Helly's theorem [8] stated in this notation is as follows: *Any family of convex objects C in $\mathbb{R}^d$ having the $(d + 1, 1)$-property is 1-piercable.*

Helly-type theorems have been widely studied for different settings (see surveys [5, 6]). Danzer and Grünbaum [4] considered the following generalised version of Helly's theorem:

For every positive integer k, does there exist a finite $g(k, d)$ such that if any family of convex objects C in $\mathbb{R}^d$ has the $(g(k, d), k)$-property, then C is k-piercable?

They showed that $g(k, d)$ is infinite even for families of boxes in $\mathbb{R}^d$. Specifically, they gave a generic construction and showed that $g(k, d)$ is infinite for all $k \geq 3$, $d \geq 2$ and $(k, d) \neq (3, 2)$. The same construction also works as a counterexample for hypercubes in $\mathbb{R}^d$. Katchalski et al [9] showed that $g(k, d)$ is infinite for translates of a symmetric convex hexagon.

Positive results are known for small values of k (i.e. $k = 2$). Danzer and Grünbaum [4] showed that for a family of boxes in $\mathbb{R}^d$, $g(2, d) = 3d - 1$ if d is even and $g(2, d) = 3d$ if d is odd. They also proved that $g(3, 2) = 16$ for a family of rectangles in $\mathbb{R}^2$. Katchalski et al [9] showed that for a family of homothetic triangles in $\mathbb{R}^2$, $g(2, 2) = 9$.

In this paper, we obtain the first positive results for general k. We show that for a family of pseudolines in $\mathbb{R}^2$, $g(k, 2)$ is finite for all $k \geq 2$. Specifically, we prove the following:

Theorem 1 *Let C be a family of pseudolines in $\mathbb{R}^2$ with $|C| \geq k^2 + k + 1$. For any integer $k \geq 2$, if C has the $(k^2 + k + 1, k)$-property then C is k-piercable.*

We extend the above theorem for families of objects C with the following property: any subfamily of $p + 1$ distinct objects in C intersect in at most one point. Note that $p = 1$ for a family of pseudolines.

Theorem 2 *Let C be a family of objects with the property that any subfamily of $p+1$ distinct objects in C intersect in at most one point. Let $|C| \geq k(kp+1)+1$. For any integer $k \geq 2$, if C has the $(k(kp + 1), k)$-property then C is k-piercable.*

The proof of Theorem 1 and 2 are combinatorial and exploit only the intersection property. In fact, Theorem 2 is true for set systems with the property that any subfamily of $p + 1$ distinct sets intersect in at most one element. Also the proofs lead naturally to a FPT algorithm for the minimum piercing problem on these objects. Note that the minimum piercing problem is NP-hard and APX-hard even for lines in $\mathbb{R}^2$ [12, 3].

Since $g(k, d)$ is infinite for most families of geometric objects in the above problem, we define the following relaxed variant, which we refer to as the k-Helly problem:

k-Helly problem: *For every positive integer k, determine the smallest $f(k, d)$ such that if any family of convex objects C in $\mathbb{R}^d$ has the $(g(k, d), k)$-property for some $g(k, d)$, then C is $f(k, d)$-piercable.*

*Department of Computer Science and Automation, Indian Institute of Science, Bangalore, India, {subramanya,chintanraoh,pradeesha,gsat}@csa.iisc.ernet.in

The k-Helly problem is related to the weak ϵ-net [1] and Hadwiger-Debrunner (p, q)-problem [7] as follows:

Weak ϵ-nets are a special case of the k-Helly problem : In the weak ϵ-net problem, we ask for a piercing set for objects containing $> \epsilon n$ points. By the pigeon hole principle, in any subcollection of $\frac{1}{\epsilon}+1$ objects, two will intersect. Therefore the objects satisfy the $(\frac{1}{\epsilon}+1, \frac{1}{\epsilon})$-property.

The k-Helly problem is a special case of the Hadwiger-Debrunner (p, q)-problem since the $(g(k, d), k)$-property implies the Hadwiger-Debrunner (p, q)-property for $p = g(k, d), q = g(k, d)/k$. Also, the finiteness of $g(k, d), f(k, d)$ is implied by the Hadwiger Debrunner (p, q) theorem [2], which shows a finite piercing set. However, the bounds given by the Hadwiger Debrunner (p, q) theorem are large (roughly $O(p^6)$ for convex objects in $\mathbb{R}^2$).

We show the following result for boxes in $\mathbb{R}^d$:

Theorem 3 *Let C be a family of boxes in $\mathbb{R}^d$. For any $k \geq 2$, if C has the (k^{2d}, k)-property, then C is $2^d k$-piercable.*

Note that for boxes in $\mathbb{R}^d$, $f(k, d) > k$ since otherwise $g(k, d)$ is infinite. The proof of Theorem 3 directly leads to a 2^d-approximate FPT algorithm for the minimum piercing problem on boxes. We note that the minimum piercing problem for boxes is NP-hard as well as W[1]-hard [11].

2 Lines and Pseudolines

Any two lines in $\mathbb{R}^2$ intersect in at most one point. This can be generalized in the following way.

Definition 2 *A family of geometric objects C in $\mathbb{R}^2$ is called a family of pseudolines if for every $l_i, l_j \in C$, l_i and l_j intersect in at most one point.*

Let C be a finite family of pseudolines in $\mathbb{R}^2$.

Definition 3 *Let a point x lie in the intersection of a set of pseudolines $l_1, l_2, \cdots, l_s \in C$. We call x k-degenerate in C if $s > k$.*

Lemma 4 *Let H be a set of points that pierces C. If x is k-degenerate in C and $x \notin H$, then $|H| \geq k+1$.*

Proof. If $x \notin H$ then we need at least s points to hit the s pseudolines passing through x. Since $s > k$ the lemma follows. □

Lemma 5 *Let $|C| \geq (k^2+k+1)$ and G be the set of all k-degenerate points in C. If C has the (k^2+k+1, k)-property, then $1 \leq |G| \leq k$*

Proof. Let S be a subset of C such that $|S| = k^2+k+1$. S is k-piercable. By pigeon hole principle, there exists a point x that pierces at least $k+1$ pseudolines in S. Hence $|G| \geq 1$. Also if $G \geq k+1$, there exists $S' \subset C$ which contains $k+1$ pseudolines passing through each of the first k points in G and one pseudoline passing through the $(k+1)$th point which does not pass through the first k points. Clearly $|S'| \leq k^2+k+1$ and S' is not k-piercable, a contradiction. Hence $1 \leq |G| \leq k$. □

Proof of Theorem 1. Let G be the set of all k-degenerate points in C. From Lemma 5, we know that $1 \leq |G| \leq k$. Let C' be the set of pseudolines not pierced by any of the points in G. We claim that if $|G| = k$ then $C' = \emptyset$. For if $C' \neq \emptyset$ then there is a $l \in C'$ such that it is not pierced by any point in G. For each point in G, pick $k+1$ pseudolines passing through it. This together with l gives a set C'' of at most $k(k+1)+1 = k^2+k+1$ pseudolines which is not k-piercable, a contradiction.

Hence let $|G| = k-r$ where $k > r \geq 1$ and $C' \neq \emptyset$. We claim $|C'| \leq rk$. Assume, for contradiction, that $|C'| > rk$. Then, for each point in G, pick $k+1$ pseudolines passing through it. This together with $rk+1$ lines from C' to give a set C'' of (at most) $(k-r)(k+1)+rk+1 = k^2+(k-r)+1 < k^2+k+1$ pseudolines. C'', being a subset of C, has the (k^2+k, k)-property and hence can be pierced by k points. Any point in G can pierce only $k+1$ pseudolines in C'' and no r points outside G can pierce the remaining $rk+1$ pseudolines in C'', a contradiction.

Now as before we pick $k+1$ pseudolines from each of the $k-r$ k-degenerate points together with at most rk pseudolines from C' to get a system of (at most) $(k-r)(k+1)+rk = k^2+k-r < k^2+k+1$ pseudolines. This can be pierced by k points. We have to choose each of the $k-r$ k-degenerate points in a piercing set for this system. This means that the rk pseudolines from C'(none of them are pierced by the degenerate points) have to be pierced by r points. This implies that C is k-piercable.

Lemma 6 *Let C be a family of pseudolines with $|C| \geq 6$. If C has the $(6, 2)$-property then C is 2-piercable.*

Proof. As C has the $(6, 2)$-property, there exist two cases. There exist some 6 pseudolines out of which 5 do not intersect or out of every 6 pseudolines 5 intersect.

In the first case there are two sub cases. There exist $l_1, \ldots, l_6 \in C$ such that l_1, l_2, l_3, l_4 intersect or in the second sub case l_1, l_2, l_3 and l_4, l_5, l_6 intersect respectively. Let $l \in C$. If l_1, l_2, l_3, l_4 intersect, then l is incident on the intersection of l_1, l_2, l_3 or on the intersection of l_5, l_6. Otherwise $l_1, l_2, l_3, l_5, l_6, l$ is a set of 6 pseudolines which are not 2-piercable. If l_1, l_2, l_3 and l_4, l_5, l_6 intersect, then l is incident on the intersection of l_1, l_2, l_3 or l_5, l_6. Otherwise $l_1, l_2, l_3, l_5, l_6, l$ is a set of

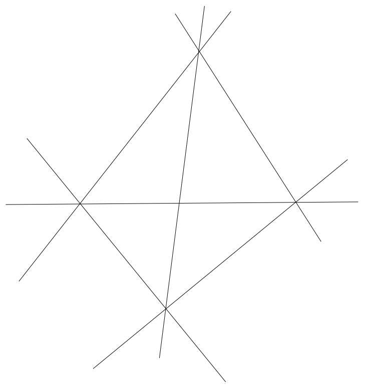


Figure 1: A family of 6 lines with the $(5, 2)$-property which is not 2-piercable.

6 lines which are not 2-piercable. Hence in both sub cases C is 2-piercable.

In the second case when out of every 6 pseudolines 5 intersect, all the lines except one have a common intersection and hence C is 2-piercable.

Hence in either case C is 2-piercable. □

The above result is tight since there is a family of 6 lines with the $(5, 2)$-property which is not 2-piercable (shown in Figure 1).

Consider a collection of pseudolines C. We wish to determine if C is k-piercable or not. There is a naive FPT algorithm which is implied by the above combinatorial result which takes $O(n^2 + k^3k^{4k})$ time. However one can use the techniques given in [10] to get a faster FPT algorithm which takes $O(n^2 + k^{2k+2})$ time.

3 Objects with discrete intersection

We extend the results of the previous section to a more general family of objects. We consider families of objects C with the following property : any subfamily of $p + 1$ distinct objects intersect in at most one point. This is a notion similar to the one in [10]. Unit circles, curves in the plane obtained by polynomial equations of bounded degree are some examples of objects with the above property.

Definition 4 *Let a point x lie in the intersection of a set of objects $c_1, c_2, \cdots, c_s \in C$. We call x k-degenerate in C if $s > kp$.*

Lemma 7 *Let H be a set of points that pierces C. If x is k-degenerate in C and $x \notin H$, then $|H| \geq k + 1$.*

Proof. Any point $y \neq x$ can pierce at most $p-1$ of the objects passing through x. Hence we need at least $k+1$ points to pierce $kp+1$ objects passing thorugh x. □

Consider a set of objects C with $|C| \geq k(kp + 1) + 1$ which has the $(k(kp + 1) + 1, k)$-property. Consider a subset $S \subseteq C$ with $|S| = k(kp+1)$. Note that it can be pierced by a set H of size k. Then there is some point $x_1 \in H$ which pierces at least $kp + 1$ objects in C. We construct a set of degenerate points as follows. Let $C_1 = C$, $G_1 = \{x_1\}$ where x_1 is obtained as before. Construct G_{i+1}, $i \geq 1$, as long as possible, in the following way: $C_{i+1} = C_i \setminus \{c \in C_i : c \text{ pierced by } x_i\}$. Let x_{i+1} be any k-degenerate point in C_{i+1}. Now $G_{i+1} = G_i \cup x_{i+1}$.

Lemma 8 *Consider a family of objects C with $|C| \geq k(kp+1)+1$ which has the $(k(kp+1), k)$ property. Let G be a set of k-degenerate points in C with maximum cardinality. Then $1 \leq |G| \leq k$.*

Proof. Clearly $|G| \geq 1$. Suppose $|G| \geq k + 1$. Let C^i be a subset of objects pierced by $x_i \in G$ with $|C^i| = kp + 1$. Consider $X = \cup_{1 \leq i \leq k} C^i \cup \{c\}$ where $c \in C^{k+1}$. Clearly $|X| \leq k(kp + 1) + 1$. Hence X is k-piercable. Any k-piercing set for C' must contain $\{x_1, \cdots x_k\}$ (by Lemma 7). This is a contradiction as $\forall x_i \in G, 1 \leq i \leq k$, x_i cannot pierce c. □

Proof of Theorem 2. Let G be a set of k-degenerate points in C with maximum cardinality and let $|G| = k - r$ where $k > r \geq 0$ (by Lemma 8). Let C^i be a subset of objects pierced by $x_i \in G$ with $|C^i| = kp + 1$.

Let C' be the set of remaining objects not pierced by any of these points. If $C' = \emptyset$ then C is k-piercable. Hence let us assume $C' \neq \emptyset$. We claim that $|C'| \leq r(kp + 1)$.

Assume, for contradiction, that $|C'| = r(kp + 1) + 1$. Then a subset of objects $X = C^1 \cup \cdots C^{k-r} \cup C'$ is k-piercable since $|X| \leq k(kp + 1) + 1$. Any k-piercing set for X must contain all $k - r$ points in G. If $r = 0$ this means that a object in C' is not pierced, a contradiction. Else if $r > 0$ this implies $r(kp + 1)$ objects in C' is pierced by r points, all of which are not k-degenerate, a contradiction.

Hence $|C'| \leq r(kp + 1)$. Again as before a subset of objects $X = C^1 \cup \cdots C^{k-r} \cup C'$ is k-piercable since $|X| \leq k(kp + 1) + 1$. Any k-piercing set for X must contain all $k - r$ points in G. This implies C' is r-piercable(if $r = 0$ this means $C' = \emptyset$). Hence C is k-piercable.

We extend the result on lines in the previous section to hyperplanes in 3 dimensions. The idea of replacing degenerate hyperplanes by a line is from [10].

Lemma 9 *Let C be a family of hyperplanes in R^3 with the $(k(k + 1)^3, k)$ property. Then C is k-piercable.*

Proof. We obtain a family of objects C' from C as follows. If at least $k + 1$ hyperplanes intersect in a line then we replace them with the line.

It is obvious that if C' is k-piercable then C is k-piercable. We note that if any $k + 1$ hyperplanes in C

intersect in a line l then any k-piercing set must contain a point from l. Hence C' is k-piercable if and only if C is k-piercable.

We claim that any set of $k+2$ objects in C' intersect in at most 1 point. There are two cases - the set contains at least two lines or the set contains at most one line. The claim is true if there are at least two lines in this set of $k+2$ objects. In the other case, the set contains at least $k+1$ hyperplanes and these cannot intersect in a line. Hence the $k+2$ objects intersect at most 1 point.

From Theorem 2 if C' has the $(k(k(k+2)+1), k)$ property then C' is k-piercable. Any line in C' can be realized as the intersection of at most $k+1$ hyperplanes in C. Hence if C has the $((k+1)k(k(k+2)+1), k)$ property then C is k-piercable which proves the claim. $\square$

This result can be extended to higher dimensions.

4 Boxes in $\mathbb{R}^d$

In this section we consider the k-Helly problem for families of boxes in $\mathbb{R}^d$.

Lemma 10 *Let I be a family of intervals in $\mathbb{R}$ with the $(k+1, k)$ property. Then I is k-piercable.*

Proof. We note that I satisfies the Hadwiger Debrunner $HD(k+1, 2)$ property. Hence I has a piercing set of size k [7]. $\square$

Lemma 11 *Let S be a family of vertical and horizontal slabs with the $(k+1, k)$-property. S is k-piercable.*

Proof. Let S_1 be the set of vertical slabs and S_2 the set of horizontal slabs. Clearly from Lemma 10, S_1 and S_2 are k-piercable. Without loss of generality let S_1 and S_2 be 'pierced' by k points on the x axis $v_1, \ldots, v_k$ and k points on the y axis $h_1, \ldots, h_k$ respectively. $(v_1, h_1), \ldots, (v_k, h_k)$ is a k-piercing set for S. $\square$

Proof of Theorem 3. Let C be a family of boxes in $\mathbb{R}^d$ with the (k^{2d}, k)-property. Figure 2 provides an illustration of the proof for rectangles in the plane(the case $d = 2$). We orthogonally project each box $r \in C$ to the coordinate axes. For each axis i, $1 \leq i \leq d$, we get a set of intervals C_i with the (k^{2d}, k)-property. Hence C_i has the $(k+1, k)$-property and is k-piercable (Lemma 10). Let H_i, $|H_i| \leq k$, be such a piercing set (the small hollow points on the x and y axes in Figure 2). Consider the grid points $H = \{(x_1, \ldots, x_d) : x_1 \in H_1, \ldots, x_d \in H_d\}$ (the small hollow circles in Figure 2). For $r \in C$, let r_i be the projection of r on axis i. There exist $x_1 \in H_1, \ldots, x_d \in H_d$, such that x_i pierces r_i. Thus $(x_1, \ldots, x_d) \in H$ pierces r. Hence every $r \in C$ can be pierced by one of the (at most k^d) grid points in H.

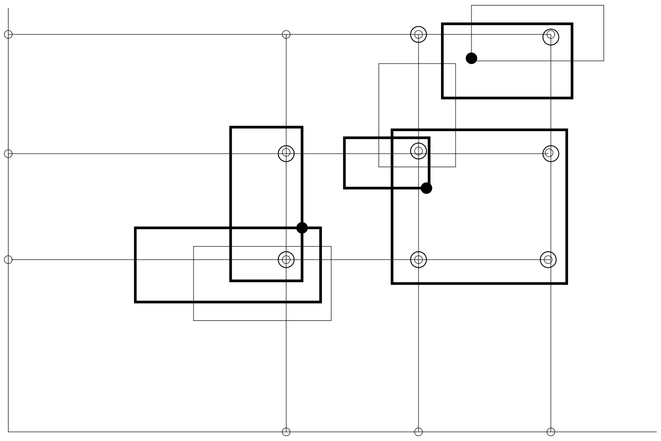


Figure 2: Grid points, representative rectangles and piercing set for a collection of rectangles.

For $X \subseteq H$ we define $S_X \subseteq C$ as follows:

$$S_X = \{r \in C : r \cap H = X\}$$

We note that C is partitioned into the sets S_X, i.e. $C = \dot{\bigcup}_{X \subseteq H} S_X$.

The subset of H 'induced' by a box $r \in C$ will be of the form of a 'rectangular sub block' of H. Any rectangular sub block of H is uniquely determined by its diagonal endpoints. Hence there are at most $\binom{k^d}{2}$ distinct subsets of H induced by boxes. Therefore there are at most $\binom{k^d}{2} \leq k^{2d}$ distinct nonempty S_X.

Let $S' \subseteq C$ be a set of representative boxes obtained by picking exactly one box from each of the nonempty sets S_X, $X \subseteq H$ (the bold rectangles in Figure 2). Note that $|S'| \leq k^{2d}$. Since C has the (k^{2d}, k) property, S' can be pierced by a set of points $W \subset \mathbb{R}^d$, $|W| \leq k$ (the filled points in Figure 2). For $p \in W$, let $N(p)$ denote the set of (at most 2^d) grid points of H which form the gridcell containing p. Let $P = \cup_{p \in W} N(p)$, $|P| \leq 2^d k$ (the big hollow points in Figure 2). If p pierces some box $r \in S_X$, then the points in $N(p)$ pierce all boxes in S_X. Since points in W pierce all the boxes in S', points in P pierce all the boxes in $C = \dot{\bigcup}_{X \subseteq H} S_X$. Thus C is $2^d k$-piercable.

The proof of Theorem 3 directly leads to a 2^d-approximate FPT algorithm for the minimum piercing problem on boxes. Given a collection of boxes C Algorithm 1 returns no if C is not k-piercable and returns a piercing set of size atmost $2^d k$ otherwise.

Obtaining C_i takes $O(dn)$ time. Checking if each C_i is k-piercable takes $O(dn \log n)$ time. Obtaining S' takes $O(dn \log k)$ time. The bruteforce check takes $O(k^{4k})$ time. Hence the whole algorithm takes $O(dn \log n + k^{4k})$ time.

Algorithm 1 FPT algorithm to give a 2^d approximation for piercing boxes in R^d

```
Orthogonally project each box $r \in C$ to the $d$ axes to get a set of intervals $C_i$ for each axis $i$
if All the $C_i$ are $k$-piercable then
  Obtain $S'$
  Bruteforce check if $S'$ is $k$-piercable

  if $S'$ is $k$-piercable then
    return Grid neighbours of piercing set
  else
    return false
  end if
else
  return false
end if
```

5 Conclusion

In this paper we prove that any family of pseudolines with the (k^2+k+1, k)-property is k-piercable. We extend this result for other families of geometric objects with discrete intersection, i.e., polynomial curves and hyperplanes. It is an interesting question to fully charaterise such families of objects for which $g(k,d)$ is finite. We also pose a relaxed variant of this problem as the k-Helly problem and show non-trivial bounds for a family of boxes in $\mathbb{R}^d$. An interesting open problem is to obtain tight bounds on the k-Helly problem for other families of geometric objects in $\mathbb{R}^d$.

References

[1] N. Alon, I. Barany, Z. Furedi, and D. J. Kleitman. Point selections and weak ϵ-nets for convex hulls. *Combinatorics, Probability and Computing*, 1:189–200, 1992.

[2] N. Alon and D. Kleitman. Piercing convex sets and the Hadwiger Debrunner (p,q)-problem. *Advances in Mathematics*, 92:103–112, 1992.

[3] B. Broden, M. Hammar and B. J. Nilsson. Guarding lines and 2-link polygons is APX-hard. *Proceedings of the Canadian Conference on Computational Geometry*, 45–48, 2001.

[4] L. Danzer and B. Grünbaum. Intersection properties of boxes in $\mathbb{R}^d$. *Combinatorica*, 2:237–246, 1982.

[5] L. Danzer, B. Grünbaum, and V. Klee. Helly's theorem and its relatives. *Convexity, American Mathematical Society*,7:101–179, 1963.

[6] J. Eckhoff. Helly, Radon, and Caratheodory type theorems. *Handbook of Convex Geometry*, 389–448, 1993.

[7] H. Hadwiger and H. Debrunner. Uber eine variante zum hellyschensatz. *Archiv der Mathematik*, 8:309–313, 1957.

[8] E. Helly. Uber mengen konvexer kŏrper mit gemeinschaftlichenpunkten, *Jber. Deutsch. Math. Verein.*, 32:175–176, 1923.

[9] M. Katchalski and D. Nashtir. On a conjecture of Danzer and Grünbaum. *American Mathematical Society*, 124:3213-3218, 1996.

[10] S. Langerman and P. Morin. Covering things with things. *Discrete & Computational Geometry*, 33:717–729, 2005.

[11] D. Marx. Efficient approximation schemes for geometric problems? *Proceedings of the 13th annual European Symposium on Algorithms*, 448–459, 2005.

[12] N. Megiddo and A. Tamir. On the complexity of locating linear facilities in the plane. *Operations Rsearch Letters*, 1:194–197, 1982.

Adaptive Techniques to find Optimal Planar Boxes

J. Barbay [*] G. Navarro [†] P. Pérez-Lantero [‡]

Abstract

Given a set P of n planar points, two axes and a real-valued score function $f()$ on subsets of P, the Optimal Planar Box problem consists in finding a box (i.e. an axis-aligned rectangle) H maximizing $f(H \cap P)$. We consider the case where $f()$ is monotone decomposable, i.e. there exists a composition function $g()$ monotone in its two arguments such that $f(A) = g(f(A_1), f(A_2))$ for every subset $A \subseteq P$ and every partition $\{A_1, A_2\}$ of A. In this context we propose a solution for the Optimal Planar Box problem which performs in the worst case $O(n^2 \lg n)$ score compositions and coordinate comparisons, and much less on other classes of instances defined by various measures of difficulty. A side result of its own interest is a fully dynamic *MCS Splay tree* data structure supporting insertions and deletions with the *dynamic finger* property, improving upon previous results [Cortés et al., J.Alg. 2009].

1 Introduction

Consider a set P of n planar points, and two axes x and y forming a base of the plane, such that the points are in general position (i.e. no pair of points share the same x or y coordinate). We say that a real-valued function $f()$ on subsets of P is *decomposable* [2, 7] if there exists a *composition function* $g()$ such that $f(A) = g(f(A_1), f(A_2))$ for every subset $A \subseteq P$ and every partition $\{A_1, A_2\}$ of A. Without loss of generality, we extend $f()$ to P such that $f(p) = f(\{p\})$. A decomposable function is *monotone* if the corresponding composition function $g()$ is monotone in its two arguments. A *box* is a rectangle aligned to the axes, and given a monotone decomposable function $f()$, such a box is $f()$-*optimal* if it optimizes $f(H \cap P)$. Without loss of generality, we assume that we want to maximize $f()$ and that its composition function $g()$ is monotone increasing in its two arguments. Given a monotone decomposable function $f()$ well defined for the empty set $\varnothing$, a point p of P is *positive* if $f(p) > f(\varnothing)$. Otherwise, this point p is *negative*. Observe that if p is positive then $f(A \cup \{p\}) = g(f(A), f(p)) > g(f(A), f(\varnothing)) = f(A)$ by monotonicity of $g()$: hence a point p is positive if and only if $f(A \cup \{p\}) > f(A)$ for every subset $A \subset P$ not containing p. A *stripe* is an area delimited by two lines parallel to the same axis. A *positive stripe* (resp. *negative stripe*) is one which contains only positive (resp. negative) points. A *monochromatic stripe* is a stripe in which all points have the same sign.

Given a set of planar points, a simple example of such monotone decomposable functions is counting the number of points the box contains. Other examples include counting the number of blue points; returning the difference between the number of blue points and the number of red points contained; returning the number of blue points in the box or $-\infty$ if it contains some red points; summing the weights of the points contained; taking the maximum of the weights of contained points; etc.

Given a set P of n planar points and a real-valued function $f()$ on subsets of P, the Optimal Planar Box problem consists in finding an $f()$-optimal box. Depending on $f()$, this problem has various practical applications, from identifying rectangular areas of interest in astronomical pictures to the design of optimal rectangular forest cuts or the analysis of medical radiographies. We present various adaptive techniques for the Optimal Planar Box problem:

- In the worst case over instances composed of n points, our algorithm properly generalizes Cortés et al.'s solution [5] for the Maximum Weight Box problem, within the same complexity of $O(n^2 \lg n)$ score compositions.
- For any $\delta \in [1..n]$ and $n_1, \dots, n_\delta \in [1..n]$ summing to n, in the worst case over instances composed of δ monochromatic stripes of alternating signs when the points are sorted by their y-coordinates, such that the i-th stripe contains n_i points, our algorithm executes $O(\delta n(1 + \mathcal{H}(n_1, \dots, n_\delta))) \subset O(\delta n \lg(\delta + 1))$ score compositions (Theorem 4), where $\mathcal{H}(n_1, \dots, n_\delta) = \sum_{i=1}^{\delta} (n_i/n) \lg(n/n_i)$ is the usual entropy function.
- Assuming the same y-coordinate order, for any $\lambda \in [0..n^2]$, in the worst case over instances where

[*]Department of Computer Science, University of Chile, Chile, jeremy.barbay@dcc.uchile.cl. Partially funded by grant FONDECYT 1-120054, Chile.

[†]Department of Computer Science, University of Chile, Chile, gnavarro@dcc.uchile.cl. Partially funded by Millennium Nucleus Information and Coordination in Networks ICM/FIC P10-024F, Mideplan, Chile.

[‡]Escuela de Ingeniería Civil en Informática, Universidad de Valparaíso, Chile, pablo.perez@uv.cl. Partially supported by grant FONDECYT 11110069 (Chile) and project MEC MTM2009-08652 (Spain).

λ is the sum of the distances between the insertion positions of the consecutive points according to their x-coordinate, our algorithm makes $O(n^2(1+\lg(1+\lambda/n))$ score compositions (Lemma 5). Measure λ relates to the local insertion sort complexity [11] of the sequence of x-coordinates. It holds $\lambda \in O(n+\texttt{Inv})$, where $\texttt{Inv}$ is the number of inversions in the sequence. When the points are grouped into δ monochromatic stripes, the complexity drops to $O(n\delta(1+\lg(1+\texttt{Inv}/n))$ (Theorem 7).

- Assuming the same y-coordinate order, for a minimal cover of the same sequence of x-coordinates into $\rho \leq n$ *runs* (i.e. contiguous increasing subsequences) of lengths $r_1, \dots, r_\rho$, our algorithm executes $O(n^2(1+\mathcal{H}(r_1,\dots,r_\rho))) \subset O(n^2\lg(\rho+1))$ score compositions (Lemma 6). When the points can be grouped into δ monochromatic stripes, this complexity decreases to $O(n\delta(1+\mathcal{H}(r_1,\dots,r_\rho))) \subset O(n\delta\lg(\rho+1))$ (Theorem 7 again).

- Using an approach orthogonal to the one of Cortés et al. [5], we partition (via a clever strategy considering axis-parallel lines) the point set P into subsets called diagonal blocks, so that a new adaptive algorithm is obtained (Theorem 14). The algorithm solves the Optimal Planar Box problem in each block and combine the solutions. Extending this algorithm, we obtain another adaptive algorithm running in $O(n\lg n+\sigma n)$ comparisons and $O(\sigma n\lg n)$ score compositions, where $\sigma \in [1..n]$ is a measure of difficulty of the instance that depends on the partition in diagonal blocks (Theorem 18).

Due to the lack of space, several proofs are omitted. A longer version of this paper is available at `http://arxiv.org/abs/1204.2034`.

2 Optimal Boxes and Related Problems

Given a set P of n weighted planar points, in which the weight of a point can be either positive or negative, the Maximum Weight Box problem [5] consists in finding a box R maximizing the sum of the weights of the points in $R \cap P$. Cortés et al. [5] gave an algorithm solving this problem in time $O(n^2\lg n)$ using $O(n)$ space, based on *MCS trees*, a data structure supporting in $O(\lg n)$ time the dynamic computation of the Maximum-Sum Consecutive Subsequence problem [3] (hence the name "MCS").

The Maximum Weight Box problem [5] and, by successive reductions, the Maximum Subarray problem [14], the Maximum Box problem [5, 8, 10], and the Maximum Discrepancy Box problem [5, 6] can all be reduced to a finite number of instances of the Optimal Planar Box problem by choosing adequate definitions for the score functions $f()$ to optimize.

Cortés et al.'s algorithm [5] first sorts the points by their y-coordinate in $O(n\lg n)$ time and then traverses the resulting sequence of points $p_1, p_2, \dots p_n$ as follows. For each p_i, it sets an MCS tree (described in more details in Section 3) with points $p_i, \dots p_n$, where the key is their x-coordinate x_i, and all have value $f(\varnothing)$. It then successively activates points p_j for $j \in [i..n]$, setting its weight to value $f(p_j)$, updating in time $O(\lg n)$ the MCS tree so that to compute the optimal box contained between the y-coordinate of p_i to that of p_j. The whole algorithm executes in time $O(n^2\lg n)$.

3 Fully Dynamic MCS Trees

The MCS tree [5] is an index for a fixed sequence $S=(x_i)_{i\in[1..n]}$ of n elements, where each element x_k of S has a weight $w(x_k) \in \mathbb{R}$, so that whenever a weight $w(x_k)$ is updated, a consecutive subsequence $(x_i)_{i\in[l..r]}$ of S maximizing $\sum_{i\in[l..r]} w(x_i)$ is obtained (or recomputed) in $O(\lg n)$ time. This behavior is dynamic in the sense that it allows modification of element weights, yet it is only partially dynamic in the sense that it admits neither addition nor deletion of elements.

Existing dynamic data structures can be easily adapted into a truly dynamic data structure with the same functionalities as MCS trees. We start by generalizing MCS trees [5] from mere additive weights to monotone decomposable score functions in Lemma 1. We further generalize this solution to use an AVL tree [1] in Lemma 2 and a Splay tree [13] in Lemma 3, whose "finger search" property will play an essential role in the results of Sections 4 and 5.

Lemma 1 *Let S be a static sequence of n elements, and $f()$ be a monotone decomposable score function receiving as argument any subsequence of S, defined through the activation and deactivation of each element of S. There exists a semi-dynamic data structure for maintaining S using linear space that supports the search for an element in $O(\lg n)$ comparisons; the activation or deactivation of an element in $O(\lg n)$ score compositions; and $f()$-optimal sub range queries in $O(\lg n)$ comparisons and score compositions.*

The MCS tree data structure can be converted into a truly dynamic data structure supporting both insertions and deletions of elements. This data structure can be used to index a dynamic sequence $S=(x_i)_{i\in[1..n]}$ of n elements so that whenever an element is inserted or removed, a consecutive subsequence $S'=(x_i)_{i\in[l..r]}$ of S optimizing $f(S')$ can be (re)computed in $O(\lg n)$ score compositions and comparisons. The following lemma establishes the property of this data structure, which we call *MCS AVL tree*.

Lemma 2 *Let S be a dynamic sequence of n elements, and $f()$ be a monotone decomposable score function receiving as argument any consecutive subsequence of S. There exists a fully dynamic data structure for maintaining S using linear space that supports the search for an element in $O(\lg n)$ comparisons; the update of the score of an element in $O(\lg n)$ score compositions, the insertion or deletion of an element in $O(\lg n)$ comparisons and score compositions; and $f()$-optimal subrange queries in $O(\lg n)$ comparisons and score compositions.*

The Splay tree is a self-adjusting binary search tree created by Sleator and Tarjan [13]. It supports the basic operations search, insert and delete, all of them called *accesses*, in $O(\lg n)$ amortized time. For many sequences of accesses, splay trees perform better than other search trees, even when the specific pattern of the sequences are unknown. Among other properties of Splay trees, we are particularly interested in the *Dynamic Finger Property*, conjectured by Sleator and Tarjan [13] and proved by Cole et al. [4]: every sequence of m accesses on an arbitrary n-node Splay tree costs $O(m+n+\sum_{j=1}^{m}\lg(d_j+1))$ rotations where, for $j = 1..m$, the j-th and $(j-1)$-th accesses are performed on elements whose ranks among the elements stored in the Splay tree differ by d_j. For $j = 0$, the j-th element is the element stored at the root. It is easy to see that in the MCS AVL tree we can replace the underlying AVL tree by a Splay tree, and obtain then the next lemma, which describes the *MCS Splay tree* data structure.

Lemma 3 *Let S be a dynamic sequence of n elements and $f()$ be a monotone decomposable function receiving as argument any consecutive subsequence of S. There exists a data structure for maintaining S that uses linear space and supports the search in $O(\lg n)$ amortized comparisons, the update of the score of an element in $O(\lg n)$ amortized score compositions, and the insertion and deletion of elements in $O(\lg n)$ amortized comparisons and score compositions. Joint with the insertion or deletion of any element, the consecutive subsequence S' of S maximizing $f(S')$ is recomputed. The Dynamic Finger Property is also satisfied for each operation (search, insertion and deletion), both for the number of comparisons and for the number of score compositions performed.*

4 Taking Advantage of Monochromatic Stripes

Consider an instance where positive and negative points can be clustered into δ positive and negative stripes along one given axis, of cardinalities $n_1, \dots, n_\delta$. Such stripes can be easily identified in $O(n \lg n)$ comparisons and $O(n)$ score accesses. On such instances one does not need to consider boxes whose borders are in the middle of some stripes: all optimal boxes will start at the edge of a stripe; specifically, the top (resp. bottom) of an optimal box will align with a positive point at the top (resp. bottom) of a positive stripe.

This very simple observation not only limits the number of boxes for which we need to compute a score, but also it makes it easier to compute the score of each box: adding the n_i points of the i-th stripe in increasing order of their coordinates in a MCS Splay tree of final size n amortizes to $O(n+\sum_{i=1}^{\delta} n_i \lg(n/n_i))$ coordinate comparisons and score compositions. The reason is that the n_i distances $d_j + 1$ of Lemma 3 telescope to at most $n + n_i$ within stripe i, and thus by convexity the cost $O(n + \sum_{j=1}^{n}\lg(d_j + 1))$ is upper bounded by $O(n + \sum_{i=1}^{\delta} n_i \lg(1 + n/n_i))$ which is $O(n + \sum_{i=1}^{\delta} n_i \lg(n/n_i)) = O(n(1 + \mathcal{H}(n_1, \dots, n_\delta))) \subset O(n \lg(\delta + 1))$. Combining this with the fact that the top of an optimal box is aligned with a positive point at the top of a positive stripe yields the following result.

Theorem 4 *For any $\delta \in [1..n]$ and $n_1, \dots, n_\delta \in [1..n]$ summing to n, in the worst case over instances composed of δ stripes of alternating signs over an axis such that the i-th stripe contains n_i points, there exists an algorithm that finds an $f()$-optimal box in $O(\delta n(1 + \mathcal{H}(n_1, \dots, n_\delta))) \subset O(\delta n \lg(\delta + 1))$ score compositions and $O(\delta n(1 + \mathcal{H}(n_1, \dots, n_\delta)) + n \lg n) \subset O(\delta n \lg(\delta + 1) + n \lg n)$ coordinate comparisons.*

5 Taking Advantage of Point Alignments

Running the algorithm outlined in the first paragraph of Section 4 over the MCS Splay tree has further consequences. In this section we show how it makes the algorithm adaptive to local point alignments.

The cost of our algorithm using the MCS Splay tree can be upper bounded as follows. Let λ denote the sum of the distances between the insertion positions of the consecutive points according to their x-coordinate. When we insert the points in the MCS Splay tree starting from p_1, the total cost is $O(n + \sum_{j=1}^{n}\lg(d_j + 1)) \subset O(n + n\lg(1 + \lambda/n))$ score compositions, by convexity of the logarithm and because $\sum_{j=1}^{n} d_j + 1 \leq \lambda + n$. A simple upper bound when considering all the n passes of the algorithm can be obtained as follows.

Lemma 5 *There exists an algorithm that finds an $f()$-optimal box in $O(n^2(1+\lg(1+\lambda/n)))$ score compositions and $O(n^2(1 + \lg(1 + \lambda/n)) + n \lg n)$ coordinate comparisons, where $\lambda \leq n^2$ is the local insertion complexity of the sequence of x-coordinates of the points sorted by y-coordinates.*

In the worst case this boils down to the $O(n^2 \lg n)$-worst-case algorithm, whereas in the best case $\lambda = 0$ and the cost corresponds to $O(n^2)$ operations.

We can upper bound λ by using other two measures of disorder in permutations. For example, let us consider `Inv`, the number of *inversions* in the permutation π, or said another way, the number of pairs out of order in the sequence [12]. The measure `Inv` corresponds to a cost where the "finger" is always at the end of the sequence. This can be as small as $(\lambda - n)/2$, for example consider the permutation $\pi = (m, m-1, m+1, m-2, m+2, \ldots, 1, 2m-1)$ for $m = (n+1)/2$ and odd n. However, `Inv` can be much larger than λ because it is not symmetric on decreasing sequences, for example when the points are semi-aligned in a decreasing diagonal and the permutation is $\pi = (n, n-1, n-2, \ldots, 1)$. Thus replacing λ by `Inv` in Lemma 5 yields a valid upper bound in terms of big-O complexity.

Another well-known measure of permutation complexity is the number of *increasing runs* ρ, that is, the minimum number of contiguous monotone increasing subsequences that cover π [9]. Let $r_1, \ldots, r_\rho$ be the lengths of the runs, computed in $O(n \lg n)$ comparisons. Then the sum of the values $|\pi_{j+1} - \pi_j|$ within the i-th run telescopes to at most n, and so does the sum of the d_j values. Therefore $\sum_{j=1}^{n} \lg(d_j + 1) \leq \sum_{i=1}^{\rho} r_i \lg(1+n/r_i) \leq n + \sum_{i=1}^{\rho} r_i \lg(n/r_i)$ by convexity. This leads to the following alternative upper bound.

Lemma 6 *There exists an algorithm that finds an $f()$-optimal box in $O(n \lg n)$ coordinate comparison and $O(n^2(1+\mathcal{H}(r_1, \ldots, r_\rho)) \subset O(n^2 \lg(\rho+1))$ score compositions, where $r_1, \ldots, r_\rho$ are the lengths of ρ maximal contiguous increasing subsequences that cover the sequence of x-coordinates of the points sorted by y-coordinate.*

6 Taking Advantage of both Stripes and Alignments

The combination of the techniques of Sections 4 and 5 can be elegantly analyzed. A simple result is that we need to start only from δ different p_i values, and therefore an upper bound to our complexity is $O(n\delta((1 + \lg(1 + \lambda/n)))$. We can indeed do slightly better by sorting the points by increasing x-coordinates within each monochromatic stripe. While the measure λ' resulting from this reordering may be larger than λ, the upper bounds related to `Inv` and ρ, namely `Inv'`, ρ', and $\mathcal{H}(n'_1, \ldots, n'_{\rho'})$, do not increase. In particular it is easy to see that the upper bound of Theorem 4 is dominated by the combination since $\rho' \leq \delta$ and $\mathcal{H}(r'_1, \ldots, r'_{\rho'}) \leq \mathcal{H}(n_1, \ldots, n_\delta)$ (because no run will cut a monochromatic stripe once the latter is reordered).

Theorem 7 *There exists an algorithm that finds an $f()$-optimal box in $O(n \lg n)$ coordinate comparisons and $O(n\delta(1 + \min(\lg(1 + \texttt{Inv}/n), \mathcal{H}(r_1, \ldots, r_\rho)))) \subset O(n\delta \lg(\rho + 1))$ score compositions, where δ is the minimum number of monochromatic stripes in which the points, sorted by increasing y-coordinate, can be partitioned; X is the corresponding sequence of x-coordinates once we (re-)sort by increasing x-coordinate the points within each monochromatic stripe;* `Inv` $\leq n^2$ *is the number of out-of-order pairs in X; and $r_1, \ldots, r_\rho$ are the lengths of the minimum number $\rho \leq \delta$ of contiguous increasing runs that cover X. A similar result holds by exchanging x and y axes.*

Note that if these new measures are not particularly favorable, the formula boils down to the $O(n\delta \lg \delta)$ time complexity of Section 4.

7 Taking Advantage of Diagonals of Blocks

In this section we present an approach orthogonal to the previous ones, which considers partitions of the point set into subsets and yields to a new adaptive algorithm which solves the Optimal Planar Box problem in each of them and combine the solutions. Its difficulty measure depends on such a partition.

For any subset $A \subseteq P$, a *diagonalization* of A is a partition $\{A_1, A_2\}$ of A induced by two lines ℓ_1 and ℓ_2, respectively parallel to axes x and y, so that the elements of A_1 and the elements of A_2 belong to opposite quadrants with respect to the point $\ell_1 \cap \ell_2$. Note that if $p_1, p_2, \ldots, p_m$ denote the elements of A sorted by x-coordinate, then any diagonalization of A has the form $\{\{p_1, \ldots, p_k\}, \{p_{k+1}, \ldots, p_m\}\}$ for some index $k \in [1..m-1]$. Not all point sets admit a diagonalization, the simplest case consists of four points placed at the four corners of a square whose sides are slightly rotated from the axes. We call such a point set a *windmill*, due to the characteristic position of its points. Given any bounded set $S \subset \mathbb{R}^2$, let $\texttt{Box}(S)$ denote the smallest box enclosing S and let the *extreme* points of any subset $A \subseteq P$ be those belonging to the boundary of $\texttt{Box}(A)$.

Lemma 8 *Let A be a point set that does not admit a diagonalization. Then A has exactly four extreme points. Furthermore, A has a windmill which contains at least one extreme point of A.*

Definition 9 *A diagonalization tree of P, D-tree, is a binary tree such that: (i) each leaf u contains a subset $S(u) \subseteq P$ which does not admit a diagonalization, (ii) set $\{S(u) \mid u \text{ is a leaf}\}$ is a partition of P, and (iii) each internal node v has exactly two children v_1 (the left one) and v_2 (the right one) and satisfies that $\{A(v_1), A(v_2)\}$ is a diagonalization of $A(v)$, where for each node v $A(v)$ denotes the union of the sets $S(u)$ for all leaves u descendant of v (See Figure 1).*

Lemma 10 *Let P be a set of n points in the plane. Every D-tree of P has the same number of leaves. Furthermore, the i-th leaves from left to right of any two D-trees of P contain the same subset $S(\cdot)$ of P.*

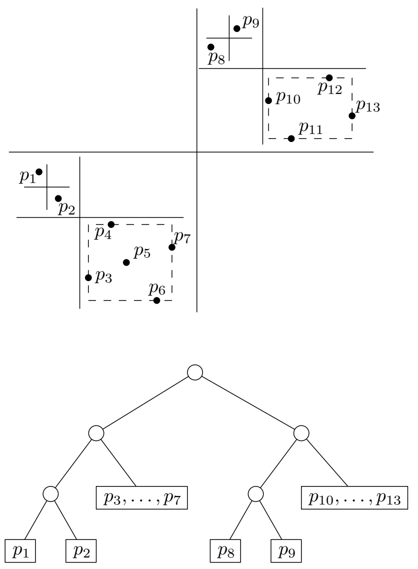


Figure 1: A D-tree of the point set $\{p_1,\dots,p_{13}\}$.

From Lemma 10 we can conclude that every D-tree T of P induces the same partition $\{S(u_1),\dots,S(u_\beta)\}$ of P, where $u_1,\dots,u_\beta$ are the leaf nodes of T.

Lemma 11 *A D-tree of P requires $O(n)$ space and can be built in $O(n\lg n)$ comparisons.*

Proof. (Sketch) Let $p_1,p_2,\dots,p_n$ be the elements of P sorted by x-coordinate, and let $p_{\pi_1},p_{\pi_2},\dots,p_{\pi_n}$ be the elements of P sorted by y-coordinate. Considering the computation of permutation π a preprocessing, we can show that: If P admits a diagonalization $\{\{p_1,\dots,p_k\},\{p_{k+1},\dots,p_n\}\}$ then it can be determined in $O(\min\{k,n-k\})$ comparisons. Otherwise, if P does not admit a diagonalization, then it can be decided in $O(n)$ comparisons. We can then build a D-tree of P recursively as follows. If a diagonalization $\{\{p_1,\dots,p_k\},\{p_{k+1},\dots,p_n\}\}$ of P exists, which was determined in $O(t)$ comparisons where $t=\min\{k,n-k\}$, then create a root node and set as left child a D-tree of $\{p_1,\dots,p_k\}$ and as right child a D-tree of $\{p_{k+1},\dots,p_n\}$. Otherwise, if P does not admit a diagonalization, which was decided in $O(n)$ comparisons, then create a leaf node whose set $S(\cdot)$ is equal to P. This results in the next recurrence equation for the total number $T(n)$ of comparisons, where $1\le t\le\lfloor n/2\rfloor$:

$$T(n)=\begin{cases} O(t)+T(t)+T(n-t) & n>1\text{, a diagonalization exists.}\\ O(n) & \text{otherwise.}\end{cases}$$

By applying induction and using the binary entropy function $H(x)=x\lg(1/x)+(1-x)\lg(1/(1-x))$, with the fact $x\le H(x)$ for $x\le 1/2$, it can be proved that $T(n)$ is $O(n\lg n)$. $\square$

Definition 12 *For any non-empty subset $A\subseteq P$ the set of ten $f()$-optimal boxes of A, denoted by $\texttt{Ten}(A)$, consists of the following $f()$-optimal boxes of A, all contained in $\texttt{Box}(A)$:*

1. *$\texttt{Box}(A)$;*
2. *$\texttt{B}_{opt}(A)$, without restriction;*
3. *$\texttt{B}_1(A)$, with the bottom-left vertex of $\texttt{Box}(A)$;*
4. *$\texttt{B}_2(A)$, with the bottom-right vertex of $\texttt{Box}(A)$;*
5. *$\texttt{B}_3(A)$, with the top-right vertex of $\texttt{Box}(A)$;*
6. *$\texttt{B}_4(A)$, with the top-left vertex of $\texttt{Box}(A)$;*
7. *$\texttt{B}_{1,2}(A)$, with the bottom vertices of $\texttt{Box}(A)$;*
8. *$\texttt{B}_{2,3}(A)$, with the right vertices of $\texttt{Box}(A)$;*
9. *$\texttt{B}_{3,4}(A)$, with the top vertices of $\texttt{Box}(A)$;*
10. *$\texttt{B}_{4,1}(A)$, with the left vertices of $\texttt{Box}(A)$.*

Lemma 13 *For any non-empty subset $A\subseteq P$ and any diagonalization $\{A_1,A_2\}$ of A, $\texttt{Ten}(A)$ can be computed in $O(1)$ score compositions from $\texttt{Ten}(A_1)$ and $\texttt{Ten}(A_2)$.*

Theorem 14 *There exists an algorithm that finds an $f()$-optimal box of P in $O(n\lg n+\sum_{i=1}^{\beta}h_c(n_i))$ comparisons (on coordinates and indices) and $O(\sum_{i=1}^{\beta}h_s(n_i)+\beta)$ score compositions, where $\{P_1,\dots,P_\beta\}$ is the partition of P induced by any D-tree of P and β is the size of this partition, n_i is the cardinality of P_i, and $h_c(n_i)$ and $h_s(n_i)$ are the numbers of coordinate comparisons and score compositions used, respectively, to compute the ten $f()$-optimal boxes of P_i.*

Proof. Build a D-tree T of P in $O(n\lg n)$ comparisons (Lemma 11). Let $u_1,\dots,u_\beta$ be the leaves of T which satisfy $S(u_i)=P_i$ for all $i\in[1..n]$. Compute the set $\texttt{Ten}(S(u_i))=\texttt{Ten}(P_i)$ in $h_c(n_i)$ coordinate comparisons and $h_s(n_i)$ score compositions. By using a post-order traversal of T, for each internal node v of T compute $\texttt{Ten}(A(v))$ from $\texttt{Ten}(A(v_1))$ and $\texttt{Ten}(A(v_2))$, where v_1 and v_2 are the children nodes of v, in $O(1)$ score compositions (Lemma 13). The $f()$-optimal box of P is the box $\texttt{B}_{opt}(A(r))$, where r is the root node of T and satisfies $A(r)=P$. In total, this algorithm runs in $O(n\lg n)+\sum_{i=1}^{\beta}h_c(n_i)=O(n\lg n+\sum_{i=1}^{\beta}h_c(n_i))$ coordinate comparisons and $\sum_{i=1}^{\beta}h_s(n_i)+\sum_{i=1}^{\beta-1}O(1)=O(\sum_{i=1}^{\beta}h_s(n_i)+\beta)$ score compositions. $\square$

Corollary 15 *There exists an algorithm that finds an $f()$-optimal box of P in $O(n\lg n+\sum_{i=1}^{\beta}n_i\lg n_i)$ comparisons and $O(\sum_{i=1}^{\beta}n_i^2\lg n_i+\beta)$ score compositions, where β is the size of the partition $\{P_1,\dots,P_\beta\}$ of P induced by any D-tree of P, and $n_i=|P_i|$ for all i.*

8 Dealing with Windmills

In this section we use Lemma 8 to obtain a variant of the algorithm in Theorem 14. The set $S(u)$ of every leaf node u of any D-tree of P does not admit a diagonalization and has a windmill containing an extreme point of $S(u)$. The idea is to remove the extreme points of $S(u)$ and then recursively build a D-tree of the remaining points. This approach yields a diagonalization in depth of the point set, potentially reducing the number of score compositions.

Definition 16 *An extended diagonalization tree of P, D^*-tree, is defined recursively as follows: Each leaf node u of a D-tree of P satisfying $|S(u)| > 1$ is replaced by a node u' containing the set $X(u)$ of the four extreme points of $S(u)$, and if the set $S(u) \setminus X(u)$ is not empty then u' has as its only one child a D^*-tree of $S(u) \setminus X(u)$.*

Lemma 17 *Every D^*-tree of P has the same number σ of one-child nodes, contains $n - 4\sigma$ leaves nodes, and every leaf node u satisfies $|S(u)| = 1$ or $|S(u)| = 4$. A D^*-tree of P requires $O(n)$ space and can be built in $O(n \lg n + \sigma n)$ comparisons.*

Proof. The first part of the lemma can be seen from Lemma 10 and Definition 16. A D^*-tree of P can be built in $O(n \lg n + \sigma n)$ comparisons by following the same algorithm to build a D-tree of P until finding a leaf node u such that $S(u)$ does not admit a diagonalization. At this point we pay $O(n)$ comparisons in order to continue the algorithm with the set $S(u) \setminus X(u)$ according to Definition 16. Since this algorithm finds σ nodes u, the total comparisons are $O(n \lg n + \sigma n)$. The D^*-tree has n nodes of bounded degree and hence can be encoded in linear space. □

Theorem 18 *There exists an algorithm that finds an $f()$-optimal box of P in $O(n \lg n + \sigma n)$ coordinate comparisons and $O(\sigma n \lg n)$ score compositions, where σ is the number of one-child nodes of every D^*-tree of P.*

Proof. Build a D^*-tree T of P in $O(n \lg n + \sigma n)$ comparisons (Lemma 17). For each of the $n - 4\sigma$ leaves nodes u of T compute $\texttt{Ten}(S(u))$ in constant score compositions. Then, using a post-order traversal of T, compute $\texttt{Ten}(S(u))$ for each internal node u as follows: If v has two children v_1 (the left one) and v_2 (the right one), then $\{A(v_1), A(v_2)\}$ is a diagonalization of $A(v)$ and $\texttt{Ten}(A(v))$ can be computed in $O(1)$ score compositions from $\texttt{Ten}(A(v_1))$ and $\texttt{Ten}(A(v_2))$ (Lemma 13). Otherwise, if v is one of the σ one-child nodes, then $\texttt{Ten}(A(v))$ can be computed in $O(n \lg n)$ worst-case comparisons and score compositions. Namely, if a box of $\texttt{Ten}(A(v))$ contains a at least one point of $X(u)$ in the boundary then it can be found in $O(n \lg n)$ comparisons and score compositions [5]. Otherwise, it is a box of $\texttt{Ten}(A(v'))$, where v' is the child of v. We pay $O(1)$ score compositions for each of the $O(n)$ two-child nodes and $O(n \lg n)$ score compositions for each of the σ one-child nodes. Then the total score compositions is $O(n + \sigma n \lg n)$. □

9 Future work

The definition of $\texttt{Ten}(\cdot)$ can be used for further results: Suppose the point set P can be partitioned into subsets $P_1, P_2, \ldots, P_k$ so that bounding boxes $\texttt{Box}(P_1), \texttt{Box}(P_2), \ldots, \texttt{Box}(P_k)$ are pairwise disjoint and any axis-parallel line stabs at most one of them. Once $\texttt{Ten}(P_1), \texttt{Ten}(P_2), \ldots, \texttt{Ten}(P_k)$ have been computed, an optimal box of P can be found in $O(k \lg k)$ comparisons and $O(k^2 \lg k)$ score compositions. Finding an optimal decomposition $P_1, P_2, \ldots, P_k$ is our main issue.

References

[1] G. Adelson-Velskii and E. M. Landis. An algorithm for the organization of information. In *Proc. of the USSR Academy of Sciences*, volume 146, pages 263–266, 1962.

[2] C. Bautista-Santiago, J. M. Díaz-Báñez, D. Lara, P. Pérez-Lantero, J. Urrutia, and I. Ventura. Computing optimal islands. *Oper. Res. Lett.*, 39(4):246–251, 2011.

[3] J. Bentley. Programming pearls: algorithm design techniques. *Commun. ACM*, 27(9):865–873, 1984.

[4] R. Cole, B. Mishra, J. Schmidt, and A. Siegel. On the dynamic finger conjecture for splay trees. Part I: Splay sorting log n-block sequences. *SIAM J. Comp.*, 30(1):1–43, 2000.

[5] C. Cortés, J. M. Díaz-Báñez, P. Pérez-Lantero, C. Seara, J. Urrutia, and I. Ventura. Bichromatic separability with two boxes: A general approach. *J. Algorithms*, 64(2-3):79–88, 2009.

[6] D. P. Dobkin, D. Gunopulos, and W. Maass. Computing the maximum bichromatic discrepancy, with applications to computer graphics and machine learning. *J. Comput. Syst. Sci.*, 52(3):453–470, 1996.

[7] D. P. Dobkin and S. Suri. Dynamically computing the maxima of decomposable functions, with applications. In *FOCS*, pages 488–493, 1989.

[8] J. Eckstein, P. Hammer, Y. Liu, M. Nediak, and B. Simeone. The maximum box problem and its application to data analysis. *Comput. Optim. App.*, 23(3):285–298, 2002.

[9] D. E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, 1968.

[10] Y. Liu and M. Nediak. Planar case of the maximum box and related problems. In *CCCG*, pages 14–18, 2003.

[11] H. Mannila. Measures of presortedness and optimal sorting algorithms. In *IEEE Trans. Comput.*, volume 34, pages 318–325, 1985.

[12] A. Moffat and O. Petersson. An overview of adaptive sorting. *Australian Comp. J.*, 24(2):70–77, 1992.

[13] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.

[14] T. Takaoka. Efficient algorithms for the maximum subarray problem by distance matrix multiplication. *Electronic Notes in Theoretical Computer Science*, 61:191–200, 2002. CATS'02.

A Fast Dimension-Sweep Algorithm for the Hypervolume Indicator in Four Dimensions

Andreia P. Guerreiro*† Carlos M. Fonseca† Michael T. M. Emmerich‡

Abstract

The Hypervolume Indicator is one of the most widely used quality indicators in Evolutionary Multiobjective Optimization. Its computation is a special case of Klee's Measure Problem (KMP) where the upper end of all rectangular ranges coincides with a given reference point (assuming minimization, without loss of generality). Although the time complexity of the hypervolume indicator in two and three dimensions is known to be $\Theta(n \log n)$, improving upon the $O(n^{d/2} \log n)$ complexity of Overmars and Yap's algorithm for the general KMP in higher dimensions has been a challenge. In this paper, a new dimension-sweep algorithm to compute the hypervolume indicator in four dimensions is proposed, and its complexity is shown to be $O(n^2)$.

1 Introduction

In multiobjective optimization, solutions may be seen as points in a decision space, $\mathcal{S}$, which are mapped onto a multi-dimensional objective space, $\mathbb{R}^d$, by means of a vector-valued objective function, $f : \mathcal{S} \rightarrow \mathbb{R}^d$. Minimization of all objective function components is assumed throughout this work without loss of generality.

In this context, a solution $x \in \mathcal{S}$ is said to dominate another solution $z \in \mathcal{S}$ iff $f(x) \leq f(z)$ and $f(x) \neq f(z)$, where the inequality $\leq$ applies componentwise. A solution $x \in \mathcal{S}$ is said to be Pareto-optimal iff $\forall z \in \mathcal{S}, f(z) \leq f(x) \Rightarrow f(z) = f(x)$. The set of all Pareto-optimal solutions in decision space is called the *Pareto-optimal set*, and the corresponding set of points in objective space is called the *Pareto-optimal front.*

Since enumerating the whole Pareto-optimal set, or even the Pareto-optimal front, is usually infeasible, multiobjective optimization typically aims at finding a good, discrete approximation to the Pareto-optimal front. In comparative studies, the quality of such approximations is often assessed in a quantitative manner by means of quality indicators, which map a set of points in objective space to a real value. Such quality indicators have also been integrated into some evolutionary multiobjective optimizers [3, 14], which leads to the quality indicator being evaluated many times in the course of an optimization run, and imposes a need for efficient algorithms to compute it.

One of the most widely used quality indicators is the hypervolume indicator [18]. Given a set of points $\mathrm{X} \subset \mathbb{R}^d$ and a reference point $r \in \mathbb{R}^d$, the hypervolume indicator $H(\mathrm{X})$ is the Lebesgue measure, $\lambda(\cdot)$, of the region dominated by X and bounded above by r, i.e. $H(\mathrm{X}) = \lambda(\{q \in \mathbb{R}^d \mid \exists p \in \mathrm{X} : p \leq q \wedge q \leq r\})$. Alternatively, the hypervolume indicator may be written as the measure of the union of n isothetic hyperrectangles in d dimensions:

$$H(\mathrm{X}) = \lambda\left(\bigcup_{\substack{p \in \mathrm{X} \\ p \leq r}} [p, r]\right)$$

where $[p, r] = \{q \in \mathbb{R}^d \mid p \leq q \wedge q \leq r\}$, which highlights its connection to Klee's Measure Problem (KMP) [13]. Furthermore, given a point $p \in \mathrm{X}$ and a reference point $r \in \mathbb{R}^d$, the individual contribution of p is the Lebesgue measure of the region exclusively dominated by p. It can be obtained by subtracting $H(\mathrm{X} - \{p\})$ from $H(\mathrm{X})$.

It is known that the hypervolume indicator and, thus, Klee's measure problem cannot be computed exactly in time polynomial in the number of dimensions unless P = NP [6]. While asymptotically optimal, $O(n \log n)$-time algorithms for the hypervolume indicator in two and three dimensions are known [2], no tight complexity bounds are available for $d \geq 4$. For a long time, the best upper bounds for $d \geq 4$ stemmed from algorithms for the more general KMP. Chan's algorithm [8], with time bound $O(n^{d/2}\, 2^{O(\log^* n)})$, where $\log^*$ denotes the iterated logarithm function, provides the best general upper bound to date. It slightly improves upon Overmars and Yap's algorithm, which has time complexity $O(n^{d/2} \log n)$ [13]. Beume [1] developed a simplified version of Overmars and Yap's algorithm for the hypervolume indicator, but with the same complexity.[1]

*INESC-ID, Instituto Superior Técnico, Technical University of Lisbon, Av. Rovisco Pais 1, 1049-001 Lisboa, Portugal apg@dei.uc.pt

†CISUC, Department of Informatics Engineering, University of Coimbra, Pólo II, 3030-290 Coimbra, Portugal cmfonsec@dei.uc.pt

‡LIACS, Faculty of Science, Leiden University, P.O. Box 9512, 2300 RA Leiden, The Netherlands emmerich@liacs.nl

[1]A gap in the analysis presented in N. Beume and G. Rudolph, *Faster S-Metric Calculation by Considering Dominated Hypervolume as Klee's Measure Problem*, in Proc. 2nd IASTED Conf. on Comp. Intelligence, 231–236, 2006, is acknowledged in [1].

Better bounds have been obtained for the KMP on unit cubes [7] and on fat boxes [5], for example, but reducing the hypervolume indicator to such problems is not possible in general. Only very recently has a tighter, general upper bound of $O(n^{(d-1)/2} \log n)$ on the time complexity of the hypervolume indicator been obtained [17].

In this paper, a fast dimension-sweep algorithm for the hypervolume indicator in four dimensions is proposed. Although its quadratic time complexity exceeds Yıldız and Suri's new upper bound by a factor of $n^{1/2}/\log n$, it can be easily implemented based on simpler data structures, and runs much faster than the currently available implementations of alternative algorithms on standard benchmark instances [11].

The paper is structured as follows: the next section reviews some of the existing algorithms for the hypervolume indicator and their time complexities. Section 3 describes the proposed algorithm to compute the hypervolume indicator in four dimensions, and states its complexity. Concluding remarks are drawn in Section 4.

2 Related work

Several algorithms for the hypervolume indicator have been proposed in the literature in addition to the aforementioned simplified algorithm by Beume, known as HOY [1]. Fonseca *et al.*'s algorithm [10] is a recursive dimension-sweep algorithm which improves upon a previous algorithm known as HSO (Hypervolume by Slicing Objectives) [16] by caching intermediate results without sacrificing linear space complexity, and using the asymptotically optimal $O(n \log n)$ algorithm later detailed by Beume *et al.* [2] as its three-dimensional base case. Its $O(n^{d-2} \log n)$ time complexity for $d > 2$ matches HOY's for $d = 4$, but is worse for greater values of d.

Two other algorithms, IIHSO (Iterated Incremental HSO) [4] and WFG (Walking Fish Group) [15], have been reported to be the fastest known algorithms in practice, the former for $d = 4$ and the latter for $d > 4$, based on experimental results on a set of benchmark instances [15]. However, IIHSO has $O(n^{d-1})$ time complexity and WFG is reported to be exponential in the number of points in the worst case [15]. Finally, Yıldız and Suri's new algorithm [17] is asymptotically the fastest to date, and it will be interesting to see how well it performs in practice.

3 New dimension-sweep algorithm for the four-dimensional case

Like WFG [15], the new algorithm proposed here follows a dimension-sweep approach, and implements an iterated incremental computation of the hypervolume indicator. However, its time complexity can be shown to be at most quadratic.

The next subsection presents the main ideas behind the proposed algorithm. An algorithm to compute the individual contribution of a point in three dimensions, on which the main algorithm relies, is described in Subsection 3.2. The data structures used and the operation of the overall algorithm are described in Subsections 3.3 and 3.4, respectively. Subsection 3.5 discusses in detail how the $O(n^2)$ time complexity is achieved.

3.1 General description

An asymptotically optimal, $O(n \log n)$-time algorithm to compute the hypervolume indicator in three-dimensions is described by Beume *et al.* [2], and is a natural extension of Kung *et al.*'s algorithm for maxima in three dimensions [12]. In a minimization setting, the algorithm operates by sweeping input points in ascending order of z-coordinate values. While sweeping, the set S of the points seen so far whose (orthogonal) projections on the (x,y)-plane are not dominated by the projection of any other points already seen is efficiently maintained, using a balanced search tree with either the x or y coordinate as the key. At each step, the volume of a slice bounded below by the current point and bounded above by the next point is computed.

Algorithm 1 details this approach, and generalizes it to any number of dimensions. For each new point, p, the volume of a slice is computed by determining the measure, v, of the region dominated by the projection of $\mathrm{S} \cup \{p\}$ onto $(d-1)$-dimensional space, denoted $\mathrm{S}^* \cup \{p^*\}$, and multiplying it by the difference between the last coordinate of the next point, q, and that of p. Note that the asterisk is used to denote projection onto $(d-1)$-dimensional space.

The currently dominated $(d-1)$-dimensional hypervolume, v, is updated by finding the points in S^* that are dominated by p^* and subtracting their individual contributions from v in turn as they are removed from S^*, before adding the individual contribution of p^* to v and inserting p^* into S^*. Once v has been updated, the height, and thus the hypervolume, of the current slice can be easily computed by fetching the next point.

In Algorithm 1, $\mathsf{contribution}(p^*, \mathrm{S}^*, r^*)$ denotes the individual contribution of a point p^* to a non-dominated point set S^*, given a reference point r^*. Each point is swept and removed at most once, resulting in a total of $O(n)$ computed contributions. In the 3-dimensional case, individual 2-dimensional contributions can be computed in constant time, and the complexity of the algorithm is dominated by the time needed to find each dominated projection, s^*. In the 4-dimensional case, the contribution of each point (in three dimensions) to a non-dominated point set can be computed in (amortized) $O(n)$ time as it will be seen next. Since all dominated projections, s^*, can also be found in linear time, the 4-dimensional hypervolume indicator may be com-

Algorithm 1 General algorithm

```
**Input:** X // a set of $n$ points in $\mathbb{R}^d$
**Input:** $r$ // $r \in \mathbb{R}^d$ is the reference point
**Output:** $h$ // Total hypervolume
 1: $s \leftarrow (-\infty, ..., -\infty, r^d) \in [-\infty, +\infty]^d$
 2: Q is a queue containing $X \cup s$ sorted in *ascending* order of dimension $d$
 3: $S^* \leftarrow \varnothing$
 4: $v \leftarrow 0$
 5: $h \leftarrow 0$
 6: $p \leftarrow$ dequeue(Q)
 7: **while** $Q \neq \varnothing$ **do**
 8:     **for all** $s^* \in S^* : p^* \leq s^*$ **do**
 9:         $S^* \leftarrow S^* - \{s^*\}$
10:         $v \leftarrow v -$ contribution$(s^*, S^*, r^*)$
11:     $v \leftarrow v +$ contribution$(p^*, S^*, r^*)$
12:     $S^* \leftarrow S^* \cup \{p^*\}$
13:     $q \leftarrow$ dequeue(Q)
14:     $h \leftarrow h + (q^d - p^d) \cdot v$
15:     $p \leftarrow q$
16: **return** $h$
```

puted in $O(n^2)$ time.

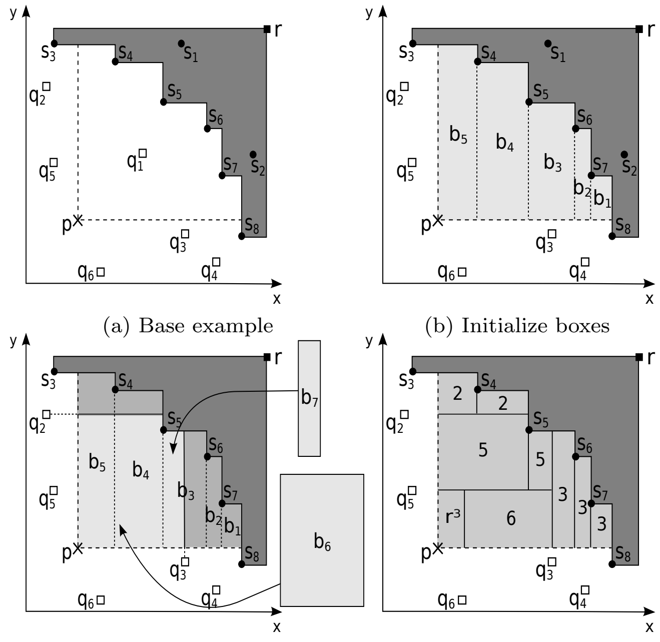
3.2 Individual contribution of a point in three dimensions

In order to achieve $O(n)$ time complexity in the computation of a single point contribution to a non-dominated point set S^* in the conditions imposed by Algorithm 1 (i.e., $p^* \in \mathbb{R}^3$, $S^* \subset \mathbb{R}^3$ and $\nexists q^* \in S^* : p^* \leq q^*$), a method inspired in Emmerich and Fonseca's algorithm [9] is proposed.

Given a non-dominated point set $S \subset \mathbb{R}^3$ and a reference point $r \in \mathbb{R}^3$, the individual contribution of each point p in S may be efficiently determined using the method proposed by Emmerich and Fonseca [9]. The volume dominated exclusively by each point is divided into cuboids (or boxes), and the sum of their volumes is computed. To this end, S is swept in ascending order of the z-coordinate[2] and the region of the (x,y)-plane exclusively dominated by each point p at $z = p^z$ is partitioned into smaller non-overlapping rectangular areas. This partitioning can be obtained by sweeping those points that have coordinate z lower than p^z along one of the dimensions x or y, and is used in the algorithm proposed here. Unlike in [9], where all contributions are computed, in Algorithm 2 only the contribution of a single point needs to be updated.

The example in Figure 1 will be used throughout the remainder of this Section to illustrate the computation of single-point contributions. The base problem is depicted in Figure 1a. Ignoring the presence of q_1 in the example of Figure 1a, as it would have been removed in a previous step, the contribution of p would be computed as the sum of the volumes of the boxes depicted in Figure 1d, where the numbers indicate the corresponding heights.

[2]Note that, in [9], maximization is assumed. For clarity and consistency, the description here considers minimization instead.



(a) Base example

(b) Initialize boxes

(c) Simulate closeBoxesLeft and closeBoxesRight

(d) Expected result

Figure 1: Example of a problem in 3 dimensions, where the goal is to determine the contribution of p to S ($S = \{q_1, ..., q_6\} \cup \{s_1, ..., s_8\}$), given the reference point r. In this problem, $s_t^z \leq p^z$ ($t = 1, ..., 8$) and $q_i^z > p^z$ ($i = 1, ..., 6$). It is assumed that $p^z = 0$ and $q_i^z = i$.

Algorithm 2 HV4D - contribution

```
**Input:** $p \in \mathbb{R}^3$
**Input:** $S \subset \mathbb{R}^3$
**Input:** $r \in \mathbb{R}^3$ // The reference point
**Output:** $c$ // contribution
1: $S_1, S_2 \leftarrow$ split$(S, p^z)$ // $S_1 = \{q \mid q \in S : q^z \leq p^z\}$,
2:                                      // $S_2 = \{q \mid q \in S : q^z > p^z\}$
3: $B \leftarrow$ initializeBoxes$(p, S_1)$
4: $c \leftarrow$ determineContrib$(p, S_2, B, r)$
5: **return** $c$
```

The main steps of the computation of the contribution of a point $p \in \mathbb{R}^3$ to a set $S \subset \mathbb{R}^3$, as described above, are detailed in Algorithm 2. All of them can be implemented in $O(n)$ time, as long as S is a non-dominated point set and there are no points in S which are dominated by p, which is guaranteed to happen by Algorithm 1. Furthermore, points must be kept sorted with respect to dimension two, in order to delimit the base of the boxes (see Figure 1b), and to dimension three, to allow their heights to be determined.

3.3 Data structures

Algorithm 1 receives a non-dominated point set $\mathrm{X} \subset \mathbb{R}^4$ as input, and sets up a queue Q containing all points in X in ascending order of the fourth coordinate. A sentinel is added to Q in order to ensure that point q, which is used to determine the height of the slice, always exists in line 13. The set S^* is stored in a data structure that maintains all points sorted in ascending order of coordinates y and z, using two doubly-linked lists. Such sorted lists are used also for the two subsets S_1 and S_2 of S in Algorithm 2, and support the following operations:

next$^y(p, \mathrm{S})$ The point following p in S with respect to coordinate y, for $p \in \mathrm{S}$.

higher$^y(p, \mathrm{S})$ The point $q \in \mathrm{S}$ with the least $q^y > p^y$, for $p \notin \mathrm{S}$.

getXRightBelow(p, S) The point $q \in \mathrm{S}$ with the least $q^x \geq p^x$ such that $q^y \leq p^y$

Operations next^z and higher^z, analogous to next^y and higher^y, are available as well. Operation next is performed in constant time as long as p itself is in S, while the remaining operations are performed in linear time.

In Algorithm 2, the volume exclusively dominated by a point is partitioned into cuboids, here referred to as boxes. Each box b is defined by its lower corner (l^x, l^y, l^z) and its upper corner (u^x, u^y, u^z). Boxes are kept in a doubly-linked list, B, so that boxes that need to be updated or removed may be accessed easily (in constant time). Since there is no overlap between boxes in the list, it is possible to keep the list of boxes sorted in ascending order of coordinate x. When a box is created, only (l^x, l^y, l^z) and (u^x, u^y) are known. Boxes are kept in the list as long as the corresponding value of u^z is not known. Once this value is determined, the box is closed, i.e., its volume is computed, and the box is removed from the list. Then, the volume is added to the accumulated volume c.
In order to manage the list of boxes, the following operations are implemented:

pushLeft(B, b) Add box b to the left of the box list B

closeAllBoxes(B, z) Close all boxes in list B, setting the corresponding value of u^z to z, and return the sum of the volumes of those boxes.

closeBoxesLeft(B, y, z) From left to right, close all boxes in list B for which $u^y > y$, setting the corresponding value of u^z to z and l^y to y. After closing those boxes, push to the left of B a new box whose lower corner coincides with p, and has $u^y = y$ and u^x equal to the u^x of the last box removed. Finally, return the total volume of the closed boxes.

Algorithm 3 HV4D - contribution - initializeBoxes

```
Input: $p \in \mathbb{R}^3$
Input: $\mathrm{S}_1 \subset \mathbb{R}^3$ // $\forall q \in \mathrm{S} \Rightarrow q^z \leq p^z$
Input: $r \in \mathbb{R}^3$ // The reference point
Output: B // Box list
 1: $\mathrm{S}_1 \leftarrow \mathrm{S}_1 \cup \{(r^x, -\infty, -\infty), (-\infty, r^y, -\infty)\}$
 2: $\mathrm{B} \leftarrow \varnothing$
 3: $q \leftarrow \mathsf{higher}^y(p, \mathrm{S}_1)$
 4: $m \leftarrow \mathsf{getXRightBelow}(p, \mathrm{S}_1)$
 5: while $q^x > p^x$ do
 6:     if $q^x < m^x$ then
 7:         $b \leftarrow ((q^x, p^y, p^z), (m^x, q^y, p^z))$
 8:         $\mathsf{pushLeft}(\mathrm{B}, b)$
 9:         $m \leftarrow q$
10:     $q \leftarrow \mathsf{next}^y(q, \mathrm{S}_1)$
11: $b \leftarrow ((p^x, p^y, p^z), (m^x, q^y, p^z))$
12: $\mathsf{pushLeft}(\mathrm{B}, b)$
13: return B
```

closeBoxesRight(B, x, z) From right to left, close all boxes in list B for which $u^x > x$, setting their u^z to z. If the last removed box is such that $l^x < x$, l^x is updated to x before closing it, and a new box is pushed to the right of B with the same corners, but with u^x set to x. Return the total volume of the closed boxes.

Operation pushLeft is performed in constant time. Operation closeAllBoxes is performed in k steps, and the remaining operations in $k+1$ steps. Therefore, all have a cost of $O(k)$, where $k \leq n$ represents the number of boxes removed.

3.4 Detailed description

Algorithm 1 sweeps through every point p in Q and determines the contribution of its projection on (x,y,z)-space, p^*, to the volume dominated by S^*. This may cause the removal of points in S^* that are dominated by p^*. Point removal can be performed in constant time, but requires the computation of the corresponding contributions, as well. After computing its individual contribution, p^* is added to S^* while keeping the lists used to maintain S^* sorted in ascending order of both y and z coordinates, which can be implemented in linear time. Furthermore, Algorithm 1 guarantees that, when calculating the contribution of any point p^*, all points in S^* are kept sorted in ascending order of coordinates y and z, and that no point in S^* is dominated by any other point in S^* or by p^* itself. As explained before, these constraints allow linear-time computation of a point's contribution.

Algorithm 2 computes the 3-dimensional contribution of p to a set of points S. The computation consists of two parts: the bases of an initial set of boxes are

Algorithm 4 HV4D - contribution - determineContrib

```
Input: $p \in \mathbb{R}^3$
Input: $\mathrm{S}_2 \subset \mathbb{R}^3$ // $\forall q \in \mathrm{S} \Rightarrow q^z > p^z$
Input: B is a list of boxes
Output: $c$ // contribution
 1: $\mathrm{S}_2 \leftarrow \mathrm{S}_2 \cup \{(-\infty, -\infty, r^z)\}$
 2: $q \leftarrow \mathsf{higher}^z(p, \mathrm{S}_2)$
 3: while not $empty(B)$ do
 4:   if $q^x \leq p^x$ then
 5:     if $q^y \leq p^y$ then
 6:       $c \leftarrow c + \mathsf{closeAllBoxes}(\mathrm{B}, q^z)$ // Case 3
 7:     else
 8:       $c \leftarrow c + \mathsf{closeBoxesLeft}(\mathrm{B}, q^y, q^z)$ // Case 1
 9:   else
10:     $c \leftarrow c + \mathsf{closeBoxesRight}(\mathrm{B}, q^x, q^z)$ // Case 2
11:   $q \leftarrow \mathsf{next}^z(q, \mathrm{S}_2)$
12: return $c$
```

determined first (Algorithm 3), and then box heights are found (Algorithm 4).

To determine the bases of the boxes, the points in S whose z coordinate is lower than or equal to p^z (S_1) and which are dominated by p with respect to the x and y coordinates, but not by any other point in S_1, are swept (points $s_4, ..., s_7$ in Figure 1a). Boxes are created from right to left by sweeping through points in S_1 in ascending order of coordinate y, starting from point $\mathsf{higher}^y(p, \mathrm{S}_1)$ (s_7), which is the lowest point in S_1 higher than p^y, and stopping when a point to the left of p is found (s_3). Note that such points always exist because of the presence of the sentinel $(-\infty, r^y, -\infty)$, although this is not shown in Figure 1. All points between the starting point (s_7) and the end point (s_3) that do not fulfill all the above conditions are skipped (s_2). Each of the points that satisfy the above conditions (s_7, s_6, s_5, s_4) defines l^x and u^y of a box as well as u^x of the next box. For example, in Figure 1b, point s_5 defines l^x and u^y of box b_3 and u^x of box b_4. The value of u^x for the first and rightmost box created is determined by $\mathsf{getXRightBelow}(p)$ (s_8), and is guaranteed to exist due of the sentinel $(r^x, -\infty, -\infty)$. Finally, the end point s_3 only defines u^y of the last and leftmost box. In the example of Figure 1a, after executing the first part of the algorithm, B contains $b_1, ..., b_5$ as depicted in Figure 1b.

The next step, determineContrib, consists of determining the height of boxes and closing them and, in some cases, initializing a new box (Algorithm 4). The total area covered by boxes in the (x, y)-plane shrinks after each step. Only points with coordinate z higher than p^z (S_2) need to be considered ($q^2, ..., q^6$). Therefore, points in S_2 are swept in ascending order of coordinate z as long as there are still boxes to be closed. While processing each point q, three cases are considered, depending on the projection of q on the (x, y)-plane:

Case 1: q is to the left of and above p (e.g. q_2, q_5)

Case 2: q is to the right of and below p (e.g. q_3, q_4, q_6)

Case 3: q dominates p (e.g. the sentinel $(-\infty, -\infty, r^z)$, which is not represented in Figure 1)

Note that q is never dominated by p on the (x,y)-plane, because it would also be dominated in (x, y, z)-space in that case, but those points were previously removed in Algorithm 1.

Cases 1, 2 and 3 cause the algorithm to call functions closeBoxesLeft, closeBoxesRight and closeAllBoxes, respectively. Figure 1c shows an example of what happens when cases 1 or 2 occur. The darker regions of boxes b_5 and b_4 (respectively, b_1, b_2 and b_3) represent the boxes that are shrunk and closed when case 1 (case 2) occurs while processing q_2 (q_3). After closing those boxes, box b_6 (b_7) is inserted to the left (right) of the box list to account for the area left uncovered due to the shrinking of the boxes before they are closed. Every function returns the total volume of the closed boxes. When case 3 occurs, all boxes are closed and the algorithm can terminate, as there are no more box heights to be determined.

3.5 Complexity

It is not difficult to see that the complexity of Algorithm 2 is $O(n)$. Splitting S into two subsets (S_1 and S_2), and each of the two remaining stages of the algorithm can be performed in $O(n)$ time. Regarding the first stage (initializeBoxes), note that for each point in S_1 with coordinate y greater than p^y, of which there are at most n points, at most one box is created (in constant time). Therefore, $O(n)$ complexity is achieved. The second stage processes all points with coordinate z greater than p^z, which are also at most n points. For each of these points, k boxes are closed, $k \in [0, n]$. Moreover, at most one box is created, which can happen only if at least one box is closed. Note that if there are t points with third coordinate lower or equal to p^z, then the first stage can create up to t boxes, while the second stage can create at most $n - t$ boxes, which gives a total of up to n boxes created. Therefore, the maximum number of closed boxes is also n. Independently of which function is used to close boxes (closeBoxesRight, closeBoxesLeft or closeAllBoxes) k steps are performed if boxes are only closed, or $k + 1$ steps, if a box is also created, leading to $O(k)$ cost either way. Therefore, the total cost of the operations of Algorithm 2 amortizes to $O(n)$.

Algorithm 1 sweeps through n points and, for each point p, it determines the points in S^* with greatest y and z coordinates lower than p^y and p^z, respectively, in order to keep the lists of points associated with S^* sorted, at a cost of $O(n)$. Moreover, for each point swept, the individual contributions of k dominated points and the contribution of the current point are computed. Since each point in the original set X is added to S* and removed from it at most once, the algorithm computes at most $2n$ contributions, each at a cost of $O(n)$ using Algorithm 2, leading to $O(n^2)$ amortized time complexity.

4 Concluding remarks

A C-language implementation of the algorithm proposed here confirms that it can be implemented efficiently, and that no large constants hide in the O notation [11]. Following the same ideas, it may be possible to obtain a dimension-sweep algorithm for $d = 5$ with complexity $O(n^2 \log n)$, since at least a 4-dimensional analogue of initializeBoxes with $O(n \log n)$ complexity would be easily constructed. This would match Yıldız and Suri's upper bound [17] for the 5-dimensional case.

Acknowledgement Andreia P. Guerreiro gratefully acknowledges funding from Fundação para a Ciência e Tecnologia (FCT), Portugal, through Ph.D. grant SFRH/BD/77725/2011.

References

[1] N. Beume. S-metric calculation by considering dominated hypervolume as Klee's measure problem. *Evol. Comput.*, 17:477–492, Dec. 2009.

[2] N. Beume, C. M. Fonseca, M. López-Ibáñez, L. Paquete, and J. Vahrenhold. On the complexity of computing the hypervolume indicator. *IEEE Trans. Evol. Comput.*, 13(5):1075–1082, 2009.

[3] N. Beume, B. Naujoks, and M. Emmerich. SMS-EMOA: Multiobjective selection based on dominated hypervolume. *Eur. J. Oper. Res.*, 181(3):1653–1669, 2007.

[4] L. Bradstreet, L. While, and L. Barone. A fast many-objective hypervolume algorithm using iterated incremental calculations. In *IEEE Congress on Evolutionary Computation (CEC 2010)*, pages 179–186, July 2010.

[5] K. Bringmann. Klee's measure problem on fat boxes in time $O(n^{(d+2)/3})$. In *26th Symposium on Computational geometry (SoCG)*, pages 222–229, New York, NY, USA, 2010. ACM.

[6] K. Bringmann and T. Friedrich. Approximating the volume of unions and intersections of high-dimensional geometric objects. In S.-H. Hong et al., editors, *Algorithms and Computation*, volume 5369 of *LNCS*, pages 436–447. Springer Berlin / Heidelberg, 2008.

[7] T. M. Chan. Semi-online maintenance of geometric optima and measures. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '02, pages 474–483, Philadelphia, PA, USA, 2002. Society for Industrial and Applied Mathematics.

[8] T. M. Chan. A (slightly) faster algorithm for Klee's measure problem. *Computational Geometry*, 43:243–250, 2010.

[9] M. Emmerich and C. M. Fonseca. Computing hypervolume contributions in low dimensions: Asymptotically optimal algorithm and complexity results. In R. H. C. Takahashi et al., editors, *EMO 2011*, volume 6576 of *LNCS*, pages 121–135. Springer Berlin / Heidelberg, 2011.

[10] C. M. Fonseca, L. Paquete, and M. López-Ibáñez. An improved dimension-sweep algorithm for the hypervolume indicator. In *IEEE Congress on Evolutionary Computation (CEC 2006)*, pages 1157–1163, Piscataway, NJ, July 2006. IEEE Press.

[11] A. P. Guerreiro. Efficient algorithms for the assessment of stochastic multiobjective optimizers. Master's thesis, IST, Technical University of Lisbon, Portugal, 2011.

[12] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *Journal of the ACM*, 22(4):469–476, 1975.

[13] M. H. Overmars and C.-K. Yap. New upper bounds in Klee's measure problem. *SIAM J. Comput.*, 20(6):1034–1045, 1991.

[14] T. Wagner, B. Nicola, and B. Naujoks. Pareto-, aggregation-, and indicator-based methods in many-objective optimization. In S. Obayashi et al., editors, *EMO 2007*, volume 4403 of *LNCS*, pages 742–756, Berlin, Heidelberg, 2007. Springer-Verlag.

[15] L. While, L. Bradstreet, and L. Barone. A fast way of calculating exact hypervolumes. *IEEE Trans. Evol. Comput.*, 16(1):86–95, 2012.

[16] L. While, P. Hingston, L. Barone, and S. Huband. A faster algorithm for calculating hypervolume. *IEEE Trans. Evol. Comput.*, 10(1):29–38, Feb. 2006.

[17] H. Yıldız and S. Suri. On Klee's measure problem on grounded boxes. In *28th Symposium on Computational Geometry (SoCG)*, Chapel Hill, North Carolina, USA, June 2012.

[18] E. Zitzler and L. Thiele. Multiobjective optimization using evolutionary algorithms – A comparative case study. In A. E. Eiben et al., editors, *Parallel Problem Solving from Nature, PPSN V*, volume 1498 of *LNCS*, pages 292–301. Springer, Heidelberg, 1998.

An Efficient Transformation for Klee's Measure Problem in the Streaming Model

Gokarna Sharma* Costas Busch* Ramachandran Vaidyanathan† Suresh Rai† Jerry L. Trahan†

Abstract

Given a stream of rectangles over a discrete space, we consider the problem of computing the total number of distinct points covered by the rectangles. This can be seen as the discrete version of the two-dimensional Klee's measure problem for streaming inputs. We provide an (ϵ, δ)-approximation for fat rectangles. For the case of arbitrary rectangles, we provide an $\mathcal{O}(\sqrt{\log U})$-approximation, where U is the total number of discrete points in the two-dimensional space. The time to process each rectangle, the total required space, and the time to answer a query for the total area are polylogarithmic in U. Our approximations are based on an efficient transformation technique which projects rectangle areas to one-dimensional ranges, and then uses a streaming algorithm for the Klee's measure problem in the one-dimensional space. The projection is deterministic and to our knowledge it is the first approach of this kind which provides efficiency and accuracy trade-offs in the streaming model.

1 Introduction

The well-known two-dimensional *Klee's measure problem* (KMP) [7] can be stated as follows: given a collection of m axis-aligned rectangles, how quickly can we compute the area of their union? This problem has been studied extensively in the literature [3, 6, 12] with the best known bounds of $\mathcal{O}(m \log m)$ and $\mathcal{O}(m)$ for the time and space requirements for an exact answer for m rectangles.

In this paper, we consider the problem of estimating the discrete version of the classical two-dimensional KMP in the streaming model. In this case, the data stream consists of rectangular elements over a discrete two-dimensional grid of points, and the task is to efficiently estimate at any time the number of distinct grid points occupied by the rectangles that have arrived so far. Following the literature, we hereafter denote this problem of finding the number of distinct elements by F_0 (referred to as the *zeroth frequency moment*) [1, 10, 11].

The motivation to study KMP in the streaming model is due to spatial and temporal data that arise in many domains such as VLSI layout processing and sensor networks. A spatial database, e.g. OpenGIS[1], deals with a large collection of relatively simple geometric objects, of which rectangles are the most basic types. Moreover, query processing in the constraint database model [8] can also be seen as a computation over the set of geometric objects, e.g. [2]. The streaming setting makes sense in online scenarios of the aforementioned applications when the workspace is very limited such that rescanning the entire dataset is not feasible.

A recent work in the F_0 of KMP in the streaming model is due to Tirthapura and Woodruff [11], who gave an (ϵ, δ)-approximation algorithm with space as well as processing time per rectangle $(\frac{1}{\epsilon}\log(\frac{mU}{\delta}))^{\mathcal{O}(1)}$, $0 < \epsilon, \delta < 1$, where U is the total number of grid points in the two-dimensional space[2]. Comparing to their algorithm, the algorithm we present here is very simple, does not depend on m, and exhibits different tradeoffs.

Contributions. We consider a $2^n \times 2^n$ two-dimensional grid with $U = 2^{2n}$ points. The input stream consists of a set of m elements $\Upsilon = \{x_0, x_1, \ldots, x_{m-1}\}$, where each x_i is an $a_i \times b_i, 0 \leq a_i, b_i < \sqrt{U}$, rectangle of discrete points, and x_{m-1} is the last element that has arrived so far. Let A denote the total area (number of distinct discrete points) of the rectangles in Υ which have arrived so far in the stream. We present and analyze an algorithm that returns an estimate $\text{est}(A)$ of A.

Our first result is for "fat" rectangles − we say that a rectangle is *fat* if the ratio g of its side lengths (i.e., the aspect ratio) is between $\frac{1}{c} < g < c$, where $c \geq 1$ is some constant, e.g. [4]. We give an algorithm which provides an (ϵ, δ)-approximation of A (that is, F_0). Given $0 < \epsilon, \delta < 1$, an approximation algorithm is said to (ϵ, δ)-approximate F_0 if the estimated output $\hat{F}_0$ satisfies $\mathbf{Pr}[|\hat{F}_0 - F_0| < \epsilon F_0] > 1 - \delta$. Moreover, our streaming approximation algorithm achieves the following time and space complexities for fat rectangles: (i) the amortized processing time per rectangle is $\mathcal{O}(\frac{1}{\epsilon}\log\frac{U}{\epsilon}\log\frac{1}{\delta})$; (ii) the workspace needed is $\mathcal{O}(\frac{1}{\epsilon^2}\log U \log\frac{1}{\delta})$ bits; and

*Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803, USA, {gokarna,busch}@csc.lsu.edu

†Department of Electrical and Computer Engineering, Louisiana State University, Baton Rouge, LA 70803, USA, {vaidy,srai,jtrahan}@lsu.edu

[1]http://www.opengeospatial.org/

[2]For a discrete d-dimensional space, their algorithm has space as well as processing time per rectangle $(\frac{d}{\epsilon}\log(\frac{m\omega}{\delta}))^{\mathcal{O}(1)}$, where ω is the maximum coordinate along any dimension.

(iii) the time to answer a query for F_0 is $\mathcal{O}(1)$.

For the general case of arbitrary rectangles (the rectangles with any ratio of side lengths), we present a streaming algorithm which provides an $\mathcal{O}(\sqrt{\log U})$-approximation of A, such that $\Omega(1/\sqrt{\log U}) \leq \mathrm{est}(A)/A \leq \mathcal{O}(\sqrt{\log U})$; the approximation bound holds with constant probability. Moreover, it ensures that: (i) the amortized processing time per rectangle is $\mathcal{O}(\log U \cdot \log\log U)$; (ii) the workspace needed $\mathcal{O}(\log^2 U \cdot \log\log U)$ bits; and (iii) the time to answer a F_0 query is $\mathcal{O}(\log U)$.

The main idea is to transform each input rectangle to an interval (range), such that the estimate of F_0 for the intervals provides an estimate for A. Our two-dimensional approximation is based on the *range-efficient*[3] (ϵ, δ)-approximation algorithm of [10].

Our algorithm implements an efficient proximity transformation technique of rectangles to ranges based on a Z-ordering [9] (note that a depth-first traversal of a quadtree is essentially a Z-ordering). The proximity-based transformation is deterministic and partitions the data stream into buckets according to the aspect ratio of the rectangles. In the general case we use $\mathcal{O}(\log U)$ buckets, while for the case of fat rectangles we only use one bucket. We then apply a range-efficient algorithm for each bucket instance independently. The algorithm requires first to normalize a rectangle and then project it to a range. The normalization helps to preserve the intersection properties of the rectangles even when they are transformed to ranges, which further helps to obtain good approximations. In the analysis, we bound the error due to normalization and also due to the projection on ranges.

To the best of our knowledge, this is the first transformation algorithm for KMP that improves significantly on the previous solutions for fat rectangles [4, 11].

Related Work. For the classical (non-streaming) KMP, Bentley [3] described a deterministic time-optimal $\mathcal{O}(m \log m)$ time and $\mathcal{O}(m)$ workspace solution. This solution is based on reducing the problem to m one-dimensional KMPs [7] by sweeping a vertical line across the area. Some recent work tried to minimize the space and time requirements for the efficient computation of the area of the union, e.g. [6, 12]. Particularly, Chen and Chan [6] gave an algorithm that runs in $\mathcal{O}(m^{3/2} \log m)$ time but needs $\mathcal{O}(\sqrt{m})$ extra workspace. Vahrenhold [12] minimizes the extra space to $\mathcal{O}(1)$ with the same running time. All the aforementioned solutions for KMP are deterministic and compute the *exact* area. Moreover, no deterministic algorithm has improved the best known bounds of $\mathcal{O}(m \log m)$ and $\mathcal{O}(m)$ for the time and space. Therefore, recent focus has been on approximation algorithms. To this end, Bringmann and Friedrich [5] gave a $(1 \pm \epsilon)$-approximation algorithm for any $0 < \epsilon < 1$. However, it has space complexity that is still linear in the size of the input.

A difficulty in obtaining tighter bounds on time and space complexities of KMP stems from the fact that [3, 6, 12] use explicit sorting algorithms and tree-based data structures (e.g. quadtrees) to handle rectangles, where such data structures need at least $\Omega(\log m)$ time to process an individual element using $\mathcal{O}(m)$ space. A natural question that arises is whether tighter bounds on time and space requirements are achievable. According to the literature, computing F_0 *exactly* requires space linear in the number of distinct values [1]. Therefore, we opt to design streaming approximation algorithms for KMP that require very limited space.

Outline of Paper. We give a KMP streaming algorithm in Section 2 and discuss the normalization in Section 3. In Section 4, we give an algorithm to transform a rectangle to a normalized approximation based on a Z-ordering. We analyze the transformation algorithm for the one bucket case (fat rectangles) in Section 5 and do the same for many buckets (general case) in Section 6. We conclude the paper in Section 7. Proofs are deferred to the full version due to space limitations.

2 KMP Streaming Algorithm

Algorithm 1 is an approach for estimating the total number of distinct points A covered by a streaming set Υ of m rectangles in $\mathbb{Z}_n^2$. The basic idea is to transform each rectangle to one or more one-dimensional ranges and then use a range-efficient algorithm to estimate the number of discrete points used by the stream of ranges. The challenge is to perform the transformation in such a way that the estimate from the ranges is a good approximation of A. In order to achieve good approximations, we first *normalize* the rectangles, by aligning them into appropriate space points whose coordinates are multiples of 2. We then separate the rectangles into different buckets according to their normalization. Each bucket has range mapping characteristics which help to accurately estimate the respective covered areas. We run a range-efficient algorithm to each bucket, which we combine to obtain the resulting estimate for A.

Algorithm 1 initializes χ buckets $\mathbb{B}_0, \ldots, \mathbb{B}_{\chi-1}$ of normalized rectangles in $\mathbb{Z}_n^2$. Each rectangle $x \in \Upsilon$ is transformed to one or more normalized rectangles $y'_1, y'_2, \ldots$ whose union is an approximation of x. Then, each y'_j is inserted into some appropriate bucket $\mathbb{B}_{i_j}$. Each $y'_j \in \mathbb{B}_{i_j}$ is immediately mapped to a one-dimensional range using a projection function appropriate for $\mathbb{B}_{i_j}$. In this way, bucket $\mathbb{B}_{i_j}$ produces a stream of ranges. Some known range-efficient (ϵ, δ)-approximation algorithm is applied to the stream of ranges to find an es-

[3]In the range-efficient model, the elements of a data stream contain a range (or interval) of items.

Algorithm 1: A rectangle area estimation algorithm

```
    Input: A streaming set $\Upsilon$ of $m$ rectangles in $\mathbb{Z}_n^2$;
    Output: An estimate $\mathrm{est}(A)$ of total number of
            distinct discrete points $A$ covered by the
            rectangles in $\Upsilon$;
1   Initialization:
2       Define $\chi$ buckets $\mathbb{B}_0, \ldots, \mathbb{B}_{\chi-1}$ of normal rectangles in
        $\mathbb{Z}_n^2$ (initially the buckets are empty);
3   When a new rectangle $x \in \Upsilon$ arrives:
4       Normalization: Transform $x$ into a sequence of
        normal rectangles $y'_1, y'_2, \ldots$, and then assign each $y'_j$
        to some appropriate bucket $\mathbb{B}_{i_j}$;
5       One-dimensional mapping: Map each $y'_j$ to a range
        $r_j$ using an appropriate projection based on $\mathbb{B}_{i_j}$;
6       Apply a range-efficient $(\epsilon, \delta)$-approximation
        algorithm to update with respect to $r_j$ an estimate of
        $A'_{i_j}$, the total area (total discrete points) of the
        normal rectangles in $\mathbb{B}_{i_j}$;
7       Maintain $E_{\max} = \max_i \mathrm{est}(A'_i)$ and
        $E_{\mathrm{sum}} = \sum_{i=0}^{\chi-1} \mathrm{est}(A'_i)$; the maximum and the sum
        among the $\chi$ bucket estimates;
8   When an estimate for $F_0$ is asked for:
9       Return $\mathrm{est}(A) = \sqrt{E_{\max} E_{\mathrm{sum}}}$;
```

timate of A'_i, the total number of discrete points occupied by the normalized rectangles in $\mathbb{B}_i$. Algorithm 1 then maintains $E_{\max}$ and E_{sum}, the maximum and the sum among all the bucket estimates, respectively. When an estimate $\mathrm{est}(A)$ is asked for, the algorithm returns $\mathrm{est}(A) = \sqrt{E_{\max} E_{\mathrm{sum}}}$.

One can apply any range-efficient (ϵ, δ)-approximation algorithm for F_0 on each bucket $\mathbb{B}_i$ (step 6). Here, we use the Hits algorithm of [10]. It has the following time and space complexities for F_0 for each bucket $\mathbb{B}_i$, where $U = 2^{2n}$.

Theorem 1 (Pavan and Tirthapura [10]) *Given $0 < \epsilon < 1$ and $0 < \delta < 1$, algorithm* Hits *(ϵ, δ)-approximates F_0 with space complexity $\mathcal{O}(\frac{1}{\epsilon^2} \log U \log \frac{1}{\delta})$ bits, amortized time taken to process a range $\mathcal{O}(\log \frac{U}{\epsilon} \log \frac{1}{\delta})$, and time taken to process a query for F_0 at any time $\mathcal{O}(1)$.*

In section 4 we give a transformation Proximity of rectangles to ranges which will enable us to provide two versions of Algorithm 1, one with a single bucket (fat rectangles), and the other with multiple buckets (arbitrary rectangles). We continue with first describing the normalization that we use.

3 Normalization

We first start with some basic definitions for ranges and their normalizations, and then we extend the definitions for normalized rectangles.

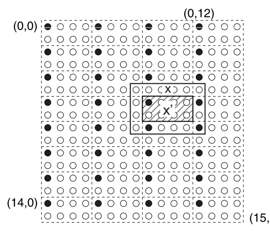
Ranges. For integer $n \geq 0$, let $\mathbb{Z}_n = \{0, 1, \ldots, 2^n - 1\} \subset \mathbb{Z}$ be a one-dimensional space of 2^n discrete integer points. A range (or interval) $r = [p_1, p_2]$, where $0 \leq p_1 \leq p_2 < 2^n$, consists of all the points between p_1 and p_2. Denote with $|r| = p_2 - p_1 + 1$ the size of range r which is the number of points in it.

The *α-normal subset* of $\mathbb{Z}_n$, denoted $\mathbb{W}_n^\alpha$, for integer $0 \leq \alpha \leq n$, consists of every $(2^\alpha)^{\text{th}}$ element of $\mathbb{Z}_n$, namely, $\mathbb{W}_n^\alpha = \{p \in \mathbb{Z}_n \ : \ p = i2^\alpha \wedge i \in \mathbb{Z}\}$. We will refer to the elements of $\mathbb{W}_n^\alpha$ as *normal points.* The normal subset $\mathbb{W}_n^\alpha$ induces $2^{n-\alpha}$ *normal ranges* of size 2^a such that each starts at a normal point. We will also use the notation $\mathbb{W}_n^\alpha$ to denote the normal ranges. Let $\mathbb{W}_n = \bigcup_{\alpha=0}^{n-1} \mathbb{W}_n^\alpha$ denote the set of all possible normal ranges (and respective normal points).

Rectangles. All the definitions for one-dimensional space $\mathbb{Z}_n$ extend to the two-dimensional space $\mathbb{Z}_n^2 = \mathbb{Z}_n \times \mathbb{Z}_n$ of discrete integer points. Space $\mathbb{Z}_n^2$ can be viewed as an array of points such that for any point $(p, q) \in \mathbb{Z}_n^2$, p corresponds to a row and q to a column (there are 2^n rows and 2^n columns). The upper left corner of $\mathbb{Z}_n^2$ is point $(0, 0)$, and the lower right corner is point $(2^n - 1, 2^n - 1)$. (See the figure below.)

A rectangle $x = \langle (p_1, q_1), (p_2, q_2) \rangle$ is a subset of $\mathbb{Z}_n^2$, where $p_1, p_2, q_1, q_2 \in \mathbb{Z}_n$ with $p_1 \leq p_2$ and $q_1 \leq q_2$, such that x contains all points $\{(p, q) \ : \ p_1 \leq p \leq p_2 \text{ and } q_1 \leq q \leq q_2\}$. Note that (p_1, q_1) is the north-west corner of x, while (p_2, q_2) is the south-east corner. We say that x is a $a \times b$ rectangle with side lengths $a = p_2 - p_1 + 1$ and $b = q_2 - q_1 + 1$. We denote with $|x| = a \cdot b$ the size of rectangle x which is the number of points in it.

For any integers $0 \leq \alpha, \beta \leq n$, define the *$(\alpha, \beta)$-normal subset* $\mathbb{W}_n^{\alpha,\beta} = \mathbb{W}_n^\alpha \times \mathbb{W}_n^\beta \subseteq \mathbb{Z}_n^2$. Each element $(p, q) \in \mathbb{W}_n^{\alpha,\beta}$ is a *normal point* for which it holds $p = i2^\alpha$ and $q = j2^\beta$, for integers i and j. In other words, set $\mathbb{W}_n^{\alpha,\beta}$ selects every $(2^\alpha)^{\text{th}}$ point in the vertical direction and every $(2^\beta)^{\text{th}}$ point in the horizontal direction of $\mathbb{Z}_n^2$. Each normal point $w \in \mathbb{W}_n^{\alpha,\beta}$ corresponds to a $2^\alpha \times 2^\beta$ *normal rectangle*, whose north-west corner is w. We will also use the notation $\mathbb{W}_n^{\alpha,\beta}$ to denote the set of normal rectangles. Let $\mathbb{W}_n = \bigcup_{\alpha=0}^{n} \bigcup_{\beta=0}^{n} \mathbb{W}_n^{\alpha,\beta}$ denote the set of all possible normal rectangles (and respective normal points). The figure above shows a $(1, 2)$-normal subset and the respective $2^1 \times 2^2$ normal rectangles $\mathbb{W}_4^{1,2}$ of $\mathbb{Z}_4^2$.

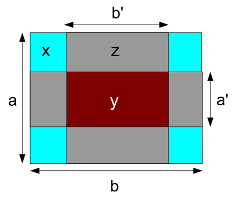


Lemma 2 *Any $a \times b$ rectangle $x \in \mathbb{Z}_n^2$, contains an $a' \times b'$ normal rectangle $x' \subseteq x$ with $1 \leq |x|/|x'| \leq 16$, such that $1 \leq a/a' \leq 4$ and $1 \leq b/b' \leq 4$.*

Given a rectangle x, the internal normal rectangle x' of Lemma 2 can be computed in constant time. See the figure above for an example rectangle x and its respective internal normalized rectangle x'.

Aspect Ratios. The aspect ratio of an $a \times b$ rectangle is $\frac{b}{a}$. Each normal rectangle induced by $\mathbb{W}_n^{\alpha,\beta}$ has an aspect ratio of $2^{\beta-\alpha}$. All rectangles induced by $\mathbb{W}_n^{\alpha+i,\beta+i}$ (for integer i) have aspect ratio $2^{\beta-\alpha}$. Thus, given g, $0 \le g \le n$, $\mathbb{W}_n^{\alpha,g+\alpha}$ corresponds to normal rectangles of aspect ratio 2^g and size $2^\alpha \times 2^{g+\alpha}$. Let $\mathbb{W}_n^{(g,+)} = \bigcup_{\alpha=0}^{n-g} \mathbb{W}_n^{\alpha,g+\alpha}$ denote the normal rectangles of aspect ratio 2^g. Similarly, let $\mathbb{W}_n^{(g,-)} = \bigcup_{\alpha=0}^{n-g} \mathbb{W}_n^{g+\alpha,\alpha}$ denote the normal rectangles of aspect ratio 2^{-g}. Let $\mathbb{W}_n^{(+)} = \bigcup_{g=0}^{n} \mathbb{W}_n^{(g,+)}$ be the set of all possible normal rectangles with aspect ratio at least one. Let $\mathbb{W}_n^{(-)} = \bigcup_{g=1}^{n} \mathbb{W}_n^{(g,-)}$ be the set of all possible normal rectangles with aspect ratio < 1.

Union of Rectangles. In our KMP approach, given a rectangle x_i we compute inside it a y_i which is either a normal rectangle (for example given by Lemma 2), or y_i is a rectangle that consists of multiple normal rectangles (as will be the case of Section 5). Consider a sequence of rectangles $x_0, x_1, \ldots, x_{m-1}$. Denote the union of these rectangles as $X = x_0 \cup x_1 \cup \cdots \cup x_{m-1}$ and the area of X as $|X|$. Consider also a sequence of rectangles $y_0, y_1, \ldots, y_{m-1}$, where each y_i is contained in x_i, with union $Y = y_0 \cup y_1 \cup \cdots \cup y_{m-1}$. We are interested in estimating the area of X based on calculating the area of Y.



Let x be an $a \times b$ rectangle (see the figure above) with $a \cdot b$ discrete points in $\mathbb{Z}_n^2$, $0 \le a, b < 2^n$. Denote by $|x|$ the area of x, namely, $|x| = a \cdot b$. Let y denote an $a' \times b'$ rectangle in x. Write $a = a^{top} + a' + a^{bottom}$ and $b = b^{left} + b' + b^{right}$, where $a = a'c_{x,v}$ and $b = b'c_{x,h}$, for some integers $c_{x,v}, c_{x,h} \ge 1$. Note that $|x| = a'c_{x,v} \cdot b'c_{x,h} = c_x|y|$, where $c_x = c_{x,v} \cdot c_{x,h}$.

We define rectangle z^{top} of dimensions $a^{top} \times b'$ that resides on top of y. Similarly, we define z^{bottom} as an $a^{bottom} \times b'$ rectangle that resides on the bottom of y. Symmetrically, we define z^{left} and z^{right} as the $a' \times b^{left}$ and $a' \times b^{right}$ rectangles that reside on the left and right of y. Note that $z^{top}, z^{bottom}, z^{left}$, and z^{right} are all in x. Finally, we define the *cross* polygon z to be: $z = y \cup z^{top} \cup z^{bottom} \cup z^{left} \cup z^{right}$.

Given X and Y, we define the corresponding sequence of cross polygons $z_0, z_1, \ldots, z_{m-1}$, with union $Z = z_0 \cup z_1 \cup \cdots \cup z_{m-1}$. Denote $c_v = \max_i c_{x_i,v}$, and $c_h = \max_i c_{x_i,h}$. It can be shown that $|Z| = \alpha|Y|$, for $1 \le \alpha \le 2c_v + 2c_h - 3$. It can also be shown that $|X - Z| = \beta|Z|$, for $0 \le \beta \le 2 \cdot \min\{c_v, c_h\} - 2$. Therefore, we obtain:

Lemma 3 *$|X| = \gamma|Y|$, where $1 \le \gamma \le (2c_v + 2c_h - 3)(2 \cdot \min\{c_v, c_h\} - 1)$.*

4 Proximity Transformation

Here we describe the Proximity transformation which deterministically maps each normal rectangle to a one-dimensional range based on a Z-ordering.

Without loss of generality consider a bucket $\mathbb{B}_g$ which contains normal rectangles from $\mathbb{W}_n^{(g,-)}$. For the one-dimensional mapping (step 5 of Algorithm 1), we show how Algorithm Proximity takes a normalized rectangle $x' \in \mathbb{B}_g$ and maps it to a linear interval (range) r in a manner that preserves the intersection properties of the rectangles of $\mathbb{B}_g$.

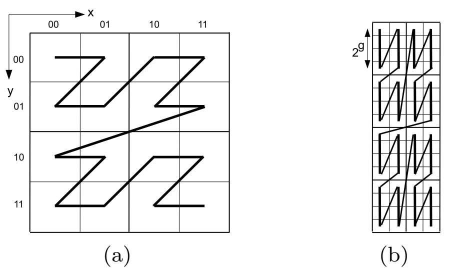


Figure 1: A Z-curve for aspect ratio (a) 1, (b) 1/4

Define $f_n^{(0,+)} : \mathbb{Z}_n^2 \longrightarrow \mathbb{Z}_{2n}$ as the well-known Z-ordering [9] (see Fig. 1a). The figure shows the Z-values for the case $g = 0$ of points for the two dimensional space $0 \le x \le 3, 0 \le y \le 3$ (shown in binary). Interleaving the binary coordinate values gives binary Z-values and connecting the Z-values in their order from the lowest (0000) to the highest (1111) produces the recursively Z-shaped curve, i.e., a Z-order. For aspect ratio < 1, define $f_n^{(g,-)} : \mathbb{Z}_n^2 \longrightarrow \mathbb{Z}_{2n}$ as follows. Partition $\mathbb{Z}_n^2$ into 2^g contiguous vertically aligned elements, then connect them in the order provided by a Z-ordering (see Fig. 1b). For aspect ratio > 1, partition as horizontally aligned 2^g elements. Note that $f_n^{(g,-)}$ preserves the intersection properties of normalized rectangles of $\mathbb{W}_n^{(g,-)}$, that is, for any $x'_1, x'_2 \in \mathbb{W}_n^{(g,-)}$, $|x'_1 \cap x'_2| = |(f_n^{(g,-)}(x'_1)) \cap (f_n^{(g,-)}(x'_2))|$. Similarly, $f_n^{(g,+)}$ preserves the intersection properties of normalized rectangles in $\mathbb{W}_n^{(g,+)}$.

Each element i of $\mathbb{Z}_{2n} = \{0, 1, \cdots, 2^{2n} - 1\}$ (the codomain of $f_n^{(g,-)}(f_n^{(g,+)})$) is a $2n$-bit binary representation of i. The domain $\mathbb{Z}_n^2$ of $f_n^{(g,-)}$ $(f_n^{(g,+)})$ consists of doublets (p, q), where $p, q \in \mathbb{Z}_n$. So, each (p, q) expressed as the concatenation of the binary representations of p and q is a $2n$-bit quantity. This $2n$-bit number (belonging to $\mathbb{Z}_{2n}$) is called the *row-major index* of (p, q). Thus, function $f_n^{(g,-)}$ ($f_n^{(g,+)}$) can be viewed as the transformation of one $2n$-bit number into another $2n$-bit number.

Since $n = \mathcal{O}\left(\log |\mathbb{Z}_n^2|\right)$, the function $f_n^{(g,-)}(\cdot)$ ($f_n^{(g,+)}(\cdot)$) can be computed using a logarithmic-size (albeit non-standard) operation that could be assumed to be computable in constant time. Therefore, we obtain:

Lemma 4 *For any normalized rectangle $x \in \mathbb{B}_g$, $f_n^{(g,-)}(x)$ ($f_n^{(g,+)}(x)$) is a set of $|x|$ contiguous integers (range) in $\mathbb{Z}_{2n}$. This ordering can be computed in constant time for a given rectangle and preserves the intersection properties of rectangles of $\mathbb{B}_g$.*

5 One Bucket

We describe how to efficiently estimate the area of a stream with fat rectangles. We first describe the special case of a stream of squares $\Upsilon = \{x_0, x_1, \ldots, x_{m-1}\}$ and then we extend the result to include fat rectangles with aspect ratios other than 1. The normalization (step 4) of Algorithm 1 uses only one bucket $\mathbb{B}$ which contains normal rectangles of aspect ratio 1 (namely, normal squares in $\mathbb{W}_n^{(0,+)}$). Using the following result it is possible to transform each square $x \in \Upsilon$ into a set of normal squares which are then used to estimate the total area A of the stream Υ.

Lemma 5 *Given an $a \times a$ square x and η, $0 < \eta \leq 1$, there is an internal $a' \times a'$ square x' which consists of at most c/η normal squares (of possibly various sizes), for some positive constant c, such that $a(1-\eta) \leq a' \leq a$.*

In Algorithm 1, since $\mathrm{est}(A) = \sqrt{E_{\max} E_{\mathrm{sum}}}$ and we use only one bucket, we get $E_{\max} = E_{\mathrm{sum}}$. Thus, the result of the algorithm is directly the output of Hits (Theorem 1) on bucket $\mathbb{B}$. For each square $x_i \in \Upsilon$, let x_i' be the respective square given by Lemma 5. Suppose that the set $\Upsilon' = \{x_0', x_1', \ldots, x_{m-1}'\}$ has total area A'. Each x_i is replaced with c/η normal squares that cover the respective x_i' (as specified by Lemma 5), and further each such normal square is projected to a range through the Proximity transformation of Section 4. In other words, Hits returns an (ϵ', δ) estimate on A', for $0 < \epsilon', \delta < 1$. This is then used to obtain an estimate of A. Using Lemma 3 we can relate the areas of A and A' through η, since $c_v, c_h \leq 1/(1-\eta)$. Finally, by appropriately substituting ϵ' and η with linear functions of ϵ, we obtain an (ϵ, δ)-approximation for $\mathrm{est}(A)$. The space complexity and query time remain asymptotically the same as in Hits, while the time complexity increases by a factor of c/η (due to the number of normalized rectangles inside each x_i'). Therefore, we can obtain:

Theorem 6 *For the special case of a stream Υ of squares, given $0 < \epsilon, \delta < 1$, Algorithm 1 (ϵ, δ)-approximates the area of A with space complexity $\mathcal{O}(\frac{1}{\epsilon^2} \log U \log \frac{1}{\delta})$ bits, amortized time taken to process a square $\mathcal{O}(\frac{1}{\epsilon} \log \frac{U}{\epsilon} \log \frac{1}{\delta})$, and time taken to process a query for A at any time $\mathcal{O}(1)$.*

Theorem 6 can be also applied to fat rectangles. Each fat rectangle of aspect ratio h can be converted to a stream of at most $h + 1$ (or $1/(h+1)$ if $h < 1$) squares that cover the rectangle area. Thus, we can use the approach above for a constant factor increase in time.

6 Many Buckets

Here we give an analysis of Algorithm 1 for the general case of arbitrary input rectangles. Section 6.1 establishes a relation of (ϵ, δ)-estimators to ξ-approximations.

6.1 Area Estimation

Consider a set $\Upsilon = \{x_0, x_1, \ldots, x_{m-1}\}$ of m rectangles in $\mathbb{Z}_n^2$, such that each x_j is an $a_j \times b_j$ rectangle, $0 \leq a_j, b_j < \sqrt{U}$, with $a_j b_j$ discrete points, and total area A. Partition Υ arbitrarily into χ subsets $\{\Upsilon_0, \Upsilon_1, \ldots, \Upsilon_{\chi-1}\}$ where Υ_i has m_i rectangles and $m = \sum_{i=0}^{\chi-1} m_i$. Let A_i denote the area of the union of the rectangles in Υ_i. In each x_j of size $a_j \times b_j$, let x_j' denote an $a_j' \times b_j'$ rectangle contained in x_j, where $a_j' \leq a_j$ and $b_j' \leq b_j$ Define $\Upsilon', \Upsilon_i', A'$, and A_i' correspondingly. Let $0 < \Delta_1 \leq \frac{A'}{A} \leq \Delta_2 \leq 1$; we fix Δ_1 and Δ_2 later in Section 6.2.

Let us assume that there is an (ϵ, δ)-estimator algorithm $\mathcal{A}$ (similar to Theorem 1) that computes an estimate $\mathrm{est}(A_i')$ of the area in the rectangles in Υ_i' and returns: (i) $E_{\max} = \max_i \mathrm{est}(A_i')$, the maximum among $\mathrm{est}(A_i'), 0 \leq i \leq \chi - 1$, and (ii) $E_{\mathrm{sum}} = \sum_{i=0}^{\chi-1} \mathrm{est}(A_i')$, the sum of each $\mathrm{est}(A_i'), 0 \leq i \leq \chi - 1$. The algorithm $\mathcal{A}$ then uses the estimates $E_{\max}$ and E_{sum} to estimate quantity $\mathrm{est}(A)$ of A as $\mathrm{est}(A) = \sqrt{E_{\max} E_{\mathrm{sum}}}$. Define relative error as $\varrho = \frac{|\mathrm{est}(A) - A|}{A}$. We have that:

Lemma 7 *The (ϵ, δ)-estimator algorithm $\mathcal{A}$ computes an estimate $\mathrm{est}(A)$ of the total area A such that $A \cdot \epsilon_1 \leq \mathrm{est}(A) \leq A \cdot \epsilon_2$ with probability $(1-\delta)^\chi$, where χ is the number of blocks in a partition of set Υ of input rectangles, $\epsilon_1 = \frac{\Delta_1(1-\epsilon)}{\sqrt{\chi}}$, $\epsilon_2 = \Delta_2(1+\epsilon)\sqrt{\chi}$, and $0 < \Delta_1 \leq \Delta_2 \leq 1$.*

6.2 Approximation

We project each bucket $\mathbb{B}_i$ to $\mathbb{Z}_{2n}$ (as described in Section 4) and apply a range-efficient algorithm for F_0. Let Υ_i denote the original rectangles in Υ that are projected with bucket $\mathbb{B}_i$, and let Υ_i' denote the normal rectangles assigned to $\mathbb{B}_i$ (according to their aspect ratio).

Thus, $\Upsilon = \{\Upsilon_0, \Upsilon_1, \ldots, \Upsilon_{\chi-1}\}$, and after normalization $\Upsilon' = \{\Upsilon'_0, \Upsilon'_1, \ldots, \Upsilon'_{\chi-1}\}$. For each $x \in \Upsilon$ we use a single internal normal rectangle x' as given by Lemma 2, which is assigned to a bucket $\mathbb{B}_i$ according to its aspect ratio. Let A_i (resp. A'_i) denote the total area of Υ_i (resp. Υ'_i) . This results to $A_i/A'_i = \gamma$, $\gamma \leq 91$, from the area analysis in Lemma 3.

The Hits algorithm (Theorem 1) yields an (ϵ, δ)-approximation of F_0 for the ranges in each $\mathbb{B}_i$. The estimate est(A) of the total area A of Υ using $E_{\max}$ and E_{sum} is computed from the (ϵ, δ)-estimates of A'_i given by the Hits algorithm for each bucket.

From Lemma 7, substituting $\Delta_1 = 1/\gamma$ and $\Delta_2 = 1$, Algorithm 1 computes an estimate est(A) of the total area A of the set Υ of m rectangles mapped to χ buckets such that $A \cdot \epsilon_1 \leq \text{est}(A) \leq A \cdot \epsilon_2$, where $\epsilon_1 = \frac{1}{\gamma}\frac{(1-\epsilon)}{\sqrt{\chi}}$ and $\epsilon_2 = (1+\epsilon)\sqrt{\chi}$ with probability $(1-\delta)^{\chi}$. We set $\chi = 2n + 1 = \log U + 1$ (the total number of normal aspect ratios), and hence, $\epsilon_1 = \frac{1}{\gamma}\frac{(1-\epsilon)}{\sqrt{\log U+1}}$ and $\epsilon_2 = (1+\epsilon)\sqrt{\log U + 1}$ with probability $(1-\delta)^{\log U+1}$.

Algorithm 1 reaches est(A) with space complexity $\mathcal{O}(\frac{1}{\epsilon^2}\log^2 U \log\frac{1}{\delta})$ bits, amortized time taken to process a rectangle $\mathcal{O}(\log\frac{U}{\epsilon}\log\frac{1}{\delta})$, and the time taken to process a query for F_0 is $\mathcal{O}(\log U)$. These follow from Theorem 1 by combining the space and time complexities of $\log U + 1$ instances of Hits. In comparison to Hits, the space needed by our approximation algorithms increases by a factor of $\mathcal{O}(\log U)$ due to the $\log U + 1$ buckets. The amortized time taken to process a rectangle remains the same as we need to run Hits on only a single bucket for that rectangle. Since it is necessary to compute the median of $\log U + 1$ instances of Hits, one for each bucket, the time taken to process a query for A increases by a factor of $\mathcal{O}(\log U)$. By setting $\epsilon = 1/2$ and $\delta = \frac{1}{\log U+1}$, our algorithm approximates est(A), such that $\Omega(1/\sqrt{\log U}) \leq \text{est}(A)/A \leq \mathcal{O}(\sqrt{\log U})$, with constant probability.

Theorem 8 *Algorithm 1 $\mathcal{O}(\sqrt{\log U})$-approximates A with constant probability and achieves space complexity $\mathcal{O}(\log^2 U \cdot \log\log U)$ bits, amortized time to process a rectangle $\mathcal{O}(\log U \cdot \log\log U)$, and time taken to process a query at any time $\mathcal{O}(\log U)$.*

The asymptotic notation $\mathcal{O}(\sqrt{\log U})$ hides large constants, due to the fact that for each x_i we use some internal normalized square. We can improve the constants and bring them down to 1, by tiling each x_i with poly-log number of normalized rectangles as specified by the lemma below. This has the side effect of increasing the time complexity by a poly-log factor which can be traded-off for the enhanced accuracy.

Lemma 9 *A rectangle r can be $(1-\eta)$-approximately tiled with $4\log^2\frac{1}{\eta^2}$ normal rectangles, for $0 < \eta \leq 1/2$.*

7 Conclusions

We presented a randomized approximation algorithm with poly-log bounds on time and space complexity with approximation factor $1 \pm \epsilon$ for fat rectangles and $\mathcal{O}(\sqrt{\log U})$ for general rectangles for the KMP in the streaming model. Our technique will give an $\mathcal{O}((\sqrt{\log U})^{d-1})$-approximation for the general case of d-dimensional KMP. For future work, it would be interesting to explore techniques that will reduce the current approximation factor $\mathcal{O}(\sqrt{\log U})$ for general rectangles to $\mathcal{O}(1)$ or $1 \pm \epsilon$, using deterministic transformations. Moreover, it would be interesting to evaluate our algorithm experimentally in a real-time setting.

Acknowledgements: We are indebted to Srikanta Tirthapura for helpful suggestions for the problem.

References

[1] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *STOC*, pages 20–29, 1996.

[2] M. Benedikt and L. Libkin. Exact and approximate aggregation in constraint query languages. In *PODS*, pages 102–113, 1999.

[3] J. Bentley. Algorithms for Klee's rectangle problems, Unpublished notes, Computer Science Department, Carnegie Mellon University. 1978.

[4] K. Bringmann. An improved algorithm for Klee's measure problem on fat boxes. *Comput. Geom. Theory Appl.*, 45(5-6):225–233, 2012.

[5] K. Bringmann and T. Friedrich. Approximating the volume of unions and intersections of high-dimensional geometric objects. *Comput. Geom. Theory Appl.*, 43(6-7):601–610, 2010.

[6] E. Y. Chen and T. M. Chan. Space-efficient algorithms for Klee's measure problem. In *CCCG*, pages 27–30, 2005.

[7] V. Klee. Can the measure of $\cup[a_i, b_i]$ be computed in less than O$(n \log n)$ steps? *American Mathematical Monthly*, 84(4):284–285, 1977.

[8] G. Kuper, L. Libkin, and J. Paredaens. *Constraint Databases*. Springer, 1st edition, 2010.

[9] J. A. Orenstein and T. H. Merrett. A class of data structures for associative searching. In *PODS*, pages 181–190, 1984.

[10] A. Pavan and S. Tirthapura. Range-efficient counting of distinct elements in a massive data stream. *SIAM J. Comput.*, 37(2):359–379, 2007.

[11] S. Tirthapura and D. P. Woodruff. Rectangle-efficient aggregation in spatial data streams. In *PODS*, pages 283–294, 2012.

[12] J. Vahrenhold. An in-place algorithm for Klee's measure problem in two dimensions. *Inf. Process. Lett.*, 102(4):169–174, 2007.

Finding Shadows among Disks

Nataša Jovanović* Jan Korst† Zharko Aleksovski† Wil Michiels† Johan Lukkien* Emile Aarts*

Abstract

Given a set of n non-overlapping unit disks in the plane, a line ℓ is called blocked if it intersects at least one of the disks and a point p is called a shadow point if all lines containing p are blocked. In addition, a maximal closed set of shadow points is called a shadow region. We derive properties of shadow regions, and present an $\mathcal{O}(n^4)$ algorithm that outputs all shadow regions. We prove that the number of shadow regions is $\Omega(n^4)$ for some instances, which implies that the worst-case time complexity of the presented algorithm is optimal.

1 Introduction

Let $\mathcal{D}$ be a set of n closed and non-overlapping unit disks, i.e., disks with radius 1, in the two-dimensional plane. A line ℓ is called *blocked* if it intersects at least one of the disks in $\mathcal{D}$. A point p is called a *shadow point*, if all lines containing p are blocked. A point that is not a shadow point, is called a *light point*.

For a light point p it holds that there is at least one line in the plane that does not intersect any of the disks in $\mathcal{D}$. It follows that all the points outside the convex hull spanned by the disks are light points. In other words, all shadow points defined by the disks in $\mathcal{D}$ are inside the convex hull spanned by the disks, denoted as $H(\mathcal{D})$.

A closed shape S in the plane is a *shadow region* if each point in S is a shadow point and if S is maximal in the sense that there is no shape S' containing only shadow points for which $S \subset S'$. It follows that the collection of shadow regions partitions the set of shadow points. By definition, each disk $\delta \in \mathcal{D}$ is contained in a shadow region.

Shadow Regions Problem. *Given $\mathcal{D}$, determine the set $\mathcal{S}$ of shadow regions in the plane.*

In other words, we are interested in designing an efficient algorithm that outputs the set of all shadow regions, for a given set $\mathcal{D}$ of disks. Figure 1 illustrates a set of 17 shadow regions defined by 14 randomly positioned unit disks.

Motivation. Hollemans et al. [4] describe a method for detecting objects. It uses light emitters and sensors placed on the boundary of a rectangular detection area.

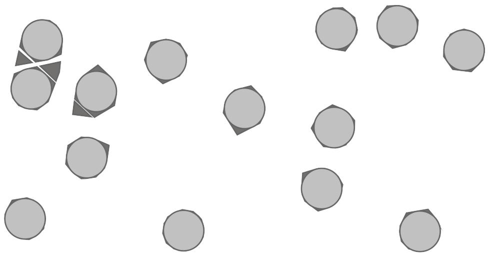


Figure 1: The shadow regions defined by 14 unit disks.

The shadow regions problem is related to the accuracy of the method in the following way. Each sensor continuously determines the set of emitters from which it receives light and the set of emitters from which it does not receive light because the line of sight is blocked by an object. Using this information, one can determine the set of shadow regions. As each object is located in a shadow region, this gives an approximation of the placement of the objects. Ideally, we have n shadow regions, each with a size that is exactly equal to the object it contains. However, this ideal situation will not occur since, besides being part of an object, a point can also be a shadow point because: (1) the density of emitters and sensors is too low, and (2) all lines going through the point can be blocked by surrounding objects. The shadow points resulting from the latter cause are an intrinsic shortcoming of the method. By subtracting the objects from the solution of the shadow regions problem we get the shadow areas where detection fails due to this occlusion.

Related work. The problem considered by Dumitrescu and Jiang in [3] is to some extent related to the shadow regions problem. The authors show the existence of dark points [10] in maximal disk packings. A point is called dark within a set of disks if any ray with apex in that point intersects at least one of the disks. Note that any dark point is by definition a shadow point, but not vice versa. In addition, they present an algorithm for finding all of the dark points that are on the boundary of disks in a given set. While these authors' interest is in the dark points, we focus on the shadow points. Furthermore, we present an optimal algorithm to determine all shadow points in the plane defined by the disks, not only the ones on the disk boundaries. The problem of detecting circular objects in the plane is considered by Jovanović, Korst and Pronk in [5], where the

*Eindhoven University of Technology, n.jovanovic@tue.nl
†Philips Research Eindhoven, jan.korst@philips.com

authors present two algorithms to approximate the objects by convex polygons, using a finite set of line segments, defined by a given set of emitters and sensors. More remotely related problems are the problems on illumination of convex bodies [9, 11] and the visibility problems concerning hiding or blocking points and unit disks by a set of unit disks [6, 7, 8].

2 Introducing shadow regions

Let ℓ be a line in the plane such that it intersects the convex hull $H(\mathcal{D})$ of disks. The line ℓ is called a *defining line* for a shadow region S if it contains an edge of S. One can prove the following lemma.

Lemma 1 *Let ℓ be a defining line for a shadow region S. Then the following holds:*

- *ℓ does not intersect any disk in $\mathcal{D}$ in more than one point*
- *ℓ is tangent to at least two disks in $\mathcal{D}$*
- *ℓ is not tangent to any three disks δ_1, δ_2 and δ_3, where δ_1 and δ_2 are on the same side of ℓ and δ_3 is such that its point of tangency with ℓ is between the points of tangency of δ_1 and δ_2 with ℓ.*

Now, let us take a look at a small example of $\mathcal{D}$ consisting of only three disks, so that we can get a notion on the size, shape and the number of shadow regions defined by the disks. Each two non-tangent disks define four common tangent lines: a pair of parallel tangent lines and a pair of crossing (intersecting) tangent lines. The four tangent lines define four shadow areas that are attached to the disks; see Figure 2. By definition, a disk and all its attached shadow areas represent one shadow region. Note that the size of these shadow regions depends on the distance between the disks: the closer the disks, the larger the shadow regions. Depending on the

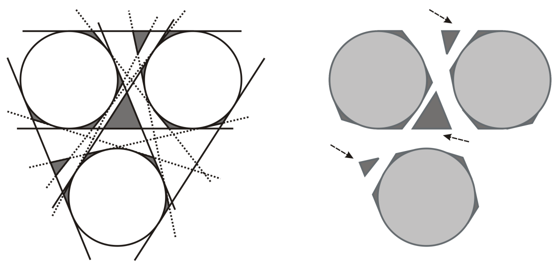


Figure 2: The shadow regions defined by 3 unit disks; the arrows point at the free shadow areas.

mutual distance, the three disks may define one or more *free* shadow regions, i.e., shadow regions that are not attached to any of the disks; see Figure 2. A free shadow region is bounded by line segments only, thus, it has the shape of a polygon. It can be shown that three disks can define at most 4 free shadow regions, which implies that they can define 1 to 7 shadow regions in total.

A shadow region can be formally represented by a cyclic sequence of points $p_0, p_1, \ldots, p_k$, where each two neighboring points are connected by either a line segment or a circular arc of radius 1.

Generally, n disks define at most $2n(n-1)$ common tangent lines, which can partition the plane into $\mathcal{O}(n^4)$ non-overlapping convex polygons that contain either shadow points only or light points only. In Section 5, we will prove that there are instances for which the number of shadow regions defined by n disks is $\Omega(n^4)$.

Lemma 2 *A shadow region is convex.*

Proof. We prove the lemma by contradiction. Hence, assume that a shadow region S is not convex. Let p be a light point inside the convex hull $H(S)$ of S and outside S. Each line containing p intersects the shadow region S, which implies that it is blocked. This implies that p is a shadow point, which is in contradiction with the assumption of p being a light point. $\square$

As a consequence of Lemma 1, in the process of determining the shadow regions, we consider only the set $\mathcal{T}$ of defining lines.

Let $t \in \mathcal{T}$ be a line tangent to two disks δ_1 and δ_2 in $\mathcal{D}$. The points of tangency between the line t and the disks δ_1 and δ_2 divide t into three parts: one line segment denoted by s, and two rays denoted by r and r'. One can prove the following lemma.

Lemma 3 *If disks δ_1 and δ_2 are not on the same side of t, line segment s does not define a shadow region. If disks δ_1 and δ_2 are on the same side of t, the rays r and r' do not define a shadow region.*

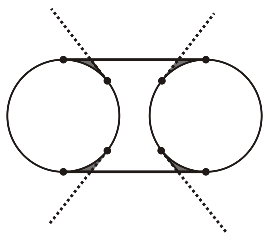


Figure 3: Parts of the tangent lines that define the shadow regions.

From Lemma 3, each crossing tangent line may be involved in the definition of shadow regions through the pair of rays with apices in the points of tangency. The parallel tangent lines are involved in the definition of shadow regions through the line segments connecting the points of tangency; see Figure 3.

3 Modelling light corridors

Let $\mathcal{L}$ be the set of all lines in the plane that do not intersect any disk, hence, $\mathcal{L}$ is the set of lines that only contain light points. Set $\mathcal{L}$ can be partitioned into two subsets, dividing lines and non-dividing lines. For a non-dividing line all disks are on the same side of that line. Each dividing line specifies a bipartition of the set of disks into non-empty sets. All dividing lines specifying the same bipartition of disks in $\mathcal{D}$ form a light corridor; see Figure 4. Note that each light point is contained in one or more light corridors. This means that the collection of shadow regions is given by the difference between $H(\mathcal{D})$ and the union of all light corridors.

Let $\mathcal{T}$ be the set of all defining lines. A light corridor can be characterized by its two crossing tangent lines t and t' in $\mathcal{T}$ that are clockwise fixed and counterclockwise fixed, respectively; see Figure 4. For t this means that it cannot be rotated in clockwise direction around any point on the line over any angle θ such that it does not intersect at least one of the disks in more than one point. An analogous interpretation holds for t'. These lines define the "in" and "out" of the corridor through $H(\mathcal{D})$. Inside the convex hull $H(\mathcal{D})$, each light corridor is an open non-convex area, bounded by a set of line segments and a set of circular arcs of radius 1.

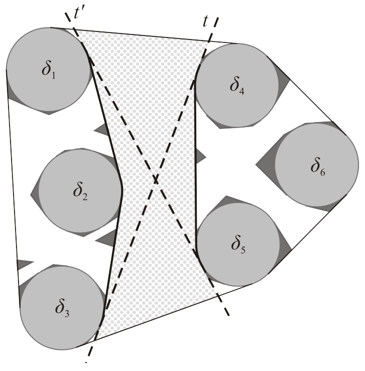


Figure 4: An example of a light corridor.

Lemma 4 *The number of light corridors defined by n non-overlapping unit disks is at most $\frac{n(n-1)}{2}$.*

Proof. The n disks define at most $n(n-1)$ crossing tangent lines in $\mathcal{T}$. Each crossing tangent line in $\mathcal{T}$ defines one bipartition of disks, which corresponds to exactly one light corridor. Hence, the number of light corridors is not larger than the number of crossing tangent lines in $\mathcal{T}$. Moreover, each light corridor is characterized by a pair of crossing tangent lines, which implies that the number of light corridors is at most $n(n-1)/2$. $\square$

4 Algorithm

In this section, we present an algorithm for determining the set of all shadow regions defined by n non-overlapping unit disks. We give the algorithm in a step-by-step manner and discuss its overall time complexity.

The algorithm for determining all shadow regions defined by n unit disks consists of the following four main steps:

1. Determine the convex hull $H(\mathcal{D})$;
2. Determine the set $\mathcal{T}$ of all defining tangent lines;
3. Determine all light corridors inside $H(\mathcal{D})$;
4. Determine the union U of all light corridors and next, the set of all shadow regions, by finding the set difference between $H(\mathcal{D})$ and U.

Let us now take a closer look at each step of the algorithm and its worst-case time complexity. In the first step, it is needed to compute first the convex hull of the disk centers, and then to compute an offset polygon, which can be done in $\mathcal{O}(n \log n)$ time [2].

As defined in Section 2, the set $\mathcal{T}$ of defining lines are the lines tangent to at least two disks in $\mathcal{D}$ that do not intersect any of the disks in $\mathcal{D}$ in more than one point. Now, we can determine the set $\mathcal{T}$ of all defining lines in $\mathcal{O}(n^2 \log n)$ time, as follows. For each disk in $\mathcal{D}$, we sort radially the other $n-1$ disks, which takes $\mathcal{O}(n \log n)$ time. This structure allows to find all defining lines of one disk in linear time. In addition to each defining line determined, we keep the information on tangent disks and the tangency points, the type of the tangent line, i.e., whether it is a crossing line or not, and the part(s) of the line which are involved in the definition of the shadow regions, i.e., the rays or the line segment, as explained in Lemma 3. Hence, it takes $\mathcal{O}(n \log n)$ time to determine all defining lines of one disk and all the additional properties. Therefore, finding the set $\mathcal{T}$ of defining tangent lines for all n disks takes $\mathcal{O}(n^2 \log n)$ time.

In order to determine the set of all light corridors, for each of the disks we need the sorted list of all its points of tangency, in a cyclic order. Such a list can be determined in $\mathcal{O}(n^2 \log n)$ time since all the defining lines are determined, hence, all the points of tangency for each of the disks.

As mentioned in the proof of Lemma 4, a crossing tangent line in $\mathcal{T}$ characterizes one light corridor. Starting with a crossing line from $\mathcal{T}$, we determine the corresponding light corridor as follows. We start by including one ray of the chosen crossing line. Then, we simply look up the corresponding point of tangency on the tangent disk and take the successor point of tangency from the sorted list of points for that disk. That point is a starting point for either a line segment, or a ray of some other tangent line. In the case of a starting point of a line segment, we look up the ending point on the next disk, etc. The computation of one side of the corridor is finished when a ray occurs in the sequence. The other side is determined in the same way, starting with the other ray of the originally chosen crossing tangent line.

From Lemma 4, the number of light corridors is $\mathcal{O}(n^2)$. In addition, the number of all defining tangent lines is also quadratic in the number of disks, which implies that the total number of all rays (2 rays per crossing tangent) and line segments (1 line segment per parallel tangent) together is also $\mathcal{O}(n^2)$. In this way, amortized over all iterations, the light corridors can be determined in $\mathcal{O}(n^2)$ time, which implies that the total time complexity of the third step of the algorithm is $\mathcal{O}(n^2 \log n)$.

The problem of determining the union U of all light corridors comes down to the problem of finding the intersections of a set of line segments and circular arcs. This is a well-known and extensively studied problem [2]. Using the deterministic algorithm by Balaban [1], the intersections of N line or curve segments can be determined in $\mathcal{O}(N \log N + K)$ time, where K is the number of intersecting pairs. Given that we have $\mathcal{O}(n^2)$ line segments and circular arcs, the number K of intersecting pairs is $\mathcal{O}(n^4)$. Therefore, using this algorithm, the union U of all light corridors can be determined in $\mathcal{O}(n^4)$ time. The set of all shadow regions is then simply determined as a complement set of U within the convex hull $H(\mathcal{D})$.

With the discussion above, we get to the following result.

Theorem 5 *The set of all shadow regions defined by n non-overlapping unit disks can be determined in $\mathcal{O}(n^4)$ time.*

5 Determining the number of shadow regions

In the previous section we presented an $\mathcal{O}(n^4)$ algorithm for deriving all shadow regions created by a set of n disks. From this it follows that a set of n disks defines $\mathcal{O}(n^4)$ shadow regions. In this section we show that this bound is tight, i.e., that problem instances exist with $\Omega(n^4)$ shadow regions. This implies the interesting result that the $\mathcal{O}(n^4)$ worst-case time complexity of the presented algorithm is optimal.

To construct a problem instance with $\Omega(n^4)$ shadow regions, we place the disks in two "columns", where each column contains n equidistant disks, such that each disk of one column is directly opposite to a disk of the other column. The idea behind the construction is to obtain a quadratic number of thin light corridors that pass between the disks of the two columns, i.e., in the left to right direction. If these corridors do not intersect within some finite area of width w, then adding another $2n$ disks that, in the same way, create quadratic number of light corridors in the top-bottom direction, results in $\Theta(n^4)$ shadow regions; see an illustration in Figure 5. If we need to add only linear number of mutually tangent disks to block the light corridors that come from other (e.g., diagonal) directions, we then have a linear number of disks creating $\Theta(n^4)$ shadow regions.

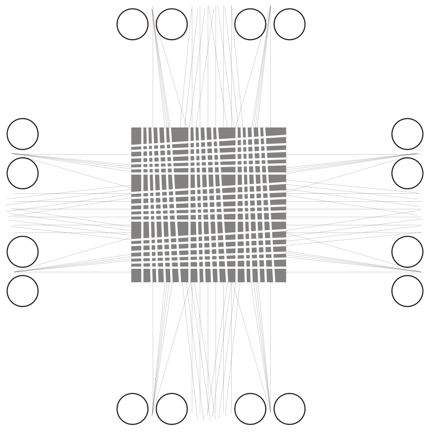


Figure 5: Constructing $\Omega(n^4)$ shadow regions with a linear number of disks - an illustration.

Let L be the line connecting the centers $O_1, O_2, \ldots, O_n$ of the disks $\delta_1, \delta_2, \ldots, \delta_n$ in the left column and, in the same fashion, let R be the line connecting the centers $O'_1, O'_2, \ldots, O'_n$ of the disks $\delta'_1, \delta'_2, \ldots, \delta'_n$ in the right column; see Figure 6. Furthermore, let h denote the distance between the columns, i.e., the distance between L and R, and let d denote the distance between two neighboring disks in one column, measured from center to center. Given h, the distance d is chosen so that the top two disks of one column and the bottom two disks of the other column are all tangent to the same line. In this way, there is no light corridor defined by these top-bottom pairs of disks, however, there is exactly one light corridor between any other two pairs of neighboring disks in different columns. From the congruence of the two gray triangles in Figure 6, the relation between the distances d and h is given by

$$d = \frac{2h}{\sqrt{h^2 - 4(n-2)^2}} \tag{1}$$

For the time being, we only consider the light corridors between pairs (δ_i, δ_{i+1}) of neighboring disks from the left column and pairs $(\delta'_j, \delta'_{j+1})$ of neighboring disks from the right column, where $i, j \in \{1, \ldots, n-1\}$.

The distance d between the neighboring disks determines the width of the corridors. From Equation (1), we get that if $h \to \infty$, then $d \to 2$. Furthermore, using elementary calculus, it can be shown that increasing

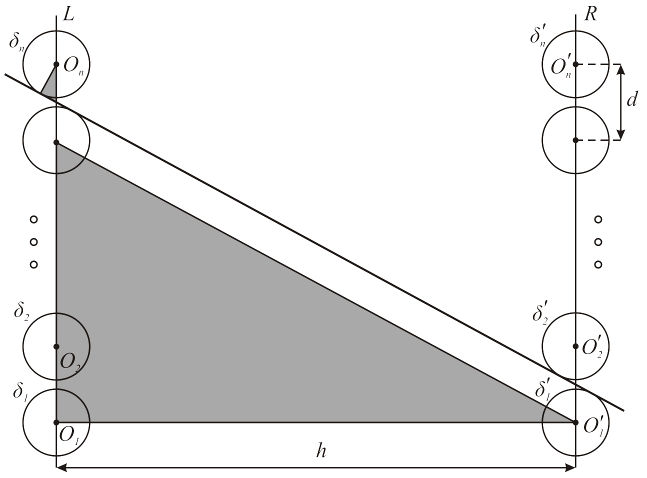


Figure 6: The columns of disks, each column containing n disks.

the distance h between the columns results in decreasing the width of the corridors. Note that the corridors are not all of the same width, i.e., the longer corridors are thinner than the shorter corridors.

It remains to be shown that there is an area between the columns where no two corridors intersect. Furthermore, we want to show that for some h, the width w of that area can be at least nd. In this way, overlapping (or intersecting) this area containing the left to right corridors with the area containing the top to bottom non-intersecting corridors, results in creating $\Theta(n^4)$ shadow regions.

In Section 3 we showed that each light corridor is characterized by a pair of two crossing tangent lines. In this special case of disks being placed in two columns, one can easily show that, between the columns, each corridor is bounded by a pair of parallel line segments. Considering the left column as the beginning and the right column as the end of the corridors, among the intersection points of the corridors' bounding line segments, we can distinguish two subsets of points: the splitting points and the meeting points; see Figure 7. The splitting point of two light corridors that begin between the same pair of disks is the common (intersection) point of these corridors furthest from L. In a similar way, the meeting point of two corridors that do not begin between the same pair of disks is the intersection point of these two corridors closest to L. Let P_s denote the vertical line containing the splitting point(s) furthest from L and let P_m denote the vertical line containing the meeting point(s) closest to L. Clearly, if the distance $\bar{h}_s$ between P_s and L is smaller than the distance $\bar{h}_m$ between P_m and L, the area between the two vertical lines P_s and P_m, gives an area inside which no two corridors intersect. In addition, the width w of the area is given by

$$w = \bar{h}_m - \bar{h}_s \tag{2}$$

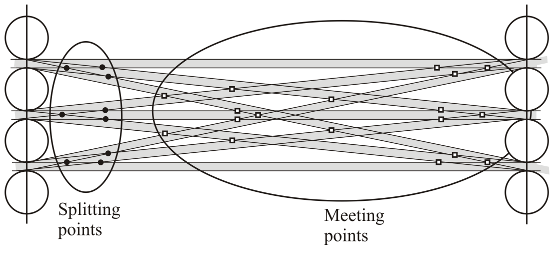


Figure 7: The splitting points and the meeting points of nine light corridors passing between eight disks in the columns.

Next, we express the distance $\bar{h}_s$ as a function of the distance h between the columns of disks. Let us consider only the $n-1$ light corridors that all begin between one pair of neighboring disks in the left column. One can show that among the splitting points of these corridors, the splitting point furthest from L, is the splitting point of two neighboring light corridors, i.e., the corridors that end between the neighboring pairs of disks in the right column. Let h_s be the distance from the splitting point P of an arbitrary pair of neighboring corridors to the line L. By definition, $\bar{h}_s$ is the maximum of all distances h_s of the splitting points of all neighboring corridors. Using elementary calculus, one can prove the following lemma.

Lemma 6 *For an arbitrary pair of neighboring light corridors C_j and C_{j+1}, it holds that*

$$\lim_{h\to\infty} h_s = 0.$$

In other words, for h large enough, all light corridors split on distance ϵ from the line L and "enter" the area in which they do not intersect.

From Equation (2), to determine the width w of the area where the light corridors do not intersect, besides the distance $\bar{h}_s$, we also need to determine the distance $\bar{h}_m$, i.e., the distance from the closest meeting point(s) to the line L. We first determine the light corridors that define the closest meeting point(s).

One can show that the light corridors that define the closest meeting point(s) begin between two neighboring pairs of disks; see Figure 7. More precisely, the bottom-most corridor C_b of all corridors beginning between the pair of disks (δ_{j+1}, δ_j) and the top-most corridor C_t of all corridors beginning between the pair of disks (δ_j, δ_{j-1}) define (one of) the closest meeting point(s) to the line L. Let h_m be the distance from the splitting point P' of the corridors C_b and C_t to the line L. Note that C_b ends between the bottom pair of disks (δ'_1, δ'_2) and C_t ends between the top pair of disks $(\delta'_n, \delta'_{n-1})$ in the right column. In a similar way as Lemma 6, using elementary calculus, one can prove the following lemma.

Lemma 7 *For the distance h_m, it holds that $h_m \to \infty$, when $h \to \infty$.*

From Lemma 6 and Lemma 7 and Equation (2), we can conclude that for h large enough, the width w of the area where corridors do not intersect can be of size nd. Note that the area is not in the middle between the columns. Instead, we have two such areas of non-intersecting corridors adjacent to the left and to the right column, respectively. In the next step of the construction, we add

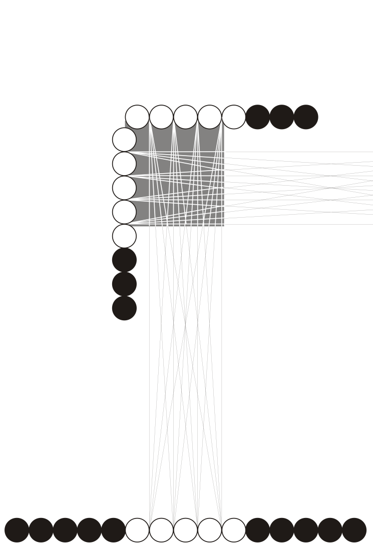


Figure 8: A linear number of unit disks defining $\Theta(n^4)$ shadow regions - the thin light corridors pass between the white disks; the black disks are mutually tangent, hence, representing the blocking disks.

$2n$ disks organized in two rows that are on the top and the bottom side, as we mentioned earlier in this section, and such that the areas of non-intersecting corridors completely overlap. Each of the $\mathcal{O}(n^2)$ light corridors in the left to right direction intersects each of the $\mathcal{O}(n^2)$ light corridors in the top to bottom direction. Hence, they partition the square area of size $(nd)^2$ into $\Theta(n^4)$ regions. In order for these regions to be the *shadow* regions, the light coming from directions different than left, right, top or bottom must be blocked. Therefore, in addition to the $4n$ disks used in this construction, we "close the gaps" by extending, for example, the top row and the left column by $\lceil \frac{n}{2} \rceil$ tangent disks each and the right column and the bottom row with $2n$ tangent disks each; see Figure 8. These blocking disks ensure that there are no additional corridors intersecting the area partitioned into shadow regions by the constructed light corridors.

6 Concluding remarks

We considered the problem of determining all shadow regions defined by a set of n non-overlapping unit disks in the plane. We discussed the basic properties of the shadow regions and we presented an $\mathcal{O}(n^4)$ algorithm for determining them. We showed that the number of shadow regions can be $\Omega(n^4)$. Hence, the presented algorithm determines all the shadow regions in worst-case optimal time.

References

[1] I.J. Balaban. An optimal algorithm for finding segment intersections. In *Proc. of the 11th ACM Symposium on Computational Geometry*, 1995, Vancouver, Canada, 211–219.

[2] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*, 2nd Edition, 2000, Springer.

[3] A. Dumitrescu, and M. Jiang. The forest hiding problem. In *Proc. of the 21st ACM-SIAM Symposium on Discrete Algorithms*, 2010, Austin, Texas, USA.

[4] G. Hollemans, T. Bergman, V. Buil, K. van Gelder, M. Groten, J. Hoonhout, T. Lashina, E. van Loenen, and S. van de Wijdeven. Entertaible: multi-user multi-object concurrent input. In *Proc. of the 19th Annual ACM Symposium on User Interface Software and Technology*, 2006, Montreux, Switzerland, 55–56.

[5] N. Jovanović, J. Korst, and V. Pronk. Object detection in flatland. In *Proc. of the 3rd International Conference on Advanced Engineering Computing and Applications in Sciences*, 2009, Sliema, Malta.

[6] N. Jovanović, J. Korst, and A.J.E.M. Janssen. Minimum blocking sets of circles for a set of lines in the plane. In *Proc. of the 20th Canadian Conference on Computational Geometry*, 2008, Montréal, Canada, 91–94.

[7] N. Jovanović, J. Korst, R. Clout, V. Pronk, and L. Tolhuizen. Candle in the woods: asymptotic bounds on minimum blocking sets. In *Proc. of the 25th ACM Symposium on Computational Geometry*, 2009, Aarhus, Denmark, 148–152.

[8] N. Jovanović, J. Korst, Z. Aleksovski, and R. Jovanović. Hiding in the crowd: asymptotic bounds on minimum blocking sets. In *Proc. of the 26th European Workshop on Computational Geometry*, 2010, Dortmund, Germany, 197–200.

[9] H. Martini, and V. Soltan. Combinatorial problems on the illumination of convex bodies. *Aequationes Mathematicae 57*, 1999, 121–152.

[10] J. Mitchell. Dark points among disks. In *Open Problems from the 2007 Fall Workshop in Computational Geometry*, Hawthorne, New York, USA.

[11] L. Szabo, and Z. Ujvary-Menyhart. Clouds of planar convex bodies. In *Aequationes Mathematicae 63*, 2002, 292–302.

Computing the Coverage of an Opaque Forest [*]

Alexis Beingessner Michiel Smid [†]

Abstract

We consider the problem of taking an opaque forest and determining the regions that are covered by it. We provide a tight upper bound on the complexity of this problem, and an algorithm for computing this area, which is worst-case optimal.

1 Introduction

Let a *region* be any bounded, closed, and connected set of points in $\mathbb{R}^2$. Then a *barrier*, or *opaque forest*, of a finite set R of regions, is any finite set B of closed and bounded line segments, such that for any line ℓ: if ℓ intersects R then ℓ also intersects B. A previously studied problem is as follows: given some set R or regions, compute a barrier B such that the length of all the segments in B, $|B|$, is minimal. The exact solution to this problem is not known even for specific cases, such as when R is a unit square. The best known bounds for this instance of the problem are $2 \leq |B| \leq \sqrt{2} + \sqrt{6}/2$. [2]

The general problem of computing a minimal barrier for a given set of regions is a very difficult one. Currently there are no proven algorithms for computing this precisely, nor even known solutions for specific cases. For the internally optimal barrier, there is also no known algorithm. However, by further restricting the problem, it is reducible to well studied problems. If the internally optimal barrier is restricted to a single connected component, then this is easily reducible to the *Minimal Steiner Tree Problem.* If the barrier is further restricted to a single polygonal chain, then the problem is reducible to the *Travelling Salesman Problem.* Both of these problems are known to be NP-Hard in general, but can be much more easily computed or approximated when the input points are in convex position, which is the case for this problem [2].

In this paper we consider the following problem: given some barrier B, compute a maximal set R of regions such that B is a barrier for R. More precisely, given a set B of n line segments, compute $R(B) = \{p \in \mathbb{R}^2 :$ every line through p intersects $B\}$. We say that $R(B)$ is the *coverage* of B.

We give an algorithm that computes the coverage of an opaque forest in $O(n^4)$ time. We also provide an example of an opaque forest whose coverage has size $\Omega(n^4)$. Thus, our algorithm is worst-case optimal.

2 Maximal Regions

Let a *maximal region* of a set P of points be a region R such that for every point p in R, there exists an open ball A centered at p such that $A \cap R = A \cap P$.

Lemma 1 *If a maximal region of $R(B)$ is a line segment, then that line segment is part of B.*

Proof. Assume this is not the case. Then there is some line segment $S \in R(B)$ that is a maximal region, but is not in B. Therefore all lines that pass through a point p in S intersect B, and there exists an open ball A of points around p such that every point q in A that is not in S has a line ℓ through it which does not intersect B.

Consider such a point q. The line ℓ through q that does not intersect B cannot intersect S, or else the points it intersects in S are not actually in $R(B)$. We can select a point q' such that it is arbitrarily close to p, and the line ℓ' must therefore become ever more parallel to the line S lays on to avoid intersection. Therefore it must be the case that the line collinear with S intersects B, but the line ℓ' that is parallel to S and arbitrarily close to it does not. Therefore, there must exist some line segment $S' \in B$ that is parallel to S. Further, there must be some opaque forests to the left and right of S that do not meet each other or S', or else ℓ can pass through S. Therefore, there is a space for parallel lines to the left and right of S. However, this implies that there is a line ℓ'' that enters through one space and exits through the other which does not intersect B but passes through S, which means there are points in S which are not in $R(B)$. If this were not the case, then ℓ' would intersect B. So we have a contradiction, therefore if S is in $R(B)$, S is in B. □

Lemma 2 *$R(B)$ may contain maximal regions that are single points, but are not part of B.*

Proof. Consider the construction of three line segments found in Figure 2.

p is not part of B. Every line that passes through p intersects B, so $p \in R(B)$. Yet there exists an open ball of points centred at p such that every point in this ball

[*]This work was supported by NSERC

[†]School of Computer Science, Carleton University

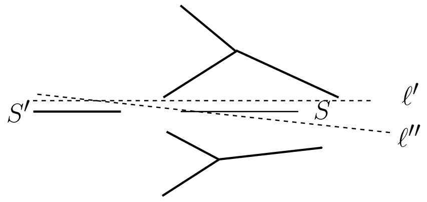


Figure 1: There is a line ℓ'' which does not intersect B but passes through S

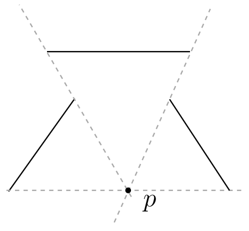


Figure 2: A construction that creates a maximal region that is exactly one point

except for p has a line through it that does not intersect B. Therefore p is a maximal region of $R(B)$. □

3 Connected Components

B is a set of n line segments consisting of m connected components $B_1, \ldots, B_m$. Further, $Conv(B_i)$ is the convex hull of the connected component B_i. Then for some point $p \in \mathbb{R}^2$, we define $L_p(B_i)$ as follows:

1. If B_i is a single line segment, and p is collinear to B_i, then $L_p(B_i) = \emptyset$
2. Otherwise, if p lies on a vertex of $Conv(B_i)$, then $L_p(B_i)$ is the double-wedge defined by the lines of the two edges of $Conv(B_i)$ that meet at p.
3. Otherwise, if p lies inside $Conv(B_i)$, or on its boundary, $\partial Conv(B_i)$, then $L_p(B_i) = \mathbb{R}^2$
4. Otherwise, $L_p(B_i)$ is the double-wedge defined by the tangents of $Conv(B_i)$ that pass through p.

Intuitively, $L_p(B_i)$ can be thought of as the set of all lines that pass through p and intersect B_i. However this is not strictly true. For parts (3) and (4) of the definition, this does in fact hold. However, (2) describes the limiting behaviour of a point as it tends towards a vertex of $Conv(B_i)$ from outside. (1) ignores the behaviour of points collinear to a single disjoint line segment. This definition may seem counter-intuitive, but it is useful for us. Further, we will consider $L_p(B_i)$ to be a subset of $\mathbb{R}^2$, and not the actual lines that pass through p.

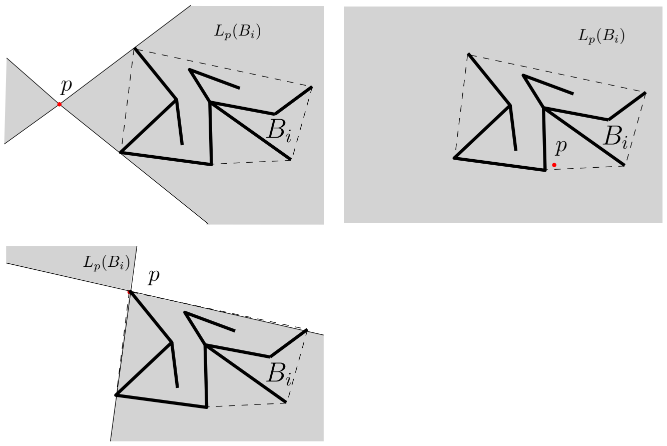


Figure 3: Various possible cases for $L_p(B_i)$

4 Clear and Blocked Points

Let a *blocked point* be a point p with respect to some barrier B such that for every line ℓ which passes through p, ℓ intersects B. Then a *clear point* is a point which is not blocked. Every point of B is a blocked point. Moreover, $R(B)$ is the set of all blocked points with respect to B, and the complement $\overline{R(B)}$ or $R(B)$ is the set of all clear points.

Theorem 3 *For every barrier B, each maximal region $C \subseteq R(B)$ is the intersection of halfplanes defined by lines that pass through two vertices of B.*

Proof. Assume there exists some line ℓ which is tangent to the boundary of a maximal region $C \subseteq R(B)$, but ℓ does not touch B. Then, because the complement of B is an open set, ℓ can be translated to intersect C without intersecting B. However that would mean C contains clear points, which is a contradiction. Therefore, ℓ must be tangent to B at at least one point. Now assume ℓ is tangent to B at exactly one point. Then ℓ can still be rotated around the point of tangency, once more intersecting C. This once more contradicts the fact that C is a subset of $R(B)$. Therefore ℓ must be tangent to at least two points of B. Further, since B is a set of line segments, only the end points of these segments need be considered, as tangency to a line segment is simply tangency to its two end points. □

Remark that this also implies that we need only finitely many halfplanes to define a maximal region of $R(B)$, and that every maximal region of $R(B)$ is convex.

Lemma 4 *Every point in $\overline{L_p(B_i) \cup B_i}$ is a clear point with respect to B_i.*

Proof. In case (1), where B_i is a single line segment and p is a point collinear with it, this follows trivially, as $R(B_i) = B_i$. Therefore, even though $\overline{L_p(B_i)} = \mathbb{R}^2$,

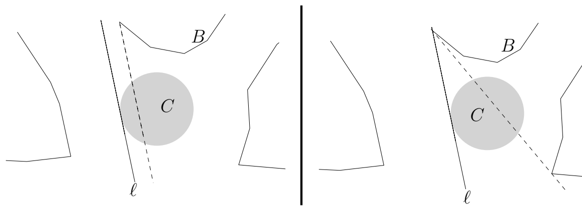


Figure 4: The line ℓ must be tangent to B at two vertices, if it defines the boundary of part of $R(B)$

the only points that aren't clear are those of B_i itself. In case (2), because B_i is a closed set, we can get points arbitrarily close to the vertex p lies on. Therefore while it is not the case that some line ℓ that makes up $\overline{L_p(B_i)}$ passes through p and does not intersect B_i, there exists some line ℓ' arbitrarily close to ℓ for which this is true. In case (3), where p lies inside of or on $Conv(B_i)$ this also follows trivially, as $\overline{L_p(B_i)}$ is empty. In case (4), where p lies outside of $Conv(B_i)$, the set of all lines that pass through p and intersect B_i are exactly those that are between the two tangents of B_i with respect to p. Therefore $\overline{L_p(B_i)}$ is a set of clear points. $\square$

We now define $L_p(B) = \bigcup_{i=1}^{m} L_p(B_i)$ to be the set of all lines which intersect B and pass through p, ignoring previously established special cases.

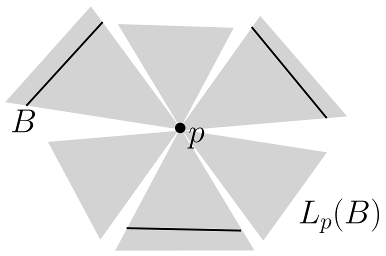


Figure 5: $L_p(B)$ is the set of all lines that intersect B and pass through some point p

Since $\overline{L_p(B_i) \cup B_i}$ is a set of clear points with respect to B_i, we can further conclude that $\overline{L_p(B) \cup B}$ has this property with respect to the whole of B. Further, for some points r and s, since $\overline{L_r(B) \cup B}$ and $\overline{L_s(B) \cup B}$ have this property, $\overline{L_r(B) \cup B} \cup \overline{L_s(B) \cup B}$ also has this property. By DeMorgan's law for set compliments, we can also conclude that $\overline{(L_r(B) \cap L_s(B)) \cup B}$ has this property as well. Therefore given

$$L(B) = \bigcap_{i=1}^{m} \bigcap_{p:\ \text{vertex of } Conv(B_i)} L_p(B)$$

we know $\overline{L(B) \cup B}$ is a set that also has this property.

Theorem 5 *Let CI be the closure of the interior of a set of points, then $CI(L(B)) \cup B \subseteq R(B) \subseteq L(B) \cup B$. Further, $R(B) \setminus (CI(L(B)) \cup B)$ is a finite set of disjoint points.*

Proof. Since $\overline{R(B)}$ is the set of all clear points with respect to B, and $\overline{L(B) \cup B}$ is a set of some clear points with respect to B, $\overline{R(B)} \supseteq \overline{L(B) \cup B}$. Therefore, $R(B) \subseteq L(B) \cup B$.

From Lemmas 1 and 2, we know that the only zero area maximal regions of $R(B)$ that aren't in B are individual points. Remark that $CI(L(B))$ differs from $L(B)$ in that only the zero area maximal regions of $L(B)$ have been removed. Therefore, if $CI(R(B)) = CI(L(B))$, all that $R(B)$ and $CI(L(B))$ may differ by are disjoint points. Since $R(B) \subseteq L(B) \cup B$, and B has zero area, $CI(R(B)) \subseteq CI(L(B))$, so all that remains to be proven is $CI(L(B)) \subseteq CI(R(B))$. Equivalently, $\overline{CI(R(B))} \subseteq \overline{CI(L(B))}$

Assume some postive-area region P of points is in $\overline{CI(R(B))}$. Consider a point $p \in P$. There is some line ℓ through p that does not intersect B. Then ℓ can be rotated around p without intersecting B until it is tangent with some connected component B_i at some point p'. We will call this rotated line l'. Now if $p \notin \overline{CI(L(B))}$, then there exists some $L_q(B_j)$, $j \neq i$, which p is in. This would mean there is some line ℓ'' through p and p' such that ℓ'' intersects B_j.

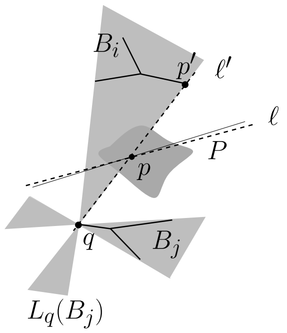


Figure 6: There exists some $L_q(B_j)$, $j \neq i$, which p is in

However ℓ'' is ℓ', and if ℓ' intersects B_j then there are three possibilities. Either ℓ intersects B_j, we should have stopped at B_j before we got to B_i, or ℓ' is tangent to B_j as well. For the first two cases we have a contradiction, so ℓ' must be tangent to B_j. However, since p is part of some region with positive area, we may take a point $p'' \notin \ell'$ adjacent to p such that it lies on no such tangent, and for which this case is therefore not possible. Therefore $p'' \in CI(L(B))$ or else there is a contradiction. Remark that this argument holds for any choice of p'' that does not lie on a tangent between two connected components. If the points on these tangents were not in $\overline{CI(L(B))}$ this would imply a region of zero area exists in $CI(L(B))$, but this is impossible. Therefore all points around p must be in $\overline{CI(\overline{L(B)})}$, and therefore p must be as well. Therefore, if $p \in \overline{CI(R(B))}$, $p \in \overline{CI(L(B))}$, and therefore $CI(L(B)) \subseteq CI(R(B))$.

Since $CI(R(B)) = CI(L(B))$, $R(B) \setminus CI(L(B))$ is a set of disjoint points. To prove that there are finitely many points, recall that by Theorem 3 each maximal region of $R(B)$ is an intersection of halfplanes defined by the vertices of B. The only way to get a point from this process is where three or more halfplane boundaries intersect at a point. Since there are finitely many vertices and therefore finitely many halfplanes, it follows that there are finitely many points. □

5 Computing the Coverage of a Barrier

Theorem 5 provides a procedure for computing $R(B)$.

We will assume our input is given as a list B of m connected components $B_1, \ldots, B_m$, totalling n line segments. The first step of our algorithm will be to compute the convex hulls of all m components. Next, for each vertex p_k of each $Conv(B_i)$, we will compute $L_{p_k}(B_j)$ for each $Conv(B_j)$, and union together these $L_{p_k}(B_j)$ into $L_{p_k}(B)$ by sorting them by angle. Then we will construct an arrangement using all the lines of the $L_{p_k}(B)$. We can then determine our final result by determining how many $L_{p_k}(B)$ one cell is part of, and then traversing the dual while changing our count according to whether a given edge exits or enters an $L_{p_k}(B)$. Then we simply output those regions which were in every $L_{p_k}(B)$, as well as B itself. However this process returns $CI(L(B)) \cup B$, so we may still be missing a finite number of points.

To compute these points, recall that they must lay at the intersection of 3 or more halfplanes. While this is necessary, it is not sufficient. The only way we know of to be certain a point is in $R(B)$ is to perform a radial plane sweep on B from that point. Since there are $O(mn)$ lines in the arrangement, there are $O(m^2n^2)$ candidate points. We will consider a line ℓ that makes up the arrangement. There are $O(mn)$ points of intersection on this line. First we will perform a radial plane sweep on one of these points p to construct a set $\Theta = \{\theta_1, \ldots, \theta_k\}$ of points on the interval 0 to π, where each point θ_i represents the angle of a tangent to some B_j from p, and each point is labelled with the number of connected components the line through p at the angle $\theta_i + \epsilon$ intersects. If every θ_i is labelled with a non-zero value, then $p \in R(B)$ and we return it. Now consider the intersection point q on ℓ that is adjacent to p. While most of the exact values of Θ will change, the ordering and labelling of the points will only change for those related to the tangents that bound this segment of ℓ. We can store this data in the vertices of the arrangement during construction, so we can just query p and q for this information. By updating just these values and checking if any are now labelled with 0, we now know if q is in $R(B)$. Repeating this process for all the points on ℓ, and then for all choices of ℓ, we will have determined all the points in $R(B)$.

Computing the convex hulls will take $O(n \log n)$ time. Computing $L_{p_k}(B_j)$ requires computing two lines. In all the special cases this takes constant time, however in the case where we must actually compute the lines as tangents, we take $O(\log n)$ time to binary search $Conv(B_j)$'s vertices for the most extreme points. Since there are $O(m)$ B_j, it takes $O(m \log n)$ time to compute them all for one p_k. Further, to union them together into $L_{p_k}(B)$, we need to sort their lines by angle, which will take $O(m \log m)$ time. Since there are $O(n)$ p_k, we take $O(nm(\log n + \log m))$ time to compute them all. Since we now have $O(nm)$ lines from all our $L_{p_k}(B)$, our arrangement will take $O(m^2n^2)$ time to compute, whose dual we can navigate in $O(m^2n^2)$ time.[1]

For each line of the arrangement we take at most $O(m^2)$ time to perform the plane sweep of the first point. Then for each other point, there are an amortized $O(1)$ other lines intersecting at this point, and we do $O(1)$ work per intersection, so we do $O(mn)$ work per line. Therefore this step takes $O(m^2n^2)$ time.

Therefore our algorithm runs in $O(m^2n^2)$ time. Now we must determine whether this is good or not.

Since m is at most n, our algorithm will run in $O(n^4)$ time in the worst case. Consider the following barrier: Take a regular n-gon, and shrink all the edges by a small amount, so that there are gaps where the vertices were. Now there are small regions of space where lines can travel between each pair of vertices. These regions are equivalent to the planar embedding of K_n. This partitions the space into $\Theta(n^4)$ convex regions [3]. So to even *write* the output it would take $\Omega(n^4)$ time and space. Therefore, our algorithm is indeed worst-case optimal.

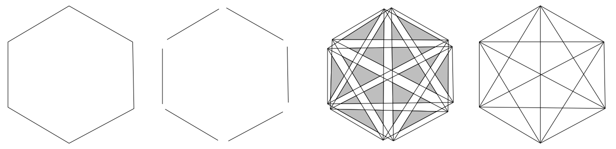


Figure 7: The worst known case barrier and its coverage

6 Deciding Whether a Point is Part of a Barrier's Coverage

Given a barrier B one can fairly simply determine whether a point p is in $R(B)$ in $O(n \log n)$ time and $O(n)$ space using a plane sweep. However if $R(B)$ is already constructed, point queries can be done in $O(\log k)$ time using a structure that takes $O(k^2)$ extra space and

$O(k^2 \log k)$ time to construct [4], where k is the number of edges in $R(B)$.

References

[1] M. de Berg. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, 2008.

[2] A. Dumitrescu and J. Pach. Opaque sets. *CoRR*, abs/1005.2218, 2010.

[3] J. W. Freeman. The number of regions determined by a convex polygon. *Mathematics Magazine*, 49(1):23–25, 1976.

[4] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

Open Problems from CCCG 2011

Erik D. Demaine[*] Joseph O'Rourke[†]

The following is a description of the problems presented on August 10, 2011 at the open-problem session of the *23rd Canadian Conference on Computational Geometry* held in Toronto, Ontario, Canada.

Blocking visibility with cylinders

Joseph O'Rourke
Smith College
orourke@cs.smith.edu

Suppose you have a supply of infinite-length, opaque, unit-radius cylinders, and you would like to block all visibility from a point $p \in \mathbb{R}^3$ to infinity with as few cylinders as possible. (The cylinders are infinite length in both directions.) The cylinders may touch but not interpenetrate, and they should be disjoint from p, leaving a small ball around p empty. (Another variation would insist that cylinders be pairwise disjoint, i.e., not touching one another.)

A collection of parallel cylinders arranged to form a "fence" around p do not suffice, leaving two line-of-sight $\pm$ rays to infinity. Perhaps a grid of cylinders in the pattern illustrated in Figure 1 (left) suffice, but at least if there are not many cylinders, there is a view from an interior point to infinity (Figure 1, right).

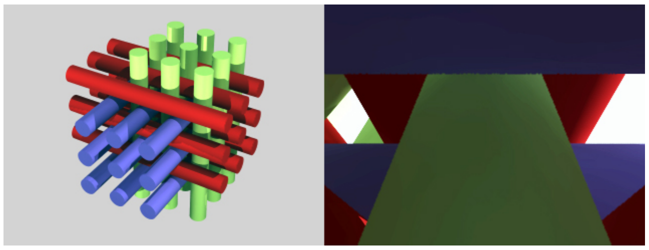


Figure 1: A grid of cross cylinders. A view from inside shows not all visiblity is blocked.

This question was originally posed on MathOverflow [OR11a], and several ideas contributed there suggest to start with the six cylinder arrangment in Figure 2 (left), supplemented by a circular "forest" to block the remaining lines of sight, three-quarters of which are illustrated in Figure 2 (right). The illustrated configuration needs 18 cylinders, but perhaps as few as 10 suffice for this plan?

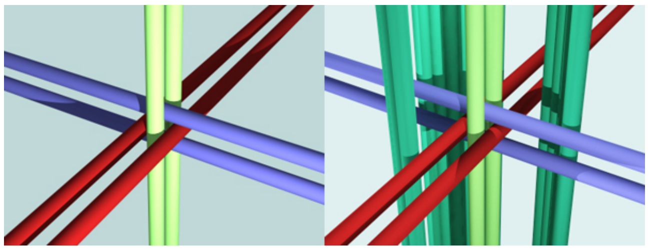


Figure 2: Six cylinders block all but some "diagonal" lines of sight. Erecting a vertical fence should then block all lines of sight.

What is the minimum number of infinite cylinders that can block visibility from a point?

References

[OR11a] J. O'Rourke. Blocking visibility with cylinders. http://mathoverflow.net/questions/69963/ 11 May 2011.

The Rain Hull and the Rain Ridge

Joseph O'Rourke
Smith College
orourke@cs.smith.edu

Rain falls steadily on an island, a 2-manifold M, which you may assume, as you prefer, is: (a) smooth, or (b) a PL-manifold, or perhaps (c) a triangulated irregular network (TIN). After a time, M is saturated, in the sense that every raindrop drains into the ocean rather than filling yet-unfilled crevices or basins. At this point, we have what I will dub the *rain hull* of M, $H_R(M)$, a unidirectional version of the the reflex-free hull defined by Jack Snoeyink at the 13th CCCG [ACCS04]

(1) How difficult is to compute the rain hull $H_R(M)$?

This question was originally posed on MathOverflow [OR11b] and a respondent there (Joel Hamkins) argued that at least it can be computed in polynomial time. Nonlocal effects such as that illustrated in Figure 3 must be accommodated.

[*]MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar Street, Cambridge, MA 02139, USA, edemaine@mit.edu
[†]Department of Computer Science, Smith College, Northampton, MA 01063, USA. orourke@cs.smith.edu

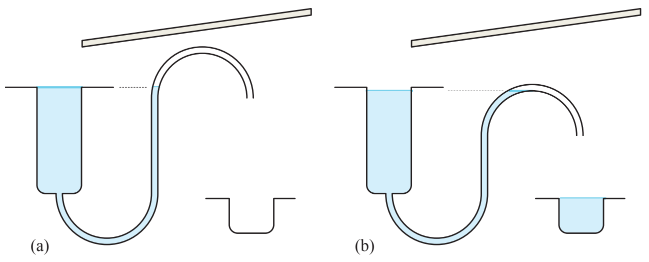


Figure 3: (a) The connecting tube rises too high to fill the other, protected basin. (b) A lower tube does overflow into the other catch-basin.

Let us assume we have $\overline{M} = H_R(M)$ computed or given. A raindrop falling on $p \in \overline{M}$ might follow a unique *trickle path* (that is the technical term: e.g., see [dBHT11]) to the ocean, or the drop may randomly 'fracture' to follow distinct paths to the ocean. Define the *rain ridge* (my terminology) $R(\overline{M})$ to be the complement of the points of $\overline{M}$ that have a unique trickle path.

So points on the rain ridge are akin to points on a *cut locus*, in that they have two or more distinct paths to $\partial\overline{M}$. They are, in a sense, *continental-divide* points [Hay09].

(2) What can be said about the structure of the rain ridge $R(\overline{M})$? And how quickly can it be computed?

Unlike the cut locus or "ridge tree," the rain ridge is not always a tree. All the points in a filled basin are in the rain ridge, for when a raindrop lands in a filled basin, it is natural to assume it "spreads out" and spills in equal portions over every boundary point of the basin. But surely there are substantive properties to investigate. Surely the rain ridge $R(\overline{M})$ cannot be an arbitrary subset of $\overline{M}$?

(3) Can an extended metric be assigned to $\overline{M}$ so that its geodesics are its trickle paths?

An *extended metric* is one that permits $d(x, y) = \infty$ (e.g., for points not on the same trickle path). What I am hoping for here is a way to view the rain ridge as a cut locus of $\partial\overline{M}$, and then apply a century of knowledge on the cut locus to the rain ridge.

References

[ACCS04] H.K. Ahn, S.W. Cheng, O. Cheong, and J. Snoeyink. The reflex-free hull. *International Journal of Computational Geometry and Applications*, 14(6):453–474, 2004.

[dBHT11] M. de Berg, H. Haverkort, and C. Tsirogiannis. Implicit flow routing on terrains with applications to surface networks and drainage structures. In *Proc. 22nd ACMSIAM Symp. on Discrete Algorithms (SODA)*, pages 285–296, 2011.

[Hay09] B. Hayes. Dividing the Continent. In *Group Theory in the Bedroom, and Other Mathematical Diversions.* Hill and Wang, 2009. pp. 107–123.

[OR11b] J. O'Rourke. The rain hull and the rain ridge. http://mathoverflow.net/questions/69963/ 10 July 2011.

Long Alternating Paths

Jorge Urrutia
Universidad Nacional Autónoma de México
urrutia@matem.unam.mx

Let P_{kn} be a point set with kn points in general position. A k-coloring of P_{kn} is a partitioning of P_{kn} into k disjoint subsets $S_1, \ldots, S_k$, each with n elements. The sets $S_1, \ldots, S_k$, are called the chromatic classes of P_{kn}.

An *alternating* path Π of P_{kn} is a simple polygonal path connecting a subset of the points of P_{kn} such that there are no monochromatic edges in the path.

Conjecture Any 3-colored point set P_{3n} contains an alternating path with at least $2n$ elements.

We have been unable to prove that P_{3n} always contains an alternating path with $\frac{3}{2}n$ points; this seems to be a challenging weaker open problem. For 3-colored point sets P_{3n} in convex position, it is known there always exists a path that covers $2n$ points, and that this bound is tight [MSU]. Tight bounds for 2-colored point sets are not known for point sets in convex, or in general position [AGHNP].

References

[AGHNP] M. Abellanas, J. Garcia, G. Hernandez, M. Noy, and P. Ramos. Bipartite embeddings of trees in the plane. Discrete Appl. Math. 93 (1999), 141148.

[AU] J. Akiyama, and J. Urrutia. Simple alternating path problem. *Discrete Math*, (1990) 84: 101-103.

[MSU] C. Merino, G. Salazar and J. Urrutia. On the length of the longest alternating path for multicoloured point sets in convex position. *Discrete Mathematics*, Vol. 360, no. 15, pp. 1791-1797, 2006.

[PKT] J. Pach, J. Kynčl, and G. Tóth. Long alternating paths in bicolored point sets. *Discrete Mathematics*, 308 (2008), 4315–4322.

Monochromatic Empty Triangles

Jorge Urrutia
Universidad Nacional Autónoma de México
urrutia@matem.unam.mx

Let P_n be a set of n points in general position on the plane, each of which is colored red or blue. A triangle with vertices in P is called *empty* if it contains no point of P in its interior, it is called *monochromatic* if all of its vertices are red, or all are blue.

Conjecture Any bicolored point set P_n contains $\Omega(n^2)$ monochromatic empty triangles.

A liner bound was established in [DHKS]. It was improved to $cn^{\frac{5}{4}}$ in [AFHU], and to $cn^{\frac{4}{3}}$ in [PT].

References

[AFHU] O. Aichholzer, R. Fabila-Monroy, D. Flores-Peñaloza, T. Hackl, C. Huemer, and J. Urrutia, Monochromatic empty triangles. *Computational Geometry,* Vol. 42, Issue 9, November 2009, Pages 934–938.

[DHKS] O. Devillers, F. Hurtado, Gy. Károlyi, and C. Seara, Chromatic variants of the Erdõs-Szekeres theorem on points in convex position, *Computational Geometry,* Vol. 26, Issue 3, 2003, pp. 193–208.

[PT] J. Pach, and G. Tóth. Monochromatic empty triangles in two-colored point sets. Geometry, Games, Graphs and Education: the Joe Malkevitch Festschrift (S. Garfunkel, R. Nath, eds.), COMAP, Bedford, MA, 2008, 195-198. Also in: Discrete Applied Mathematics, submitted.

Shortest Periodic Light Ray

Boaz Ben-Moshe
Ariel University Center of Samaria
benmo@g.ariel.ac.il

Given a simple polygon, find the shortest periodic path of a light ray reflecting from the polygon edges as perfect mirrors. This problem is solved for *rational triangles*, those whose angles are rational multiples of π, but seems to be open for arbitrary triangles.

The Geometry of Golf

Alejandro López-Ortiz
University of Waterloo
alopez-o@uwaterloo.ca

After repeated unsuccessful attempts to get the ball in the hole from a particular point in the green, a golfer walks away in frustration and declares: That shot is impossible!

A mathematician happens to be standing nearby and says outloud: Hmmm, is it true that one can always putt a golf ball into the hole on this or any other arbitrary green?

A computer scientist overhears the mathematician and thinks: for given a green and ball location can I use my smartphone to determine if the shot is possible and if so in what direction and speed should I hit the ball?

More formally, the mathematician's question becomes: does every smooth two-dimensional manifold under a gravitational potential field is "connected" in the sense that a point particle at an arbitrary point on it can be made to roll into any other point on the manifold given a proper nudge (initial velocity vector) in the right direction.

The answer for this question is NO. A simple counterexample folds the surface into caves, but this is not strictly necessary: there are C^∞ manifolds which are described by the plot of a function $f : R \times R \to R$ and yet do not allow rolling motions into the hole. To see this consider a green with a mountain ridge between the ball and the hole, as depicted in Figure 4. If the slope on the hole side of

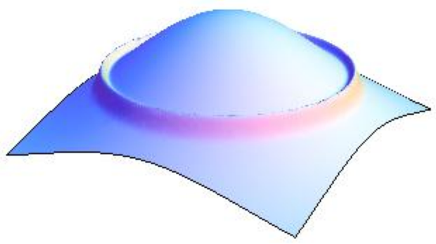


Figure 4: The hole is at the top center. This particular example is due to Jaap Eldering at http://mathoverflow.net/questions/84033/.

the ridge is sufficiently steep the ball can be made to become airborne and overfly the hole; see Figure 5.

The computer scientist's questions, posed formally, become: first, given a description of a green (perhaps discretized as a TIN) give an efficient algorithm that determines if the hole can always be

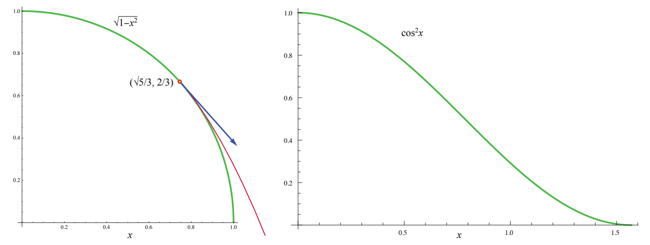


Figure 5: (a) Not gently falling. (b) Gently falling. Figure from [OR11c].

reached from all points, and second, given a ball position in said green can we compute the direction and speed of the putting action that will roll the ball into the hole?

Several variations of the question

Putting. Under what conditions can a given ball on a C^∞ manifold, with a quadratic gravitational field, reach the hole?

Golf green design. Under what conditions can every putt on a C^∞ manifold, with a quadratic gravitational field, reach the hole?

Hole location design. Given a C^∞ manifold, which points on it are reachable from all others and hence would be reasonable choices for location of the hole?

Chipping. What if you can chip, i.e., loft the ball?

Driving. Under what conditions is it possible to achieve a "hole-in-one" from the driving tee if we consider obstacles such as trees?

Sand Save. Under what shape conditions can you chip out of a sand trap and always move closer to the hole.

To understand the physics of the problem we study the 2D setting of a ball rolling down a curve.

First consider the instantaneous version of the problem: given a ball on the curve and moving at a certain speed will it become airborne at this instant?

We consider first the case where the particle was at rest. In this setting two forces are acting on the ball, namely gravity and surface resistance. Gravity is a vertical vector pointing downwards with magnitude $9.8m/s^2$. The surface resistance is a vector perpendicular to the tangent to the curve.

Observe that the magnitude of the resistance vector is exactly equal to to the projection of the gravitational vector on the normal direction to the tangent to the curve at all times. This is easier to see when the particle is at rest: if the forces on the direction of the surface were not perfectly canceled with that of gravity then the particle would either burrow into the surface until equilibrium is achieved (think of a really heavy ball making a dent) or would magically start hovering over it, both of which do not happen with a rolling golf ball.

Hence the only movement possible for a particle at rest is in the direction of the tangent to the surface and the surface resistance must perfectly cancel any force in any other direction.

The ball will then move along the direction of the tangent at a speed which is given by the addition of the resistance vector to that of the force of gravity. Let $\Gamma(t) = (x(t), y(t))$ denote the trajectory of the particle parameterized by the time t. Then the speed vector $v(t)$ is given by $d\Gamma/dt$, and the instantaneous change in speed is given by the differential equation $v'(t) = ||\Gamma'(t)||^{-1}(\Gamma'(t)\cdot(0,-g))+(0,-g)$.

The particle becomes airborne if the speed vector ever lies above the tangent to the curve which would result in a ski-like take off along the direction of the speed vector.

If the particle is already in motion then the same equations apply and the only change is in the initial condition $v(t)$ which for the particle at rest case was $v(t_0) = \vec{0}$ and now becomes $v(t_0) = (v_x(t_0), v_y(t_0))$.

We can test for the airborne state if we recall that the cross product of two vectors $\vec{a} = (a_1, a_2), = \vec{b} = (b_1, b_2)$ in the plane is the vector $(0, 0, a_1b_2 - a_2b_1)$, where the last coordinate is positive if and only if $\vec{a}$ is below $\vec{b}$. Substituting the speed vector and the tangent vector above we get that the ball remains on the surface iff

$$v_x(t)a_y(t) - v_y(t)a_x(t) \leq 0$$

We can now use this equation together with an iterative differential equation solver to numerically test this property along the entire trajectory of a putting path.

Update. In [OR11c] differential equations are derived for what can be considered a one-dimensional version of the putting problem.

References

[OR11c] J. O'Rourke. Gently falling functions. MathOverflow. http://mathoverflow.net/questions/68114/ 18 June 2011.

Polygon Triangulation Without Large Angles
Alexander Rand
University of Texas at Austin
arand@ices.utexas.edu

Let P be a generic convex polygon with vertices $V_1, V_2, \ldots, V_n$ (and define $V_0 := V_n$ and $V_{n+1} := V_1$ for simplicity). For $\gamma < \pi$, we will say that P belongs to the set S_γ if for any $i \notin \{j, j+1\}$ then $\angle V_j V_i V_{j+1} < \gamma$, i.e., no vertex forms a large angle with any opposite side of the polygon. See Figure 6.

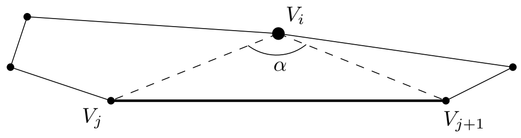


Figure 6: If $\angle V_j V_i V_{j+1}$ (denoted by α in the figure) is large, then no triangulation exists without a large angle. If this angle is bounded for all pairs of vertices and opposite edges, we expect some acceptable triangulation can be formed.

Open Problem For $\gamma < \pi$, give an algorithm that, for any convex polygon in S_γ, adds some vertices to the *interior* of the polygon and produces a triangulation with no angles larger than $\theta(\gamma) < \pi$.

- Most related problems/algorithms in the literature (e.g., [BMR95, MPS07]) involve inserting vertices on the boundary of the polygon, which we have disallowed.
- The restriction to the set S_γ is essential: an obtuse triangle with largest angle very near π cannot be triangulated satisfying our requirements.
- The specific relationship between θ and γ can be selected at the discretion of the solver. The best solution hopefully has a form $\pi - \theta(\gamma) = \Omega(\pi - \gamma)$.
- The number of points added by the algorithm is unimportant for the original motivation of the problem, but it makes sense to ask what is the fewest number of vertices which can be added (which examples suggest is small; see conjecture below). Simple approaches often involve $O(n)$ vertices. Adding a very large number of vertices is not particularly helpful as any vertex placed too close to a boundary edge produces a large angle in the triangulation.
- At least one vertex can be necessary. Any triangulation of the regular n-gon without any additional vertices will produce at least one triangle of three consecutive vertices and, for large n, this has a large angle. Adding a single vertex at the center of the polygon gives an acute triangulation. See Figure 7.
- Moreover, it appears that at least two vertices must be inserted in some cases. See Figure 8.

Conjecture There exists an algorithm and some function $\theta(\gamma)$ which solve the problem above using at most two additional vertices.

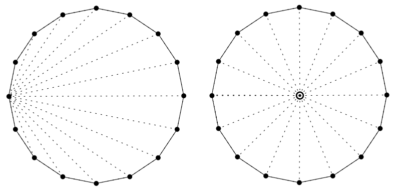


Figure 7: For a regular n-gon with n large, any triangulation without additional vertices includes a triangle of three consecutive vertices and thus a large angle. Adding a single vertex in the center yields an acute triangulation.

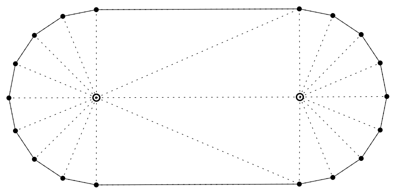


Figure 8: An example which requires two additional vertices (or at least appears to). This can be extended to any extreme γ or θ thresholds by adding more vertices to the semi-circles and making the full example wider in the horizontal direction.

Update. This problem is a special case of one less formally stated in the conclusion of [BDE92]. Specifically, Bern et al. ask for an algorithm and class of polygons yielding quality triangulations (i.e. without large angles) using only interior Steiner points.

References

[BDE92] M. Bern, D. Dobkin, and D. Eppstein. Triangulating polygons without large angles. *Proceedings of the 8th Annual Symposium on Computational Geometry*, pages 222–231, 1992.

[BMR95] M. Bern, S. Michell, and J. Ruppert. Linear-size nonobtuse triangulation of polygons. *Discrete Computational Geometry*, 14(1):411–428, 1995.

[MPS07] G. Miller, T. Phillips, and D. Sheehy. Size competitive meshing without large angles. In *34th International Colloquium on Automata, Languages and Programming*, pages 655–666, 2007.

Disk Constrained 1-Center Queries

Luis Barba *

Abstract

We show that a set P of n points in the plane can be preprocessed in $O(n \log n)$-time to construct a data structure supporting $O(\log n)$-time queries of the following form: Find the minimum enclosing circle of P with center on a given disk.

1 Introduction

Given a set P of n points in the plane, the minimum enclosing circle problem, originally posed by Sylvester in 1857 [14], asks to identify a point c_P in the plane such that the maximum Euclidean distance from the points of P to c_P is minimized. Therefore, this problem can be thought as that of finding the center of the minimum enclosing circle of P. For ease of notation we say that every circle containing P is a P-circle. An $O(n^2)$-time algorithm was presented by Elzinga and Hearn [5] to find the minimum P-circle. Later, Preparata in [11], and Shamos and Hoey in [13], independently proposed two algorithms to solve this problem in $O(n \log n)$-time. Lee presented the farthest-point Voronoi diagram, which can be also used to solve this problem in $O(n \log n)$-time [9]. Finally, Megiddo proposed an optimal $O(n)$-time algorithm to find the center of the minimum P-circle using a prune and search approach [10]. Furthermore, the problem of finding the minimum enclosing d-sphere that contains a given set of n points in $\mathbb{R}^d$ can be solved in $O(n)$-time for any fixed d [3][4].

Several constrained versions of the minimum P-circle problem have been studied lately. Hurtado, Sacristan and Toussaint presented an optimal $O(n+m)$-time algorithm to find the minimum P-circle whose center is constrained to satisfy m linear inequalities [6]. Bose and Toussaint considered the generalized version of this problem by restricting the center of the P-circle to lie inside a simple polygon of size m. They proposed an $O((n+m)\log(n+m)+k)$-time algorithm to solve this problem, where k is the number of intersections of Q with the farthest-point Voronoi diagram of P [2]. Megiddo studied the problem of finding the minimum P-circle with center on a given straight line and proposed an $O(n)$-time algorithm to solve this problem [10]. He also posed the on-line version of this problem in which a preprocessing of the point set P is allowed and the objective is to answer the following query: Given a straight line ℓ, find the minimum P-circle with center on ℓ. Das, Karmakar, Nandy and Roy first addressed this problem and proposed an $O(n \log n)$-time preprocessing on P, which allows them to answer these queries in $O(\log^2 n)$-time [12]. They improved the query running time to $O(\log n)$ using $O(n^2)$ preprocessing time and space [7]. Finally, Bose, Langerman and Roy showed an $O(n \log n)$-time preprocessing to construct a linear space data structure that answers queries in $O(\log n)$-time [1].

In this paper, we address a generalized version of this problem in which the center of the minimum P-circle is constrained to lie on a query disk. This problem has a direct application in wireless communication: think of a set of locations that need to receive a certain message (represented by P) and think of a main moving antenna that is broadcasting a message within a certain range (represented by a query disk Q). Our objective is then to determine the location for a re-transmitter C, inside the range of Q, such that every location in P receives the message from C at the lowest cost.

We propose an $O(n \log n)$-time preprocessing on the point set P, to construct a linear space data structure that answers both disk and line queries, in $O(\log n)$-time.

2 Preliminaries

In this paper, the words *disk* and *circle* refer to the same geometric object. The former refers to the query objects while the latter to the solutions of the query. Given $S \subset \mathbb{R}^2$, ∂S denotes its boundary while $int(S)$ denotes its interior.

Let P be a set of n points in the plane. A circle containing P is called a P-circle. Given a disk Q with center on q, let p_Q be a point in P such that q lies in the farthest-point Voronoi region associated with p_Q. The farthest-point Voronoi diagram of P can be seen as a tree with n unbounded edges and is denoted in this paper by $\mathcal{V}(P)$. For any point p of P, let $R(p)$ be the farthest-point Voronoi region associated with p. Let C_P be the minimum enclosing circle of P and let c_P be its center. If c_P is not a vertex of $\mathcal{V}(P)$, we insert it into $\mathcal{V}(P)$ by splitting the edge where it belongs. We consider $\mathcal{V}(P)$ as a rooted tree at c_P. Given a point x on $\mathcal{V}(P)$, π_x denotes the unique path joining c_P with x

*Département d'Informatique, Université Libre de Bruxelles, Brussels, Belgium, lbarbafl@ulb.ac.be

contained in $\mathcal{V}(P)$. Given a set $S \subseteq \mathbb{R}^2$, let $C(S)$ be the minimum P-circle with center on S and let $c(S)$ be its center. We say that $c(S)$ is the query center of S. Whenever $S = \{x\}$, $C(x)$ denotes the minimum P-circle with center on x and we let $\rho(x)$ be its radius. If x belongs to $R(p)$ for some vertex p of P, then $C(x)$ passes through p and hence, $\rho(x) = d(p, x)$, where $d(*, *)$ represents the Euclidean distance between two points in the plane. Given a disk Q with center on q and a point x outside Q, the projection of x on Q, denoted by $\sigma(x, Q)$, is the intersection of the segment $[x, q]$ with the circumcircle of Q.

3 The minimum P-circle with center on Q

Proposition 1 *Let w be a point on an edge of $\mathcal{V}(P)$. The function ρ is monotonically increasing along the path π_w starting at c_P.*

Proof. In [12] (Result 2), it is shown that if x is an ancestor of y on the rooted tree $\mathcal{V}(P)$, then $\rho(x) < \rho(y)$. Since $\rho(x)$ is a convex function [1] (Lemma 3), it is also convex when restricted to any segment of π_w. □

Let Q be a disk with center on q. The following results characterize the position of $c(Q)$ with respect to $\mathcal{V}(P)$.

Proposition 2 *The point $c(Q)$ lies on ∂Q if and only if $c_P \notin int(Q)$.*

Proof. $\rightarrow$) If $c_P \in int(Q)$, then $C(Q) = C_P$ and hence $c(Q) = c_P$ lies in $int(Q)$.

$\leftarrow$) Assume that $c(Q)$ lies in the interior of Q but c_P does not. Let p be a point of P such that $c(Q)$ belongs to $R(p)$. Two cases arise:

If $c(Q) \in int(R(p))$, then there is a point x in the vicinity of $c(Q)$, slightly closer to p, such that x belongs to $Q \cap R(p)$. Therefore, $\rho(x) < \rho(c(Q))$ which is a contradiction. Otherwise, if $c(Q)$ lies on an edge of $\mathcal{V}(P)$, we can consider a point x slightly closer to c_P along the path $\pi_{c(Q)}$, such that x still belongs to Q. Thus, by Proposition 1 $\rho(x) < \rho(c(Q))$ which is also a contradiction. Therefore, if c_P does not belong to the interior of Q, then $c(Q)$ lies on the boundary of Q. □

From now on we assume that c_P is not contained in Q. Otherwise, c_P is trivially the query center of Q.

Lemma 3 *Given a disk Q with center on q. If p_Q is a point of P such that $q \in R(p_Q)$, then:*

1. *The circumcircle of $C(Q)$ passes through exactly one point p of P, if and only if $p = p_Q$ and $\sigma(p_Q, Q) \in int(R(p_Q))$. In this case, $c(Q) = \sigma(p_Q, Q)$.*
2. *The circumcircle of $C(Q)$ passes through at least two points of P, if and only if $c(Q)$ lies on an edge of $\mathcal{V}(P)$.*

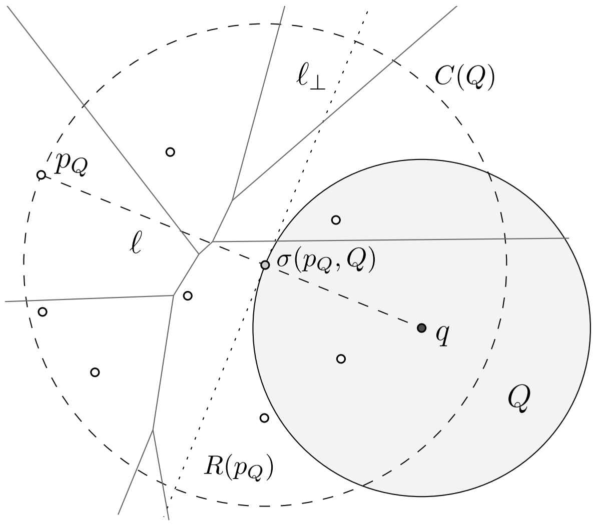


Figure 1: Case 1 of Lemma 3 where the projection of p_Q on Q lies inside $R(p_Q)$ and determines the center of $C(Q)$.

Proof. 1 $\rightarrow$) If $C(Q)$ passes through only one point p of P, then $c(Q) \in int(R(p))$. Let ℓ be the line joining p with $c(Q)$ and let $\ell_\perp$ be the perpendicular line to ℓ that passes through $c(Q)$; see Figure 1. Note that $\ell_\perp$ must leave all points of Q on the halfplane defined by $\ell_\perp$ that is farther away from p. Otherwise, we can choose a point x inside $Q \cap R(p)$ such that x is closer to p than $c(Q)$—a contradiction since $C(x)$ would be a P-circle with smaller radius than $C(Q)$. Since $\ell_\perp$ leaves all points of Q in one halfplane, $\ell_\perp$ is tangent to Q and hence $c(Q) = \sigma(p, Q)$. Moreover, the points $q, c(Q)$ and p are collinear. Thus, the circle with center on q and passing through p is also a P-circle, which means that $q \in R(p)$, i.e. $p = p_Q$.

1 $\leftarrow$) Let C be the circle with center on $\sigma(p_Q, Q)$ and radius $d(\sigma(p_Q, Q), p_Q)$ Since $\sigma(p_Q, Q)$ is the closest point of Q to p_Q, C is the smallest circle containing p_Q with center on Q. Moreover, since $\sigma(p_Q, Q) \in int(R(p_Q))$, C is a P-circle passing only through p_Q and any other P-circle with center on Q must contain p_Q. Thus, $C(Q) = C$ and it passes only through one point of P.

2) Follows from the definition of the farthest-point Voronoi diagram; see Figure 2. □

If case 1 of Lemma 3 holds we are done since $C(Q)$ will be the circle with center on $\sigma(p_Q, Q)$ and radius $d(\sigma(p_Q, Q), p_Q)$. Therefore, we assume from now on that $c(Q)$ is a point lying on an edge of $\mathcal{V}(P)$.

4 Sketch of the algorithm

The idea behind the algorithm that we will present is to shrink the disk Q, obtaining in this way a new disk

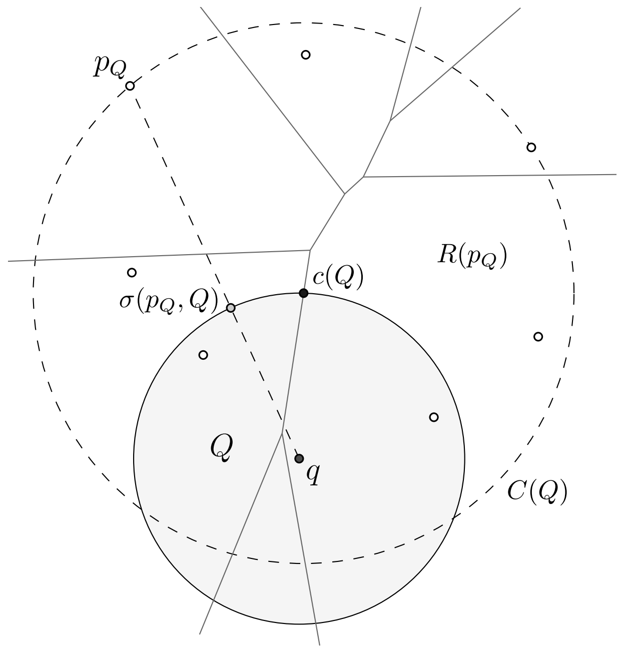


Figure 2: Case 2 of Lemma 3 where $c(Q)$ lies on an edge of $\mathcal{V}(P)$ and $C(Q)$ passes through two points of P. Moreover, the projection of p_Q on Q lies outside of $R(p_Q)$.

Q' with the same center, such that case 1 of Lemma 3 holds for Q'. Thus, $c(Q')$ can be efficiently computed and we can scale Q' back to its original size, tracking the position of $c(Q')$ during this scaling.

Let ℓ be the line joining q with p_Q and let ω be the intersection of ℓ with $\partial R(p_Q)$. It is well known that this intersection is unique. Let Q' be the circle with center on q and radius $d(q, \omega)$. Note that Q' can be seen as the disk Q scaled down such that ω is the projection of p_Q on Q' and ω lies in $R(p_Q)$. Thus, by Lemma 3, $C(Q')$ is the circle with center on ω and radius $d(\omega, p_Q)$; see Figure 3.

The idea is now to scale back Q' to Q, without losing the position of the query center of Q' along the process. In order to do that, we construct a family of disks representing this scaling as follows. Assume that r and r' are the radius of Q and Q', respectively, and let $Q(t)$ be the disk with center on q and radius $r' + t(r - r')$, $t \in [0, 1]$. Note that $Q(t)$ represents a continuous scaling starting with $Q(0) = Q'$ and ending with $Q(1) = Q$. Let $\gamma(t)$ be the curve described by query center of $Q(t)$, $t \in [0, 1]$.

Lemma 4 *The curve $\gamma(t)$ is a continuous curve such that $\gamma(0) = \omega$, $\gamma(1) = c(Q)$ and $\gamma(t)$ lies on π_ω for every $0 \le t \le 1$.*

Proof. The curve $\gamma(t)$ is continuous since ρ is a continuous function. Thus, it only remains to prove that $\gamma(t)$ is contained in $\mathcal{V}(P)$.

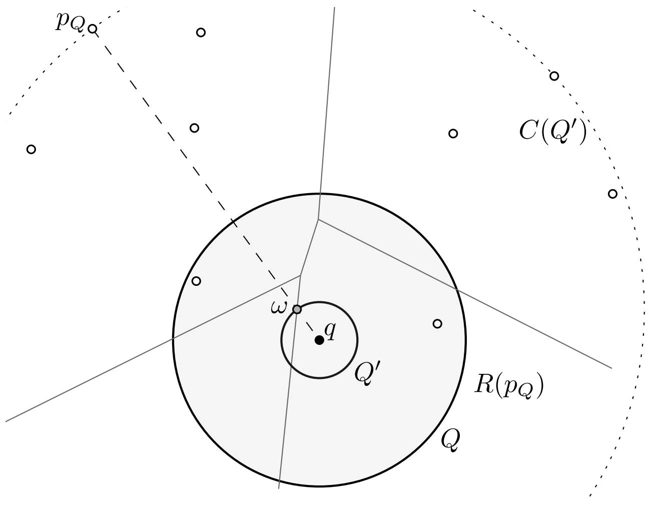


Figure 3: The disk Q' as the reduction of Q. In this case, $c(Q') = \omega$ is the projection of p_Q on Q'.

Since every $Q(t)$ is centered on q, $p_Q = p_{Q(t)}$ for every $0 \le t \le 1$. Furthermore, for every $0 < t \le 1$, the projection of p_Q on $Q(t)$ lies outside $R(p_Q)$; see Figure 3. Therefore, Lemma 3 implies that every $Q(t)$ has its query center lying on an edge of $\mathcal{V}(P)$. In other words, the curve $\gamma(t)$ is completely contained in $\mathcal{V}(P)$.

Since we assumed that c_P lies outside Q and since $Q(t) \subseteq Q(t')$ for every $0 \le t < t' \le 1$, the value of $\rho(\gamma(t))$ decreases monotonically as t increases. Thus, because $\gamma(t)$ is contained in $\mathcal{V}(P)$, Proposition 1 implies that $\gamma(t)$ is contained on the path joining c_P with ω. $\square$

Our objective will be to find $c(Q)$ along the path π_ω using a binary search. However, the boundary of a disk may intersect a path on $\mathcal{V}(P)$ more than once. Thus, we need the following result.

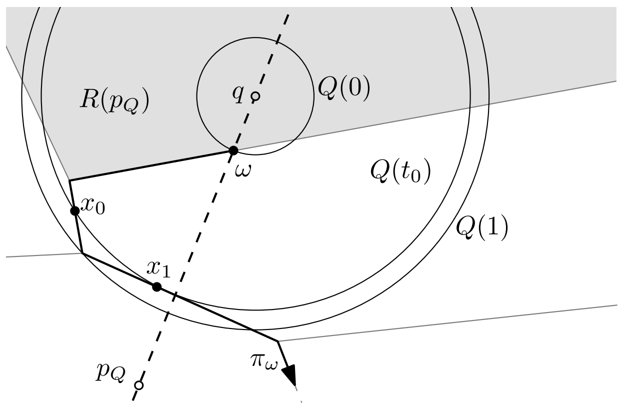


Figure 4: The point x_0 is an accumulation point of $\gamma(t)$ while x_1 represents a discontinuity of $\gamma(t)$.

Lemma 5 *There is a unique intersection point between the path π_ω and the boundary of Q.*

Proof. Proceed by contradiction and assume that the boundary of $Q = Q(1)$ intersects π_ω in at least two points. Recall that $Q' = Q(0)$ intersects π_ω at a unique point $\omega = \sigma(p_Q, Q')$. Let t_0 be the minimum value in $[0, 1]$ such that $\partial Q(t_0)$ intersects π_ω in more than 2 points. Let $x_0, \ldots, x_k$ be the points of intersection between $Q(t_0)$ and π_ω, and assume that they are sorted in decreasing order with respect to their depth on the tree $\mathcal{V}(P)$. Note that, for every $0 \leq t < t_0$, $Q(t)$ intersects π_ω in exactly one point and, by Lemma 4, this intersection defines the position of $\gamma(t)$. Therefore, x_0 is an accumulation point of the curve $\gamma(t)$; see Figure 4. However, Proposition 1 implies that $\rho(x_0) > \ldots > \rho(x_k)$ and hence $\gamma(t_0)$ must be equal to x_k. This represents a discontinuity of the curve $\gamma(t)$ and hence a contradiction. □

Using both lemmas presented in this section, we obtain the following result.

Corollary 6 *The point $c(Q)$ is the unique intersection point between π_ω and ∂Q.*

5 The algorithm

Recall that our objective is to design a data structure on P to answer the following query: Given any disk Q, find the minimum P-circle with center on Q.

In the previous section we presented the relation existing between the query center of Q and $\mathcal{V}(P)$. In this section, we use that relation to construct a data structure on $\mathcal{V}(P)$, that allow us to perform a binary search for $c(Q)$ along the paths contained in $\mathcal{V}(P)$.

5.1 Preprocessing

Compute $\mathcal{V}(P)$ and c_P in $O(n \log n)$-time [13]. Assume that $\mathcal{V}(P)$ is stored as a binary tree with n (unbounded) leaves, so that every edge and every vertex of the tree has a set of pointers to the vertices of P defining it. Every vertex p of P has a pointer to $R(p)$ which is stored as a convex polygon. Construct a point location data structure on top of the farthest-point Voronoi diagram in $O(n \log n)$-time [8] so that we can answer furthest-point queries in $O(\log n)$-time. If c_P is not a vertex of $\mathcal{V}(P)$, we insert it to $\mathcal{V}(P)$ by splitting the edge that it belongs to.

We will use an operation on the vertices of $\mathcal{V}(P)$ called PointBetween with the following properties. Given two vertices u, v in π_ω, PointBetween(u, v) returns a vertex z that splits the path on π_ω joining u and v into two subpaths. Moreover, if we discard the subpath that does not contain $c(Q)$ and we proceed recursively on the other, then, after $O(\log n)$ iterations, the search interval becomes only an edge of π_ω containing $c(Q)$.

A data structure that supports this operation was presented in [12]. This data structure can be constructed on top of $\mathcal{V}(P)$ in $O(n)$ time and uses linear space.

5.2 The search for $c(Q)$

Given a query disk Q with center on q and radius r, we present an algorithm to determine the position of $c(Q)$ in $O(\log n)$-time using the data structure described in the previous section. Let p_Q be a point of P such that q belongs to $R(p_Q)$. To find p_Q, an $O(\log n)$-time point-location query on the farthest-point Voronoi diagram suffices.

Let ℓ be the line joining q with p_Q and let ω be the intersection of the boundary of $R(p_Q)$ with ℓ. Since $R(p_Q)$ is a convex polygon, this intersection can be computed in $O(\log n)$-time. Let Q' be the disk with center on q and radius $d(q, \omega)$. By Corollary 6, $c(Q)$ is the unique intersection between π_ω and ∂Q. Thus, we search on π_ω for $c(Q)$ as follows:

The procedure PointBetween(w, c_P) provides a point z that splits π_ω into two subpaths. Let π (resp. π') be the subpath joining z with c_P (resp. ω with z) contained in π_ω. If $z \in Q$ (resp. $z \notin Q$), then $c(Q)$ lies on π (resp. π'). Thus, we can discard either π or π' and continue the search on the subpath containing $c(Q)$. We proceed until finding two consecutive vertices on π_ω, such that the first one lies inside Q but the second one does not. The details can be found in Algorithm 1.

Algorithm 1 Algorithm to find $c(Q)$ given the path $\pi_\omega = (\omega = u_0, \ldots, u_r = c_P)$

```
1: Define the initial search interval:
   $u \leftarrow u_0, v \leftarrow u_r$.
2: if $uv$ is an edge of $\pi_\omega$ then
3:     Finish and report the segment $s = [u, v]$.
4: end if
5: $z \leftarrow$ PointBetween$(u, v)$.
6: if $z \in Q$ then
7:     Move forward, let $u \leftarrow z$ and return to step 2.
8: else
9:     Move backwards, let $v \leftarrow z$ and return to step 2.
10: end if
```

When our algorithm finishes, it reports an edge $s = [u, v]$ of the path π_ω, such that u is contained in Q but v is not. By Corollary 6, we conclude that $c(Q)$ is the intersection point between s and ∂Q. Since the number of steps on this binary search is $O(\log n)$ [12] and since each step requires a constant number of operations, the overall running time of the algorithm is $O(\log n)$.

Theorem 7 *After preprocessing a set P of n points in $O(n \log n)$-time, the minimum P-circle with center on a query disk Q can be found in $O(\log n)$-time.*

References

[1] P. Bose, S. Langerman, and S. Roy. Smallest enclosing circle centered on a query line segment. In *CCCG'08*, pages 167–170, 2008.

[2] P. Bose and G. Toussaint. Computing the constrained euclidean, geodesic and link centre of a simple polygon with applications. In *Computer Graphics International, 1996. Proceedings*, pages 102 –111, jun 1996.

[3] B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. In *Proceedings of SODA*, pages 281–290, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[4] M. E. Dyer. On a multidimensional search technique and its application to the Euclidean one centre problem. *SIAM J. Comput.*, 15:725–738, August 1986.

[5] J. Elzinga and D. W. Hearn. Geometrical solutions for some minimax location problems. *Transportation Science*, 6(4):379–394, 1972.

[6] F. Hurtado, V. Sacristan, and G. Toussaint. Some constrained minimax and maximim location problems. *Studies in Locational Analysis*, 15:17–35, 2000.

[7] A. Karmakar, S. Roy, and S. Das. Fast computation of smallest enclosing circle with center on a query line segment. *Information Processing Letters*, 108(6):343 – 346, 2008.

[8] D. G. Kirkpatrick. Optimal search in planar subdivisions. Technical report, University of British Columbia, Vancouver, BC, Canada, Canada, 1981.

[9] D. Lee. Farthest neighbor Voronoi diagrams and applications. *Report 80-11-FC-04, Dept. Elect. Engrg. Comput. Sci.*, 1980.

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

[11] F. Preparata. Minimum spanning circle. *Steps in Computational Geometry*, pages 3–5, 1977.

[12] S. Roy, A. Karmakar, S. Das, and S. C. Nandy. Constrained minimum enclosing circle with center on a query line segment. *Computational Geometry*, 42(6-7):632 – 638, 2009.

[13] M. Shamos and D. Hoey. Closest-point problems. In *Proceedings of FOCS*, pages 151–162, Washington, DC, USA, 1975. IEEE Computer Society.

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

Circle Separability Queries in Logarithmic Time

Greg Aloupis[*] [†] Luis Barba[*] Stefan Langerman[*][‡]

Abstract

In this paper we preprocess a set P of n points so that we can answer queries of the following form: Given a convex m-gon Q, report the minimum circle containing P and excluding Q. Our data structure can be constructed in $O(n \log n)$ time using $O(n)$ space, and answers queries in $O(\log n + \log m)$ time.

1 Introduction

The planar separability problem consists of constructing, if possible, a boundary that separates the plane into two components such that two given sets of geometric objects become isolated. Typically this boundary is a single curve such as a line, circle or simple polygon, meaning that each component of the plane is connected. Probably the most classic instance of this problem is to separate two given point sets with a circle (or a line, which is equivalent to an infinitely large circle). A separating line can be found, if it exists, using linear programing. This takes linear time by Megiddo's algorithm [9]. For circle separability (in fact spherical separability in any fixed dimension), O'Rourke, Kosaraju and Megiddo [10] gave a linear-time algorithm for the decision problem improving earlier bounds [3, 8]. They also gave an $O(n \log n)$ time algorithm for finding the largest separating circle and a linear-time algorithm for finding the minimum separating circle between any two finite point sets. With these ideas, Boissonat et al. [4] gave a linear-time algorithm to report the smallest separating circle for two simple polygons, if any exists.

Augustine et al. [1] showed how to preprocess a point set (or a simple polygon) P, so that the largest circle isolating P from a query point can be found in logarithmic time. For the line separability problem, Edelsbrunner showed that a point set P can be preprocessed in $O(n \log n)$ time, so that a separating line between P and a query convex m-gon Q can be computed in $O(\log n + \log m)$ time [7]. In 3D, Dobkin and Kirkpatrick showed that two convex polyhedra of size n and m can be preprocessed in linear time, so that a separating plane, if any exists, can be computed in $O(\log n \cdot \log m)$ time [6]. In this paper we show that a set P on n points can be preprocessed in $O(n \log n)$ time, using $O(n)$ space, so that for any given convex m-gon Q we can find the smallest circle enclosing P and excluding Q in $O(\log n + \log m)$ time. This improves the $O(\log n \cdot \log m)$ bound presented in [2], which is described in this paper as well.

2 Preliminaries

Let P be a set of n points in the plane and let Q be a convex m-gon. A *P-circle* is a circle containing P and a *separating circle* is a P-circle whose interior does not intersect Q. A *separating line* is a straight line leaving the interiors of P and Q in different halfplanes.

Let $\mathbf{C}^*$ denote the minimum separating circle and let $\mathbf{c}^*$ be its center. Note that $\mathbf{C}^*$ passes through at least two points of P, hence $\mathbf{c}^*$ lies on an edge of the farthest-point Voronoi diagram $\mathcal{V}(P)$, which is a tree with leaves at infinity [5]. For each point p of P, let $R(p)$ be the farthest-point Voronoi region of p.

Let C_P be the minimum enclosing circle of P. If C_P is constrained by three points of P then its center, c_P, is at a vertex of $\mathcal{V}(P)$. Otherwise C_P is constrained by two points of P (forming its diameter). In this case, c_P is on the interior of an edge of $\mathcal{V}(P)$ and we insert c_P into $\mathcal{V}(P)$ by splitting the edge where it belongs. Thus, we can think of $\mathcal{V}(P)$ as a rooted tree on c_P. For any given point x on $\mathcal{V}(P)$ there is a unique path along $\mathcal{V}(P)$ joining c_P with x. Throughout this paper we will denote this path by π_x.

Given any point y in the plane, let $C(y)$ be the minimum P-circle with center on y and let $\rho(y)$ be the radius of $C(y)$. We say that y is a *separating point* if $C(y)$ is a separating circle.

3 Properties of the minimum separating circle

In this section we describe some properties of $\mathbf{C}^*$, and the relationship between $\mathbf{c}^*$ and $\mathcal{V}(P)$. Several results in this section have been proved in [2].

Let $\mathrm{CH}(P)$ denote the convex hull of P. We assume that the interiors of Q and $\mathrm{CH}(P)$ are disjoint, otherwise there is no separating circle. Also, if Q and C_P have disjoint interiors, then C_P is trivially the minimum separating circle.

[*]Département d'Informatique, Université Libre de Bruxelles, Brussels, Belgium, aloupis.greg@gmail.com, {lbarbafl,slanger}@ulb.ac.be
[†]Chargé de recherches du F.R.S.-FNRS
[‡]Maître de recherches du F.R.S.-FNRS

Observation 1 *Every P-circle contained in a separating circle is also a separating circle.*

Lemma 2 [2] *Let x be a point on $\mathcal{V}(P)$. The function ρ is monotonically increasing along every edge of the path π_x starting at c_P.*

We remark that Lemma 2 has also been shown to hold on vertices of π_x (not edge interiors), in [12].

Theorem 3 [2] *Let s be a point on $\mathcal{V}(P)$. If s is a separating point, then $\mathbf{c}^*$ belongs to π_s.*

Given a separating point s, we claim that if we move a point y continuously from s towards c_P on π_s, then $C(y)$ will shrink and approach Q, becoming tangent to it for the first time when y reaches $\mathbf{c}^*$. To prove this claim in Lemma 6, we introduce the following notation.

Let x be a point lying on an edge e of $\mathcal{V}(P)$ such that e lies on the bisector of $p, p' \in P$. Let $C^-(x)$ and $C^+(x)$ be the two closed convex regions obtained by splitting the disk $C(x)$ with the segment $[p, p']$. Assume that x is contained in $C^-(x)$; see Figure 1.

Observation 4 *Let x, y be two points lying on an edge e of $\mathcal{V}(P)$. If $\rho(x) > \rho(y)$, then $C^+(x) \subset C^+(y)$ and $C^-(y) \subset C^-(x)$.*

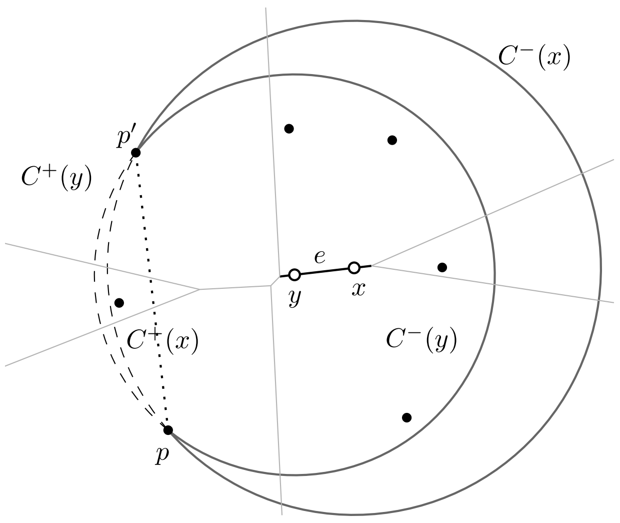


Figure 1: Observation 4 when $\rho(x) > \rho(y)$.

Lemma 5 *Let s be a point on $\mathcal{V}(P)$ and let x and y be two points on π_s. If $\rho(x) > \rho(y)$, then $C^+(x) \subset C^+(y)$ and $C^-(y) \subset C^-(x)$.*

Proof. Note that if x and y lie on the same edge, then the result holds by Observation 4. If they are on different edges, we consider the path $\Phi = (x, v_0, \ldots, v_k, y)$ contained in π_s joining x and y, such that v_i is a vertex of $\mathcal{V}(P)$, $i \in \{0, \ldots, k\}$. Thus, Observation 4 and Lemma 2 imply that $C^+(x) \subset C^+(v_0) \subset \ldots \subset C^+(v_k) \subset C^+(y)$ and that $C^-(y) \subset C^-(v_k) \subset \ldots \subset C^-(v_0) \subset C^-(x)$. □

Note that $\mathbf{C}^* = C(\mathbf{c}^*)$ must be tangent to the boundary of Q. Otherwise, $\mathbf{c}^*$ could be pushed closer to the root on $\mathcal{V}(P)$, while keeping it as a separating point until it reaches Q. From now on we refer to ϕ' as the tangency point between $\mathbf{C}^*$ and Q. We claim that ϕ' lies on the boundary of $C^+(\mathbf{c}^*)$. Assume to the contrary that ϕ' lies on $C^-(\mathbf{c}^*)$. Let $\varepsilon > 0$ and let c_ε be the point obtained by moving $\mathbf{c}^*$ a distance of ε towards c_P on $\mathcal{V}(P)$. Note that by Lemma 2, $\rho(c_\varepsilon) < \rho(\mathbf{c}^*)$. In addition, Lemma 5 implies that $C^-(c_\varepsilon) \subset C^-(\mathbf{c}^*)$. Since we assumed that ϕ' lies on the boundary of $C^-(\mathbf{c}^*)$, we conclude that ϕ' does not belong to $C(c_\varepsilon)$. This implies that, for ε sufficiently small, $C(c_\varepsilon)$ is a separating circle which is a contradiction to the minimality of $\mathbf{C}^*$. The following result was mentioned in [2] without a proof.

Lemma 6 *Let s be a separating point. If x is a point lying on π_s, then $C(x)$ is a separating circle if and only if $\rho(x) \geq \rho(\mathbf{c}^*)$. Moreover, $\mathbf{C}^*$ is the only separating circle whose boundary intersects Q.*

Proof. We know by Theorem 3 that $\mathbf{c}^*$ belongs to π_s. Let x_1 and x_2 be two points on π_s such that $\rho(x_1) < \rho(\mathbf{c}^*)$ and $\rho(\mathbf{c}^*) < \rho(x_2)$. Lemma 5 implies that $C^+(\mathbf{c}^*) \subset C^+(x_1)$ and since ϕ' belongs to the boundary of $C^+(\mathbf{c}^*)$, we conclude $C(x_1)$ contains ϕ' in its interior. Therefore $C(x_1)$ is not a separating circle.

On the other hand, $C(x_2)$ contains no point of Q. Otherwise, let $q \in Q$ be a point lying in $C(x_2)$. Two cases arise: Either q belongs to $C^-(x_2)$ or q belongs to $C^+(x_2)$. In the former case, since $\rho(s) > \rho(x_2)$, $q \in C^-(x_2) \subset C^-(s)$— a contradiction since $C(s)$ is a separating circle. In the latter case, since $\rho(x_2) > \rho(\mathbf{c}^*)$, Lemma 5 would imply that q belongs to the interior of $\mathbf{C}^*$ which would also be a contradiction. □

The basis of our algorithm is to find a separating point s and then perform a binary search on π_s to find a separating circle tangent to Q with center on this path.

4 Preprocessing

We first compute $\mathcal{V}(P)$ and c_P in $O(n \log n)$ time [13]. Assume that $\mathcal{V}(P)$ is stored as a binary tree with n (unbounded) leaves, so that every edge and every vertex of the tree has a set of pointers to the vertices of P defining it. Every Voronoi region is stored as a convex polygon and every vertex p of P has a pointer to $R(p)$. If c_P is not a vertex of $\mathcal{V}(P)$, we split the edge that it belongs to. We want our data structure to support binary search queries on any possible path π_s of $\mathcal{V}(P)$.

Thus, to guide the binary search we would like to have an oracle that answers queries of the following form: Given a vertex v of π_s, decide if $\mathbf{c}^*$ lies either between c_P and v or between v and s in π_s. By Lemma 6, we only need to decide if $C(v)$ is a separating circle.

We will use an operation on the vertices of $\mathcal{V}(P)$ called PointBetween with the following properties. Given two vertices u, v in π_s, PointBetween(u, v) returns a vertex z that splits the path on π_s joining u and v into two subpaths. Moreover, if we use our oracle to discard the subpath that does not contain $\mathbf{c}^*$ and we proceed recursively on the other, then, after $O(\log n)$ iterations, the search interval becomes only an edge of π_s containing $\mathbf{c}^*$.

A data structure that supports this operation was presented in [12]. This data structure can be constructed in $O(n)$ time and uses linear space.

5 The algorithm

Since Q is a convex m-gon, we can check in $O(\log m)$ time if C_P is a separating circle [7]. Thus, assume that C_P is not the minimum separating circle. To determine the position of $\mathbf{c}^*$ on $\mathcal{V}(P)$, we first find a separating point s and then search for $\mathbf{c}^*$ on π_s using our data structure. To find s, we construct a separating line L between P and Q in $O(\log n + \log m)$ time [7]. Let p_L be the point of P closest to L and assume that no other point in P lies at the same distance; otherwise rotate L slightly. Let $L_\perp$ be the perpendicular to L that contains p_L and let s be the intersection of $L_\perp$ with the boundary of $R(p_L)$; see Figure 2. We know that $L_\perp$ intersects $R(p_L)$ because L can be considered as a P-circle, containing only p_L, with center at infinity on $L_\perp$.

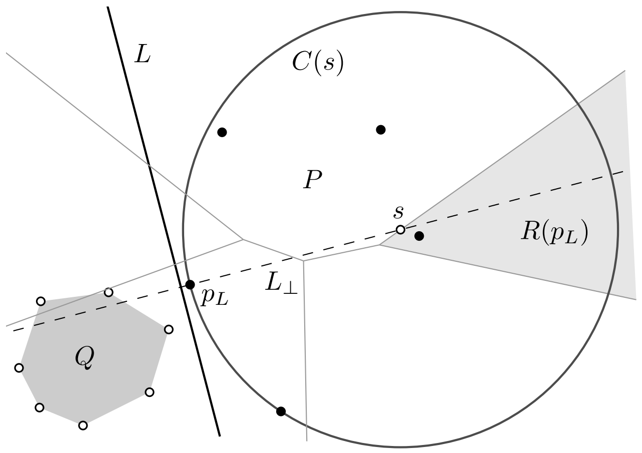


Figure 2: Construction of s. Figure borrowed from [2].

Since s is on the boundary of $R(p_L)$, $C(s)$ passes through p_L. Furthermore $C(s)$ is contained in the same halfplane defined by L that contains P. So $C(s)$ is a separating circle. Assume that s lies on the edge $\overline{xy}$ of $\mathcal{V}(P)$ with $\rho(x) > \rho(y)$ and let $\pi_s = (u_0 = s, u_1 = y, \ldots, u_r = c_P)$ be the path of length $r + 1$ joining s with c_P in $\mathcal{V}(P)$. Theorem 3 implies that $\mathbf{c}^*$ lies on π_s.

It is then possible to use our data structure to perform a binary search on the vertices of π_s, computing, at each vertex v, the radius of $C(v)$ and the distance to Q in $O(\log m)$ time. This way we can determine if $C(v)$ is a separating (or intersecting) circle. This approach finds c_P in $O(\log n \cdot \log m)$ time and was the algorithm given in [2]. However, an improvement can be obtained by using the convexity of Q.

To determine if some point v on π_s is a separating point, it is not always necessary to compute the distance between v and Q. One can first test, in $O(1)$ time, if $C(v)$ intersects a separating line tangent to Q. If not, then $C(v)$ is a separating circle and we can proceed with the binary search. Otherwise, we can try to compute a new separating line, tangent to Q, not intersecting $C(v)$. The advantage of this is that while doing so, we reduce the portion of Q that we need to consider in the future. This is done as follows:

Compute the two internal tangents L and L' between the convex hull of P and Q in $O(\log n + \log m)$ time. The techniques to construct these tangents are shown in Chapter 4 of [11]. Let q and q' be the respective tangency points of L and L' with the boundary of Q. Consider the clockwise polygonal chain $\varphi = [q = q_0, \ldots, q_k = q']$ joining q and q' as in Figure 3. Recall that ϕ' denotes the intersection point between $\mathbf{C}^*$ and the boundary of Q and note that the tangent line to $\mathbf{C}^*$ at ϕ' is a separating line. Therefore, ϕ' must lie on an edge of φ since no separating line passes through any other boundary point of Q.

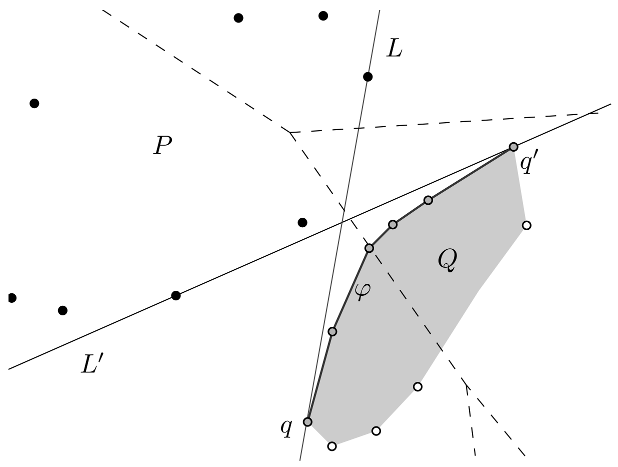


Figure 3: The construction of φ.

If $q = q'$, then $\phi' = q$ and hence we can ignore Q and compute the minimum separating circle between P and

q. As mentioned previously, this takes $O(\log n)$ time. Assume from now on that $q \neq q'$, as shown in Figure 3.

For each edge $e_i = q_i q_{i+1}$ $(0 \leq i \leq k-1)$ of φ, let ℓ_i be the line extending that edge. By construction, we know that each ℓ_i separates P and Q. We say that a point x on ℓ_i but not on e_i lies to the left of e_i if it is closer to q_i, or to the right if it is closer to q_{i+1}.

Our algorithm will essentially perform two parallel binary searches, the first one on π_s and the second one on φ, such that at each step we discard either a section of π_s or a linear fraction of φ. As we search on π_s, every time we find a separating circle, we move towards c_P. When we confirm that a P-circle intersects Q, we move away from c_P. To confirm if a vertex v is a separating point, we compare $C(v)$ to some separating line ℓ_i for intersection in constant time. If $C(v)$ is a separating circle, we discard the section of the path lying below v on $\mathcal{V}(P)$. If $C(v)$ does intersect ℓ_i, we make a quick attempt to check if $C(v)$ intersects Q by comparing $C(v)$ and the edge e_i for intersection. If they intersect, v is not a separating point and we can proceed with the binary search on π_s. Otherwise, the intersection of $C(v)$ with ℓ_i lies either to the left or to the right of e_i. However, in this case we are not able to quickly conclude whether $C(v)$ intersects Q or not. Thus, we suspend the binary search on $\mathcal{V}(P)$ and focus on $C(v)$, using its intersection with ℓ_i to eliminate half of φ. Specifically, the fact that $C(v)$ intersects ℓ_i to one side of e_i (right or left) tells us that no future P-circle on our search will intersect ℓ_i on the other side of e_i. This implicitly discards half of φ from future consideration, and is discussed in more detail in Theorem 7. Thus, in constant time, we manage to remove a section of the path π_s, or half of φ. The entire process is detailed in Algorithm 1.

Theorem 7 *Algorithm 1 finds the edge of π_s containing $\mathbf{c}^*$ in $O(\log n + \log m)$ time.*

Proof. Our algorithm maintains two invariants. The first is that $C(u)$ is never a separating circle and $C(v)$ is always a separating circle. To begin with, $C(u) = C(s)$ is a separating circle while $C(v) = C_P$ is not. If either of these assumptions does not hold, the problem is solved trivially, without resorting to this algorithm. Changes to u and v occur in steps 14 or 17, and in both the invariant is preserved. Thus, $\mathbf{c}^*$ always lies on the path joining u with v. As a second invariant, ϕ' always lies on the clockwise path joining q_a with q_b along φ. We already explained that the invariant holds when $a = 0$ and $b = k$, corresponding to the inner tangents supporting P and Q. Thus, we only need to look at steps 20 and 22 where a and b are redefined. We analyze Step 20, however Step 22 is analogous.

In Step 20 we know that $C(z)$ intersects ℓ_j to the left of e_j and that e_j does not intersect $C(z)$. We claim that for every point w lying on an edge of π_s, if $C(w)$ is a

Algorithm 1 Given $\varphi = [q = q_0, \ldots, q_k = q']$ and $\pi_s = (u_0 = s, u_1 = y, \ldots, u_r = c_P)$, find the edge of π_s that contains $\mathbf{c}^*$.

```
1: Define the initial subpath of $\pi_s$ that contains $\mathbf{c}^*$, $u \leftarrow s, v \leftarrow c_P$
2: Define the initial search interval on the chain $\varphi$, $a \leftarrow 0, b \leftarrow k$
3: **if** $u$ and $v$ are neighbors in $\mathcal{V}(P)$ and $b = a+1$ **then**
4:    Finish and report the segment $S = [u, v]$ and the segment $H = [q_a, q_b]$
5: **end if**
6: Let $z \leftarrow$ FindPointBetween$(u, v)$, $j \leftarrow \lfloor \frac{a+b}{2} \rfloor$
7: Let $e_j \leftarrow \overline{q_j q_{j+1}}$ and let $\ell_j$ be the line extending $e_j$
8: **if** $b > a+1$ **then**
9:    Compute $\rho(z)$ and let $\delta \leftarrow d(z, \ell_j)$, $\Delta \leftarrow d(z, e_j)$
10: **else**
11:    Compute $\rho(z)$ and let $\delta \leftarrow d(z, e_j)$, $\Delta \leftarrow d(z, e_j)$
12: **end if**
13: **if** $\rho(z) \leq \delta$, that is $C(z)$ is a separating circle **then**
14:    Move forward on $\pi_s$, $u \leftarrow z$ and return to Step 3
15: **else**
16:    **if** $\rho(z) > \Delta$ , that is if $C(z)$ is not a separating circle **then**
17:       Move backward on $\pi_s$, $v \leftarrow z$ and return to Step 3
18:    **else**
19:       **if** $C(z)$ intersects $\ell_j$ to the left of $e_j$ **then**
20:          Discard the polygonal chain to the right of $e_j$, $b \leftarrow \max\{j, a+1\}$
21:       **else**
22:          Discard the polygonal chain to the left of $e_j$, $a \leftarrow j$
23:       **end if**
24:       Return to Step 3
25:    **end if**
26: **end if**
```

separating circle that intersects ℓ_j, then it intersects it to the left of e_j. Note that if our claim is true, we can ignore the polygonal chain lying to the right of e_j since no separating circle will intersect it. To prove our claim, suppose that there is a point w on π_s, such that $C(w)$ is a separating circle and $C(w)$ intersects ℓ_j to the right of e_j. Let x and x' be two points on the intersection of ℓ_j with $C(w)$ and $C(z)$, respectively. First suppose that $\rho(w) < \rho(z)$ and recall that by Lemma 5, since x' lies on $C^+(z) \subset C^+(w)$, x' lies in $C(w)$. Thus, both x and x' belong to $C(w)$ which by convexity implies that e_j is contained in $C(w)$. Therefore $C(w)$ is not a separating circle which is a contradiction. Analogously, if $\rho(w) > \rho(z)$, then e_j is contained in $C(z)$ which is directly a contradiction since we assumed the opposite. Thus, our claim holds.

Note that in each iteration of the algorithm, a, b, u or v are redefined so that either a linear fraction of φ is discarded, or a part of π_s is discarded and a new call to PointBetween is performed. Recall that our data structure guarantees that $O(\log n)$ calls to PointBetween are sufficient to reduce the search interval in π_s to an edge [12]. Thus, the algorithm finishes in $O(\log n + \log m)$ iterations.

One additional detail needs to be considered when $b = a + 1$. In this case only one edge $e = [q_a, q_{a+1}]$ remains from φ, and ϕ' lies on e. Thus, if the line ℓ extending e intersects $C(z)$ but e does not, then either Step 20 or 22 is executed. However, nothing will change in these steps and the algorithm will loop. In order to avoid that, we check in Step 8 if only one edge e of φ remains. If this is the case, we know by our invariant that ϕ' belongs to e and therefore we continue the search computing the distance to e instead of computing the distance to the line extending it. This way, the search on φ stops but it continues on π_s until the edge of $\mathcal{V}(P)$ containing $\mathbf{c}^*$ is found.

Since we ensured that every edge in $\mathcal{V}(P)$ has pointers to the points in P that defined it, every step in the algorithm can be executed in $O(1)$ time. Thus, we conclude that Algorithm 1 finishes in $O(\log n + \log m)$ time. Since both invariants are preserved during the execution, Lemma 6 implies that the algorithm returns segments $[u, v]$ from π_s containing $\mathbf{c}^*$, and $[q_a, q_b]$ from φ containing ϕ'. □

From the output of Algorithm 1 it is trivial to obtain $\mathbf{c}^*$ in constant time.

Corollary 8 *After preprocessing a set P of n points in $O(n \log n)$ time, the minimum separating circle between P and any query convex m-gon can be found in $O(\log n + \log m)$ time.*

References

[1] J. Augustine, S. Das, A. Maheshwari, S. C. Nandy, S. Roy, and S. Sarvattomananda. Localized geometric query problems. *CoRR*, abs/1111.2918, 2011.

[2] L. Barba and J. Urrutia. Dynamic circle separability between convex polygons. In *Proceedings of the Spanish Meetings in Computational Geometry*, pages 43–46, 2011.

[3] B. K. Bhattacharya. Circular separability of planar point sets. *Computational Morphology*, pages 25–39, 1988.

[4] J.-D. Boissonnat, J. Czyzowicz, O. Devillers, and M. Yvinec. Circular separability of polygons. *Algorithmica*, 30:67–82, 2001.

[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, third edition, 2008.

[6] D. P. Dobkin and D. G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical Computer Science*, 27(3):241 – 253, 1983.

[7] H. Edelsbrunner. Computing the extreme distances between two convex polygons. *Journal of Algorithms*, 6(2):213 – 224, 1985.

[8] C. E. Kim and T. A. Anderson. Digital disks and a digital compactness measure. In *STOC '84: Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 117–124, New York, NY, USA, 1984. ACM.

[9] N. Megiddo. Linear programming in linear time when the dimension is fixed. *J. ACM*, 31(1):114–127, Jan. 1984.

[10] J. O'Rourke, S. Rao Kosaraju, and N. Megiddo. Computing circular separability. *Discrete and Computational Geometry*, 1:105–113, 1986.

[11] F. Preparata and M. Shamos. *Computational geometry: an introduction*. Springer-Verlag, 1985.

[12] S. Roy, A. Karmakar, S. Das, and S. C. Nandy. Constrained minimum enclosing circle with center on a query line segment. *Computational Geometry*, 42(6-7):632 – 638, 2009.

[13] M. I. Shamos and D. Hoey. Closest-point problems. In *Foundations of Computer Science, 1975., 16th Annual Symposium on*, pages 151 –162, oct. 1975.

Flip Distance Between Two Triangulations of a Point-Set is NP-complete

Anna Lubiw* Vinayak Pathak*

Abstract

Given two triangulations of a convex polygon, computing the minimum number of flips required to transform one to the other is a long-standing open problem. It is not known whether the problem is in P or NP-complete. We prove that two natural generalizations of the problem are NP-complete, namely computing the minimum number of flips between two triangulations of (1) a polygon with holes; (2) a set of points in the plane.

1 Introduction

Given a triangulation in the plane, a *flip* operates on two triangles that share an edge and form a convex quadrilateral. The flip replaces the diagonal of the convex quadrilateral by the other diagonal to form two new triangles. A sequence of flips can transform any triangulation to any other triangulation—this is true for triangulations of a convex polygon, and more generally for triangulations of a point set, and for triangulations of a polygon with holes.

In this paper we investigate the complexity of computing the *flip distance*, which is the minimum number of flips to transform one triangulation to another. This is particularly interesting for convex polygons, where the flip distance is the rotation distance between two binary trees (see below).

The main result of our paper is that it is NP-complete to compute the flip distance between two triangulations of a polygon with holes, or of a set of points in the plane.

After submitting this paper, we learned that Pilz [20] independently proved the same result. The differences between our proofs are discussed later on.

1.1 Flip distance and rotation distance

Binary search trees are a widely used data structure, and *rotations* are the basic operations used to balance them. Despite the importance of rotations, the complexity of computing the minimum number of rotations to convert one labelled binary search tree to another, called the "rotation-distance", has been open since at least 1982 [6]. It is not known if the problem is NP-complete.

*Cheriton School of Computer Science, University of Waterloo, Waterloo, Canada`{alubiw,vpathak}@uwaterloo.ca`

There is a bijection between binary trees with $n-1$ labeled leaves and triangulations of an n-vertex convex polygon. Moreover, a rotation in the tree corresponds to a flip in the polygon. Thus, computing the rotation distance between two trees is exactly equivalent to computing the flip distance between two triangulations of a convex polygon. See [23].

1.2 Generalizations and related work

Flips have been studied in the geometric setting for triangulations of point sets and of polygons. In this context, a convex polygon is equivalent to a point set in convex position. The former generalizes to simple polygons, and the latter to planar point sets. Both of these are contained in the most general case of a polygon with holes (a "polygonal region"), so long as we consider a point as a one vertex polygonal hole. There is a survey on flips by Bose and Hurtado [4]. It also covers flips in the combinatorial setting of maximal planar graphs, which we will not discuss. Flips are often studied in terms of the *flip graph* which has a vertex for every triangulation and an edge when two triangulations differ by one flip, see e.g., [10].

The foundational result is that the flip graph is connected. This was proved first by Lawson [14] for the case of point sets. He then re-proved the result [13] by arguing that any triangulation can be flipped to the Delaunay triangulation, which then acts as a "canonical" triangulation from which any other triangulation can be reached. The constrained Delaunay triangulation can be used in the same way to argue that any polygonal region has a connected flip graph [2]. For more direct proofs see [9, 12, 18].

Regarding the number of flips needed to transform one triangulation to another, flipping via the [constrained] Delaunay triangulation takes $O(n^2)$ flips—in fact, a more exact bound is the number of visibility edges, see [2]. Hurtado, Noy and Urrutia [12] proved that $\Omega(n^2)$ flips may be required even for triangulations of a polygon. For the case of a convex polygon, Sleator et al. [23] proved that for large values of n, the flip distance between two triangulations of an n-gon is at most $2n-10$, and that $2n-10$ flips are sometimes necessary.

The problem of computing the exact flip distance between two given triangulations is especially interesting for convex polygons, as mentioned above. Lucas [16] gave a polynomial time algorithm for special cases. The best approximation factor is trivially 2, and can be improved in some special cases [15]. Recently it was proved that the problem is fixed parameter tractable in the flip distance [5]. Attempts have also been made to compute good upper and lower bounds on the flip distance efficiently. See, for example, [1, 19, 17, 7].

The more general problem of computing the flip distance between two triangulations of a point set is stated as an open problem in the survey by Bose and Hurtado [4], and the book by Devadoss and O'Rourke [8, Unsolved Problem 12]. Hanke et al. [11] proved that the flip-distance is upper bounded by the total number of intersections between the overlap of the initial and final triangulations. Eppstein [10] provided an algorithm to compute a lower bound on the flip-distance efficiently. He also showed that the lower bound is equal to the flip-distance for certain special kinds of point-sets.

2 Triangulations of polygonal regions

Theorem 1 *The following problem is NP-complete: Given two triangulations of a polygon with holes and a number k, is the flip distance between the two triangulations at most k?*

2.1 Proof idea

Note that the problem lies in NP. We prove hardness by giving a polynomial time reduction from vertex cover on 3-connected cubic planar graphs [3, 24], which is known to be NP-complete [3, 24].

The idea is to take a planar straight-line drawing of the graph and create a polygonal region by replacing each edge by a "channel" and each vertex by a "vertex gadget". We then construct two triangulations of the polygonal region that differ on the channels, and show that a short flip sequence corresponds to a small vertex cover in the original graph.

We begin by describing channels and their triangulations, because this gives the intuition for the proof. A *channel* is a polygon that consists of two 7-vertex reflex chains joined by two *end* edges, as shown in Figures 1(a) and 1(b). Note that every vertex on the upper reflex chain sees every vertex on the lower reflex chain and vice versa. We identify two triangulations of a channel: a *left-inclined triangulation* as shown in Figure 1(a); and a *right-inclined triangulation* as shown in Figure 1(b).

A channel is the special case $n = 7$ of the polygons H_n of Hurtado et al. [12]. They prove in Theorem 3.8 that the flip distance between the right-inclined and left-inclined triangulations of H_n is $(n-1)^2$. We include a different proof in order to generalize:

Property 1 *Transforming a left-inclined triangulation of a channel to a right-inclined triangulation takes at least 36 flips.*

Proof. In any triangulation of a channel, each edge of the upper reflex chain is in a triangle whose apex lies on the bottom reflex chain. This apex must move from lower right (B_7) to lower left (B_1), in order to transform the left-inclined triangulation to the right-inclined triangulation. Similarly, each edge of the lower reflex chain is in a triangle whose apex lies on the upper reflex chain, and must move from upper left to upper right. However, one flip can only involve one edge of the upper chain and one edge of the lower chain (no other 4 vertices form a convex quadrilateral), and thus can only move one upper and one lower apex, and only by one vertex along the chain. Twelve triangles times six apex moves per triangle divided by two apex moves per flip gives a lower bound of 36 flips. □

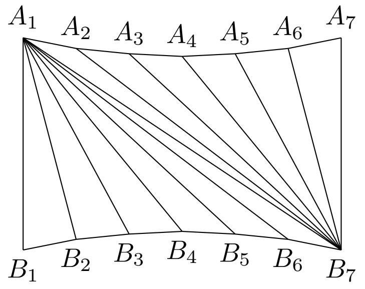
We now show that the number of flips goes down if a channel has a *cap*, an extra vertex that is visible to all the channel vertices, as shown in Figure 1(c).

Property 2 *The flip distance from a left-inclined to a right-inclined triangulation of a capped channel is 24.*

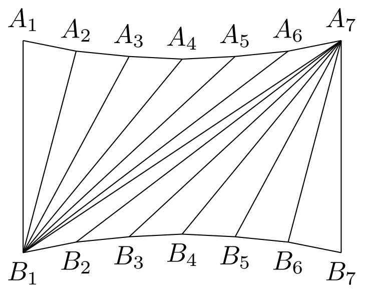
Proof. The "canonical" triangulation shown in Figure 1(d) is 12 flips away from both the left-inclined and the right-inclined triangulations of a capped channel: To flip the left-inclined triangulation to the canonical triangulation, flip edges $A_1B_1, \ldots, A_1B_7$ followed by edges $A_2B_7, \ldots, A_6B_7$ in that order. Similarly for the right-inclined triangulation.

For the lower bound, we follow the same idea as above. In any triangulation, each edge of the upper [lower] reflex chain is in a triangle whose apex is either the cap or a vertex of the lower [upper] chain. There are only two kinds of flips: (1) a flip involving the cap vertex, an edge of one chain, and a vertex of the other chain; and (2) a flip involving one edge of each chain. A flip of type (1) moves the apex of only one triangle, and moves the apex to or from the cap. If a triangle is altered by a flip of type (1) then at least two such flips are required, one to move the apex to the cap and one to move the apex from the cap. If a triangle is only altered by flips of type (2), then, as above, it costs 3 flips to get the apex to its destination. Thus the 12 triangles require at least 24 flips. □

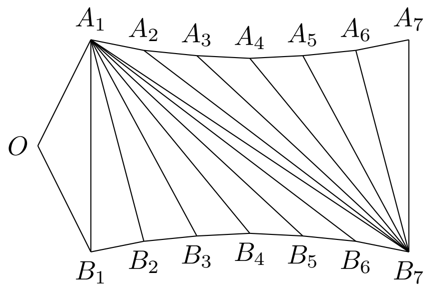
We now elaborate on the idea of our reduction. We create a polygonal region by replacing each edge in the planar drawing by a channel, and each vertex by a vertex gadget. We make two triangulations of the polygonal region. In triangulation T_1 all edge channels are left-inclined and in T_2 all edge channels are right-inclined. The triangulations are otherwise identical. We design vertex gadgets so that making a few flips in a vertex gadget creates a cap for a channel connected to it. Since transforming a channel from left-inclined to right-inclined is less costly if it is capped, the minimum flip


A_1 A_2 A_3 A_4 A_5 A_6 A_7
B_1 B_2 B_3 B_4 B_5 B_6 B_7

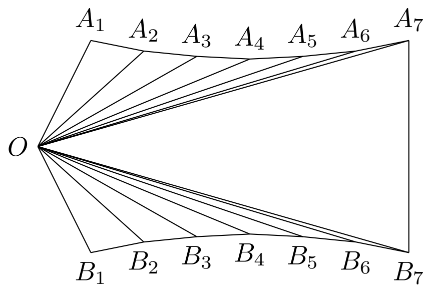
(a) A left-inclined channel.


A_1 A_2 A_3 A_4 A_5 A_6 A_7
B_1 B_2 B_3 B_4 B_5 B_6 B_7

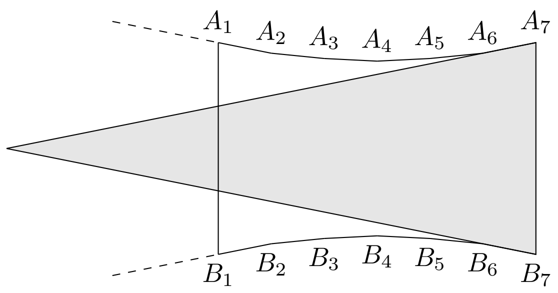
(b) A right-inclined channel.


A_1 A_2 A_3 A_4 A_5 A_6 A_7
O
B_1 B_2 B_3 B_4 B_5 B_6 B_7

(c) A capped channel.


A_1 A_2 A_3 A_4 A_5 A_6 A_7
O
B_1 B_2 B_3 B_4 B_5 B_6 B_7

(d) The canonical channel


A_1 A_2 A_3 A_4 A_5 A_6 A_7
B_1 B_2 B_3 B_4 B_5 B_6 B_7

(e) Narrow (shaded) and wide (dashed) mouths.

Figure 1: Channels

sequence that transforms all the channels is obtained by choosing the smallest set of vertices that covers all the edges and using them to cap all the channels. Thus, intuitively, a minimum flip sequence corresponds to a minimum vertex cover.

One complication is that we cannot construct a vertex gadget for a *sharp* vertex—a vertex of degree 3 where one of the three incident angles in the planar drawing is $\geq \pi$. Therefore, we first show how to eliminate sharp vertices. Let G be our given 3-connected cubic planar graph. Using a result of Rote and Bárány [21], we can find, in polynomial time, a *strictly convex* drawing of G on a polynomial-sized grid. *Strictly convex* means that each face is a strictly convex polygon. Thus the only sharp vertices of this drawing are the vertices of the outer face. We replace each sharp vertex v by a 3-vertex chain v_1, v_2, v_3 as shown in Figure 2. We claim that G has a vertex cover of size $\leq k$ if and only if the modified graph has a vertex cover of size $\leq k + t$, where t is the number of vertices on the outer face of G. This is because any minimum vertex cover of the modified graph can be adjusted to use either $\{v_1, v_3\}$ (corresponding to v being in the vertex cover of G), or $\{v_2\}$ (corresponding to v not being in the vertex cover of G).

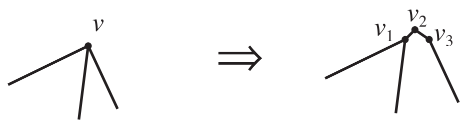


Figure 2: Eliminating sharp vertices

We remark that Pilz's independent NP-hardness reduction [20] is from general (non-planar) vertex cover. His construction begins with the same channel gadgets, but then uses channels that overlap geometrically while flipping independently.

2.2 Details of the reduction

For the remainder of the proof we will assume that we have a graph G with vertices of degree 2 and 3, and a straight-line planar drawing, Γ, of the graph on a polynomial sized grid with no sharp vertices.

We define the *narrow* and *wide* mouths of a channel as shown in Figure 1(e). Any point inside the narrow mouth but outside the channel can be a potential cap for the channel. We show below that a vertex outside the wide mouth does not reduce the flip distance.

We now describe the triangulated vertex gadgets. See Figures 3(a) and 3(b). Each of the 2 or 3 channels attached to the vertex gadget will have one potential cap. We place a convex quadrilateral $CDEF$ with diagonal

CE, called the *lock*, that separates each channel from its potential cap. Thus the lock CE must be flipped, or "unlocked", in order to cap any channel.

For the degree-2 gadget (see Figure 3(a)), place point C in the smaller angular sector (of angle $< \pi$) between the two channels, so that C is outside the wide mouths of both channels. Place points D, E, and F in the other angular sector, with D inside channel 1's narrow mouth and outside channel 2's wide mouth, E outside the wide mouth of both channels, and F inside channel 2's narrow mouth and outside channel 1's wide mouth. Triangulate as shown. Thus D is a potential cap for channel 1 and F is a potential cap for channel 2.

For the degree-3 gadget (see Figure 3(b)), note that because the vertex is not sharp, the mouth of each channel exits between the other two channels. We place vertices in the angular sectors as shown in the figure. Place D inside the intersection of the narrow mouths of channels 1 and 2, and outside the wide mouth of channel 3. Place F inside channel 3's narrow mouth and outside channel 1 and 2's wide mouths. Place C and E outside the wide mouths of all the channels. Triangulate as shown. Thus D is a potential cap for both channel 1 and 2 and F is a potential cap for channel 3.

Observe that every channel is blocked from its unique potential cap by exactly 3 edges. (For example, in Figure 3(b), channel 1 is separated from its potential cap D by edges FA, FE, and CE.) Observe furthermore that for each vertex gadget, the sets of blocking edges of the channels have one edge in common, namely the locking edge CE, and are otherwise disjoint. These properties are crucial for correctness.

We will say that a vertex gadget is *locked* if the diagonal CE exists and *unlocked* otherwise. We first show what is possible with unlocked vertex gadgets.

Property 3 *If we unlock a vertex gadget then, for each channel attached to it, there is a sequence of 28 flips that transforms the channel triangulation and returns the vertex gadget to its (unlocked) state.*

Proof. We first claim that there is a 2-flip sequence that caps the channel. We enumerate the possibilities (refer to Figure 3). Note that we handle channels one at a time, not simultaneously. For the degree-2 gadget: flip CF followed by CA for channel 1; flip CD followed by CB' for channel 2. For the degree-3 gadget: flip FE followed by FA for channel 1; flip CF followed by CA' for channel 2; flip ED followed by EA'' for channel 3. Once the channel is capped, we can transform the left-inclined triangulation to the right-inclined triangulation in 24 flips by Property 2. Then we undo the 2 flips that capped the channel. The total number of flips is 28. □

Next we give lower bounds on the number of flips. First, note that the proof of Property 1 carries over to:

Property 4 *Transforming a left-inclined triangulation of a channel to a right-inclined triangulation takes at least 36 flips even in the presence of other vertices, so long as the other vertices lie outside the wide mouths at either end of the channel.*

We now consider what happens when we unlock some vertex gadgets. Let T_1' be the triangulation obtained from T_1 by unlocking some vertex gadgets. Let T_2' be the triangulation obtained from T_2 by unlocking the same vertex gadgets. Let C be the set of channels that have a locked vertex gadget at both ends. Then:

Property 5 *If the vertex gadgets at the ends of the channels of C remain locked, then the number of flips required to transform T_1' to T_2' is at least $28|E-C|+36|C|$.*

Proof. Consider a channel of C, with a locked vertex gadget at both ends. The cap vertices of the channel are not useable. By construction, the other vertices are outside the wide mouths of the channel. Therefore, by Property 4, we need 36 flips to transform it.

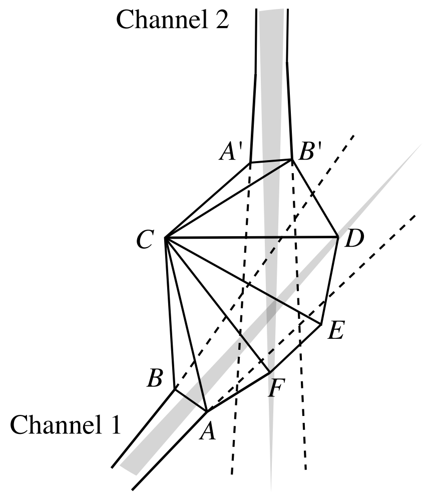
Consider the channels with an unlocked vertex gadget at one end. We only save flips by capping the channel. To do this, we must flip the two edges that block the channel from its cap. Because the edges that block one channel are disjoint from the edges that block any other channel, we must do two flips per channel, and we must re-flip those edges to return to the original state. Finally, by Property 2 it takes at least 24 flips to transform a capped channel. (Note that the proof of Property 2 carries over even if the channel is capped at both ends.) The total number of flips is 28 per channel. □

2.3 Putting it all together

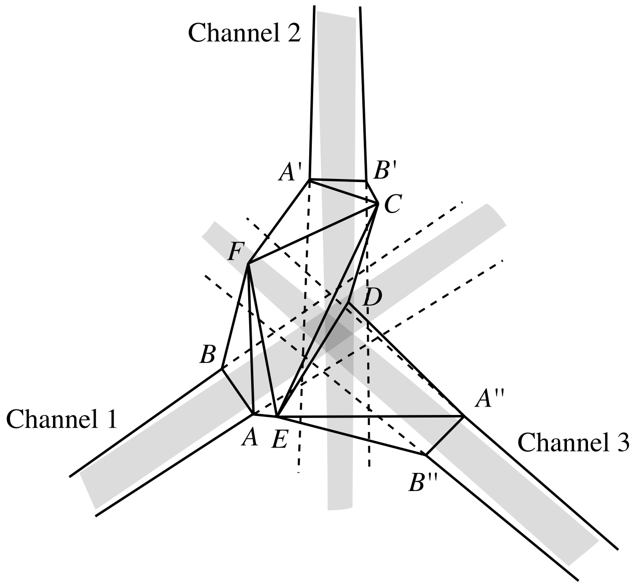
Lemma 2 *G has a vertex cover of size $\leq k$ if and only if the flip distance between the two triangulations T_1 and T_2 is $\leq 2k + 28|E|$.*

Proof. Suppose that G has a vertex cover of size k. Unlock the corresponding k vertex gadgets. Each edge channel has an unlocked gadget at one end, so by Property 3 we can transform between the two triangulations of the channel in 28 flips. When all channels have been transformed, we relock the k vertex gadgets. The total number of flips is $2k + 28|E|$.

For the other direction, suppose that there is a flip sequence between T_1 and T_2 of length $\leq 2k + 28|E|$. Let L be the set of vertices whose gadgets are unlocked in the flip sequence. Let C be the set of edges not covered by vertex set L. By adding one vertex to cover each edge of C, we observe that G has a vertex cover of size $|L| + |C|$. Thus it suffices to show that $|L| + |C| \leq k$. By Properties 4 and 5 the number of flips is at least $2|L| + 36|C| + 28(|E - C|) \geq 2|L| + 28|E| + 8|C|$.



(a) Degree 2



(b) Degree 3

Figure 3: Gadgets for vertices

By assumption, the number of flips was $\leq 2k + 28|E|$. Therefore $2|L| + 8|C| \leq 2k$, which implies that $|L| + |C| \leq k$, as required. □

The last ingredient of the NP-completeness proof is to show that the reduction takes polynomial time. We need the following claim.

Claim 1 *The size of the coordinates used in the construction is bounded by a polynomial in n.*

Proof. We give the main idea here, with further details in the full version. We begin with a straight line drawing of a graph on a polynomial size grid. Expand the grid, and allocate a square region around each vertex for the vertex gadget. Expand each edge to two parallel line segments. These line segments will become the channel, but for now, the reflex vertices of the channel are all collinear, which means that the channel's wide mouth is equal to its narrow mouth. The points C, D, E, F of the vertex gadget go in *feasible* regions defined by the wide and narrow mouths (e.g. in the 3-channel gadget, point D lies in the narrow mouth of channels 1 and 2, but outside the wide mouth of channel 3). We make the channels narrow enough so that all the feasible regions intersect the square region allocated to the gadget. We claim that we can choose the channel end points A, B, A', B', A'', B'' on the expanded grid so that the resulting channels satisfy this property.

Now we pick points C, D, E, F inside the appropriate regions. Because the boundaries of the feasible regions are determined by pairs of points on the expanded grid, the new points can be chosen to have polynomial size (because solutions to linear programs have polynomial size as shown in Theorem 10.1 of [22]). Finally we place the reflex points of each channel. The feasible region wherein each set of reflex points can be placed is bounded by lines through pairs of points already placed. Thus, we can choose reflex points of polynomial size. □

3 Triangulations of point-sets

We prove the NP-hardness of computing the flip distance between two triangulations of a point set by reducing from computing the flip distance between two triangulations of a polygonal region. Given two triangulations T_1 and T_2 of a polygonal region R, we triangulate all the holes and pockets of R the same way in both triangulations. Next, we repeat each edge on the boundary of the holes and pockets n^2 times (as shown in Figure 4). This gives two triangulations of a point set. We claim that the flip distance between the two triangulations of the point set will be the same as the flip distance between the original T_1 and T_2. We use the fact that the flip distance to the constrained Delaunay triangulation is at most $\binom{n}{2}$ [2]. Thus the flip distance between T_1 and T_2 is less than n^2, but dismantling a single set of repeated edges will itself require more flips. The exact details on how we repeat the edges can be found in the full version of our paper.

Theorem 3 *The following problem is NP-complete: Given two triangulations of a point set in the plane, and a number k, is the flip distance between the triangulations at most k?*

4 Conclusion

We have shown that it is NP-complete to compute the flip distance for triangulations of a polygonal region, or

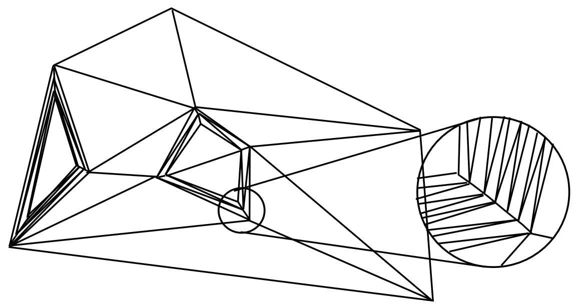


Figure 4: Repeating edges on the boundary of holes.

a point set. The problem remains open for a convex polygon, or a simple polygon.

Acknowledgements. We thank Therese Biedl for helpful suggestions.

References

[1] J.-L. Baril and J.-M. Pallo. Efficient lower and upper bounds of the diagonal-flip distance between triangulations. *Information Processing Letters*, 100:131–136, 2006.

[2] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In *Computing in Euclidean Geometry, 2nd edition*, pages 47–123. World Scientific, Lecture Notes Series on Computing, 1995.

[3] T. Biedl, G. Kant, and M. Kauffman. On triangulating planar graphs under the four-connectivity constraint. *Algorithmica*, 19(4):427–446, 1997.

[4] P. Bose and F. Hurtado. Flips in planar graphs. *Computational Geometry: Theory and Applications*, 42:60–80, 2009.

[5] S. Cleary and K. St. John. Rotation distance is fixed-parameter tractable. *Inf. Process. Lett.*, 109(16):918–922, July 2009.

[6] K. Culik II and D. Wood. A note on some tree similarity measures. *Information Processing Letters*, 15(1):39–42, 1982.

[7] P. Dehornoy. On the rotation distance between binary trees. *Advances in Mathematics*, 223(4):1316–1355, 2010.

[8] S. L. Devadoss and J. O'Rourke. *Discrete and Computational Geometry.* Princeton University Press, 2011.

[9] N. Dyn, I. Goren, and S. Rippa. Transforming triangulations in polygonal domains. *Computer Aided Geometric Design*, 10:531–536, 1993.

[10] D. Eppstein. Happy endings for flip graphs. *J. Computational Geometry*, 1, 2010.

[11] S. Hanke, T. Ottmann, and S. Schuierer. The edge-flipping distance of triangulations. *Journal of Universal Computer Science*, 2(8):570–579, 1996.

[12] F. Hurtado, M. Noy, and J. Urrutia. Flipping edges in triangulations. *Discrete and Computational Geometry*, 22:333–346, 1999.

[13] C. Lawson. Software for C^1 surface interpolation. In J. Rice, editor, *Mathematical Software III*, pages 161–194. Academic Press, New York, 1977.

[14] C. L. Lawson. Transforming triangulations. *Discrete Mathematics*, 3(4):365–372, 1972.

[15] M. Li and L. Zhang. Better approximation of diagonal-flip transformation and rotation transformation. In *Proceedings of the 4th Annual International Conference on Computing and Combinatorics*, COCOON '98, pages 85–94. Springer-Verlag, 1998.

[16] J. M. Lucas. Untangling binary trees via rotations. *The Computer Journal*, 47(2), 2004.

[17] F. Luccio, A. M. Enriquez, and L. Pagli. Lower bounds on the rotation distance of binary trees. *Information Processing Letters*, 110(21):934–938, 2010.

[18] E. Osherovich and A. Bruckstein. All triangulations are reachable via sequences of edge-flips: an elementary proof. *Computer Aided Geometric Design*, 25:157–161, 2008.

[19] J. Pallo. An efficient upper bound of the rotation distance of binary trees. *Information Processing Letters*, 73:87–92, 2000.

[20] A. Pilz. Flip distance between triangulations of a planar point set is NP-complete. http://arxiv.org/abs/1206.3179.

[21] G. Rote and I. Bárány. Strictly convex drawings of planar graphs. *Documenta Mathematica*, 11:369–391, 2006.

[22] A. Shrijver. *Theory of Linear and Integer Programming.* John Wiley & Sons, 1986.

[23] D. D. Sleator, R. E. Tarjan, and W. P. Thurston. Rotation distance, triangulations, and hyperbolic geometry. *J. Amer. Math. Soc*, 1:647–681, 1988.

[24] T. Watanabe, T. Ae, and A. Nakamura. On the node cover problem of planar graphs. In *Proceedings of the International Symposium on Circuits and Systems*, pages 78–81, 1979.

Steiner Reducing Sets of Minimum Weight Triangulations

Cynthia M. Traub[*]

Abstract

This paper develops techniques for computing the minimum weight Steiner triangulation of a planar point set. We call a Steiner point P a *Steiner reducing point* of a planar point set X if the weight (sum of edge lengths) of a minimum weight triangulation of $X \cup \{P\}$ is less than that of X. We define the *Steiner reducing set* $St(X)$ to be the collection of all Steiner reducing points of X. We provide here necessary conditions for membership in the Steiner reducing set. We prove that $St(X)$ can be topologically complex, containing multiple connected components or even holes. We construct families of sets X for which the number of connected components of $St(X)$ grows linearly in the cardinality of X. We further prove that $St(X)$ need not be simply connected, and the rank of $H_1(St(X))$ (i.e. the number of holes) can also grow linearly in the cardinality of X.

1 Introduction

We consider minimum weight triangulations of point sets that properly contain an initial input set X of n points. We examine the topology of the collection of Steiner points distinguished by the property that adding one such point to the input set results in a minimum weight triangulation of a set of $n+1$ points with weight less than that of X. We call these distinguished Steiner points *Steiner reducing points*; the collection of all Steiner reducing points is called the *Steiner reducing set* $St(X)$. We present necessary conditions for a point to be a Steiner reducing point. Our two main results prove that the number of connected components of $St(X)$ can grow linearly in the size of the input set, and $St(X)$ may fail to be simply connected, as illustrated in Figure 1. In Section 2 we will present two necessary conditions for constructing Steiner reducing sets. The topology of Steiner reducing sets is studied in Section 3.

1.1 Definitions and Techniques

A *triangulation* of a finite set X of points in $\mathbb{R}^2$ is an inclusion-maximal set of non-intersecting straight line segments between pairs of points in X. The *weight* of a triangulation is defined as the sum of the Euclidean lengths of its line segments, hence, a *minimum weight*

[*]Department of Mathematics and Statistics, Southern Illinois University Edwardsville, `cytraub@siue.edu`

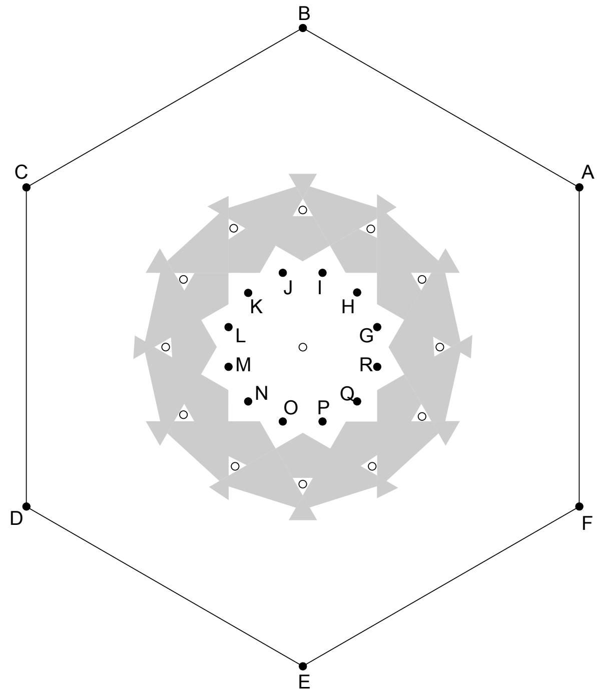


Figure 1: The Steiner reducing set of the point set whose minimum weight triangulation is illustrated in Figure 7; empty circles represent points that are not Steiner reducing points

triangulation of X is a triangulation (not necessarily unique) with weight less than or equal to that of any other triangulation of X. We denote the weight of a minimum weight triangulation of a point set X by mwt(X). An edge AB is *unavoidable* if every triangulation of X contains the edge AB. The edges of the boundary of the convex hull are all unavoidable, as are any edges interior to the convex hull which are not properly intersected by any other possible edge. We direct the reader to a standard topology book [9] for formal definitions of the topological terms used in this paper. For our purposes here, the rank of $H_0(Z)$ counts the number of connected components in the point set Z, and the rank of $H_1(Z)$ counts the number of holes or handles in a set Z. A path-connected set without holes is called *simply connected*.

Calculations of minimum weight triangulations given in this paper were made using integer programming over the universal polytope, or for small examples, verified by hand. The program universalbuilder written by De Loera and Peterson was used to generate a system of inequalities defining the universal polytope of each input set X, which was then input to the optimization engine CPLEX. This process found the weight of the minimum weight triangulation as well as the triangles used. A Bash shell script was used to test the effects of adding an additional point at each location in an $n \times n$ grid. In order to show that a minimal weight triangulation has a specific structure for each element of a 1- or 2-dimensional set of Steiner points, we appeal to the $\beta-$skeleton, unavoidable edges, and the triangle inequality, as well as the interaction between certain curves called $k-$ellipses and the chambers of the hyperplane arrangement induced by specific pairs of points.

1.2 Background and Previous Work

The minimum weight triangulation decision problem "Given a finite planar point set X and a positive integer b, is there a triangulation of X with weight b or less?" has been a problem of theoretical and computational interest for over thirty years [6]. The problem was shown to be NP-hard in 2006 [12], yet determination of minimum weight triangulations of special classes of point sets, such as polygonal domains, can be done in polynomial time [7, 11]. The shortest edge [7] and specific subsets of edges such as the β-skeleton [2, 10] have been proven to belong to all minimum weight triangulations. The simultaneous addition of many Steiner points to certain point sets can reduce triangulation weight by a significant amount, as shown in [4]. Practical motivations for the study of Steiner points include improving the ability of meshes to approximate fine details [1], and allowing approximation of the minimum weight triangulation of an input point set [4]. Our work presented here is the first to define and study the shape of the Steiner reducing set.

In order to find all edges of a minimum weight triangulation (after perhaps identifying a subset of the edges), one algorithmic approach applies integer programming to the universal polytope. This polytope has vertices corresponding to each triangulation of X. Thus, even though the minimum weight triangulation problem is known to be NP-hard, there are algorithmic means for finding the minimum weight triangulation of planar point sets of up to several hundred points that do not rely on ad-hoc methods [3]. Note that no polynomial time algorithm is known to verify that a proposed triangulation is indeed minimal.

In this paper we reveal topological complexity behind the minimum weight triangulation problem by giving conditions that allow the addition of a Steiner point to a fixed input set to reduce the weight of a minimum weight triangulation. This answers a question posed by Jesús De Loera in 2003 during the MSRI Summer Graduate Workshop on Triangulations of Point Sets.

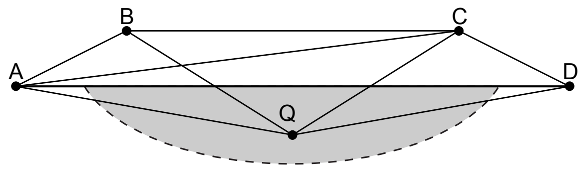


Figure 2: The shaded area is the Steiner reducing set of $Y = \{A, B, C, D\}$, shown with Steiner reducing point Q

2 Steiner Reducing Sets: Necessary Conditions

In this section, we give a means for identifying subsets of the Steiner reducing set via the combinatorial structure of a minimal triangulation and the geometric properties of an associated multi-focal ellipse (known as a k-ellipse). To our knowledge, all known examples of Steiner reducing points are located either outside the interior of the convex hull of X (as in Figure 2), or in the interior of a non-convex polygon formed by edges that belong to a minimum weight triangulation of X (as in Figure 5). The non-convexity of such a polygon will lead to Steiner reducing sets with interesting topology. It remains an open question whether there exist convex polygons which admit Steiner reducing points in the interior of their convex hulls.

It is simple to show that no set of three points admits a Steiner reducing point. Case analysis of four-point sets reveals that no such set will admit a Steiner reducing point in the interior of its convex hull. There are four-point sets, however, which allow Steiner reducing points exterior to and on the edges of their convex hulls. One illustration of such a set is shown in Figure 2.

The minimum weight triangulation of $Y \cup \{Q\}$ for $Q = (x, y)$, $y < 0$, will use edges AQ, BQ, CQ, DQ, when Q is "close" to Y, in addition to edges AB, BC, CD, as seen in Figure 2. Since the new edges replace edges AC and AD from the original triangulation, the Steiner reducing set $St(Y)$ consists of all points Q which satisfy the inequality
$|AQ| + |BQ| + |CQ| + |DQ| < |AC| + |AD| = 10 + \sqrt{65}$.

The curved boundary of $St(Y)$ is part of a curve known as a 4-ellipse. More generally, the locus of all points $P \in \mathbb{R}^2$ such that the sum of distances from P to each of k distinguished foci is constant is called a *k-ellipse.* Under this definition, the circle is a 1-ellipse, and the standard ellipse is a 2-ellipse. Let

$$E(\mathcal{Q}; d) = \left\{ P \in \mathbb{R}^2 : \sum_{i=1}^{k} |PQ_i| = d \right\}$$

denote the *k-ellipse* with *foci* in $\mathcal{Q} = \{Q_1, \ldots, Q_k\}$ and corresponding *distance sum d.* Note that $E(\mathcal{Q}; d)$ is a closed curve. For $k > 2$, the interior of this level set does not necessarily contain its foci Q_i, though the curve represented by the level set will be convex for every value of $k \geq 1$, as proven in [15]. Denote by $E_<(\mathcal{Q}; d)$ the points in its interior of the k-ellipse.

The history of the k-ellipse reaches back to Fermat, who posed the following challenge: "Given three points in a plane, find a fourth point such that the sum of its distances to the three given points is as small as possible." (The smallest possible 3-ellipse consists of this single point.) A solution to the problem of Fermat was provided by Evangelista Torricelli around 1640, and the distance minimizing point, termed the Fermat-Torricelli point [8], remains a topic of active research. In the late 1600's, Tschirnhaus generalized the standard string and pins construction of an ellipse, illustrating how to draw a 3-ellipse by hand in [18]. Due to a lack of standardized nomenclature for this object, it has been rediscovered many times throughout the literature, receiving such names as Tschirnhaussche Eiflächen [16], W_n curves [5], polyellipses [19], and egglipses [14]. A survey article of k-ellipses and their basic properties appears in [15]. Noting that the 1- and 2-ellipse both appear as the level-zero set of degree two polynomials, Nie, Parillo, and Sturmfels used semidefinite programming to show that the k-ellipse appears as part of the level zero set of a polynomial of degree 2^k if k is odd and degree $2^k - \binom{k}{k/2}$ if k is even [13]. The author's Ph.D. thesis [17] was the first work detailing the connection between k-ellipses and minimum weight triangulations.

The k-ellipse provides a criterion for measuring proximity to multiple distinct points. For the Steiner reducing set to be non-empty, the sum of lengths of edges incident to new Steiner point(s) must be less than the sum of the lengths of the replaced edges minus the lengths of new edges not incident to the Steiner point. Although it is common for k-ellipses to appear as part or all of the boundary of different Steiner reducing sets, these Steiner reducing sets are not simply unions of sets $E_<\big(X_i; d_i\big)$ for various pairs X_i, d_i of foci and distance sums. This is an issue of *feasibility*; locations of Q interior to quadrilateral $ABCD$ lie within the necessary 4-ellipse, but do not give a complete triangulation. For Q to be a Steiner reducing point, it must lie in the intersection of the 4-ellipse with the appropriate feasibility set, as shown in Figure 3. The intersection of these two sets gives the Steiner reducing set of Y.

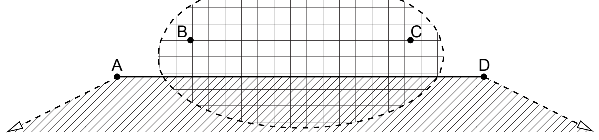


Figure 3: The intersection of $E_<(Y; 10 + \sqrt{65})$ with $F\big(Y, \Pi\big)$

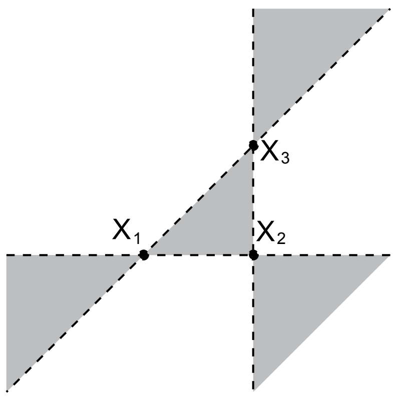


Figure 4: Any triangulation of $X \cup \{V\}$ for V within the shaded feasibility set uses the same set of edges.

We now formally define what we mean by feasibility.

Definition 1 *Given a set $X \subset \mathbb{R}^2$ of n distinct points, an abstract (yet to be located) vertex $V \notin X$, and a graph $G = (X \cup \{V\}, \Pi)$, the* feasibility set $F(X, \Pi)$ *is the set of points $P \in \mathbb{R}^2 \backslash X$ such that if $V = P$, then the straight line drawing of the graph G is a triangulation of $X \cup \{P\}$.*

In general, a feasibility set $F(X, \Pi)$ is comprised of unions of chambers (together with some boundary segments or rays) of the line arrangement formed by extending the segments of Π between pairs of points of X into lines. Thus, each feasibility set can be described via linear inequalities. Feasibility sets need not be connected, as can be shown by taking X to be a set of three non-collinear points as in Figure 4.

The intersections of feasibility sets with k-ellipses are the basic building blocks of Steiner reducing sets. In the next section we will explore the geometry and topology which occurs as we take unions of such intersections.

3 Topological Properties of Steiner Reducing Sets

3.1 Connectivity

Our first contribution to the study of the topology of Steiner reducing sets is to show that the sets need not be connected. This can be demonstrated with a point set with as few as five points, as shown in Figure 5.

Theorem 2 *There exist sets X of $5n$ points such that the rank of $H_0\big(St(X)\big)$ is at least $2n$.*

Proof. Let $Z = \{(0,0), (2,1), (8,1), (10,0), (5,18)\}$, denoted by A, B, C, D, E, respectively. We will prove that a quadrilateral of non-Steiner reducing points encircles the component of the Steiner reducing set of Z

that lies in the interior of the convex hull. Note first that edges AB, BC, CD, BE and CE are all unavoidable in a triangulation of Z since no other possible triangulation edges intersect these. It follows that Z has exactly two triangulations, and both are minimal. Choose $\mathrm{MWT}(Z)$ to be the minimum weight triangulation of Z that uses edge AC.

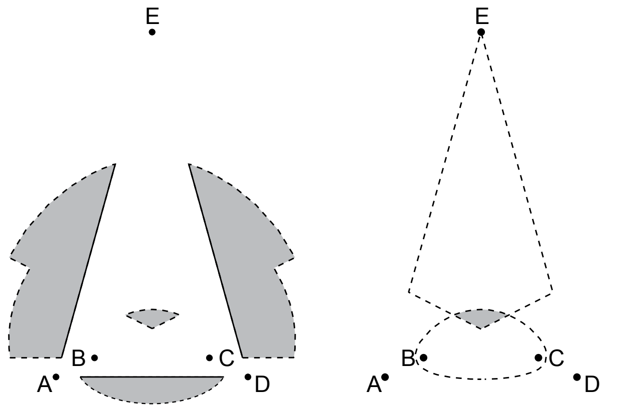


Figure 5: At left, a Steiner reducing set with four connected components; at right, the boundary of the interior component is formed by intersecting a feasibility set and a 5-ellipse

Since the minimum weight triangulation of Z contains the same configuration of edges as the minimum weight triangulation of Y from Figure 2, we know that certain points on or below segment AD will be Steiner reducing points. However, a quick calculation shows $P = (5, 3)$ is also a Steiner reducing point.

Let S_P be the connected component of the Steiner reducing set that contains P. Then $\mathsf{S}_P = E_<(Z; 2\sqrt{298}) \cap F(Z, \Phi)$, where $2\sqrt{298}$ is the length of the edges replaced from $\mathrm{MWT}(Z)$, and the feasibility set is the collection of points within the convex hull that can be connected to all five points of Z without intersecting any edges of a minimal triangulation of $Y = \{A, B, C, D\}$.

In order to establish that the subset S_P of the Steiner reducing set is not path connected to $St(Y)$, consider quadrilateral $EFGH$, where $F = \overleftrightarrow{BE} \cap \overleftrightarrow{CD}, G = \overleftrightarrow{AB} \cap \overleftrightarrow{CD}$, and $H = \overleftrightarrow{CE} \cap \overleftrightarrow{AB}$. This shape surrounds S_P, and its edges contain no Steiner reducing points, a fact which we now prove. To verify that no points of FG or GH are Steiner reducing points, we consider without loss of generality points $R = (x, \frac{1}{2}x)$ within the convex hull of Z and for $x \geq 5$. Such points R lie on the ray $\overrightarrow{GH}$, which makes edges BE, AB, BC, and CD all unavoidable in any triangulation of $Z \cup \{R\}$. Since the triangle inequality implies $|CR| + |ER| \geq |CE|$, the point R is not a Steiner reducing point. A similar argument shows that no point Q on segments FE or EH will be a Steiner reducing point. Therefore there are at least two connected components in $St(Z)$.

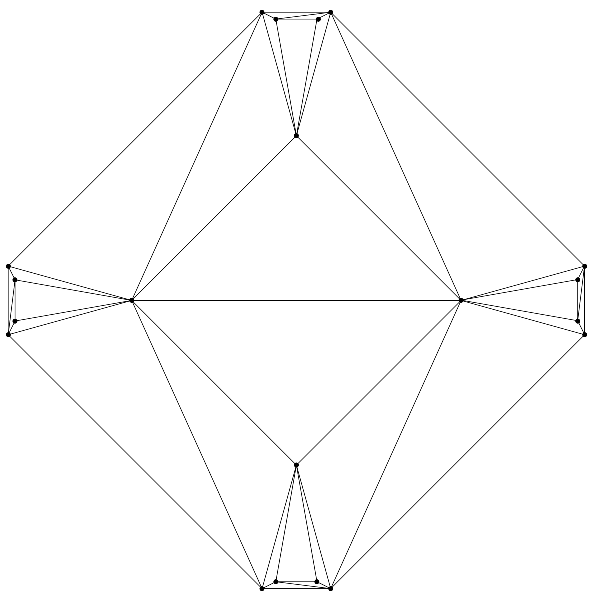


Figure 6: $St\left(Z^{[4]}\right)$ has at least 12 connected components

To prove that $St(Z)$ has four connected components, we verify that the three components outside the convex hull in Figure 5 are not path connected to one another. This can be done, as above, by showing no points of lines $y = 1/2$, $y = 16$ are Steiner reducing points.

By arranging four copies of the point set Z as shown in Figure 6, we can construct a point set with a Steiner reducing set that has at least 8 connected components. The left and right exterior components of the Steiner reducing set from each individual copy of Z are lost, but the the other two components from each copy of Z remain. For this particular example, the midpoint of each boundary edge is a Steiner reducing point, each belonging to its own connected component. By arranging copies of Z so that the images of E from the original set lie on the vertices of an n-gon with sufficiently long edges (relative to the size of Z), we can get a point set denoted $Z^{[n]}$ whose minimum weight triangulation contains all edges of the minimum weight triangulation of each copy of Z. Thus, $X = Z^{[n]}$ is a point set with $5n$ points that has at least $2n$ connected components in its Steiner reducing set. □

In the previous example, we had $Y \subseteq Z$ and the corresponding Steiner reducing sets $St(Y) \subseteq St(Z)$. We note that in general, $X \subseteq W$ does not imply either $\mathrm{MWT}(X) \subseteq \mathrm{MWT}(W)$ or $St(X) \subseteq St(W)$. Even if we have containment of both the point sets $X \subseteq W$ and the edge sets $\mathrm{MWT}(X) \subseteq \mathrm{MWT}(W)$, we may not have $St(X) \subseteq St(W)$. We further note that it is possible for the number of connected components of the Steiner reducing set to exceed the number of points in the set.

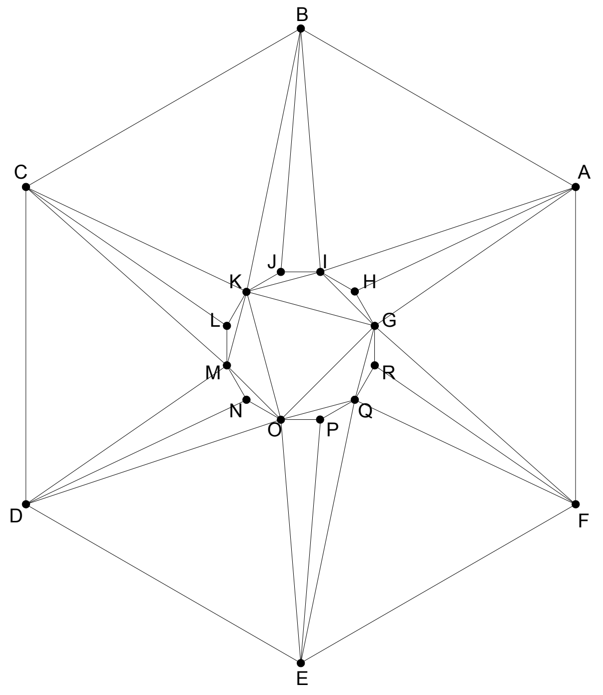


Figure 7: A minimum weight triangulation of Y using the edge set $\mathrm{MWT}(Y)$

An example of a 15-point set X that admits a Steiner reducing set $St(X)$ such that $H_0\big(St(X)\big)$ has rank at least 20 is given in the author's Ph.D. thesis [17].

3.2 Simple Connectivity

Our second topological result is to show that Steiner reducing sets need not be simply connected.

Theorem 3 *There exists a set Y of 18 points such that the rank of $H_1\big(St(Y)\big)$ is at least 13.*

We first describe the structure of a minimum weight triangulation of the point set Y under consideration, then find a connected subset of the Steiner reducing set of Y. We prove this subset is not simply connected by demonstrating 13 curves that lie in the Steiner reducing set of Y and are representatives of linearly independent homology classes within $H_1(St(Y))$.

Proof. Let $Y = G_6 \cup G_{12}$, where

$$G_6 = \left\{ \left(83\cos\frac{\sigma_j}{12}, 83\sin\frac{\sigma_j}{12} \right) \middle| j = 1, \ldots, 6 \right\}, \text{ and}$$

$$G_{12} = \left\{ \left(20\cos\frac{\sigma_j}{24}, 20\sin\frac{\sigma_j}{24} \right) \middle| j = 1, \ldots, 12 \right\},$$

where $\sigma_j = 2\pi(2j-1)$. An illustration of Y together with a minimal weight triangulation is given in Figure 7.

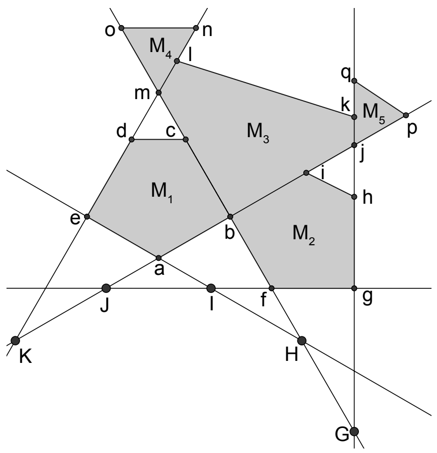


Figure 8: Five subsets M_i of the Steiner reducing set of Y

The set Y is preserved under the standard group action of D_6, the dihedral group of order 12. Let Γ be the orbit of AG under the induced action of D_6 on the edges. The edges of the interior 12-gon formed from the points in G_{12} belong to the β-skeleton of Y, and thus will belong to every minimum weight triangulation of Y. The edges in $\Gamma \cup \{AI, BK, CM, DO, EQ, FG\}$ triangulate the region between the convex hulls of G_6 and G_{12}. No other subset of 18 edges which lie in the annular region between G_6 and G_{12} has smaller weight, so these edges belong to a minimum weight triangulation of Y. Denote by $\mathrm{MWT}(Y)$ a fixed minimum weight triangulation of Y which uses these 18 edges.

Let $\mathcal{H}$ be the line arrangement formed by extending into lines the segments that form the boundaries of $\mathrm{conv}(G_6)$ and $\mathrm{conv}(G_{12})$. In Figure 8 we illustrate five polygons $M_i, 1 \leq i \leq 5$, whose interiors lie in specific chambers of this line arrangement and belong to the Steiner reducing set $St(Y)$. The chamber will determine the available triangulation edges. For example, if a Steiner point Z is added in chamber associated with M_2, Z can be adjacent to any of G, H, I, or J, but cannot be adjacent to K since the edges of the interior 12-gon still belong to a minimum weight triangulation of the augmented point set. We further simplify our search for the Steiner reducing set by utilizing the convexity of k-ellipses. Namely, if $W \subseteq Y$ is a set of d points whose convex hull lies inside a specific feasibility region, and if the points of W all lie in a region bounded by an appropriate k-ellipse, then $\mathrm{conv}(W) \subseteq St(Y)$. By checking the weight of the proposed triangulation at the vertices of the polygons M_i, we use convexity to determine that the interior of each M_i is indeed a subset of a corresponding k-ellipse, and build the Steiner reducing set by verifying minimal triangulations for points that fall on chamber boundary lines.

To finish the proof of Theorem 3, we prove the existence of 13 holes within the Steiner reducing set $St(Y)$. To do so, we find generators of linearly independent homology classes in $H_1(St(Y))$. The technique requires finding 13 points in the interior of $\text{conv}(St(Y))$ that are not Steiner reducing points, together with 13 closed curves γ_i such that $\gamma_i \subset St(Y)$ for $1 \leq i \leq 13$ and each bounds a compact subset of $\mathbb{R}^2$ that contains a point which is not a Steiner reducing point.

None of the points $(0, 35)$, $(18, 32)$, or $(0, 0)$, as well as rotations of these by multiples of $\frac{\pi}{3}$ are Steiner reducing points. It is not difficult to use sets M_i to construct three generators $\gamma_1, \gamma_2, \gamma_3$ of linearly independent homology classes in $H_1(St(Y))$ such that $(0, 35)$ lies interior to curve γ_1, $(18, 32)$ lies interior to γ_2, and $(0, 0)$ lies interior to γ_3. It follows that no path homotopy exists within the set $St(Y)$ from γ_i to γ_j, $i \neq j$, so the holes detected are distinct. Define γ_i for $4 \leq i \leq 13$ to be the distinct images of γ_1 and γ_2 under rotations about the origin by integer multiples of $\pi/3$ radians. $\square$

As in the proof of the first theorem, by aligning copies of Y with the vertices of an n-gon with sufficiently long edges (relative to the size of Y), we can get a point set $Y^{[n]}$ whose minimum weight triangulation contains all edges of the minimum weight triangulation of each copy of Y, thus giving a point set with $18n$ points that has at least $13n$ holes.

4 Future Work

These results have implications for the design of algorithms to search for minimum weight Steiner triangulations. Questions remain about whether two points can be better than one: in a case when the Steiner reducing set $St(X)$ is empty, is it possible that two Steiner points P and Q can team up to cause the weight of a minimal weight triangulation of $X \cup \{P, Q\}$ to be less than that of X? It may be possible to use the structures of the k-ellipse and feasibility sets to design faster algorithms to find the minimal weight triangulation of certain classes of point sets in an efficient manner.

References

[1] M. Bern and D. Eppstein. Mesh generation and optimal triangulation. In *Computing in Euclidean geometry*, volume 1 of *Lecture Notes Ser. Comput.*, pages 23–90. World Sci. Publishing, River Edge, NJ, 1992.

[2] S.-W. Cheng and Y.-F. Xu. On β-skeleton as a subgraph of the minimum weight triangulation. *Theoret. Comput. Sci.*, 262(1-2):459–471, 2001.

[3] J. A. De Loera, J. Rambau, and F. Santos. *Triangulations: Structures for Algorithms and Applications*, volume 25 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin, 2010.

[4] D. Eppstein. Approximating the minimum weight Steiner triangulation. *Discrete Comput. Geom.*, 11(2):163–191, 1994.

[5] P. Erdős and I. Vincze. On the approximation of convex, closed plane curves by multifocal ellipses. *J. Appl. Probab.*, (Special Vol. 19A):89–96, 1982.

[6] M. R. Garey and D. S. Johnson. *Computers and intractability: A guide to the theory of NP-completeness.* W. H. Freeman and Co., San Francisco, 1979.

[7] P. Gilbert. New results in planar triangulations. Master's thesis, University of Illinois, 1979.

[8] C. Groß and T.-K. Strempel. On generalizations of conics and on a generalization of the Fermat-Torricelli problem. *Amer. Math. Monthly*, 105(8):732–743, 1998.

[9] A. Hatcher. *Algebraic topology.* Cambridge University Press, Cambridge, 2002.

[10] J. M. Keil. Computing a subgraph of the minimum weight triangulation. *Comput. Geom.*, 4(1):13–26, 1994.

[11] G. T. Klincsek. Minimal triangulations of polygonal domains. *Ann. Discrete Math.*, 9:121–123, 1980.

[12] W. Mulzer and G. Rote. Minimum weight triangulation is NP-hard. In *Computational geometry (SCG'06)*, pages 1–10. ACM, New York, 2006.

[13] J. Nie, P. A. Parrilo, and B. Sturmfels. Semidefinite representation of the k-ellipse. In *Algorithms in algebraic geometry*, volume 146 of *IMA Vol. Math. Appl.*, pages 117–132. Springer, New York, 2008.

[14] P. V. Sahadevan. The theory of the egglipse—a new curve with three focal points. *Internat. J. Math. Ed. Sci. Tech.*, 18(1):29–39, 1987.

[15] J. Sekino. n-ellipses and the minimum distance sum problem. *Amer. Math. Monthly*, 106(3):193–202, 1999.

[16] G. Sz.-Nagy. Tschirnhaus'sche Eiflächen und Eikurven. *Acta Math. Acad. Sci. Hungar.*, 1:36–45, 1950.

[17] C. Traub. *Topological Effects Related to Minimum Weight Steiner Triangulations.* PhD thesis, Washington University in St. Louis, 2006.

[18] E. W. v. Tschirnhaus. *Medicina mentis, sive artis inveniendi praecepta generalia.* J. Thomann Fritsch, Leipzig, 1695.

[19] A. Varga and C. Vincze. On a lower and upper bound for the curvature of ellipses with more than two foci. *Expo. Math.*, 26(1):55–77, 2008.

On the Space-Efficiency of the "Ultimate Planar Convex Hull Algorithm"

Jan Vahrenhold*

Abstract

The output-sensitive "ultimate planar convex hull algorithm" of Kirkpatrick and Seidel [16] recently has been shown by Afshani *et al.* [1] to be *instance-optimal.* We revisit this algorithm with a focus on space-efficiency and prove that it can be implemented as an in-place algorithm, i.e., using $\mathcal{O}(1)$ working space.

1 Introduction

How much working space does an algorithm require? This question may be asked on its own accord, but there are important practical implications as well. In so-called *resource-constrained* systems, at least one of the resources needed for working on a problem instance is limited by practical constraints and thus scarce relative to the size of the problem instance. Examples include sensors or smartphones where memory and energy are limited, but also workstations where the main memory is scarce relative to terabyte-sized or larger data sets.

In this note, we are working in the space-efficient model of computation, where the primary objective is to analyze the space an algorithm needs in addition to representing the input. An algorithm is said to be *in situ* if it requires $\mathcal{O}(\log N)$ extra words of memory, and it is said to be *in-place* if the extra space requirement is $\mathcal{O}(1)$ words. Since the input elements must not be destroyed or modified, an in-place algorithm can be seen as permuting the input such that it represents the output. As usual, we assume that a word of memory is capable of representing an input element or an index, thus, when measured in bits, these space bounds translate to $\mathcal{O}(\log^2 N)$ bits and $\mathcal{O}(\log N)$ bits.

The study of space-efficient algorithm goes back to fundamental one-dimensional problems such as sorting, merging, and selecting [12, 17, 21]; in the past decade, it has been extended to problems for point sets in two and three dimensions [4, 5, 6, 7, 8, 10, 13, 14, 19, 20]. Recently, also problems for polygons and special classes of graphs have been investigated in the space-efficient model of computation [2, 3].

*Department of Computer Science, Westfälische Wilhelms-Universität Münster, `jan.vahrenhold@uni-muenster.de`. Part of this work has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis" (`http://sfb876.tu-dortmund.de`), project A2.

Among the above results, two are of particular importance for our work: Brönnimann *et al.* [8] investigated the space-efficiency of planar convex hull algorithms, i.e., the complexity of computing the H-element convex hull of a set of N points in two dimensions. The authors proved that both the "ultimate" algorithm by Kirkpatrick and Seidel [16] and its simplified variant by Chan, Snoeyink, and Yap [11] can be implemented as *in situ* algorithms, i.e., using $\mathcal{O}(\log N)$ extra words of space, while maintaining an output-sensitive, optimal running time of $\mathcal{O}(N \log H)$. Since, however, not only Graham's optimal algorithm [15] but also Chan's optimal, output-sensitive algorithm [9] was shown be implementable as an in-place algorithm, Brönnimann *et al.* [8] conjectured Chan's algorithm to be the "more ultimate" planar convex hull algorithm. The second relevant result was proved by Bose *et al.* [6]; we developed a framework for simulating a balanced divide-and-conquer scheme using only $\mathcal{O}(1)$ working space. In particular, we showed how such a recursive scheme can be implemented using a constant-sized (in terms of words) stack and, among other results, developed linear-time, in-place algorithms for selecting, un-selecting, and k-selection in sorted arrays.

The optimal, output-sensitive algorithms by Kirkpatrick and Seidel [16] and by Chan, Snoeyink, and Yap [11] recently have been shown by Afshani *et al.* [1] to be *instance-optimal* in the sense that the maximum running time (over all possible permutations of the input) of the algorithm does not differ by more than a constant factor from the minimum of the average running times (again, over all possible permutations of the input) over all algorithms that solve this problem. Afshani *et al.* pointed out that such an instance-optimal algorithm is "immediately also competitive against *randomized* (Las Vegas) algorithms" [1, p. 130]. In this note, we prove:

Theorem 1 *The deterministic, time-optimal, output-sensitive, and instance-optimal planar convex hull algorithm by Kirkpatrick and Seidel can be realized as an in-place algorithm, i.e., using $\mathcal{O}(1)$ working space.*

Since Chan's algorithm is known to be time-optimal, output-sensitive, and implementable as an in-place algorithm but not known to be instance-optimal, the above theorem improves the state-of-the-art of characterizing the "ultimate planar convex hull algorithm" in favor of Kirkpatrick and Seidel's algorithm.

2 The Algorithm by Kirkpatrick and Seidel

The convex hull algorithm by Kirkpatrick and Seidel [16] uses a divide-and-conquer approach where, just like in the quicksort algorithm, the bulk of the work is done prior to recursion. To compute the upper hull of a given point set, the algorithm first finds a "bridge", i.e., a hull edge crossing the median x-coordinate of the point set, removes all points in the slab spanned by the endpoints of the bridge, and recurses on the remaining non-trivial point sets. The lower hull is computed analogously. Assuming that the bridge can be found in linear time, Kirkpatrick and Seidel upper-bound the running time $f(N, H)$ for computing the H-element convex hull of an N-element point set as follows [16, p. 290]:

$$f(N,H) \le \begin{cases} cn & \text{if } H = 2 \\ cn + \max\limits_{H_\ell + H_r = H} \left\{ f\left(\frac{N}{2}, H_\ell\right) + f\left(\frac{N}{2}, H_r\right) \right\} & \text{if } H > 2 \end{cases}$$

Here, H_ℓ and H_r denote the number of hull points left and not to the left of the median x-coordinate. Solving this equation yields an upper bound of $\mathcal{O}(N \log H)$.

To determine the bridge (or, rather, its endpoints) in linear time, the authors first determine the median x-coordinate using a k-selection algorithm. They then consider pairs of points and again use a k-selection algorithm to find a pair of points inducing a line with median slope. If this line cannot be shown to induce the bridge, its slope is used to prune away a constant fraction of the points, and the algorithm recurses on the remaining points.[1] Each invocation of this algorithm runs in linear time as long as both k-selection and the pruning procedure can be executed in linear time.

Special care must be taken to handle pairs of points inducing lines with infinite slope (in this case, the pair is excluded from the k-selection algorithm, and the lower point is pruned immediately). Also, to ensure instance-optimality, Afshani *et al.* [1] require that all points strictly below the line through the leftmost and rightmost point of the point set need to be pruned prior to determining the median x-coordinate.

3 Space-Efficient Building Blocks

As we will see, there are three building blocks that need to be made in-place: selecting a subset of the input, finding the k-th element according to some order, and running a divide-and-conquer algorithm. Previously, we gave linear-time algorithms for the first two tasks [6]. For the sake of self-containedness, we present the pseudocode for the selection task:

[1]There is a simplified version of this algorithm using randomized 2D linear programming [8, 18]. It is unclear, however, if instance-optimality can be shown for the resulting convex hull algorithm and, more important, whether such a characterization is meaningful for a randomized algorithm.

Algorithm 1 SUBSETSELECTION(A, b, e, f): selecting a subset from an array $A[b, \ldots, e-1]$ using a $(0,1)$-valued function f that can be evaluated in constant time [6]. If A is sorted, there is a linear-time inverse oblivious of f.

```
Ensure: $A[b, \ldots, i-1]$ contains all elements of
    $A[b, \ldots, e-1]$ for which $f$ evaluates to one.
1: $i \leftarrow b$, $j \leftarrow b$ and $m \leftarrow b + 1$.
2: while $i < e$ and $j < e$ do
3:     while $i < e$ and $f(A[i]) = 1$ do
4:         $i \leftarrow i + 1$.    ▷ Move $i$ such that $f(A[i]) = 0$.
5:     $j \leftarrow \max\{i + 1, j + 1\}$;
6:     while $j < e$ and $f(A[j]) = 0$ do
7:         $j \leftarrow j + 1$.    ▷ Move $j$ such that $f(A[j]) = 1$.
8:     if $j < e$ then
9:         swap $A[i] \leftrightarrow A[j]$.
10: Return $i$.
```

Also, we discussed how to use a bit stack of $\mathcal{O}(\log n)$ bits, i.e., $\mathcal{O}(1)$ words, to implement the following template for the case of an almost perfectly balanced divide-and-conquer, i.e., for the case that the size of the "left" part of the recursion always is a power of two.

Algorithm 2 RECURSIVE(A, b, e): Standard template for recursive divide-and-conquer algorithms [6].

```
1: if $e - b \le 2^{h_0}$ (=size of the recursion base) then
2:     BASECODE$(A, b, e)$          ▷ Solve small instances
3: else
4:     PRECODE$(A, b, e)$
         ▷ Setup Subproblem 1 in $A[b, \ldots, \lfloor (b+e)/2 \rfloor - 1]$
5:     RECURSIVE$(A, b, \lfloor (b+e)/2 \rfloor)$       ▷ Recurse left
6:     MIDCODE$(A, b, e)$
         ▷ Setup Subproblem 2 in $A[\lfloor (b+e)/2 \rfloor, \ldots, e-1]$
7:     RECURSIVE$(A, \lfloor (b+e)/2 \rfloor, e)$       ▷ Recurse right
8:     POSTCODE$(A, b, e)$
         ▷ Merge Subproblems 1 and 2 in $A[b, \ldots, e-1]$
```

While, in most situations, requiring an almost perfectly balanced partition is not a constraint for divide-and-conquer algorithms, such a partition cannot be guaranteed for Kirkpatrick and Seidel's algorithm which, due to several pruning steps, does not balance the sizes of the subproblems effectively handled in the recursive calls. Thus, the central algorithmic problem we need to address is how to recover the original values of b and e after returning from a call to RECURSIVE, i.e., prior to calling MIDCODE and POSTCODE, using globally no more than $\mathcal{O}(1)$ working space.

4 An Implementation With $\mathcal{O}(1)$ Working Space

In this section, we show how to implement the algorithm by Kirkpatrick and Seidel using $\mathcal{O}(1)$ working space. To

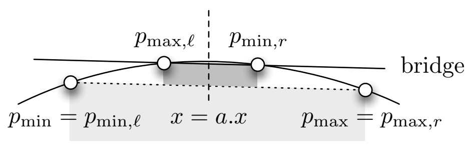


Figure 1: Extremal hull points.

facilitate the exposition, we first describe how to adapt the general divide-and-conquer template (Algorithm 2) to deal with unbalanced recursive calls (Section 4.1). We then show how to represent and combine the output of the recursive calls (Section 4.2). Finally, we present a linear-time, in-place algorithm for finding the bridge needed for the divide-step (Section 4.3).

4.1 Adapting the Divide-and-Conquer Template

Using the terminology of the preceding section, the algorithm by Kirkpatrick and Seidel will be run on an array $A[0, \ldots, n-1]$ where in each recursive step, i.e., for parameters b and e, we first invoke PreCode to do the following (see Figure 1):

1. Identify the extremal hull points $p_{\min}$ and $p_{\max}$.
2. Prune away all points strictly below $\overline{p_{\min}p_{\max}}$, i.e., in the light gray area in Figure 1 (this guarantees instance-optimality [1, Section 3.1]).
3. Determine the point a with median x-coordinate among the remaining points.
4. Compute the bridge, i.e., the segment induced by the maximal[2] hull point $p_{\max,\ell}$ to the left of a and the minimal hull point $p_{\min,r}$ not to the left of a.
5. Prune away all points below the bridge, i.e., in the dark gray area in Figure 1.
6. Adjust the indices b and e such that $A[b, \ldots, e-1]$ contains all unpruned points not to right of $p_{\max,\ell}$.

After returning from the recursive call processing the unpruned points, we need to call MidCode with the *original* values of b and e, i.e., with the same values that PreCode had been called with. Similarly, after returning from the second recursive call, these original values need to be passed to PostCode. Unlike in the standard divide-and-conquer template (Algorithm 2) we cannot simply assume that number of points passed to a recursive call is exactly half the number of points originally passed to the invoking method; in fact, the central point that allows for proving the optimal $\mathcal{O}(N \log H)$ time complexity is that this is *not* the case.

It turns out, however, that three simple invariants allow for reconstructing these values in time $\mathcal{O}(e-b)$:

Invariant (A): Prior to executing and after having executed Recursive(A, b, e), the two vertices of the upper hull that have extremal coordinates in $A[b, \ldots, e-1]$ are stored in $A[b]$ and $A[b+1]$.

[2]As usual, points lying on a hull edge, i.e., points in degenerate position, are not considered to be hull points.

Invariant (B): The points passed to Recursive(A, b, e) are exactly the points in $A[0, \ldots, n-1]$ that lie between $A[b]$ and $A[b+1]$ as characterized in Invariant (A).

Invariant (C): After having executed Recursive(A, b, e), the values of b and e have been restored.

Using one linear scan and two swap operations, Invariant (A) can be trivially established prior to invoking Recursive(A, 0, n). Since the points with extremal coordinates in $A[0, \ldots, n-1]$ are also the extremal hull vertices, Invariant (B) holds as well. It is also easy to realize that all invariants can be guaranteed to be maintained using $\mathcal{O}(1)$ time and working space for constant-sized recursion base cases.

Lemma 2 *If Invariant (A) holds prior to invoking* PreCode*, this method can be implemented as a linear-time method with $\mathcal{O}(1)$ working space such that Invariant (A) also holds prior to invoking* Recursive *for the "left" recursion.*

Proof. We consider each step of PreCode (as described above) in turn. Since Invariant (A) holds prior to invoking PreCode, Step 1 (identification of $p_{\min}$ and $p_{\max}$) trivially is a constant-time operation. Step 2 (pruning) can be implemented using SubsetSelection(A, b, e, f) where $f(p) = 0$ holds iff p is strictly below $\overline{A[b]A[b+1]}$. Since this algorithm moves the pruned elements to $A[e', \ldots, e-1]$, the resulting subarray looks as follows.

	$p_{\min}$	$p_{\max}$	unpruned	pruned	
	b			e'	e

Using $\mathcal{O}(1)$ working space (among other things, to keep track of the location of $p_{\min}$ and $p_{\max}$ to eventually restore them to $A[b]$ and $A[b+1]$), we run a linear-time, in-place k-selection algorithm [6, 17] to find the point a with median x-coordinate in $A[b, \ldots, e'-1]$. We then run an in-place version of the bridge-finding algorithm (see Section 4.3) to determine the two hull vertices $p_{\max,\ell}$ and $p_{\min,r}$ defining the bridge.

To prepare for the next recursive call, we move $p_{\max,\ell}$ to $A[b+2]$ and swap the two points in $A[b]$, i.e., $p_{\min} = p_{\min,\ell}$, and $A[b+1]$, i.e., $p_{\max}$. We then increment b by one and run SubsetSelection(A, b, e', f) where $f(p) = 1$ holds iff either $p.x < p_{\max,\ell}.x$ or $p = p_{\max,\ell}$ to prune all points not to the right of $p_{\max,\ell}$.

	$p_{\max}$	$p_{\min,\ell}$	$p_{\max,\ell}$	unpruned	pruned	
		b			e''	e'

Finally, we set $e := e''$ (the index returned by SubsetSelection) and are ready to recurse on $A[b, \ldots, e-1]$. Since, by construction, Invariants (A) and (B) hold for this call and since all steps take linear time and require $\mathcal{O}(1)$ working space, the lemma follows. □

Storing $p_{\max} = p_{\max,r}$ in front of the subarray passed to the first recursive call allows us to easily recover the indices b and e needed for invoking MIDCODE; in a slight abuse of notation, this method is invoked with the indices b and e passed to RECURSIVE (these indices are available by Invariant (C)) and does the following:

1. Recover the value of e passed to PRECODE.
2. Recover the right endpoint $p_{\min,r}$ of the bridge.
3. Select all points in $A[b, \ldots, e-1]$ that lie between $p_{\min,r}$ and $p_{\max,r}$.
4. Establish Invariant (A).

Lemma 3 *If Invariants (A) and (C) hold prior to invoking* MIDCODE *and if Invariant (B) held prior to the preceding call to* RECURSIVE*, the* MIDCODE *method can be implemented as a linear-time method with $\mathcal{O}(1)$ working space such that Invariants (A), (B), and (C) hold prior to invoking* RECURSIVE *for the "right" recursion.*

Proof. To recover the (original) value of e that was passed to PRECODE, we scan forward from $A[e]$, i.e., the first item not passed to the preceding recursive call. By Invariant (C), we know that the index of the first element in $A[e, \ldots, n-1]$ with an x-coordinate larger than $A[b-1] = p_{\max}$ (or n if no such element exists) is the value of e we are looking for. This scan takes linear time and uses $\mathcal{O}(1)$ working space. We then decrement b by one to include the element $p_{\max} = p_{\max,r}$.

To recover the right endpoint $p_{\min,r}$ of the bridge, we exploit the fact that the bridge lies on the unique line passing through $p_{\max,\ell}$ and a point to the right of $p_{\max,\ell}$ such that no point in $A[b, \ldots, e-1]$ lies above this line.[3] A simple proof by contradiction shows that this other bridge point indeed lies to the right of the point a used for originally splitting the point set.

Now that $p_{\min,r}$ and $p_{\max,r}$ have been recovered, we use the same techniques as in PRECODE to save the point $p_{\min}$, to select the points to be passed to the recursive call, to move $p_{\min,r}$ and $p_{\max,r}$ to the front of the subarray, and to adjust the index b accordingly. The resulting array then looks as follows:

	$p_{\min}$	$p_{\min,r}$	$p_{\max,r}$	unpruned	pruned	
		b			e''	e'

Finally, we set $e := e''$ (the index returned by SUBSETSELECTION) and are ready to recurse on $A[b, \ldots, e-1]$. Since, by construction, Invariants (A) and (B) hold for this call and since all steps take linear time and require $\mathcal{O}(1)$ working space, the lemma follows. □

After the second call to RECURSIVE, the call to POSTCODE is used to establish Invariants (A) and (C) for the invoking call to RECURSIVE. In the light of this, POSTCODE needs to perform the following steps:

1. Recover the value of e passed to MIDCODE.
2. Recover the left endpoint $p_{\max,\ell}$ of the bridge.
3. Establish Invariant (A).

Lemma 4 *If Invariants (A) and (C) hold prior invoking* POSTCODE *and if Invariant (B) held prior to the preceding call to* RECURSIVE*, the* POSTCODE *method can be implemented as a linear-time method with $\mathcal{O}(1)$ working space such that Invariants (A) and (C) hold.*

Proof. We proceed as in the proof of Lemma 3, i.e., we first recover the value of e that was passed to MIDCODE by checking boundary conditions w.r.t. $p_{\max,r} = A[b+2]$, adjusting b, and recovering the left bridge point. Since the value of b remained the same over the invocations of all relevant methods and since the value of e was recovered after each such invocation, Invariant (C) is established. Using a linear scan and two swap operations, Invariant (A) can be established as well. All algorithms run in linear time and use $\mathcal{O}(1)$ working space. □

4.2 Representing the Output

The description of the algorithm so far only focused on ensuring that the "boundaries" of the recursive calls can be recovered efficiently. In this subsection, we discuss how to represent the output from a call to RECURSIVE. For this, we establish a fourth invariant:

Invariant (D): After a call to RECURSIVE(A, b, e), the upper convex hull vertices (if any) between $A[b]$ and $A[b+1]$ (see Invariants (A) and (B)) are stored in increasing x-order starting at $A[b+2]$.

Obviously, this invariant can be established trivially for constant-sized recursion base cases.

Lemma 5 *If Invariants (A) and (D) hold after a call to* RECURSIVE*(A, b, e), and if Invariant (B) held prior to the preceding call to* RECURSIVE*, the upper convex hull computed during this recursive call can be recovered based upon the knowledge of b only.*

Proof. This proof exploits the fact that vertices on the upper convex hull form right turns when traversed in increasing x-order. The recovery algorithm first checks whether $A[b+2]$[4] lies strictly above the segment $\overline{A[b]A[b+1]}$. If this is not the case, the upper convex hull consists of $A[b]$ and $A[b+1]$ only, and we are done. Otherwise, the algorithm scans forward from $i = b+3$ until $A[i]$ lies right of $A[b+1]$ (in this case $i = e$, and we are done), $A[i]$ does not lie right of $A[i-1]$, or $A[i-1]$, $A[i]$, and $A[b+1]$ do not form a right turn (in the last two cases, $A[i-1]$ is the last hull vertex). □

[3]This line may not be determined by a unique point, namely if more than three points are allowed to be collinear. In this case, the rightmost of these points is the recovered bridge point since this choice minimizes the number of points passed to the next recursive call (see also Footnote 2 and Section 4.3).

[4]For the sake of simplicity, we assume that the size of the recursion base case is larger than two.

Lemma 5 implies that the upper convex hull can be reconstructed in linear time and $\mathcal{O}(1)$ working space after having returned from Recursive(A, 0, n): Scan forward from $A[0]$ to recover the index H, i.e., the number of points on the upper hull, and stably exchange $A[1](= p_{\max})$ and $A[2, \ldots, H-1]$.

We now show how to maintain Invariant (D) throughout the algorithm. Inductively assume that, after the "left" call to Recursive, we have computed the hull points (denoted by "$\frown_\ell$") between $p_{\min,\ell}$ and $p_{\max,\ell}$. Also, by Invariant (C), we have restored b and e to its original values. Then, the subarray looks as follows:

	$p_{\max}$	$p_{\min,\ell}$	$p_{\max,\ell}$	$\frown_\ell$	$\cdots$	
	b				c	e

By Lemma 5, we know that we can identify the index c such that $A[b{+}3, \ldots, c{-}1]$ stores the "$\frown_\ell$"-points. Using the (folklore) linear-time, in-place algorithm for swapping two adjacent blocks, we then move this subarray to the end of $A[b, \ldots, e-1]$.

	$p_{\max}$	$p_{\min,\ell}$	$p_{\max,\ell}$	$\cdots$	$\frown_\ell$	
	b				e'	e

We then continue as in the proof of Lemma 3. Similarly, after the "right" call to Recursive, the subarray looks as follows (b and e are available by Invariant (C), c and e' are recovered as implied by Lemma 5):

	$p_{\min}$	$p_{\min,r}$	$p_{\max,r}$	$\frown_r$	$\cdots$	$\frown_\ell$	
	b				c	e'	e

Using linear-time, in-place swapping, we rearrange the contents of the subarray such that Invariants (A) and (D) hold, and proceed as in the proof of Lemma 4.

	$p_{\min,\ell}$	$p_{\max,r}$	$\frown_\ell$	$p_{\max,\ell}$	$p_{\min,r}$	$\frown_r$	$\cdots$	
	b							e

Since Invariant (D) can be established for constant-sized recursion base cases in a straightforward way, the above discussion implies that we can maintain Invariant (D) during each execution of Recursive with linear time and $\mathcal{O}(1)$ working space as claimed.

4.3 Finding a Bridge

In the proof of Lemma 2, we assumed that there is a linear-time, in-place algorithm for finding the two endpoints $p_{\max,\ell}$ and $p_{\min,r}$ of the upper hull edge crossing the vertical line $x = a.x$. While we could simply refer to the in-place implementation proposed by Brönnimann *et al.* [8, Theorem 5], we give the details for the sake of self-containedness and to show that this computation does not interfere with maintaining the invariants.

By the discussion in the preceding subsections, we know that in this situation the subarray looks as follows:

	$p_{\min}$	$p_{\max}$	unpruned	pruned	
	b			$e'' := e'$	e

Following Kirkpatrick and Seidel, we form pairs of points and order them by increasing x-coordinate. If the number of points is odd, we use $\mathcal{O}(1)$ space to store the remaining point $\tilde{p}$. Using SubsetSelection we then move all pairs where the two points have the same x-coordinate ("$\uparrow$") to the end of the array.

	$p_{\min}$	$p_{\max}$	$\nearrow$	$\cdots$	$\nearrow$	$\uparrow$	$\cdots$	$\uparrow$	
	b					e'''			e''

From now on, we use $\mathcal{O}(1)$ space to keep track of the position of the points $p_{\max}$ and $p_{\min}$ such that they can be restored to $A[b, b+1]$ (hence establishing Invariant (A)) after the bridge has been found.

To find the bridge, we run a linear-time, in-place k-selection algorithm [6, 17] to determine the pair of points in $A[b, \ldots, e'''-1]$ inducing the line with median slope K. Based upon this slope, we twice run SubsetSelection to partition $A[b, \ldots, e'''-1]$ into three sets of pairs: Small (slope less than K), Equal (slope K), and Large (slope larger than K).

	Small	Equal	Large	$\uparrow$	$\cdots$	$\uparrow$	
	b	c	c'	e'''			e''

Just as in the original algorithm, the two endpoints of the edge with slope K are found by scanning over $A[b, \ldots, e'''-1]$ to find, among all points p maximizing $p.y - K \cdot p.x$, the points with minimum and maximum x-coordinate. This step is easily seen to both take linear time and use $\mathcal{O}(1)$ working space.

If the edge constructed this way crosses the vertical line $x = a.x$, we have found the hull vertices $p_{\max,\ell}$ and $p_{\min,r}$. We record these vertices using $\mathcal{O}(1)$ space and then spend linear time to move $p_{\max}$ and $p_{\min}$ back to $A[b, b+1]$. Finally, we discard the indices c, c', e'', and e''', keep the indices e and e', and resume the algorithm described in the proof of Lemma 2.

If, however, both endpoints of the edge lie to the right of the vertical line $x = a.x$, we need to recurse on all points in Large and the left points of all pairs in Small and Equal *plus* the upper points of all pairs with the same x-coordinate (the case that no endpoint lies to the right of the vertical line is handled symmetrically). To prepare for this (tail) recursion, we first swap Large to the beginning of $A[b, \ldots, e''-1]$ and then use SubsetSelection to select the appropriate point from each remaining pair. If the number of points we started with was odd, we also add the point $\tilde{p}$ to the points to be processed next. To prepare for the next iteration, we again group pairs of points as described above and update the indices e''' and e'' accordingly.

	$\nearrow$	$\cdots$	$\nearrow$	$\uparrow$	$\cdots$	$\uparrow$	$\cdots$	
	b			e'''			e''	

The correctness of this algorithm and its linear runtime follow from the original proofs presented by Kirkpatrick and Seidel. With respect to the space requirement, we observe that, in addition to the constant number of indices used in k-selection algorithms and the calls to SubsetSelection, the iterative algorithm outlined above requires only to maintain a constant number of "global" indices: the original indices b and e, two indices to keep track of $p_{\max}$ and $p_{\min}$, and the index e' denoting the end of the current working set. The other indices, i.e., e'', e''', c, c', and possibly the index to keep track of the "excess" element $\tilde{p}$ are indices local to each iteration and can be discarded at the end of this iteration. In summary, the above discussion implies that we can indeed find a bridge in linear time and using $\mathcal{O}(1)$ working space while maintaining the invariants.

Putting everything together, this establishes a proof of Theorem 1, i.e., we have shown that we can realize the deterministic, time-optimal, output-sensitive, and instance-optimal planar convex hull algorithm by Kirkpatrick and Seidel using $\mathcal{O}(1)$ working space. Thus, we have established one more optimality criterion to hold for the "ultimate planar convex hull algorithm".

Note added in proof An alternative in-place algorithm has been suggested by Raimund Seidel [personal communication]: Viewed holistically, the "marriage-before-conquest-algorithm" maintains an ordered sequence of upper-hull edges and gaps and proceeds always by finding a bridge in the leftmost gap until no gap is left. The suggested alternative approach realizes this using an iterative, non-recursive algorithm which requires each points to be labeled either "extreme", "dead", or "alive". Assuming that the input does not contain duplicates, these labels can be stored implicitly by locally rearranging the input points: consider blocks of consecutive 7 points in the input array; there is a canonical lexicographic order of those points; storing the points in any one of the 7! permutations allows to encode any one of the 3^7 labellings of those points, since $7! > 3^7$. We leave the details to the reader.

References

[1] P. Afshani, J. Barbay, and T. M.-Y. Chan. Instance-optimal geometric algorithms. In *Proc. IEEE Symp. Foundations of Computer Science*, pp. 129–138. 2009.

[2] T. Asano and B. Doerr. Memory-constrained algorithms for shortest path problems. In *Proc. Canadian Conf. Computational Geometry*, pp. 315–318, 2011.

[3] T. Asano, W. Mulzer, G. Rote, and Y. Wang. Constant-work-space algorithms for geometric problems. *Journal of Computational Geometry*, 2(1):46–68, 2011.

[4] H. Blunck and J. Vahrenhold. In-place randomized slope selection. In *Proc. Intl. Conf. on Algorithms and Complexity*, LNCS 3998, pp. 31–40, 2006.

[5] H. Blunck and J. Vahrenhold. In-place algorithms for computing (layers of) maxima. *Algorithmica*, 57(1):1–21, May 2010.

[6] P. Bose, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and J. Vahrenhold. Space-efficient geometric divide-and-conquer algorithms. *Computational Geometry: Theory & Applications*, 37(3):209–227, Aug. 2007.

[7] H. Brönnimann, T. M.-Y. Chan, and E. Y. Chen. Towards in-place geometric algorithms. In *Proc. Symp. Computational Geometry*, pp. 239–246. 2004.

[8] H. Brönnimann, J. Iacono, J. Katajainen, P. Morin, J. Morrison, and G. T. Toussaint. Space-efficient planar convex hull algorithms. *Theoretical Computer Science*, 321(1):25–40, June 2004.

[9] T. M.-Y. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Computational Geometry: Theory and Applications*, 16(14):361–368, Apr. 1996.

[10] T. M.-Y. Chan and E. Y. Chen. Optimal in-place and cache-oblivious algorithms for 3-d convex hulls and 2-d segment intersection. *Computational Geometry: Theory and Applications*, 43(8):636–646, Oct. 2010.

[11] T. M.-Y. Chan, J. S. Snoeyink, and C.-K. Yap. Primal dividing and dual pruning: Output-sensitive construction of four-dimensional polytopes and three-dimensional Voronoi diagrams. *Discrete & Computational Geometry*, 18(4):433–454, Dec. 1997.

[12] J.-C. Chen. Optimizing stable in-place merging. *Theoretical Computer Science*, 302(1–3):191–210, June 2003.

[13] M. De, A. Maheshwari, S. Nandy, and M. Smid. An in-place priority search tree. In *Proc. Canadian Conf. Computational Geometry*, pp. 331–336, 2011.

[14] M. De and S. Nandy. Space-efficient algorithms for empty space recognition among a point set in 2D and 3D. In *Proc. Canadian Conf. Computational Geometry*, pp. 347–353, 2011.

[15] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132–133, June 1972.

[16] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15(1):287–299, Feb. 1986.

[17] T. W. Lai and D. Wood. Implicit selection. In *Proc. Scand. Workshop on Algorithm Theory*, LNCS 318, pp. 14–23, 1988.

[18] R. Seidel. Small-dimensional linear programming and convex hulls made easy. *Discrete & Computational Geometry*, 6(4):423–434, Dec. 1991.

[19] J. Vahrenhold. An in-place algorithm for Klee's measure problem in two dimensions. *Information Processing Letters*, 102:169–174, May 2007.

[20] J. Vahrenhold. Line-segment intersection made in-place. *Computational Geometry: Theory & Applications*, 38(3):213–230, Oct. 2007.

[21] J. W. J. Williams. Algorithm 232: Heapsort. *Communications of the ACM*, 7(6):347–348, June 1964.

Divide-and-Conquer 3D Convex Hulls on the GPU

Jeffrey M. White [*] Kevin A. Wortman [†]

Abstract

We describe a pure divide-and-conquer parallel algorithm for computing 3D convex hulls. We implement that algorithm on GPU hardware, and find a significant speedup over comparable CPU implementations.

1 Introduction

The *3D convex hull problem* is to identify, for a given set of n points in $\mathbb{R}^3$, the minimal set of input points such that the convex envelope of those points contains all input points. The problem is fundamental to computational geometry and has been studied extensively. Several $O(n \log n)$ time algorithms are known, with various trade-offs in constant factors, simplicity, numerical robustness, data structure dependencies, and nondegeneracy requirements (see e.g. [1] [3] [6] [7] [9] [14] [16]). Chan's celebrated output-sensitive algorithm [4] runs in $O(n \log h)$ time, where h denotes the number of faces in the output hull, which is asymptotically optimal.

A *graphics processing unit (GPU)* is a parallel coprocessor available in commodity computers. An outgrowth of the computer gaming industry, GPUs utilize a highly-parallel single instruction multiple data (SIMD) architecture. At a high-level, GPUs work by applying a concise constant-space function called a *kernel* to all elements of an array simultaneously. Kernels are written in *domain specific embedded languages (DSELs)* such as NVIDIA's CUDA [13] or the OpenCL [10] open standard. Each kernel instance is passed an integer *global identifier (id)* which is customarily used to delineate the ranges of input that each kernel invocation applies to. The potential performance, measured in either gigaFLOPS or memory bandwidth, of GPUs is substantially greater than that of multicore CPUs. However, realizing this potential on practical problems, besides the embarrasingly-parallel graphics applications for which GPUs were originally designed, has proven challenging. By and large, existing parallel algorithms depend on facilities, such as message passing and/or synchronization primitives, which are unavailable in the GPU environment. Yet, GPUs are purpose-built for high performance computation on low-dimensional geometric objects, and the opportunity to apply them to computational geometry problems cannot be ignored.

While the 3D convex hull problem has been studied extensively in the standard computational model, precious little past work is applicable to GPU implementations. As stated above, GPU kernels cannot communicate with or synchronize against each other. This limitation rendered unusable every PRAM-model algorithm we surveyed (e.g. [2]). Further, running kernels have no provision for dynamic memory; their collective input and output must be allocated before the kernels execute *en masse* and freed afterward. Accordingly dynamic data structures are off limits. The absence of the doubly connected edge list (DCEL) structure is a particularly formidable obstacle in this context.

There are several results on computing 2D hulls purely on the GPU [8] [15] [17], but results on the more general and complex 3D problem have been elusive. While preparing this manuscript, we became aware of an independent result on the 3D problem [18]. That algorithm uses heuristics to cull many, but not all, interior points on the GPU, then feeds the remaining points to a black-box CPU hull implementation (e.g. QuickHull [3]). Experimental results show that the hybrid approach achieves a speedup factor of 10–46 times [18] on a GPU with approximately 1581 peak gigaFLOPS [11]. The algorithm presented here achieves a speedup of roughly 8 times on a GPU with 54 peak gigaFLOPS [12], while using a pure GPU divide-and-conquer approach. The pure approach is conceptually simple, and its worst case running time is not impacted by the presence of outlier points.

2 Algorithm

Our algorithm is an adaptation of Chan's *minimalist* 3D convex hull algorithm [5]. Note that this $O(n \log n)$-time algorithm is distinct from the $O(n \log h)$-time algorithm mentioned earlier, also authored by Chan. The minimalist algorithm is, by design, a straightforward top-down divide-and-conquer algorithm for computing 3D convex hulls. It was originally motivated by pedagogical needs for an algorithm that achieves a favorable $O(n \log n)$ running time, while being simple to explain and implement and avoiding dependency on difficult data structures or algorithms. Serendipitously these design constraints correspond to those imposed by the GPU.

*Department of Computer Science, California State University, Fullerton, `jeffreymarkwhite@gmail.com`

†Department of Computer Science, California State University, Fullerton, `kwortman@fullerton.edu`

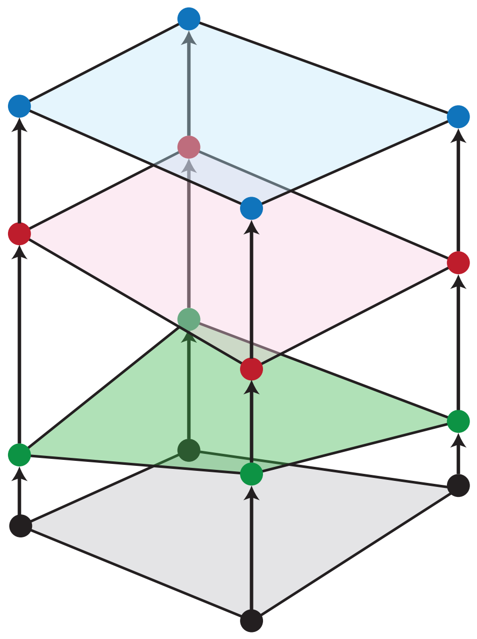


Figure 1: Algorithm Events.

The minimalist algorithm works by recasting the 3D problem as a 2D *kinetic* problem. 3D (x, y, z) points are mapped to $(x, y, \Delta y)$ points with an initial (x, y) starting point and Δy vertical rate of speed. As time t advances, the points move at distinct velocities, which triggers structural changes in the convex hull of the points (see Figure 1). Computing the convex hull of the original 3D points may be visualized as computing a *kinetic movie* of these configurations for all values $-\infty < t < \infty$. The algorithm represents this movie as a chronological sequence of *events* when input points are added to, or removed from, the hull. Input points are presorted by x-coordinate; event sequences for roughly equal-size subsets are generated recursively, then combined by a Graham-scan-like $O(n)$ merging process. In the base case a single point nominates itself as the only convex hull point.

While the minimalist algorithm boasts many of the features necessary for GPU implementation, it cannot be ported to the GPU directly. GPU kernels cannot be recursive, so the top-down divide-and-conquer approach is inappropriate. Instead, the algorithm must be reoriented into one or more mapping steps where an array of input data elements are mapped by a kernel to an array of output data elements. We achieve this reorientation by rewriting the minimalist algorithm to use *bottom-up* divide and conquer. We define a *movie array* data structure as a table of event logs. Our algorithm allocates a single movie array, and initializes one trivial event log for each input point. Then, our algorithm

```
// CPU Algorithm Point
struct Point {
  double x, y, z;
  Point *prev, *next;
  void act() {...}
};

// GPU Algorithm Point
struct Point {
  cl_float x;
  cl_float y;
  cl_float z;
  cl_int prev;
  cl_int next;
};
```

Figure 2: Differences in the Point datatype.

repeats a *merge step* that combines each pair of event logs with adjacent indices into a single event log. A merge step maps a movie array with n logs of length at most l to a new array with at most $\lceil n/2 \rceil$ logs of length at most $2l$ each. Thus, after $\lceil \log_2 n \rceil$ merge steps, the movie array contains a single event log for the entire point set. The key property of this algorithm with respect to GPU computation is that each log merge may be performed entirely independently of the others. Each kernel has a particular range of input movie array indices to read from, and a corresponding range of output indices to write to, and may perform its computation independently of other concurrent kernel instances.

3 Implementation

Our implementation of the GPU algorithm follows the bottom-up divide-and-conquer design as mentioned above. As shown in Figure 3, the point structure in the CPU algorithm uses a doubly linked list connected by pointers. The idea is to divide the sorted list down into trivial subsequences and build the list back up to the desired set of faces on the convex hull. Memory pointers are difficult (though not impossible) to move between the CPU and GPU since the two devices have distinct memory spaces. Also, on the GPU each kernel instance needs to seek to its assigned sub-input based on its global id, which could take $O(n)$ time using a list structure. For these reasons, our GPU implementation uses arrayed lists with integer indices rather than linked lists with node addresses (Figure 3).

Modifying the way data is stored impacts the way data is accessed. Figure 4 shows the differences in `act()` function used for inserting and deleting points from event logs. Figure 5 shows the differences in passing potential faces into the event-time calculations.

The implementation process began with converting the original CPU algorithm to use arrays rather then pointers to represent the data. Point data is imple-

```
// CPU Algorithm list of points
Point *P = new Point[n];
...
// Sorts points into a doubly
// linked list based x-coordinate.
Point *list = sort(P, n);

// event lists
Point **A = new Point *[2*n];
Point **B = new Point *[2*n];


// GPU Algorithm list of points
Point *P = (Point *)
    malloc(n*sizeof(Point));

// event lists
cl_int *A = (cl_int *)
    malloc(2*n*sizeof(cl_int));
cl_int *B = (cl_int *)
    malloc(2*n*sizeof(cl_int));
```

Figure 3: Differences in list creation.

```
// CPU Algorithm act() function call
point->act()

// CPU Algorithm act() function
struct Point {
...
  void act() {
    if (prev->next != this) {
      // insert point
      prev->next = next->prev = this;
    }
    else {
      // delete point
      prev->next = next;
      next->prev = prev;
    }
  }
};

// GPU Algorithm act() function call
act(pointIndex);

// GPU Algorithm act() function
void act(int pointIndex) {
  if (P[P[pointIndex].prev].next
      != pointIndex) {
    // insert point
    P[P[pointIndex].prev].next
    = P[P[pointIndex].next].prev
    = pointIndex;
  }
  else {
    // delete point
    P[P[pointIndex].prev].next
    = P[pointIndex].next;
    P[P[pointIndex].next].prev
    = P[pointIndex].prev;
  }
}
```

Figure 4: Differences in `act()` functions.

```
// CPU Algorithm time[0] calculation
t[0] = time(B[i]->prev,
            B[i],
            B[i]->next);

// GPU Algorithm time[0] calculation
t[0] = time(P[B[i]].prev,
            B[i],
            P[B[i]].next);
```

Figure 5: Differences in time calculations.

```
dataOffsetValue = 2;
totalMergesLeft = numberOfPoints/2;
do {
    numberOfThreads = totalMergesLeft;
    runGPUkernels();
    swap(A, B);
    dataOffsetValue = dataOffsetValue*2;
    totalMergesLeft = totalMergesLeft/2;
} while(totalMergesLeft > 1);
```

Figure 6: Main outer loop ran on the CPU to handle the execution of threads on the GPU.

mented as its own data type with the x, y, and z values along with indices to represent the next and previous pointers to reference other points based on their array index. Also, instead of having two pointer lists, A and B, we have two arrays of indices that reference a master list P of points.

Another significant change we made to the design is the conversion from a top-down design to a bottom-up design. Instead of using recursion, the heart of the algorithm is placed within one **while** loop as shown in Figure 6. Before implementing this routine as OpenCL kernel code, we wrote a simulation to run on the serial CPU to ensure validity of the algorithm. The ultimate goal of writing a simulation is to avoid the troublesome task of debugging GPU kernel code. This simplified the task of converting the simulation code to GPU kernel code and required only minimal modifications.

Figure 6 shows pseudocode for the main outer loop which runs on the CPU. The main loop uses two movie array structures, both of which exist on the GPU. The two structures alternate between serving as the input and output of a merge step. This approach makes it possible to avoid transferring point data between the GPU and CPU inside the loop, which is desirable as that is an expensive operation. The `dataOffsetValue` is used to calculate the location of where the head of the `leftGroupIndex` and `rightGroupIndex` exist on the globally accessed master list of points P as shown in Figure 7. To handle the way the CPU algorithm swaps lists A and B in each divide routine, we swap the kernel arguments of A and B in the `swap(A, B)` function after each iteration of merges. Following the

```
// the index of where the head of the
// left group of the list can be found
// on the globally accessed array
leftGroupIndex
 = global_ID*dataOffsetValue;

// the index of where the head of the
// right group of the list can be found
// on the globally accessed array
rightGroupIndex
 = [leftGroupIndex+((global_ID+1)
   *dataOffsetValue)]/2;

// the index of where the globally
// accessed event list begins for the
// group of merges based on the global_ID
eventListOffset = leftGroupIndex*2;
```

Figure 7: GPU kernel code: how the GPU knows which hulls should be merged and which parts of the global data to access.

`swap(A, B)` function, `dataOffsetValue` is updated to tie into the next set of group index calculations. Finally, `totalMergesLeft` is cut in half to represent the number threads to take place in the next iteration of merges. When `totalMergesLeft` reaches less than 2, the algorithm exits the main **while** loop as there is no pair of hulls left to be merged together; only one hull is left which represents the final solution.

The C++ and OpenCL source code for our implementation is freely available on the web [19].

4 Experimental Results

The GPU algorithm shows significant improvements over the CPU algorithm. Peak performance of the GPU algorithm reaches a roughly 8x speedup over the CPU algorithm (see Figure 10). Figures 8, 9 and 11 summarize the runtime of both algorithms expressed in milliseconds.

The runtime data was collected on a 2009 Apple MacBook Pro running Mac OS X 10.7.4 and OpenCL 1.2. The CPU is an Intel Core 2 Duo with two cores each running at a clock rate of 2.26 gigahertz, and together achieving approximately 13.6 gigaFLOPS according to the LINPACK benchmark tool. The CPU results are for Chan's own C++ implementation of the minimalist algorithm, which runs in a single thread. The test machine's GPU is an NVIDIA GeForce 9400M with 16 stream pipelines running at 450 megahertz, for a manufacturer-claimed throughput of 54 gigaFLOPS [12]. This CPU and GPU combination is relatively low-performance by contemporary standards.

The inputs to each algorithm are four families of point sets with various statistical properties, generated procedurally via a pseudorandom number generator. Each coordinate in the Uniform point set is selected from

n	Uniform	Normal	3 Clusters	Cube Surface
2^{12}	3.58	4.12	4.06	5.08
2^{13}	4.60	5.30	4.91	5.07
2^{14}	5.57	5.68	5.66	5.68
2^{15}	6.10	6.00	5.93	5.91
2^{16}	6.01	5.94	5.89	5.49
2^{17}	6.32	6.25	6.34	6.29
2^{18}	6.40	6.47	6.45	6.27
2^{19}	6.84	6.89	6.74	6.48
2^{20}	6.98	6.83	6.98	6.86
2^{21}	7.21	7.23	7.10	7.00
2^{22}	7.63	7.71	7.28	7.43
2^{23}	7.92	7.97	7.99	8.07

Figure 10: GPU speedup factor.

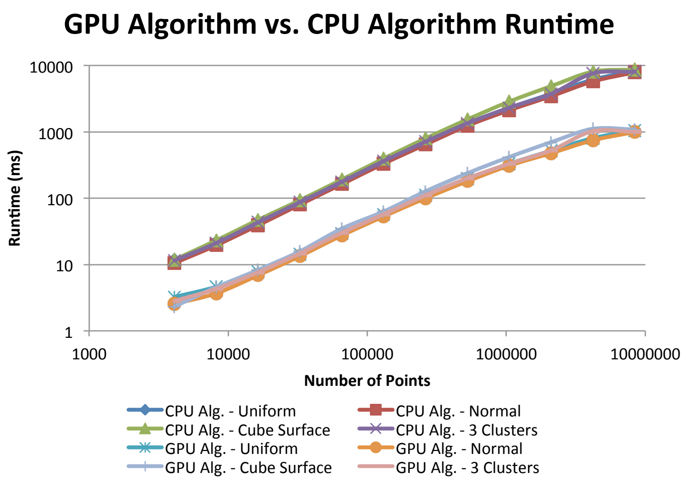


Figure 11: Runtime graph for n data points.

a uniform distribution, yielding a cube-shaped point cloud. Each coordinate in the Normal point set is an offset from 0 drawn from a normal distribution, yielding a dense cluster around the origin with a small proportion of outliers. The points in the 3 Clusters set are offset in the same way from one of three centroids; each point's centroid is chosen uniformly at random. The points in the Cube Surface set are uniform-distributed points on the surface of a cube, with a small normally-distributed inward or outward perturbation. Unlike the other distributions, a high proportion of the points in the Cube Surface are members of the convex hull.

The runtime results are the mean and standard deviation of 50 repeated trials. Elapsed times are measured with the `gettimeofday` system call which is precise to microseconds.

Originally, a hybrid approach to the GPU algorithm seemed to be a more attractive solution to solving the problem. The hybrid GPU algorithm would perform nearly all of the merge steps on the GPU, then perform the last few steps on the CPU after the

n	Uniform		Normal		3 Clusters		Cube Surface	
	Mean	σ	Mean	σ	Mean	σ	Mean	σ
2^{12}	1.16×10	1.76	1.05×10	0.885	1.14×10	0.951	1.18×10	0.616
2^{13}	2.12×10	0.591	1.95×10	0.544	2.11×10	0.580	2.31×10	0.495
2^{14}	4.35×10	1.01	3.98×10	0.340	4.33×10	0.803	4.72×10	0.800
2^{15}	8.74×10	.978	8.03×10	1.11	8.72×10	0.571	9.36×10	0.787
2^{16}	1.77×10^2	1.97	1.63×10^2	1.43	1.77×10^2	1.24	1.91×10^2	1.13
2^{17}	3.63×10^2	1.86	3.32×10^2	2.66	3.63×10^2	2.32	3.97×10^2	3.31
2^{18}	7.12×10^2	3.94	6.47×10^2	2.17	7.13×10^2	6.86	7.98×10^2	2.70
2^{19}	1.35×10^3	13.1	1.24×10^3	6.33	1.35×10^3	4.26	1.54×10^3	7.08
2^{20}	2.31×10^3	12.0	2.12×10^3	16.3	2.31×10^3	6.14	2.87×10^3	8.54
2^{21}	3.76×10^3	80.6	3.46×10^3	13.0	3.75×10^3	14.3	4.90×10^3	12.9
2^{22}	6.17×10^3	14.1	5.77×10^3	23.1	7.64×10^3	55.7	8.12×10^3	62.1
2^{23}	8.31×10^3	69.0	7.95×10^3	30.6	8.08×10^3	118.7	8.72×10^3	33.3

Figure 8: CPU algorithm runtimes.

n	Uniform		Normal		3 Clusters		Cube Surface	
	Mean	σ	Mean	σ	Mean	σ	Mean	σ
2^{12}	3.24	1.60	2.54	0.908	2.82	1.37	2.32	0.551
2^{13}	4.12	1.37	3.68	0.683	4.30	0.995	4.56	1.28
2^{14}	7.80	1.73	6.88	1.75	7.64	1.95	8.30	1.05
2^{15}	1.43×10	0.768	1.33×10	3.26	1.47×10	1.46	1.58×10	1.46
2^{16}	2.94×10	3.13	2.74×10	7.51	3.00×10	4.65	3.48×10	5.34
2^{17}	5.73×10	3.32	5.32×10	5.31	5.73×10	4.44	6.32×10	1.65
2^{18}	1.11×10^2	7.59	1.00×10^2	6.91	1.11×10^2	6.56	1.27×10^2	10.1
2^{19}	1.97×10^2	6.57	1.80×10^2	9.78	2.00×10^2	10.4	2.38×10^2	9.02
2^{20}	3.31×10^2	12.9	3.11×10^2	34.5	3.31×10^2	14.5	4.19×10^2	17.1
2^{21}	5.22×10^2	14.3	4.78×10^2	11.8	5.27×10^2	45.4	7.00×10^2	20.4
2^{22}	8.08×10^2	31.4	7.49×10^2	21.3	1.03×10^3	24.9	1.11×10^3	34.9
2^{23}	1.05×10^3	24.7	9.97×10^2	23.4	1.00×10^3	37.7	1.09×10^3	31.1

Figure 9: GPU algorithm runtimes.

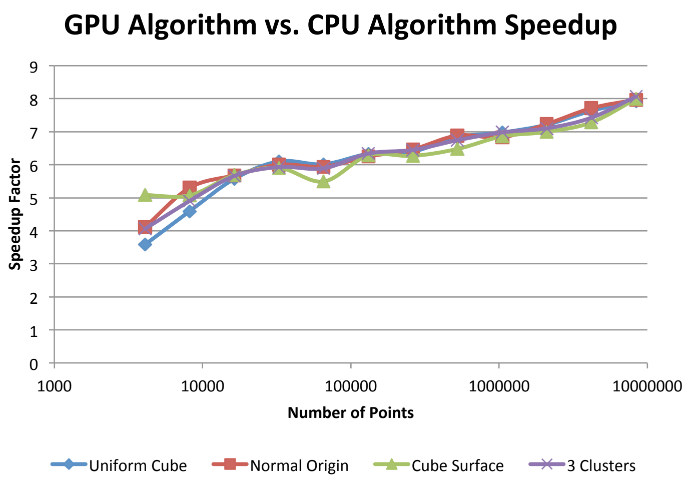


Figure 12: Speedup graph for n data points.

`totalMergesLeft` variable reached a certain value. The premise of this approach is that the last few iterations are poorly parallelizable and could be more quickly performed by a serial CPU. To accomplish this, the partially computed data would need to be copied from GPU memory to memory that the CPU has access to. On the CPU side, there would be a similar algorithm which would finish the rest of the computation using that same bottom-up style algorithm.

Surprisingly, our experimental results showed that those last few merge iterations take an insignificant amount of time – less than one millisecond. So the hybrid approach is overly-complex, and implementing it would have been an instance of premature optimization. The final design of the GPU algorithm takes place entirely on the GPU rather then on both GPU and CPU hardware. The GPU algorithm just requires the use of the CPU for the required OpenCL setup routines and ultimately to read in the data and output the data; the GPU completes all the extensive computations.

Something we found interesting is the ratio of speedup improvements over the CPU algorithm as the data set increases. For smaller data sets, the speedup is only about 4x. As the data set increases, the speedup increases to about 8x (see Figure 12).

Our roughly 8x speedup is notable since it approaches the maximum potential improvement achievable on our hardware. According to LINPACK and NVIDIA, our GPU is capable of roughly 8 times more gigaFLOPS than one of our CPU cores. Our implementation realizes practically all of this potential despite the obstacles inherent in parallelizing the 3D convex hull problem.

5 Conclusion

We have shown that bottom-up adaptation of the minimalist divide-and-conquer algorithm for 3D convex hulls is fast, practical, and reasonably straightforward. The approach is faster than CPU implementations and competitive with hybrid GPU/CPU implementations.

In performing this exercise, we did make two counterintuitive conclusions. First, while OpenCL and CUDA are intended to be high-level abstractions of GPU hardware, we nonetheless faced many obstacles related to low-level concerns such as memory management, memory hierarchies, and thread scheduling. Second, our intuition was that the overhead of starting and scheduling kernel applications would become a major bottleneck in the later steps of the algorithm. However, empirical results demonstrated this to be a non-issue.

The following are potential areas for future work:

- Higher-level libraries or tools for implementing divide-and-conquer algorithms on the GPU.
- A suite of compatible, parallel GPU implementations of fundamental computational geometry algorithms.
- In particular, an arrangement data structure, e.g. doubly connected edge list, is a prerequisite to implementing many well-motivated algorithms.

References

[1] S. G. Akl and G. T. Toussaint. A fast convex hull algorithm. *Information Processing Letters*, 7(5):219 – 222, 1978.

[2] N. M. Amato and F. P. Preparata. A time-optimal parallel algorithm for 3D convex hulls. *Algorithmica*, 14(2):169–182, Aug. 1993.

[3] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, Dec. 1996.

[4] T. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16:361–368, 1996.

[5] T. M. Chan. A minimalist's implementation of the 3-d divide-and-conquer convex hull algorithm. Technical report, University of Waterloo, 2003.

[6] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10:377–409, 1993.

[7] W. F. Eddy. A new convex hull algorithm for planar sets. *ACM Trans. Math. Softw.*, 3(4):398–403, Dec. 1977.

[8] T. Jurkiewicz and P. Danilewski. Efficient quicksort and 2D convex hull for CUDA, and MSIMD as a realistic model of massively parallel computations. Technical report, Max Planck Institute for Informatics, 2011.

[9] D. G. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15(1):287–299, 1986.

[10] A. Munshi. The OpenCL specification version 1.0. Technical report, The Khronos Group, 2009.

[11] NVIDIA. GeForce GTX 580 specifications. `http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-580/specifications`.

[12] NVIDIA. NVIDIA introduces industry-changing, highly integrated GPU. `http://www.nvidia.com/object/io_1224088545955.html`. Press Release.

[13] NVIDIA. *NVIDIA CUDA Programming Guide 2.0.* 2008.

[14] F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, 20(2):87–93, Feb. 1977.

[15] A. Rueda and L. Ortega. Geometric Algorithms on CUDA. In *Proceedings of the 3rd International Conference on Computer Graphics Theory and Applications*, 2008.

[16] R. Seidel. A convex hull algorithm optimal for point sets in even dimension. Master's thesis, Dept. of Computer Science, University of British Columbia, Vancouver, Canada, 1981.

[17] S. Srungarapu, D. Reddy, K. Kothapalli, and P. Narayanan. Fast two dimensional convex hull on the GPU. In *Advanced Information Networking and Applications (WAINA), 2011 IEEE Workshops of International Conference on*, pages 7 –12, march 2011.

[18] M. Tang, J. yi Zhao, R. Tong, and D. Manocha. GPU accelerated convex hull computation. In *Shape Modeling International (SMI) 2012*, 2012.

[19] J. White and K. A. Wortman. Divide-and-conquer 3D convex hulls on the GPU. Google Code project `http://code.google.com/p/3d-convex-hulls/`.

Basis Enumeration of Hyperplane Arrangements Up to Symmetries*

Aaron Moss† David Bremner‡

Abstract

Given a symmetry group acting on the hyperplanes of an arrangement, our goal is to report a single basis from each orbit of bases induced by this group. In this paper we extend previous techniques for finding the (feasible) bases of polyhedra up to symmetry, and for computing the symmetry groups of polyhedra, to the setting of hyperplane arrangements. We present some preliminary experiments with a C++ implementation of these techniques called `Basil`. These results show substantial speedups compared to a previous polyhedra only system using the computer algebra system `GAP`. We also measure the speedup due to a Gram matrix invariant, and show that the overhead of symmetry testing, while substantial, is dominated by the savings in reduced pivoting.

1 Introduction

Because of the large number of vertices and bases of a hyperplane arrangement, it is natural to consider generating these objects up to symmetry. One application for finding (orbits of) bases of hyperplane arrangements is the computation of the vector partition function of a matrix, a fundamental operation in parametric integer programming and representation theory. Bases of hyperplane arrangements are equivalent to bases of systems of linear equations (minimal subsystems defining zero dimensional solutions). Basis enumeration of systems of linear equations is necessary for dual-type generating function approaches to computing the vector partition function, which research by Brion, Szenes, and Vergne [6, 14] suggests may be quicker than current approaches.

This paper describes the design and implementation of a program for basis enumeration of hyperplane arrangements up to symmetries. This program, called `Basil` (“**Basis list**”) adapts the pivoting method of Bremner, Sikirić, and Schürmann [5] for basis enumeration of polyhedra up to symmetries, using a new pivot selection method to traverse hyperplane arrangements instead of polyhedra. The work of Bremner *et al.* is in turn related to the reverse-search method for basis enumeration of Avis and Fukuda [4] and earlier pivoting methods *e.g.* [7].

Sikirić’s `Polyhedral` [13] and Rehn’s `Sympol` [12] contain other solutions to the related problem of vertex enumeration up to symmetries of polyhedra; the approaches taken by both Sikirić and Rehn involve recursively decomposing the polyhedron into smaller polyhedra, and are quite different from our pivoting approach. For a survey of approaches for vertex enumeration up to symmetries of polyhedra (and the dual problem of facet enumeration up to symmetries), see [5], which covers both the recursive decomposition and pivoting approaches.

2 Background

2.1 Arrangements & Polyhedra

The structures discussed here exist in d-dimensional real space, $\mathbb{R}^d$. A *point* $\boldsymbol{x}$ in $\mathbb{R}^d$ is the ordered list of *coordinates* $\boldsymbol{x} = [x_1\ x_2\ \cdots\ x_d]$, where each of the x_i is a real number (though in this paper all explicitly defined vectors have rational coordinates for reasons of efficiency and ease of computation). A *hyperplane* H is the set of points $\boldsymbol{x} \in \mathbb{R}^d$ which satisfy a linear equation $\boldsymbol{a}^\top \boldsymbol{x} = b$ ($\boldsymbol{a} \in \mathbb{R}^d, b \in \mathbb{R}$), while a *hyperplane arrangement* $\mathcal{A}$ is the union of the points contained in a set of hyperplanes, indexed as $A_1, A_2, \cdots, A_n$. A *polyhedron* $\mathcal{P}$ is a closely related structure, the intersection of a set of *halfspaces* $P_1, P_2, \cdots, P_n$; a halfspace is the set of points $\boldsymbol{x} \in \mathbb{R}^d$ that satisfy a linear inequality $\boldsymbol{a}^\top \boldsymbol{x} \geq b$ ($\boldsymbol{a}$ and b defined as above). The hyperplane for which this inequality is satisfied with equality is known as the *bounding hyperplane* of a halfspace, while the set of bounding hyperplanes of the halfspaces defining a polyhedron is called its *bounding hyperplane arrangement.* The *size* of an arrangement is the number of hyperplanes n, while all arrangments considered will be full rank and thus have dimension d, the dimension of the underlying space. Size and dimension of polyhedra are defined analogously.

A *cobasis* $\mathbf{B}$ of a hyperplane arrangement is a set of indices of d hyperplanes which intersect at a single point, a *vertex* of the arrangement (We use here the name *cobasis* from linear programming for what is typically called a *basis* in geometry for consistency with the terminology of our linear programming-based implementation). Hyperplanes which contain a vertex are

*Research partially supported by NSERC. Computational resources provided by ACEnet.

†Faculty of Computer Science, University of New Brunswick, `moss.aaron@unb.ca`

‡Faculty of Computer Science, University of New Brunswick, `bremner@unb.ca`

said to be *incident* to that vertex. It should be noted that d hyperplanes meet in a single point if and only if the equations defining those hyperplanes are linearly independent. The problem of *basis enumeration* is thus to list all the unique cobases of a hyperplane arrangement. Vertices of polyhedra may be defined as those vertices of the polyhedron's bounding hyperplane arrangement which are contained within the polyhedron, and cobases of a polyhedron as the cobases of the bounding arrangement which correspond to those vertices. A vertex of an arrangement or polyhedron may be defined by more than one cobasis (*i.e.* if more than d hyperplanes of the (bounding) arrangement meet at that point); such a vertex is called *degenerate*. A polyhedron or arrangement with no degenerate vertices is *simple*, for such a polyhedron the *vertex enumeration* problem (reporting each unique vertex) is equivalent to the basis enumeration problem. For non-simple (*degenerate*) polyhedra and arrangements, the vertex enumeration problem can be solved by basis enumeration, though some method must be employed to filter out duplicate vertices.

The cobases of a polyhedron or hyperplane arrangement can be considered as the nodes of an implicit graph, where two cobases $\mathbf{B}_1$ and $\mathbf{B}_2$ are *adjacent* if they differ only by one element, that is, letting $\mathbf{B} = \mathbf{B}_1 \cap \mathbf{B}_2$, $\mathbf{B}_1 = \mathbf{B} \cup \{p\}$ and $\mathbf{B}_2 = \mathbf{B} \cup \{q\}$; here the $d-1$ hyperplanes defining $\mathbf{B}$ intersect in a 1-dimensional line, an *edge* of the arrangement.

A certain class of optimization problem involves finding a vertex $\boldsymbol{v}$ of a polyhedron which maximizes a linear objective function defined by a vector $\boldsymbol{c} = [c_1\ c_2\ \cdots\ c_d] \in \mathbb{R}^d$ as $c(\boldsymbol{x}) = \boldsymbol{c}^\top \boldsymbol{x}$. The field of *linear programming* has developed to solve this and related problems, some of these related problems being defined on hyperplane arrangements. One of the oldest and most studied approaches to linear programming, the simplex method pioneered by Dantzig [8], is to find an initial cobasis and then repeatedly move (or *pivot*) to some adjacent cobasis corresponding to a vertex $\boldsymbol{v}'$ with an objective value $c(\boldsymbol{v}')$ at least as good as the objective value of the current vertex. This process is repeated, proceeding until either a cobasis of an optimal vertex is reached or it can be seen that no such optimal vertex exists.

The fundamental data structure of the simplex method is the *simplex tableau*, $\mathrm{T}(\mathcal{P}, \mathbf{B})$, which re-expresses the linear inequalities defining a polyhedron $\mathcal{P}$ in terms of a cobasis $\mathbf{B}$. The first step to convert a polyhedron to tableau form is to add n new *slack variables* $\{x_{d+1}, x_{d+2}, \cdots, x_{d+n}\}$ to the existing *decision variables* $\{x_1, x_2, \cdots, x_d\}$ which define points in $\mathbb{R}^d$. The slack variables represent the "distance" between the bounding hyperplane of each halfspace in the polyhedron and the vertex represented by the tableau; the slack variables are therefore always kept non-negative by the simplex algorithm when dealing with polyhedra, though when simplex tableaux are used to represent hyperplane arrangements the slack variables may be either positive or negative, as points in an arrangement may be on either side of any hyperplane in the arrangement. To add the slack variables, each of the inequalities $\boldsymbol{a}_i^\top \boldsymbol{x} \geq b_i$ defining the halfspace P_i in $\mathcal{P}$ is rewritten as an equation $x_{d+i} = -b_i + \boldsymbol{a}_i^\top \boldsymbol{x}$, defining a matrix $A_{n\times d} = (a_{i,j})$ ($a_{i,j}$ being the j-th element of $\boldsymbol{a}_i$) and a vector $\boldsymbol{b} = [b_1\ b_2\ \cdots\ b_n]^\top$. These components are combined with the vector $\boldsymbol{c}$ defining the objective function in a matrix as follows, defining the initial simplex tableau:

$$M = \begin{bmatrix} 0 & \boldsymbol{c}^\top \\ -\boldsymbol{b} & A_{n\times d} \end{bmatrix}$$

The *basic* variables of the tableau are the set of variables x_i which are defined by the equations represented by the rows of the tableau; the set of variables x_j which are the column variables those equations are written in terms of are the *cobasic* variables of the tableau[1]. With the addition of auxiliary structures to the tableau matrix M to remember the current sets of basic and cobasic variables, the data structures needed for the simplex method are complete. The values of the cobasic variables of a simplex tableau are assumed to be zero, so that the value of any basic variable (or the objective function in the first row) can be read off from the constant term in the first column. After the initial setup of the tableau is complete, the decision variables are moved into the basis, with slack variables replacing them in the cobasis. When this process is completed, the current vertex represented by the tableau can be read off from the values of the decision variables in the first column.

In the context of linear programming, a *pivot* from a cobasis $\mathbf{B}_1$ to another adjacent cobasis $\mathbf{B}_2$ ($\mathbf{B}_1 = \mathbf{B}_2 \cup \{x_e\} \backslash \{x_l\}$) exchanges the *entering* slack variable, x_e for the *leaving* slack variable, x_l,[2] traversing an edge of the arrangement or polyhedron.

Pivot rules used in the simplex method are based on the idea of the *minimum ratio test*. Geometrically, this test can be thought of as leaving one basis and sliding along an edge of a polyhedron or hyperplane arrangement until the first new (bounding) hyperplane is reached, forming a new basis. In a simplex tableau, distance from each hyperplane is represented by its associated slack variable, and moving from one hyperplane to another (equivalently, moving to an adjacent cobasis) is accomplished by allowing the one cobasic variable to become non-zero while forcing some basic variable to zero. For a given pair x_e and x_l of cobasic and basic variables, the ratio between the constant term $\boldsymbol{b}_l$ of the

[1]Note that the cobasic variables of a simplex tableau, not the basic, correspond to a basis of the represented polyhedron or arrangement in the usual geometric definition.

[2]The variables are "entering" and "leaving" the linear programming basis, the complement of the cobasis.

leaving variable x_l and the coefficient $a_{l,e}$ of the entering variable x_e in the leaving variable's equation determines how much the entering variable can be increased or decreased. By setting x_e to $r = \boldsymbol{b}_l / a_{l,e}$, x_l is forced to zero. In polyhedra leaving and entering variables must be chosen such that $r \geq 0$, as slack variables cannot be negative, but this restriction does not hold for arrangements. Selecting x_l such that the magnitude of r is minimized (the "minimum ratio") finds the nearest adjacent cobasis to pivot to; if $r = 0$ (due to $\boldsymbol{b}_l = 0$), the pivot is *degenerate*, moving to another cobasis of the same vertex.

2.2 Symmetries

Many interesting hyperplane arrangements have a significant number of *automorphisms*: geometric symmetries (*e.g.* reflections and rotations) which leave the points in the arrangement setwise invariant. These symmetries can also be considered as permutations of the list of hyperplanes included in the arrangement[3]. Taking the group G of some set of these symmetries acting on a hyperplane arrangement, the problem of *basis enumeration up to symmetries* is listing exactly one cobasis from each orbit under the action of G. The symmetries we are particularly interested in are *isometries*, distance preserving symmetries.

One property of isometric cobases is that, given some distance metric, the set of angles according to that metric between each pair of hyperplanes which meet in a single cobasis is setwise invariant under the action of any symmetry in the automorphism group. To use this property, the angles between all pairs of hyperplanes can be precomputed, and then each distinct angle can be represented by a unique integer. The *Gram matrix* $A = (a_{i,j})$ is constructed such that $a_{i,j}$ is the value corresponding to the angle between the hyperplanes indexed i and j. Gram matrices for polyhedra can be similarly constructed with respect to the angles between the bounding hyperplanes of the polyhedron. A submatrix of a Gram matrix uniquely representing the angles between pairs of hyperplanes in a given cobasis can be constructed by selecting only the elements in the rows and columns of the Gram matrix corresponding to the indices of the cobasis hyperplanes. If each row of this submatrix is sorted, and then the rows of the submatrix are lexicographically sorted, the resulting submatrix uniquely represents the angles between each pair of hyperplanes in the cobasis, and pairs of such matrices can be compared for equality swiftly. Equality of Gram submatrices does not guarantee that the corresponding cobases are symmetric, but inequality of Gram submatrices does show that the cobases are not symmetric.

An automorphism α of the Gram matrix $G = (g_{i,j})$ of a polyhedron may be defined by a permutation σ of the row and column indices of the matrix as $\alpha(G) = (g_{\sigma(i),\sigma(j)})$ such that $G = \alpha(G)$. Such an automorphism of the Gram matrix corresponds to an automorphism of the polyhedron produced by permuting the indices of the halfspaces defining the polyhedron by σ. A full proof of this can be found in [5], but intuitively the rows and columns of the Gram matrix correspond to the halfspaces defining the polyhedron. As the Gram matrix encodes the distances between each pair of bounding hyperplanes as angles, any transformation which leaves the Gram matrix invariant will also not change the polyhedron, because the relative positions of each of the halfspaces have remained constant. If the Gram matrix is interpreted as the adjacency matrix of a graph, with the elements of the matrix representing colors of the edges, these automorphisms can also be expressed as edge-color preserving graph automorphisms.

One problem we encountered in generating Gram matrices for hyperplane arrangements that does not occur in the polyhedral case is that any hyperplane $A = \{\boldsymbol{x} \mid \boldsymbol{a}^\top \boldsymbol{x} = b\}$ can be replaced by its *negation* $\bar{A} = \{\boldsymbol{x} \mid -\boldsymbol{a}^\top \boldsymbol{x} = -b\}$ without changing the arrangement. However, the angle produced by $\bar{A}$ and another hyperplane B is the supplement of the angle produced by A and B, in general a distinct angle. When using the Gram matrix to detect non-symmetric cobases, this problem can be solved by simply using a unique up to supplements representation for each angle. This approach does not work when using the Gram matrix to determine the automorphisms of the arrangement, as spurious automorphisms are generated; essentially, these false automorphisms consider a hyperplane to be both itself and its negation simultaneously, causing the arrangement to be warped by some angles between pairs of hyperplanes being replaced by their supplements. If the arrangement is doubled such that each hyperplane is paired with its negation, then matrix automorphisms may replace a hyperplane by its negation by transposing the two in the symmetry but this warping is prevented from occurring, and correct automorphisms may be derived after reversing the doubling process on the generated permutations. This does, however, quadruple the size of the Gram matrix used for automorphism generation.

3 Algorithms

The essential idea of our algorithm for basis enumeration up to symmetries is to explore the hyperplane arrangement outward, moving from an initial cobasis to its adjacent cobases, pruning this search tree when a cobasis symmetric to one already found is reached; a full description is in Algorithm 1. The subroutine InitialCobasis() returns any coba-

[3]Automorphisms on polyhedra can be considered analogously.

sis of the arrangment; ADJACENT($\mathbf{B}$) returns a list of all cobases $\mathbf{B}_i$ which are adjacent to a cobasis $\mathbf{B}$. INNEWORBIT($\mathbf{B}$) tests whether a cobasis $\mathbf{B}$ is in an orbit already discovered, while REPORT($\mathbf{B}$) is used to output a newly discovered cobasis $\mathbf{B}$. The subroutines PUSHCOBASIS($S, \mathbf{B}$) and POPCOBASIS(S), which push and pop a cobasis to or from a stack S, (updating internal structures to be consistent with that cobasis) complete the description of the algorithm.

Algorithm 1 Basis orbit enumeration algorithm

```
function SymmetricBasisSearch(void)
                    ▷ find a cobasis of the arrangement
    $\mathbf{B} \leftarrow$ InitialCobasis()
                    ▷ explore outward from this cobasis
    $S \leftarrow$ a stack of cobases, initially empty
    Report($\mathbf{B}$)
    PushCobasis($S, \mathbf{B}$)
    repeat
        $\mathbf{B} \leftarrow$ PopCobasis($S$)
  ▷ search for new orbit representatives adjacent to $\mathbf{B}$
        for all $\mathbf{B}_i \in$ Adjacent($\mathbf{B}$) do
            if InNewOrbit($\mathbf{B}_i$) then
                Report($\mathbf{B}_i$)
                PushCobasis($S, \mathbf{B}_i$)
            end if
        end for
    until Empty($S$)
end function
```

The reader familiar with pivoting algorithms will remark upon the absence of perturbation from Algorithm 1. Practical pivoting algorithms for vertex enumeration use some form of perturbation (or equivalent pivot rule, e.g. [3]) to reduce the number of bases reported per vertex. Here our goal is to find all orbits of bases, so standard symbolic perturbation schemes that ignore the symmetry group are unlikely to work well. In [5] the authors describe an explicit *orbitwise* perturbation scheme that preserves the orbits of bases of the original input (possibly shattered into several orbits). Since this can be implemented as a preprocessor, we do not discuss it here; some of our experimental data (the E7-j examples in Table 1) is of this preprocessed type.

All the required subroutines for Algorithm 1 can be defined to act on a simplex tableau. Most of these subroutines have been known since Dantzig's original formulation of the simplex algorithm, and can be derived from most linear programming textbooks, though some simple modifications may be needed to convert processes intended for use on polyhedra to work with arrangements (such as our implementation of ADJACENT($\mathbf{B}$), detailed below). For PUSHCOBASIS($S, \mathbf{B}$) and POPCOBASIS(S), our implementation keeps an internal stack of pivots performed, reversing those pivots as necessary to return to an earlier cobasis.

Our implementation of ADJACENT($\mathbf{B}$) is based on the minimum ratio test. Our rule tries all the variables x_j in the cobasis $\mathbf{B}$ as entering variables, attempting to find valid leaving variables for each. Given an entering variable x_e, our method reports all the basic variables that are already zero as possible leaving variables (these represent degenerate pivots), as well as all the basic variables x_i which have a minimal magnitude ratio $\boldsymbol{b}_i/a_{i,e}$ in both the positive and negative directions. Taking both positive and negative ratio ensures that new adjacent cobases are found on either side of the hyperplane corresponding to x_e.

4 Implementation & Results

In order to achieve good performance, Algorithm 1 needs an efficient pivot implementation. Previous experiments by Avis [3] suggest a significant advantage for the integer pivoting method of Edmonds [9]. The implementation described in this paper, `Basil`[4], was built using David Avis' `lrslib` [2], which uses Edmonds' integer pivoting.

The design of `Basil` is quite closely based on the `Symbal` software of Bremner *et al.* [5], which performs basis enumeration up to symmetries on polyhedra. However, where `Symbal` is implemented in the `GAP` [15] computer algebra system with calls to C libraries wrapping `lrslib` for simplex operations and McKay's `Nauty` [10] for graph automorphism calculations, `Basil` has been re-implemented in C++, using Rehn's `permlib` [11] library to replace the both the group theoretic capabilities of `GAP` used by `Symbal` (which it should be pointed are relatively simple orbit membership tests) and the automorphism code in `Nauty` with matrix automorphism routines. `Basil` also uses `lrslib` for its tableau implementation.

As `Basil` is designed as an extension of `Symbal`, it is also capable of performing basis enumeration of polyhedra up to symmetries. Though this functionality is not the focus of this paper, the experimental results shown in Table 1 compare the relative performance of `Basil` and `Symbal` for basis enumeration up to symmetries of a set of polyhedra. The Ey instances discussed are the Dirchlet-Voronoi-cells (DV-cells) of the root lattices E_y, as described in section 7.2 of [5]. As can be seen from these results, `Basil` is generally about two orders of magnitude faster, attributable to the lower overhead of C++ execution than the `GAP` interpreter and the more sophisticated and efficient data structures available in C++ than `GAP`. These numbers represent only the CPU time of both programs; this is a fairly accurate representation of `Basil`'s runtime, but underestimates `Symbal`'s

[4]Software and test input available by request.

Table 1: Comparison of Basil & Symbal
(bases & vertices count *orbits*)

Problem	$n{:}d$	bases:verts	Bas(s)	Sym(s)
E7	126:8	32:2	1.82	282.16
E7-3	126:8	82:58	0.59	12.41
E7-7	126:8	1195:106	19.88	1507.72
E7-65	126:8	356:14	7.88	1308.07
E7-102	126:8	41:7	1.27	223.97
E8	240:8	2:2	0.41	2.13

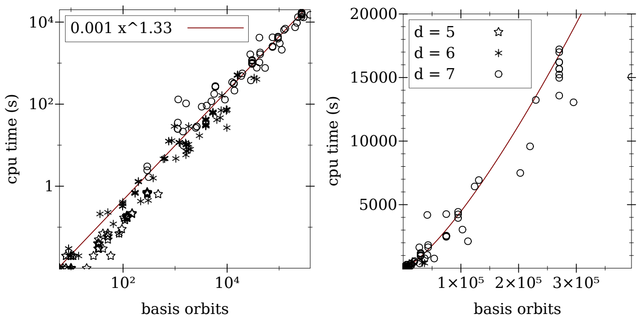


Figure 1: Time for Cd-6-3 instances by # basis orbits, on both log-log and linear plots.

by about half due to overhead from the interprocess communication needed to connect `GAP` to the external C libraries used. Timing results reported are from the Placentia ACEnet cluster [1], which has 2.3–3.0 GHz AMD Opteron processors.

Table 2 shows some early performance results for `Basil` on the arrangements. Problems DxA and EyA are the bounding hyperplane arrangements for the DV-cells of the root lattices D_x and E_y, while the Cx-y-z instances are generated by choosing z vertices of the x-cube and acting on them with a subgroup of the hyperoctohedral group with at least y orbits. Figure 1 plots runtime for 268 Cx-6-3 instances. At least for these examples, it seems that suggests that `Basil`'s runtime is super-linear but sub-quadratic in the number of orbits output (as opposed to the total number of bases, which can be exponentially larger). Table 3 shows the benefits of considering symmetries for basis enumeration; the values in this table are the results of using `Basil` to enumerate all the cobases of the given test cases.

The pivoting approach implemented in `Basil` (and `Symbal`) differs from that proposed by Avis and Fukuda [4] for the non-symmetric vertex enumeration problem in that their *reverse search* does not maintain state describing cobases already found or the path from the initial cobasis to the cobasis currently under consideration. That approach has the benefit of requiring a relatively small constant amount of memory, but also requires more simplex computations, increasing running time. For the symmetric case, we expect there to be

Table 2: Basil Timing Results
(bases & vertices count *orbits*)

Problem	$n{:}d$	bases:verts	Bas(s)
D4A	24:4	12:7	0.02
D5A	40:5	104:25	0.50
C5-6-3a	25:5	291:36	0.70
C5-6-3b	16:5	51:16	0.05
C6-6-3a	15:6	9:1	0.03
C6-6-3b	36:6	1394:91	13.90
C6-6-3c	50:6	5342:157	63.65
C7-6-3a	48:7	18720:140	456.59
E7A	126:8	12399:227	1570.66

Table 3: Non-Symmetric Timing Results
(*all* bases & vertices counted)

Problem	$n{:}d$	bases:verts	Bas(s)
D4A	24:4	5028:863	23.62
C5-6-3a	25:5	24444:852	120.80
C5-6-3b	16:5	3005:234	4.37
C6-6-3a	15:6	2530:1	16.31

relatively few orbits of cobases, allowing `Basil` to keep representatives of each in memory, and thus have not yet investigated a memory-less reverse search for this problem. Additionally, the limiting factor on the size of instances we can currently solve is the computational expense of the group theoretic calculations required to check symmetry (encapsulated in InNewOrbit in Algorithm 1), which dominate the running time of `Basil` to a significant degree. Our profiling results show that tests for orbit membership take about 60% of the runtime of `Basil`, while the only other individual operation which significantly contributes to runtime is simplex pivoting, contributing about 20% of the execution time.

Because the group theoretic computations involved in checking if two cobases are in the same orbit under the group action are so expensive, `Basil` utilizes some cheaper invariants of symmetric cobases to shrink the set of cobases that must be tested for symmetries. The simplest of these invariants is to check that the number of hyperplanes incident to the vertices defined by the two cobases is the same, as an automorphism of the hyperplane arrangement preserves the number of hyperplanes which meet at any given vertex. `Basil` also keeps a cache of recently seen cobases to avoid needing to retest previous cobases (for instance, the cobasis that was pivoted from to reach the current cobasis).

`Basil` also uses the Gram submatrix to differentiate cobases; representatives of known cobasis orbits are stored in a hash table indexed by the corresponding Gram submatrix. Comparing each newly discovered cobasis only to the cobases having Gram submatrices

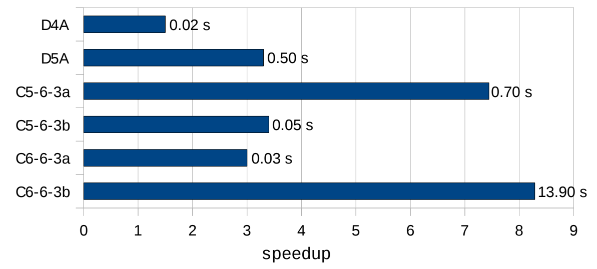


Figure 2: Speedup from using Gram matrix (bar labels are runtime *without* Gram matrix)

which are equivalent under the sorting procedure described earlier greatly reduces the number of expensive group theoretic tests which must be performed. If the Gram submatrix invariant is turned off in `Basil`, execution time on a given instance increases dramatically, as seen in Figure 2, while when activated the Gram matrix computations consume about 5% of the execution time of `Basil`.

5 Conclusion & Future Work

Basis enumeration seems to be an easier problem than the closely related problem of vertex enumeration. A pivoting algorithm can effectively explore the graph of adjacent bases. The main practical difficulty is the typically enormous output size from even moderate sized input. In certain applications, it suffices to generate one basis from each orbit under some natural symmetry group. In this paper we have described the enhancement of the symmetric pivoting software `Symbal` to produce a second generation symmetric pivoting software `Basil`. `Basil` is a native C++ application, and the speedup compared to `Symbal` can be seen as a validation of the use of C++ instead of the computer algebra system `GAP`, enabled by use of the `permlib` C++ library for group theoretic computations. The main motivation for developing `Basil` was to be able to generate orbit representatives of bases in hyperplane arrangements, based on a perceived need for this capability in certain novel approaches to integer programming. The extension from polyhedra to arrangements required defining a new ratio-test, and a modified procedure compute the symmetry group.

As the expense of the group theoretic calculations is the current limiting factor on the problem size that is feasible to solve, future directions for this research include a parallel implementation of `Basil` to bring greater computational power to bear on the problem, as well as research into invariants which may be cheaper to test than cobasis isomorphism.

Another way to reduce group theoretic calculations is to construct or approximate a *fundamental domain*, a convex cell F such that each orbit of cobases has exactly one representative in F. Such a cell can be constructed by techniques closely related to Voronoi diagrams, and could be used to prune the search for adjacent bases.

References

[1] *ACEnet.* `http://www.ace-net.ca/wiki/ACEnet`, September 2011.

[2] D. Avis. *lrs home page.* `http://cgm.cs.mcgill.ca/~avis/C/lrs.html`. accessed 26 January 2012.

[3] D. Avis. Computational experience with the reverse search vertex enumeration algorithm. *Optimization Methods and Software*, 10(2):107–124, 1998.

[4] D. Avis and K. Fukuda. A pivoting algorithm for convex hulls and vertex enumeration of arrangments and polyhedra. *Discrete & Computational Geometry*, 8(1):295–313, 1992.

[5] D. Bremner, M. D. Sikirić, and A. Schürmann. Polyhedral representation conversion up to symmetries. In D. Avis, D. Bremner, and A. Deza, editors, *Polyhedral Computation*, pages 45–71. CRM Proceedings & Lecture Notes, American Mathematical Society, 2009.

[6] M. Brion and M. Vergne. Residue formulae, vector partition functions and lattice points in rational polytopes. *Journal of the American Mathematical Society*, 10(4):797–833, October 1997.

[7] A. Charnes. The simplex method: optimal set and degeneracy. In *An introduction to Linear Programming*, Lecture VI, pages 62–70. Wiley, New York, 1953.

[8] G. B. Dantzig. Maximizing a linear function of variables subject to linear inequalities. *Activity Analysis of Production and Allocation*, pages 339–347, 1951.

[9] J. Edmonds and J.-F. Maurras. Note sur les Q-matrices d'Edmonds. *RAIRO. Recherche opérationnelle*, 31(2):203–209, 1997.

[10] B. McKay. *The nauty page.* `http://cs.anu.edu.au/~bdm/nauty/`. accessed 26 January 2012.

[11] T. Rehn. *User's Guide for PermLib.* `http://www.math.uni-rostock.de/~rehn/software/permlib.html`, October 2011. accessed 26 January 2012.

[12] T. Rehn. *User's Guide for SymPol.* `http://www.math.uni-rostock.de/~rehn/software/sympol.html`, October 2011. accessed 16 February 2012.

[13] M. D. Sikirić. *Polyhedral home page.* `http://drobilica.irb.hr/~mathieu/Polyhedral/`. accessed 10 May 2012.

[14] A. Szenes and M. Vergne. Residue formulae for vector partitions and Euler–Maclaurin sums. *Advances in Applied Mathematics*, 30:295–342, January 2003.

[15] The GAP Group. *GAP System for Computational Discrete Algebra.* `http://www.gap-system.org`, September 2008. accessed 26 January 2012.

Hardness Results for Computing Optimal Locally Gabriel Graphs

Abhijeet Khopkar[*] Sathish Govindarajan[†]

Abstract

Delaunay and Gabriel graphs are widely studied geometric proximity structures. Motivated by applications in wireless routing, relaxed versions of these graphs known as *Locally Delaunay Graphs* (*LDGs*) and *Locally Gabriel Graphs* (*LGGs*) have been proposed. We propose another generalization of *LGGs* called *Generalized Locally Gabriel Graphs* (*GLGGs*) in the context when certain edges are forbidden in the graph. Unlike a Gabriel Graph, there is no unique *LGG* or *GLGG* for a given point set because no edge is necessarily included or excluded. This property allows us to choose an *LGG*/*GLGG* that optimizes a parameter of interest in the graph. We show that computing an edge maximum *GLGG* for a given problem instance is NP-hard and also APX-hard. We also show that computing an *LGG* on a given point set with dilation $\leq k$ is NP-hard. Finally, we give an algorithm to verify whether a given geometric graph $G = (V, E)$ is a valid *LGG*.

1 Introduction

A geometric graph $G = (V, E)$ is an embedding of the set V as points in the plane and the set E as line segments joining two points in V. Delaunay graphs, Gabriel graphs and Relative neighborhood graphs (RNGs) are classic examples of geometric graphs that have been extensively studied and have applications in computer graphics, GIS, wireless networks, sensor networks, etc (see survey [7]). Gabriel and Sokal [5] defined the Gabriel graph as follows:

Definition 1 *A geometric graph $G = (V, E)$ is called a Gabriel graph if the following condition holds: For any $u, v \in V$, an edge $(u, v) \in E$ if and only if the circle with $\overline{uv}$ as diameter does not contain any other point of V.*

Gabriel graphs have been used to model the topology in a wireless network [3]. Motivated by applications in wireless routing, Kapoor and Li [8] proposed a relaxed version of Delaunay/Gabriel graphs known as k-locally Delaunay/Gabriel graphs. The edge complexity of these structures has been studied in [8, 11]. In this paper, we focus on 1-locally Gabriel graphs and call them *Locally Gabriel Graphs* (*LGG*s).

Definition 2 *A geometric graph $G = (V, E)$ is called a Locally Gabriel Graph if for every $(u, v) \in E$, the circle with $\overline{uv}$ as diameter does not contain any neighbor of u or v in G.*

The above definition implies that in an *LGG*, two edges $(u, v) \in E$ and $(u, w) \in E$ *conflict* with each other and cannot co-exist if $\angle uwv \geq \frac{\pi}{2}$ or $\angle uvw \geq \frac{\pi}{2}$. Conversely if edges (u, v) and (u, w) co-exist in an *LGG*, then $\angle uwv < \frac{\pi}{2}$ and $\angle uvw < \frac{\pi}{2}$. We call this condition an *LGG constraint.*

Study of these graphs was initially motivated by design of dynamic routing protocols for *ad hoc* wireless networks [10]. Like Gabriel Graphs, *LGGs* are also proximity-based structures that capture the interference patterns in wireless networks. An interesting point to be noted is that there is no unique *LGG* on a given point set since no edge in an *LGG* is necessarily included or excluded. Thus the edge set of the graph (used for wireless communication) can be customized to optimize certain network parameters depending on the application. While a Gabriel graph has a linear number of edges (planar graph), an *LGG* can be constructed with a super-linear number of edges [4]. A dense network can be desirable for applications like broadcasting or multicasting. The dilation or spanning ratio of a graph is an important parameter in wireless network design. Graphs with small spanning ratios are important in many applications and motivate the study of geometric spanners. In this paper, we initiate the study of dilation on *LGGs*. We show that there exists a point set such that the Gabriel Graph on it has dilation $\Omega(\sqrt{n})$ whereas there exists an *LGG* on the same point set with dilation $O(1)$.

In many situations, certain links are forbidden in a network due to physical barriers, visibility constraints or limited transmission radius. Thus, all pairs of nodes might not induce edges and this effect can be considered in *LGGs*. Thus, it is natural to study *LGGs* in the context when the network has to be built only with a set of predefined links. In this context, we define a generalized version of *LGGs* called *Generalized locally Gabriel Graphs* (*GLGGs*). Edges in a *GLGG* can be picked only from the edges in a given predefined geometric graph.

[*]Dept of Computer Science and Automation, Indian Institute of Science Bangalore, abhijit@csa.iisc.ernet.in

[†]Dept of Computer Science and Automation, Indian Institute of Science Bangalore, gsat@csa.iisc.ernet.in

Definition 3 *For a given geometric graph $G = (V, E)$ we define $G' = (V, E')$ as GLGG if G' is a valid LGG and $E' \subseteq E$.*

Previous results on $LGGs$ have focused on obtaining combinatorial bounds on the maximum edge complexity. In [8], it was shown that an LGG has at most $O(n^{\frac{3}{2}})$ edges since $K_{2,3}$ is a forbidden subgraph. Also, it was observed in [11] that any unit distance graph is also a valid LGG. Hence there exist $LGGs$ with $\Omega(n^{1+\frac{c}{\log \log n}})$ edges [4]. It is not known whether an edge maximum LGG can be computed in polynomial time.

Our Contribution: We present the following results in this paper.

1. We show that computing a $GLGG$ with at least m edges on a given geometric graph $G = (V, E)$ is NP-complete (reduction from 3-SAT) and also APX-hard (reduction from MAX-(3,4)-SAT).
2. We show that the problem of determining whether there exists an LGG with dilation $\leq k$ is NP-hard by reduction from the partition problem motivated by [6]. We also show that there exists a point set P such that any LGG on P has dilation $\Omega(\sqrt{n})$ that matches with the best known upper bound [2].
3. For a given geometric graph $G = (V, E)$, we give an algorithm with running time $O(|E| \log |V| + |V|)$ to verify whether G is a valid LGG.

2 Hardness of computing an edge maximum $GLGG$

In this section we show that deciding whether there exists a $GLGG$ on a given geometric graph $G = (V, E)$ with at least m edges for a given value of m is NP-complete by a reduction from 3-SAT. We further show that computing edge maximum $GLGG$ is APX-hard by showing a reduction from MAX-(3,4)-SAT.

A 3-SAT instance is a conjunction of several clauses and each clause is a disjunction of exactly 3 variables. Let $\mathcal{I}$ be an instance of the 3-SAT problem with k clauses $C_1, C_2, \ldots, C_k$ and n variables $y_1, y_2, \ldots, y_n$. A geometric graph $G = (V, E)$ is constructed from $\mathcal{I}$ such that there exists a $GLGG$ on G with at least m edges if and only if $\mathcal{I}$ admits a satisfying assignment. We construct a vertex set V (points in the plane) of size $(k + 3)n + k$ that is partitioned into $2n$ literal vertices denoted by $V_1 = \{x_i, x'_i \mid i \in \{1, \ldots, n\}\}$, $(k + 1)n$ variable vertices denoted by $V_2 = \{z_{i_j} \mid i \in \{1, \ldots, n\}, j \in \{1, \ldots, k + 1\}\}$ and k clause vertices denoted by $V_3 = \{c_j \mid j \in \{1, \ldots, k\}\}$. Thus, $V = V_1 \cup V_2 \cup V_3$. Two literal vertices x_i and x'_i corresponding to the same variable are called conjugates of each other.

Now let us discuss the placement of these vertices on the plane as shown in Figure 1. All literal vertices are placed closely on a vertical line l and the distance between two consecutive vertices is 10^{-5}. Two conjugate literal vertices corresponding to the same variable are kept next to each other. Let l_1 and l_2 be two horizontal lines passing through the highest and the lowest literal vertex respectively. Let b_0 be the center point of the line segment containing all the literal vertices. With b_0 as center, a circle is drawn with radius $d_1 = n^4$. All clause vertices $c_1, c_2, \ldots, c_k$ are placed along an arc a_0 of this circle (with a distance of $\frac{n}{2}$ between two consecutive vertices) with the additional restriction that these vertices cannot lie between lines l_1 and l_2. b_0c_1 and b_0c_k make an angle less than $\alpha = \frac{\pi}{4}$ with the horizontal axis. Now $k+1$ variable vertices are placed for each variable in the 3-SAT instance. Consider two horizontal lines l_{x_i} and $l_{x'_i}$ passing through literal vertices x_i and x'_i. With center at the mid point of x_i and x'_i (call it b_i) a circle is drawn with radius $d_2 = 10n^4$. Variable vertices are placed on an arc a_i of this circle on the same side of l where clause vertices are placed. These vertices $z_{i_1}, \ldots, z_{i_{(k+1)}}$ are placed a distance of $\frac{n}{4}$ apart with the restriction that no vertex should be placed between lines l_{x_i} and $l_{x'_i}$. Any line connecting these vertices with x_i and x'_i makes an angle less than α with the horizontal axis. For all the variables in $\mathcal{I}$, corresponding variable vertices are placed similarly. For simplicity variable vertices are shown corresponding to only one variable in Figure 1. For each clause C_j, there are 3 edges between

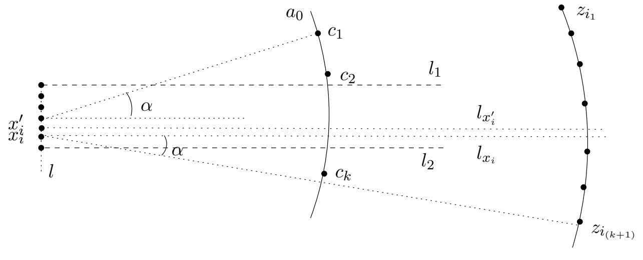


Figure 1: Placement of vertex set V

clause vertex c_j and the corresponding literal vertices. Let E_1 be the set of these edges from all the clause vertices to three corresponding literal vertices. For example, if a clause C_j has literals y_a, y_b and y'_c, then the edges $(c_j, x_a), (c_j, x_b)$ and (c_j, x'_c) are included in E_1. Another set of edges between literal vertices and variable vertices is defined

$$E_2 = \{(x_i, z_{i_1}), \ldots, (x_i, z_{i_{k+1}}), (x'_i, z_{i_1}), \ldots, (x'_i, z_{i_{k+1}}) \mid 1 \leq i \leq n\}$$

Now, $E = E_1 \cup E_2$. Let $G = (V, E)$ be the geometric graph over which an edge maximum $GLGG$ is to be computed. Let us analyze the conflicts among the edges in G. It should be noted that since a $GLGG$ is also an

LGG, it suffices to look at the *LGG* constraints to determine whether two edges conflict. Consider any *GLGG* $G' = (V, E')$ with $E' \subseteq E$ on the geometric graph G. The following constraints are observed on the edge set E'.

Since the edges (x_i, z_{i_j}) and (x'_i, z_{i_j}) conflict with each other ($\angle z_{i_j} x_i x'_i$ or $\angle z_{i_j} x'_i x_i$ is greater than $\frac{\pi}{2}$ by construction), a variable vertex z_{i_j} can have an edge incident to only x_i or x'_i.

Remark 1 *A variable vertex z_{i_j} can have only one edge ((x_i, z_{i_j}) or (x'_i, z_{i_j})) incident to it in E'.*

Similarly, we can infer Remark 2 due to *LGG* constraints.

Remark 2 *Any clause vertex c_j can be incident to at most one literal vertex in E'.*

It can be observed that two *LGG* edges that are the radii of the same circle do not conflict with each other. Here, b_i (the center of arc a_i) is close enough to both the literal vertices (x_i and x'_i) and the radius d_2 is chosen large enough so that no two edges from a literal vertex to the corresponding variable vertices conflict with each other.

Remark 3 *A literal vertex x_i (or x'_i) can have edges incident to all the corresponding variable vertices z_{i_j} in E' where $j \in \{1, \ldots, k+1\}$.*

Since a literal vertex is placed sufficiently close to b_0 (the center of arc a_0) and the radius d_1 is chosen large enough, no two edges from a literal vertex to the clause vertices conflict with each other.

Remark 4 *In E', a literal vertex x_i can have edges incident to all the clause vertices that have edges incident to x_i in E_1.*

Since d_2 is chosen large enough compared to d_1, if a literal vertex x_i is connected to a variable vertex z_{i_j}, the circle with $\overline{x_i z_{i_j}}$ as diameter would contain all the clause vertices. Therefore, x_i cannot be connected to any clause vertex due to the *LGG* constraint.

Remark 5 *In E', if a literal vertex has an edge incident to a variable vertex, it cannot have an edge incident to any clause vertex.*

Lemma 1 *If there exists a GLGG G' on G with at least $(k+1)n + k$ edges, then there exists a satisfying assignment to the given 3-SAT instance.*

Proof. Since each variable vertex can have at most one edge incident to it (refer to Remark 1), at most $(k+1)n$ edges of E' can be selected from E_2. Similarly each clause vertex can have at most one edge incident to it (refer to Remark 2), so in E' at most k edges can be selected from E_1. If there are $(k+1)n + k$ edges in E', then one edge is incident to each variable vertex and clause vertex. If there is an edge between a clause vertex c_j and the literal vertex x_i (resp. x'_i), assign $y_i = 1$ (resp. $y_i = 0$) as it satisfies the clause C_j. By this rule assign a truth value to a variable in each clause. If one clause vertex is incident to x_i, no other clause vertex can be incident to x'_i as x'_i is connected to the corresponding $k+1$ variable vertices (refer to Remark 5). Therefore, this rule would yield a consistent assignment satisfying all the clauses. Hence, the given 3-SAT instance is satisfiable. □

Lemma 2 *If there is a satisfying assignment to the given 3-SAT instance, then there exists a GLGG G' over G with at least $(k+1)n + k$ edges.*

Proof. A *GLGG* with $(k+1)n + k$ edges can be constructed based on the satisfying assignment to the given 3-SAT instance. If a variable $y_i = 1$ (resp. $y_i = 0$) then connect x'_i (resp. x_i) to the corresponding $k+1$ variable vertices $(z_{i_1}, z_{i_2}, \ldots, z_{i_{k+1}})$. Applying this rule to each variable we get $(k+1)n$ edges in E' from E_2 and these edges do not conflict with each other (refer to Remark 3). Since all the clauses will have at least one literal satisfied in this assignment, every clause vertex can have an edge incident to some literal vertex that has no edges incident to any of the variable vertices. Consider a clause C_j which is satisfied by the assignment $y_i = 1$ (resp. $y_i = 0$). Add the edge (c_j, x_i) (resp. (c_j, x'_i)) to E'. Since all the clauses are satisfied, k edges from E_1 can be added to E'. Therefore, G' has $(k+1)n + k$ edges and it is a valid *GLGG*. □

Theorem 3 *Deciding whether there exists a GLGG with at least m edges for a given value of m is NP-complete.*

Proof. By Lemma 1 and Lemma 2, this problem is NP-hard. Given a geometric graph G', it can be verified in polynomial time whether G' is a valid *GLGG* with at least m edges. Thus, this problem is NP-complete. □

This reduction to argue NP-hardness can be extended further to show inapproximability for computing an edge maximum *GLGG*. Let us consider the optimization version of 3-SAT known as MAX-3-SAT. Here the objective is to find a binary assignment satisfying the maximum number of clauses. MAX-(3,4)-SAT is a special case of MAX-3-SAT with an additional restriction that a variable is present in exactly four clauses. MAX-(3,4)-SAT is shown to be *APX*-hard in [1].

Now we enhance our existing construction such that for each variable there are 5 variable vertices instead of $k+1$ as described in the previous reduction. Let $G = (V, E)$ be this new geometric graph on which an optimal *GLGG* has to be computed. Again edge sets

E_1 and E_2 are defined as earlier. Now, we present the following lemma that helps to prove that computing an edge maximum $GLGG$ is APX-hard.

Lemma 4 *If a GLGG G'_1 computed over G has less than $5n$ edges from E_2 then we can obtain another GLGG G'_2 over G with $5n$ edges from E_2 and $|E'_2| \geq |E'_1|$.*

Proof. Initially let $G'_2 = G'_1$. In G'_2 if a variable vertex z_{i_j}, $1 \leq j \leq 5$ has an edge incident to an associated literal vertex x_i, then x_i cannot have an edge incident to a clause vertex (refer to Remark 5). Now x_i can have edges incident to all the five variable vertices (refer to Remark 3). Therefore, if a variable vertex z_{i_j} has an edge incident to x_i and some other variable vertex $z_{i_{j'}}$ corresponding to the same variable has no edge incident to it, then an edge $(x_i, z_{i_{j'}})$ can be added to E'_2 without conflicting with any existing edge.

If no vertex z_{i_j}, $1 \leq j \leq 5$ has an edge incident to x_i, the solution can be improved locally. Add the edges $\{(x_i, z_{i_j}) | 1 \leq j \leq 5\}$ to E'_2 and remove any edges connecting x_i to the clause vertices from E'_2. Note that a variable occurs only in four clauses in a MAX-(3,4)-SAT instance, so a literal vertex cannot have edges incident to more than four clause vertices. Therefore, this transformation implies $|E'_2| \geq |E'_1|$. Applying this argument to all the variable vertices, it can be ensured that in G'_2 every variable vertex has an edge incident to it. Thus, E'_2 has $5n$ edges from E_2 and $|E'_2| \geq |E'_1|$. □

Theorem 5 *Computing an edge maximum GLGG on a given geometric graph $G = (V, E)$ is APX-hard.*

Proof. Let OPT_G and OPT_S denote the optimum for the $GLGG$ instance and the MAX-(3,4)-SAT instance respectively. A clause vertex can have only one edge incident to it (refer to Remark 2) and a $GLGG$ maximizing the edges will have $5n$ edges from E_2 (edges between variables vertices and literal vertices, refer to Lemma 4). Therefore, $OPT_G = 5n + OPT_S$. Let an algorithm maximizing the number of edges selects m edges from E_1 (edges between clause vertices and literal vertices) along with $5n$ edges from E_2. Each of these m edges implies a satisfied clause in the original MAX-(3,4)-SAT instance. Since MAX-(3,4)-SAT cannot be approximated beyond 0.99948 [1], $m < 0.99948 * OPT_S$. Let c be the best approximation bound for the edge maximum $GLGG$. Therefore, $c \leq \frac{5n + 0.99948 * OPT_S}{5n + OPT_S}$. Since any binary assignment or its complement would necessarily satisfy at least half of the clauses in any given 3-SAT formula, $OPT_S \geq \frac{k}{2}$. Here $n = \frac{3}{4}k$ implying $c \leq 0.999939$. Thus, it is NP-hard to approximate edge maximum $GLGG$ within a factor of 0.999939. □

Consider the *maximum weight LGG* problem where the edges are assigned weights and we have to compute an LGG maximizing the sum of the weights of the selected edges. The edge maximum $GLGG$ problem is a special case of the maximum weight LGG problem (edge weights are either 0 or 1).

Corollary 1 *Computing a maximum weight LGG is APX-hard.*

3 Dilation of LGG

Let us define dilation of a geometric graph $G = (V, E)$. Let $D_G(u, v)$ be the distance between two vertices in the geometric graph (sum of length of the edges in the shortest path) and $D_2(u, v)$ be the Euclidean distance between u and v. Let $\delta(u, v) = \frac{D_G(u,v)}{D_2(u,v)}$. The dilation of G is defined as $\delta(G) = \max_{u,v \in V, u \neq v} \delta(u, v)$. In this section, we focus on computational and combinatorial questions on dilation for $LGGs$.

3.1 Computation of a minimum dilation LGG

In this section we show that the problem of determining whether there exists an LGG on a given point set with dilation ≤ 7 is NP-hard. The reduction from the partition problem is motivated by a technique in [6], where it was shown that computing the minimum dilation geometric graph with bounded number of edges is NP-hard. Since our problem requires us to construct an LGG instead of any geometric graph with bounded number of edges, the construction needs to be substantially modified.

The partition problem is defined as follows: Given a set S of positive integers $r_i, 1 \leq i \leq s$ s.t. $\sum_{r \in S} r = 2R$, can it be partitioned into two disjoint sets S_1 and S_2 such that $\sum_{r \in S_1} r = \sum_{r \in S_2} r = R$? Given an instance of the partition problem, we construct a point set V such that the instance of the partition problem is a *yes* instance if and only if there exists an LGG on V with dilation ≤ 7. Define a parameter λ s.t. $2sr_{max}^2 < 10^{\lambda}$ where r_{max} is the largest element of S. For each $r_i \in S$, there is a gadget G_i. Define a parameter $\eta_i = 10^{-(\lambda+1)} r_i$ to be used in gadget G_i. Note that $\eta_i \leq \frac{1}{10}$.

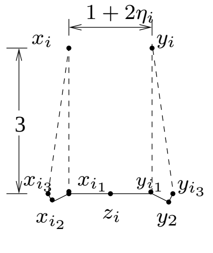


Figure 2: Structure of a basic gadget

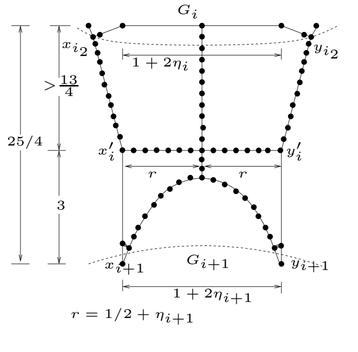


Figure 3: Basic frame structure

Now we explain the structure of a gadget G_i. Each gadget comprises of 9 points as shown in Figure 2. Points x_i and y_i are placed a distance of $1+2\eta_i$ apart. x_{i_1} and y_{i_1} are placed at the same distance such that $\overline{x_i x_{i_1}}$ and $\overline{y_i y_{i_1}}$ are parallel to each other and perpendicular to $\overline{x_{i_1} y_{i_1}}$. Vertex z_i is placed at the midpoint of the line segment $\overline{x_{i_1} y_{i_1}}$. $\overline{x_{i_1} x_{i_3}}$ is perpendicular to $\overline{x_i x_{i_1}}$ and the distance of x_{i_1} from x_{i_3} is $10\eta_i$. For $\epsilon_1 = \frac{10^{-3}}{s^2 10^{2\lambda}}$, x_{i_2} and x_{i_3} are placed at a distance of $c_1\epsilon_1$ along x-axis and $c_2\epsilon_1$ along y-axis for suitable constants c_1 and c_2, s.t. $\angle x_{i_1} x_{i_2} x_{i_3} \geq \frac{\pi}{2}$. Vertices y_{i_2} and y_{i_3} are placed similarly. We call edges $(x_{i_3}, x_{i_2}), (x_{i_2}, x_{i_1}), (x_{i_1}, z_i), (z_i, y_{i_1}), (y_{i_1}, y_{i_2})$ and (y_{i_2}, y_{i_3}) *basic edges*. It can be verified that an *LGG* over the vertices of a gadget must contain all the basic edges to keep dilation bounded by 7. It can be observed that any other edge will conflict with at least one basic edge with the exception that the point x_i can be connected to y_i, x_{i_1} or x_{i_3} and similarly y_i can be connected to x_i, y_{i_1} or y_{i_3}. Edges (x_i, x_{i_1}) and (y_i, y_{i_1}) are called *vertical edges* while (x_i, x_{i_3}) and (y_i, y_{i_3}) are called *slanted edges*. Note that the vertical edge and the slanted edge emerging from the same point x_i or y_i conflict with each other in an *LGG*. Additional points to be described later will ensure that there cannot exist a direct edge between x_i and y_i. Though both vertices x_i and y_i can have independently either a vertical or a slanted edge incident to them, if both vertices have slanted edges then $\delta(x_i, y_i) > 7$.

Remark 6 *In a gadget G_i, there can be only one slanted edge if $\delta(x_i, y_i) \leq 7$.*

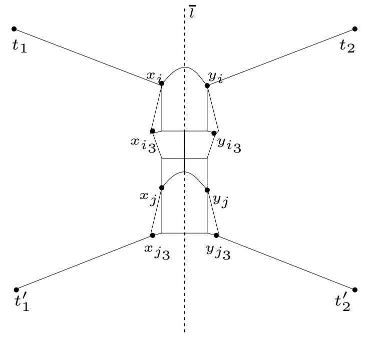


Figure 4: Layout of complete structure for $s = 2$

A frame F_i is used to connect two gadgets G_i and G_{i+1} as shown in Figure 3. It connects G_i at vertices x_{i2} and y_{i2} and connects G_{i+1} at vertices x_{i+1} and y_{i+1}. A frame also provides two symmetric paths ((x_{i+1}, x'_i, x_{i_2}) and (y_{i+1}, y'_i, y_{i_2})) between two consecutive gadgets. Let us denote this path length between i^{th} and $i+1^{th}$ gadget as $p_{i,i+1}$. All edges shown in the figure are part of the basic skeleton of a frame and these edges are included in the set of basic edges. Here we use a technique of placing vertices at very short distance (0.01 in our construction) from each other along a line. The purpose of this technique is to ensure that all these small edges are selected in the *LGG*. If such an edge is not selected then any alternate path does not bound the spanning ratio within limit. We call this technique *vertex closing*. It will ensure that in a frame, edges are taken only according to our layout. Such a sequence of vertices is called a *vertex chain*. An additional *auxiliary vertex* is placed in each gadget G_i at a distance of $\frac{\epsilon_1 \eta_i}{10s}$ from x_{i2} and y_{i2} along the lines $\overline{x_{i2} x'_i}, \overline{x_{i_2} x_{i_1}}, \overline{y_{i_2} y'_i}$ and $\overline{y_{i_2} y_{i_1}}$.

A frame also provides a convex cap on (x_i, y_i) in a gadget G_i. This is a convex point set with all the points above $\overline{x_i y_i}$ (it need not be a regular curve). There is a small edge incident to both x_i and y_i from this cap conflicting with the edge (x_i, y_i) and it ensures that x_i and y_i are not directly connected by an edge. It provides a path between x_i and y_i with spanning ratio just above 7 and for any other pair of vertices in it spanning ratio is bounded by 7. On the first gadget G_1, such a cap is placed explicitly as shown in Figure 4. Now the full structure is constructed as shown in Figure 4. There is a central vertical line $\bar{l}$ and all the gadgets are placed along it keeping vertex z of a gadget on $\bar{l}$ s.t. $\overline{x_{i_1} y_{i_1}}$ is perpendicular to $\bar{l}$ and a frame is placed between two gadgets. The vertical span for a frame F_i is $\frac{25}{4}$. There is a total of four extended arms, each of length h with *vertex closing* from G_1 and G_s, each making an angle $\sin^{-1}(\frac{220}{221})$ w.r.t. $\bar{l}$ (refer Figure 4). Here,

$$h = \frac{221}{148}(18s + (s-1)\frac{175}{4}) - \frac{k}{2} + \frac{1}{2}10^{-\lambda}R - \frac{1}{2}10^{-2\lambda}sr^2_{max}$$

where $k = \sum_{i=1}^{s-1} p_{i.i+1} + 10\sum_{i=1}^{s} \eta_i$.

Let V be the set of all points introduced above. Clearly $|V| = O(s)$. It can be verified that the description complexity of point set V is polynomial in the size of the partition instance.

Lemma 6 *If the partition problem S is solvable then there exists an LGG on V with dilation not exceeding 7.*

Lemma 7 *If there exists an LGG on V with dilation less than or equal to 7 then there exists a solution for the partition problem over S.*

Refer to full version [9] for the proofs of Lemma 6 and Lemma 7.

Theorem 8 *Given a point set P, it is NP-hard to find whether there exists an LGG with dilation less than or equal to a given value k.*

Proof. The proof can be inferred by Lemma 6 and Lemma 7. □

Let us present some simple combinatorial bounds on the dilation of *LGGs*.

Lemma 9 *There exists a point set P such that any LGG on P has dilation $\Omega(\sqrt{n})$.*

Lemma 10 *There exists a point set P such that the Gabriel Graph on P has dilation $\Omega(\sqrt{n})$ whereas there exists an LGG on P with dilation $O(1)$.*

Refer to full version [9] for the proofs of Lemma 9 and Lemma 10.

4 Verification Algorithm for *LGG*

Given a geometric graph $G = (V, E)$, let us consider the problem of deciding whether G is a valid *LGG*. It has to be verified that no two edges conflict with each other.

For any $u \in V$, let $\mathcal{L}_u$ be a circular list storing all

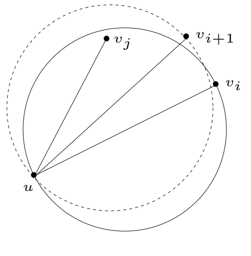


Figure 5: Checking for conflicts in an *LGG*

neighbors of vertex u in counterclockwise order. G is a valid *LGG* if edges from a vertex u to any two consecutive members in $\mathcal{L}_u$ do not conflict with each other $\forall u \in V$. This claim follows directly from the Lemma stated below.

Lemma 11 *Let u be any vertex in G and $\mathcal{L}_u = \{v_1, v_2, \ldots, v_l\}$. If edges (u, v_i) and (u, v_j) conflict with each other such that $i \leq j - 2$, then there exist a k such that $i \leq k \leq j - 1$ and the edge (u, v_k) conflicts with the edge (u, v_{k+1}).*

Proof. We give a proof by contradiction. Assume that the edges (u, v_i) and (u, v_j) conflict with each other and (u, v_k) does not conflict with (u, v_{k+1}) for any k, s.t $i \leq k < j$. Let us assume w.l.o.g. that (u, v_i) and (u, v_j) are the closest pair of conflicting and non-successive edges s.t. $i \leq j - 2$, i.e. if two edges (u, v'_i) and (u, v'_j) conflict with each other and $i \leq i' < j' \leq j$ then $j' = i' + 1$. Since (u, v_i) and (u, v_j) conflict with each other, let us assume w.l.o.g. that v_j lies within the circle with diameter $\overline{uv_i}$ as shown in Figure 5. By assumption (u, v_i) and (u, v_{i+1}) do not conflict, so v_{i+1} must lie outside this circle and similarly v_i will lie outside the circle with diameter $\overline{uv_{i+1}}$. Recall that two circles can intersect only at two points. Now it can be trivially observed that the circle with diameter $\overline{uv_{i+1}}$ will contain v_j. Thus, (u, v_{i+1}) and (u, v_j) conflict with each other. This implies that either (u, v_i) and (u, v_j) are not the closest pair of conflicting edges or (u, v_{i+1}) and (u, v_j) are successive edges and they do conflict with each other. In either case we have a contradiction of the original assumption. □

The argument above directly implies a verification algorithm for *LGG*. It involves computing $\mathcal{L}_u, \forall u \in V$ that can be done by angular sorting of the neighbors of each vertex. It can be implemented in $O(|E| \log |V|)$ time. Scanning each vertex u and verifying that edges to two consecutive members in $\mathcal{L}_u$ do not conflict takes $O(|V| + |E|)$ time. Therefore, this algorithm has time complexity of $O(|E| \log |V| + |V|)$.

References

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

[2] P. Bose, L. Devroye, W. S. Evans, and D. G. Kirkpatrick. On the spanning ratio of Gabriel graphs and beta-skeletons. *SIAM J. Discrete Math.*, 20(2):412–427, 2006.

[3] P. Bose, P. Morin, I. Stojmenović, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. In *Proceedings of the 3rd international workshop on Discrete algorithms and methods for mobile computing and communications*, pages 48–55, 1999.

[4] P. Erdős. On sets of distances of n points. *The American Mathematical Monthly*, 53(5):pp. 248–250, 1946.

[5] R. K. Gabriel and R. R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18(3):259–278, September 1969.

[6] P. Giannopoulos, R. Klein, C. Knauer, M. Kutz, and D. Marx. Computing geometric minimum-dilation graphs is NP-hard. *Int. J. Comput. Geometry Appl.*, 20(2):147–173, 2010.

[7] J. Jaromczyk and G. Toussaint. Relative neighborhood graphs and their relatives. *P-IEEE*, 80:1502–1517, 1992.

[8] S. Kapoor and X.-Y. Li. Proximity structures for geometric graphs. In *International Journal of Computational Geometry and Applications*, volume 20, pages 415–429, 2010.

[9] A. Khopkar and S. Govindarajan. On computing optimal locally Gabriel graphs. *CoRR*, abs/1110.1180, 2011.

[10] X.-Y. Li, G. Calinescu, and P.-J. Wan. Distributed construction of a planar spanner and routing for ad hoc wireless networks. In *IEEE INFOCOM*, pages 1268–1277, 2002.

[11] R. Pinchasi and S. Smorodinsky. On locally Delaunay geometric graphs. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 378–382, New York, NY, USA, 2004. ACM.

Edge Guards for Polyhedra in Three-Space

Javier Cano[*] Csaba D. Tóth[†] Jorge Urrutia[‡]

Abstract

It is shown that every polyhedron in $\mathbb{R}^3$ with m edges can be guarded with at most $\frac{27}{32}m$ edge guards. The bound improves to $\frac{5}{6}m + \frac{1}{12}$ if the 1-skeleton of the polyhedron is connected. These are the first non-trivial upper bounds for the edge guard problem for general polyhedra in $\mathbb{R}^3$.

1 Introduction

A *polyhedron* P in $\mathbb{R}^3$ is a compact set bounded by a piecewise linear manifold. Two points, a and b, are *visible* in a polyhedron P if the closed line segment ab is contained in P. For the edges of a polyhedron P, we adapt the notion of *weak visibility*: an edge e of P is visible from a point p if there is a point $q \in e$ such that p and q are visible in P. A set S of edges jointly *guard* P if every point $a \in P$ is visible from some edge in S. It is possible that a point $a \in P$ does not see any vertex of P [11], however, it is not difficult to show that every point $a \in P$ sees at least six edges of P. It follows that every polyhedron with m edges can be guarded by at most $m - 5$ edges.

It was conjectured [14] that any polyhedron of genus zero with m edges can be guarded with at most $\frac{m}{6}$ edge guards. This bound would be optimal apart from an additive constant: for every $k \in \mathbb{N}$, there are polyhedra P_k in $\mathbb{R}^3$ with $6(k+1)$ edges that require at least k edge guards [14], see Figure 1. The polyhedron P_k is the union of a flat tetrahedron T and k pairwise disjoint small tetrahedra attached to one facet of T such that their interiors cannot be seen from any of the edges of T. Since each small tetrahedron has to be guarded by one of its edges, P requires k edge guards.

In this paper, we prove that every polyhedron with m edges (and arbitrary genus) in $\mathbb{R}^3$ can be guarded by at most cm edges, where $c > 0$ is a constant strictly smaller than 1. This is the first nontrivial upper bound for the edge guard problem for general polyhedra. For every polyhedron P in $\mathbb{R}^3$, we choose a set of edges that jointly guard P as the union of two sets: (1) a set of edges that cover all vertices of P, and (2) at most 3/4 of the remaining edges.

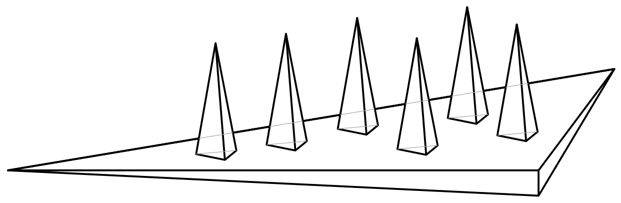


Figure 1: A polyhedron with m edges that requires $m/6 - 1$ edge guards.

The *1-skeleton* of a polyhedron P is the graph defined by the vertices and edges of P. An *edge cover* of a graph $G = (V, E)$ is a set of edges $E_1 \subseteq E$ such that every vertex $v \in V$ is incident to an edge in E_1. By placing guards at every edge in an edge cover of the 1-skeleton of P, we ensure that every point in P that sees a vertex is guarded. Note that the 1-skeleton of P is not necessarily connected (see Figure 1), even if P has genus zero. However, every connected component of the 1-skeleton is 3-connected. In Section 2, using classical matching theory, we give upper bounds for the size of a minimal edge cover in a 3-connected graph, and in a graph formed by the disjoint union of 3-connected components.

In Section 3, we 4-color the edges of P, and show that if a point $a \in P$ does not see any vertex of P, then it sees two edges of different colors. It follows that an edge cover $E_1 \subset E$ and the three smallest color classes of $E \setminus E_1$ jointly guard the entire polyhedron P.

Related work. Most of the previous research on art gallery problems focused on polygons in the plane. For example, it is well known that every simple polygon with n vertices can be guarded by at most $\lfloor \frac{n}{3} \rfloor$ point guards [3], and that every orthogonal polygon with n vertices can be guarded by $\lfloor \frac{n}{4} \rfloor$ point guards [7]. It is widely believed that every simple polygon with n vertices can be guarded by at most $\lfloor \frac{n+1}{4} \rfloor$ of its edges [10].

Everett and Rivera-Campo [6] showed that every triangulated polyhedral terrain in $\mathbb{R}^3$ with n vertices can be guarded by $\lfloor \frac{n}{3} \rfloor$ edges, as $\lfloor \frac{n}{3} \rfloor$ edges can cover all faces of a plane triangulation with n vertices. They also proved that the faces of every plane graph with n vertices can be guarded by $\lfloor \frac{2n}{5} \rfloor$ edges. See also [2] for

[*]Posgrado en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México, D.F. México, j_cano@uxmcc2.iimas.unam.mx

[†]Department of Mathematics and Statistics, University of Calgary, Calgary, AB, cdtoth@ucalgary.ca

[‡]Instituto de Matemáticas, Universidad Nacional Autónoma de México, D.F. México, urrutia@matem.unam.mx

other variants of guarding polyhedral terrains in $\mathbb{R}^3$

For *orthogonal* polyhedra with m edges in $\mathbb{R}^3$, it was conjectured that $\frac{m}{12}$ edge guards are always sufficient [14]. For every $k \in \mathbb{N}$, there are *orthogonal* polyhedra P_k in $\mathbb{R}^3$ with $12(k+1)$ edges that require at least k edge guards [14]. Recently, Benbernou et al. [1] showed that $\frac{11m}{72}$ edges are always sufficient.

Benbernou et al. [1] also introduced a variant of the problem with *open* edge guards. An open edge e of P is visible from a point p if there is a point q in the relative interior of e such that p and q are visible in P. They showed that every orthogonal polyhedron of genus g with m edges can be guarded with $\frac{11m}{72} - \frac{g}{6} - 1$ open edge guards.

2 Edge covers in 3-connected graphs

An *edge cover* of a graph $G = (V, E)$ is a set of edges $E_1 \subseteq E$ such that every vertex $v \in V$ is incident to an edge in E_1. A minimum edge cover is the union of a maximum matching $M \subset E$ and one extra edge for each vertex not covered by M. Hence the size of a minimum edge cover is $|V| - |M|$.

Nishizeki and Baybars [2, 9] proved that the maximum matching in a 3-connected planar graph with n vertices has at least $(n+4)/3$ edges; and so every such graph has an edge cover of size at most $(2n-4)/3$. An edge cover of this size can be computed in $O(n)$ time [12]. If G is a maximal planar graph (a triangulation) with $n \geq 3$ vertices and $m = 3n-6$ edges, then G has an edge cover of size at most $\frac{2}{9}m$. However, we are interested in the minimum edge cover of an *arbitrary* 3-connected graph in terms of the number of *edges*, rather than the number of vertices of the graph.

We recall a few technical terms and the Edmonds-Gallai Structure Theorem for maximal matchings [8, 15]. Let $G = (V, E)$ be a simple graph. A matching $M \subset E$ is *perfect* if it covers all vertices of G; it is *near perfect* if it covers all but one vertex of G. According to the Edmonds-Gallai Structure Theorem, if $M \subset E$ is a maximum matching of G, then there is a vertex set $U \subseteq V$ (a Berge-Tutte witness set) with the following properties:

- M contains a perfect matching on every even component of $G[V \setminus U]$;
- M contains a near perfect matching on every odd component of $G[V \setminus U]$;
- M matches all vertices of U to vertices in distinct odd components of $G[V \setminus U]$.

A minimum edge cover of G can be obtained by augmenting the maximum matching M with one extra edge for each odd component of $G[V \setminus U]$ that is not fully covered by M. We are now in the position to prove the following lemma.

Lemma 1 *Every 3-connected graph with $n \geq 4$ vertices and m edges contains an edge cover of size at most $\lfloor (m+1)/3 \rfloor$. This bound is the best possible.*

Proof. Let $G = (V, E)$ be a 3-connected planar graph $|V| \geq 4$ vertices and $m = |E|$ edges. Let $M \subseteq E$ be a maximum matching of G. The Edmonds-Gallai Structure Theorem yields a Berge-Tutte witness set $U \subset V$.

If $U = \emptyset$, then $G[V \setminus U] = G$ has a unique connected component, in which M is a perfect or near perfect matching with at least $\lfloor |V|/2 \rfloor$ edges. In this case, G has an edge cover of size $\lceil |V|/2 \rceil$. Since G is 3-connected, the minimum vertex degree is 3, and $m \geq \lceil \frac{3}{2}|V| \rceil$. Then G has an edge cover of size at most $\lfloor (m+1)/3 \rfloor$.

Assume now that $U \neq \emptyset$. Denote the components of $G[V \setminus U]$ by $G_i = (V_i, E_i)$, for $i = 1, 2, \ldots, \ell$. Let $\overline{E}_i \subset E$ denote the set of all edges incident to vertices in V_i, that is, all edges in E_i and edges between U and V_i. The edge sets E_i, $i = 1, \ldots, \ell$, are pairwise disjoint. Since G is 3-connected, the minimum vertex degree is 3, and so the sum of degrees of the vertices in V_i is at least $3|V_i|$. Also, at least 3 edges in $\overline{E}_i$ are incident to some vertices in U. Hence $|\overline{E}_i| \geq \frac{3}{2}(|V_i| + 1)$.

If $|V_i|$ is even, then M contains a perfect matching on G_i, with $\frac{1}{2}|V_i|$ edges. Hence, the maximum matching M contains less than one third of the edges of $\overline{E}_i$.

If $|V_i|$ is odd, then M contains a near perfect matching on G_i, with $\frac{1}{2}(|V_i| - 1)$ edges. A minimum edge cover of G contains one more edge of $\overline{E}_i$ between U and V_i. Altogether, a minimum edge cover of G contains at most $\frac{1}{2}(|V_i| + 1)$ edges of $\overline{E}_i$. On the other hand, $|\overline{E}_i| \geq \frac{3}{2}(|V_i| + 1)$. Hence, a minimum edge cover contains at most a third of the edges of $\overline{E}_i$. Altogether, an upper bound $m/3$ follows in this case.

The bound $\lfloor (m+1)/3 \rfloor$ is the best possible. If $m \equiv 0$ or $m \equiv 1 \mod 3$, then the lower bound construction is a bipartite graph with vertex classes U and $V \setminus U$, where every vertex in $V \setminus U$ has degree 3. If $m \equiv 2 \mod 3$, then the lower bound construction is the 1-skeleton of a pyramid with a square base with 5 vertices and 8 edges (Figure 2). The base of the pyramid can be extended to a ladder for larger values of m. $\square$

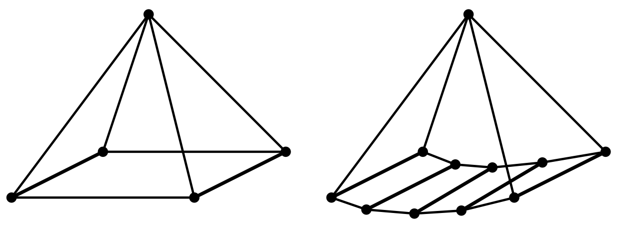


Figure 2: Lower bound constructions for $m \equiv 2 \mod 3$.

The 1-skeleton of a polyhedron in $\mathbb{R}^3$ is not necessarily connected (see Figure 1). However, each component of the 1-skeleton is 3-connected and has at least 4 vertices. For the edge cover of the 1-skeleton of a polyhedron, we derive the following corollary.

Corollary 2 *Let G be a graph such that every connected component of G is 3-connected and has at least 4 vertices. Then G has an edge cover with at most $\lfloor \frac{3m}{8} \rfloor$ edges. This bound is the best possible.*

Proof. Let $G_1, \ldots, G_k$ be the connected components of G, with $m_1, \ldots, m_k$ edges each. By Lemma 1, for each G_i we find an edge cover of size at most $\lfloor \frac{m_i+1}{3} \rfloor \leq \lfloor \frac{m_i}{3} \rfloor + 1$. Note that $\lfloor \frac{m_i+1}{3} \rfloor = \frac{m_i+1}{3}$ if $m_i \equiv 2 \mod 3$, and $\lfloor \frac{m_i+1}{3} \rfloor \leq \frac{m_i}{3}$ otherwise. Since $\sum_{i=1}^{k} m_i = m$, then

$$\sum_{i=1}^{k} \left\lfloor \frac{m_i+1}{3} \right\rfloor \leq \frac{m+k'}{3},$$

where k' is the number of components with $m_i \equiv 2 \mod 3$. Any such component has at least 8 edges, and so $k' \leq \lfloor \frac{m}{8} \rfloor$. It follows that

$$\frac{m+k'}{3} \leq \frac{m+\lfloor m/8 \rfloor}{3} \leq \frac{m+m/8}{3} = \frac{3m}{8},$$

as required. This bound is tight if each component of G is a square pyramid as in Figure 2(left). □

3 Four-coloring of edges in a polyhedron

Let P be a polyhedron with m edges (and arbitrary genus). Let $G = (V, E)$ denote the 1-skeleton of P. We may assume, by rotating P if necessary, that no edge in E is parallel to any coordinate plane. This ensures that the two endpoints of each edge $e \in E$ have distinct x- (resp., y- and z-) coordinates. We interpret above-below relation with respect to the z-axis (that is, a point a is above point b if a has a larger z-coordinate than b); and the left-right relation with respect to the y-axis. Recall that the boundary of P is a piecewise linear manifold, and so every edge $e \in E$ is incident to exactly two facets of P.

We distinguish between four types of edges in E as follows. For every edge $e \in E$, let H_e denote the plane spanned by e and a vertical line intersecting e. The plane H_e decomposes $\mathbb{R}^3$ into two halfspaces, lying on the left and the right of H_e. We say that e is a **left** edge if both facets incident to e lie in the left halfspace of H_e; edge e is a **right** edge if both facets incident to e lie in the right halfspace of H_e. The edge e is an **upper** edge if the two facets incident to e are in opposite halfspaces of H_e, and the interior of P lies below both facets. Edge e is a **lower** edge if the two facets incident to e are in opposite halfspaces of H_e, and the interior of P lies above both facets. See Figure 3 for examples.

We can now 4-color the edges of P such that the color classes correspond to the left, right, upper, and lower edges, respectively. We prove the following property of the 4-coloring.

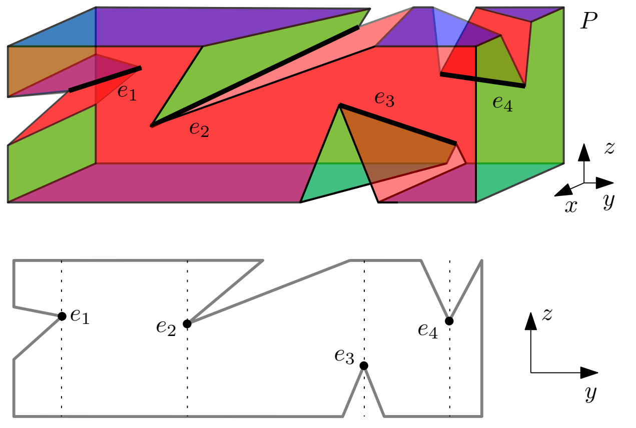


Figure 3: Top: An left edge e_1, a right edge e_2, a lower edge e_3, and an upper edge e_4 in a polyhedron P. Bottom: The cross-section of the polyhedron P with a plane parallel to the yz-plane, which is stabbed by edges $e_1, \ldots, e_4$. Dotted lines indicate the vertical lines passing through the the stabbing points of $e_1, \ldots, e_4$.

Lemma 3 *If a point $a \in P$ does not see any vertex of P, then a sees edges in at least two color classes.*

Proof. Let $a \in P$ be a point in the polyhedron P that does not see any vertex of P. Suppose that a sees edges of at most one color class. We distinguish four cases based on the color of the edges visible from a. By symmetry, it is enough to consider two out of four cases: left edges (the case of right edges is analogous), and upper edges (the case of lower edges is analogous).

Left edges. Suppose that every edge visible from a is a left edge. Consider the cross section of the polyhedron P with a plane H_a containing a and parallel to the yz-plane. Refer to Figure 4. The intersection $H_a \cap P$ may have several components, let P_a denote the component that contains a. Note that P_a is a 2-dimensional polygon, with possible holes. The vertices of P_a correspond to edges of P: each vertex of P_a is the intersection point of an edge of P with the plane H_a. Let V_a^* denote the set of reflex vertices of P_a that correspond to left edges of P. If $v \in V_a^*$, then the two edges of P_a incident to v lie on the left of v, and so the angle bisector of v is on the right side of a.

Decompose the polygon P_a as follows. Consider the vertices in V_a^* in an arbitrary order. From each vertex $v \in V_a^*$ successively shoot a ray along its angle bisector, and draw a segment along the ray from v to the first point where the ray hits the boundary of P_a or a previously drawn segment. If a ray hits a vertex, perturb the ray slightly so that it does not end at any vertex.

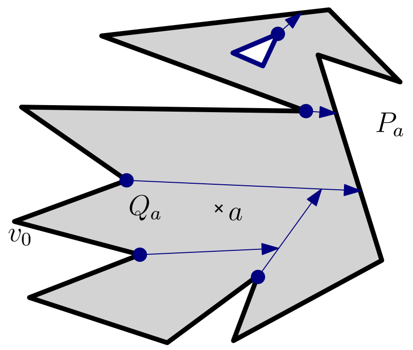


Figure 4: The polygon P_a is the cross-section of the polyhedron P with the plane H_a containing a and parallel to the xz-plane. The vertices in V_a^* are marked with large dots. P_a is decomposed into subpolygons by rays emitted by the vertices in V_a^*. The subpolygon Q_a contains a. Since Q_a is convex, a sees the leftmost vertex v_0 of Q_a.

The segments decompose P_a into subpolygons. Denote by $Q_a \subseteq P_a$ a subpolygon containing the point a, and let v_0 be the leftmost vertex of Q_a. Note that Q_a is a convex polygon, otherwise a sees a reflex vertex of Q_a which does not correspond to a left edge, since it would have no segment drawn along its bisector, contradicting the assumption that a only sees left edges. Since Q_a is convex, we have $av_0 \subset Q_a \subset P_a$, that is, v_0 is visible from a. Since all bisector rays are directed from left to right, v_0 has to be a vertex of the polygon P_a. Both edges of Q_a incident to v_0 are on the right side of v_0, as it is the leftmost vertex; and at least one of them has to be an edge of P_a, since every vertex of P_a emits at most one ray along its bisector. Therefore, v_0 does not correspond to a left edge of P. We have shown that a sees a non-left edge of P, contradicting our initial assumption.

Upper edges. Suppose that every edge visible from a is an upper edge. We decompose the polyhedron P into polyhedral cells such that each cell has exactly two nonvertical facets, which bound the cell from above and from below, respectively. We use (the first phase of) the standard vertical decomposition method [4, 13]. For every point p in every edge $e \in E$, erect a maximal vertical segment s_p such that $p \in s_p \subset P$. For an edge $e \in E$, the segments s_p, $p \in e$, form a vertical simple polygon A_e (which we call a *vertical wall*) whose upper and lower boundaries are contained in the boundary of P. The polygons A_e, $e \in E$, jointly decompose P into cells. Each cell has exactly two nonvertical facets, bounding the cell from above and below, respectively, and are contained in some facets of P; all other facets are contained in vertical walls corresponding to some edges of E. Due to the vertical walls A_e, $e \in E$, every cells has convex dihedral angles along the edges of the polyhedron P. A cell may still have a reflex dihedral angle at a *vertical* edge (e.g., consider the vertical decomposition of the polyhedron in Figure 1).

Denote by T_a a cell containing a. If point a sees some point p in a vertical wall A_e on the boundary of T_a, for some $e \in E$, then a sees the point $q \in e$ vertically above or below p. Recall that only upper edges of P are visible from a, hence every vertical wall A_e on the boundary of T_a visible from a corresponds to an upper edge $e \in E$.

We show that a sees some vertex of P. Assume first that T_a is nonconvex and so a sees some reflex edge e_r of T_a. Then e_r is a point p in a vertical edge of T_a, which lies on the boundary of two vertical walls, as noted above. Necessarily, a also sees a point vertically above p on the boundary of P, which is a vertex of P. Next assume that T_a is convex. Then every edge corresponding to a vertical wall on the boundary of T_a is incident to the top facet of T_a. Therefore, the top facet of T_a is bounded by edges of E, and hence it is a facet of P. Any vertex of the top facet of T_a is a vertex of P, and visible from a by convexity. We have shown in both cases that a sees some vertex of P. This contradicts our assumption that a does not see any vertex of P, and completes the proof. □

4 Obtaining the set of guards

The combination of the results in Sections 2 and 3 leads to the following bound on the minimum number of edge guards in a polyhedron.

Lemma 4 *Let P be a polyhedron with m edges in $\mathbb{R}^3$ (with arbitrary genus), and let E_1 be an edge cover of the 1-skeleton of P. Then P can be guarded by at most $(3m + |E_1|)/4$ edge guards.*

Proof. Four-color the edges of the 1-skeleton of P as described in Section 3. Place guards at all edges of E_1, and at the three smallest color classes of the remaining edges. Altogether, we use at most

$$|E_1| + \frac{3}{4}(m - |E_1|) = \frac{3m + |E_1|}{4}$$

edge guards. If a point $a \in P$ sees a vertex v, then it is guarded by an edge in E_1 that covers v. If a point $a \in P$ does not see any vertex of P, then it sees edges in at least two color classes by Lemma 3, and so it is guarded by an edge in one of the three smallest color classes. □

Finally, we prove our main results.

Theorem 5 *Every polyhedron in $\mathbb{R}^3$ with m edges (and arbitrary genus) can be guarded with at most $\frac{27}{32}m$ edge guards.*

Proof. Let P be a polyhedron with m edges in $\mathbb{R}^3$ with arbitrary genus. Let G be the 1-skeleton of P, and note that every connected component of G is 3-connected with at least 4 vertices. By Corollary 2, G has an edge cover E_1 of size $|E_1| \leq \frac{3m}{8}$. By Lemma 4, P can be guarded by at most

$$\frac{3m + |E_1|}{4} \leq \frac{3m + \frac{3m}{8}}{4} = \frac{27m}{32}$$

edges, as claimed. □

If the 1-skeleton of P is connected, we can establish a better upper bound.

Theorem 6 *Every polyhedron in $\mathbb{R}^3$ with m edges (and arbitrary genus) and a connected 1-skeleton can be guarded with at most $\frac{5}{6}m + \frac{1}{12}$ edge guards.*

Proof. Let P be a polyhedron with m edges in $\mathbb{R}^3$ with arbitrary genus. Let G be the 1-skeleton of P. By Lemma 1, G has an edge cover E_1 of size $|E_1| \leq \frac{m+1}{3}$. By Lemma 4, P can be guarded by at most

$$\frac{3m + |E_1|}{4} \leq \frac{3m + \frac{m+1}{3}}{4} = \frac{10m + 1}{12}$$

edges, as claimed. □

Using the same technique, one can also show that if the 1-skeleton of P is a triangulation with m edges, then it has an edge cover of size at most $\frac{2}{9}m$, and it can be guarded by at most $\frac{29m}{36}$ edge guards.

References

[1] N. M. Benbernou, E. D. Demaine, M. L. Demaine, A. Kurdia, J. O'Rourke, G. Toussaint, J. Urrutia, and G. Viglietta, Edge-guarding orthogonal polyhedra, in *23rd Canadian Conf. Comput. Geom.*, Toronto, 2011.

[2] P. Bose, D. Kirkpatrick, and Z. Li. Worst-case-optimal algorithms for guarding planar graphs and polyhedral surfaces, *Comput. Geom.* **26** (2003), 209–219.

[3] V. Chvátal, A combinatorial theorem in plane geometry, *J. Combin. Theory Se. B* **18** (1976), 39–41.

[4] K. L. Clarkson, H. Edelsbrunner, L. J. Guibas, M. Sharir, and E. Welzl, Combinatorial complexity bounds for arrangements of curves and spheres, *Discrete Comput. Geom.* **5** (1) (1990), 99–160.

[5] S. Devadoss and J. O'Rourke, Discrete and Comptuational Geometry, Princeton University Press, 2011.

[6] H. Everett and E. Rivera-Campo, Edge guarding polyhedral terrains, *Comput. Geom.* **7** (3) (1997), 201–203.

[7] J. Kahn, M. Klawe, and D. Kleitman, Traditional galleries require fewer watchmen, *SIAM J. Algebraic and Discrete Methods* **4** (1983), 194–206.

[8] L. Lovász and M. D. Plummer, Matching Theory, vol. 29 of *Ann. Discrete Math.*, North-Holland, Amsterdam, 1986.

[9] T. Nishizeki and I. Baybars, Lower bounds on the cardinalitiy of the maximum matchings of planar graphs, *Discrete Math.* **28** (1979), 255–267.

[10] J. O'Rourke, Galleries need fewer mobile guards: a variation on Chvátal's theorem, *Geometriae Dedicata* **14** (1983), 273–283.

[11] J. O'Rourke, Art Gallery Theorems and Algorithms, Oxford University Press, 1987.

[12] I. Rutter and A. Wolff, Computing large matchings fast, *ACM Trans. Algorithms* **7** (2010), article # 1.

[13] M. Sharir and P. K. Agarwal, Davenport-Schinzel Sequences and Their Geometric Applications, Cambridge University Press, 1995.

[14] J. Urrutia, Art gallery and illumination problems, in *Handbook of Computational Geometry*, pp. 973–1027, North-Holland, Amsterdam, 2000.

[15] D. B. West, A short proof of the Berge-Tutte Formula and the Gallai-Edmonds Structure Theorem, *European J. Combin.* **32** (2011), 674–676.

Hidden Mobile Guards in Simple Polygons[*]

Sarah Cannon[†] Diane L. Souvaine[‡] Andrew Winslow[§]

Abstract

We consider guarding classes of simple polygons using mobile guards (polygon edges and diagonals) under the constraint that no two guards may see each other. In contrast to most other art gallery problems, existence is the primary question: does a specific type of polygon admit *some* guard set? Types include simple polygons and the subclasses of orthogonal, monotone, and starshaped polygons. Additionally, guards may either exclude or include the endpoints (so-called *open* and *closed* guards). We provide a nearly complete set of answers to existence questions of open and closed edge, diagonal, and mobile guards in simple, orthogonal, monotone, and starshaped polygons, with some surprising results. For instance, every monotone or starshaped polygon can be guarded using hidden open mobile (edge or diagonal) guards, but not necessarily with hidden open edge or hidden open diagonal guards.

1 Definitions

We define the *boundary* of a polygon P (denoted ∂P) as a simple polygonal chain consisting of a sequence of vertices specified in counterclockwise order, and the open set enclosed by ∂P to be the *interior* of P (denoted $\text{int}(P)$). An *edge* $e = \overline{pq}$ of the polygon is an interval of ∂P between consecutive vertices p, q, and a *diagonal* $d = \overline{rs}$ of P is a straight line segment between non-consecutive vertices r, s of ∂P such $d - \{r, s\} \in \text{int}(P)$, i.e. the portion of d excluding its endpoints lies in the interior of P.

We consider guarding $\text{int}(P)$ using a subset of the edges and diagonals of P. A guard g *sees* or *guards* a location l in the polygon if l is *weakly visible* [1] from the guard: there exists a point $p \in g$ such that the interior of the segment lp lies in the interior of the polygon. Edges and diagonals selected as guards are called *edge guards* and *diagonal guards*, respectively, and a *mobile guard* [9] is either an edge or a diagonal guard. If a set S of edges and diagonals of P is such that every location in the interior of P is seen by at least one guard in S, then S is a *guard set* of P and P is said to *admit* a guard set. A *closed guard set* includes the vertices at both ends of each edge or diagonal. If all endpoints are excluded, the guard set is called an *open guard set*.

In addition to *simple polygons* or simply *polygons*, we consider a number of special classes of polygons. An *orthogonal polygon* is a polygon that can be rotated such that all edges are parallel to the x- or y-axis. A *monotone polygon* is a polygon that can be rotated such that the portion of the polygon intersecting any vertical line consists of a connected interval. A *starshaped polygon* is a polygon that can be translated such that an interior point coincides with the origin and sees all locations in the interior of the polygon, and the *kernel* of the polygon is the set of all points in the polygon with this property. These three classes (along with convex and spiral polygons) are described by O'Rourke [10] in the context of guarding problems as being "usefully distinguished in the literature."

Finally, we add the constraint that a guard set is *hidden*: no pair of guards in the set see each other. Here a pair of guards g_1, g_2 in a polygon P can see each other if there exists a pair of points $p \in g_1, q \in g_2$ such that $pq - \{p, q\} \in \text{int}(P)$.

2 Introduction

Edge, diagonal, and mobile guards in polygons have been studied extensively in the past. Avis and Toussaint [1] considered the case where a single closed edge is sufficient to guard the entire polygon. Shortly after, Toussaint gave an example of a polygon whose smallest closed edge guard set is $\lfloor n/4 \rfloor$ [9] and conjectured that an edge guard set of this size is sufficient for any polygon. O'Rourke [9] showed that closed mobile guard sets of size $\lfloor n/4 \rfloor$ are sometimes necessary and always sufficient for polygons. For closed diagonal guards, Shermer [12] has shown that guard sets of size $\lfloor (2n+2)/7 \rfloor$ are necessary for some polygons, and no polygon requires a guard set of size greater than $\lfloor (n-1)/3 \rfloor$.

More recently, open edge guards were suggested by Viglietta [14] and studied by Benbernou et al. [2] and Tóth et al. [13], who showed that open edge guard sets

[*]An abstract version of this paper was presented at the 21st Fall Workshop on Computational Geometry, 2011. A full version of this paper containing all proofs is available at `http://arxiv.org/pdf/1206.1803v1`. Research supported in part by NSF grants CCF-0830734 and CBET-0941538.

[†]Department of Computer Science, Tufts University, Medford, MA, USA `scanno01@cs.tufts.edu`

[‡]Department of Computer Science, Tufts University, Medford, MA, USA `dls@cs.tufts.edu`

[§]Department of Computer Science, Tufts University, Medford, MA, USA `awinslow@cs.tufts.edu`

of size $\lfloor n/3 \rfloor$ and $\lfloor n/2 \rfloor$ are sometimes necessary and always sufficient for simple polygons.

The study of hidden guards began with Shermer [11] who gave several results, including examples of polygons that are not guardable using hidden vertex guards. The study of hidden edges has only been initiated recently by Kranakis et al. [6] who showed that computing the largest hidden open edge set in a polygon (ignoring guarding) cannot be approximated within an arbitrarily small constant factor unless P = NP. In the same theme, Kosowoski et al. [7] have studied cooperative mobile guards, where each guard is *required* to be seen by another guard. Such a constraint is the opposite of hiddenness, which *forbids* any guard from seeing any other guard.

Here we evaluate the existence of hidden edge, diagonal, and mobile guard sets for simple polygon classes. A summary of results is seen in Table 1.

Guard class		Polygon class			
Inclusion	Type	Simple	Ortho	Mono	Star
Open	Edge	No	Yes	No	No
	Diagonal	No	No	No	No
	Mobile	No	Yes	Yes	Yes
Closed	Edge	No	No	No	No
	Diagonal	No	No	No	No
	Mobile	No	No	No	?

Table 1: New results in this paper. Entries indicate whether a hidden guard set exists for every polygon in the class.

3 Open edge guards

Recall open edge guards are edges of the polygon excluding the endpoints.

Lemma 1 *There exists a monotone polygon that does not admit a hidden open edge guard set.*

Proof. See Figure 1. We refer to the convex regions bounded by three edges in the upper left and right portions of the polygon as *ears*. Consider guarding the pair of ear regions without using any of the three edges that form each ear. The cases resulting from these attempts are seen in Figure 2. In each case, any maximal combination of non-ear edges fails to guard either ear completely. Moreover, a portion of the remaining unguarded region in each ear is not visible from any edge of the other ear. Thus any guard set contains one of the three edges in each ear. Also, every pair of ear edges in the same ear see each other, so any guard set contains exactly one edge in each ear.

Next, consider possible ear-edge pairs containing one edge from each ear. In Figure 3 it is shown that for

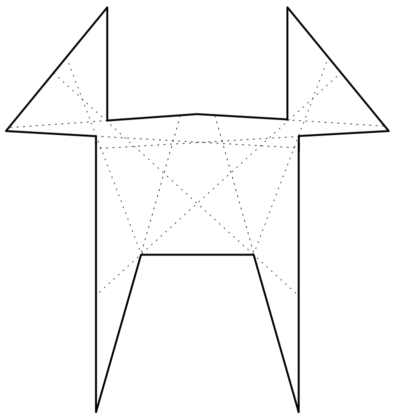


Figure 1: A monotone polygon that does not admit a hidden open edge guard set.

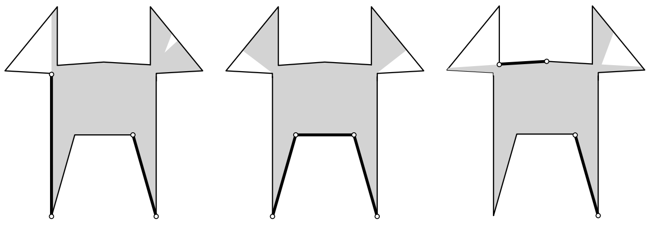


Figure 2: All maximal combinations of open edge guards that exclude the six ear edges.

each such ear-edge pair, the pair cannot be augmented to form a hidden open edge guard set for the polygon. Thus the polygon cannot be guarded with hidden open edge guards. □

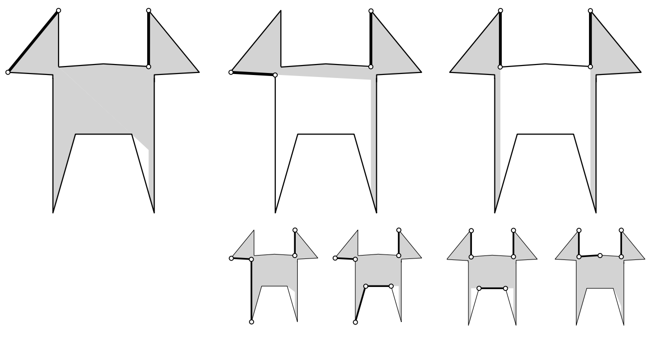


Figure 3: All combinations of ear edge pairs and the maximal hidden sets containing each ear edge pair.

Lemma 2 *There exists a starshaped polygon that does not admit a hidden open edge guard set.*

Proof. See Figure 4. The polygon consists of a central convex region with numerous spikes emanating from it. Figure 6 provides a labeled version of the polygon, with two sets of four large spikes each ($\{a_i\}$ and $\{b_i\}$) and

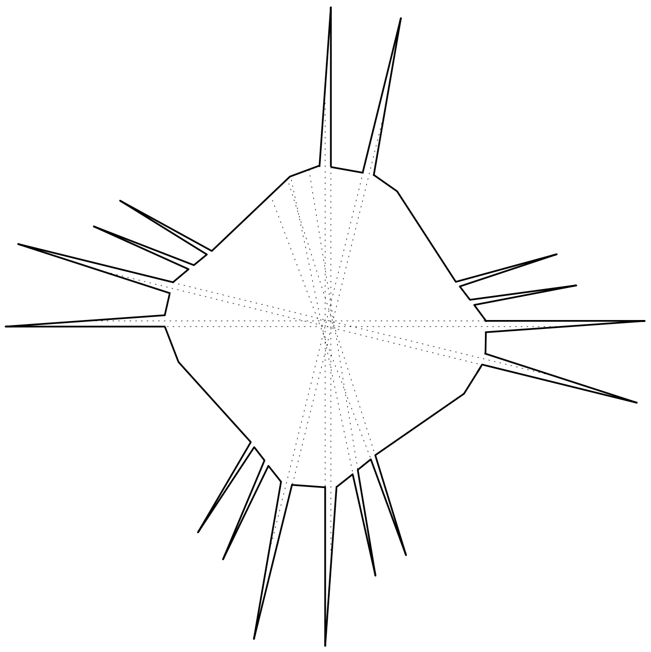


Figure 4: A starshaped polygon that does not admit a hidden open edge guard set.

four sets of two small spikes each ($\{c_1, c_2\}$ forms one such set). Call edges on the central convex region *central edges* and the spike pairs $\{a_1, a_3\}$, $\{a_2, a_4\}$, $\{b_1, b_3\}$, $\{b_2, b_4\}$ *opposing spike pairs.* Consider guarding the four spikes $\{a_i\}$ without using central edges (see Figure 5).

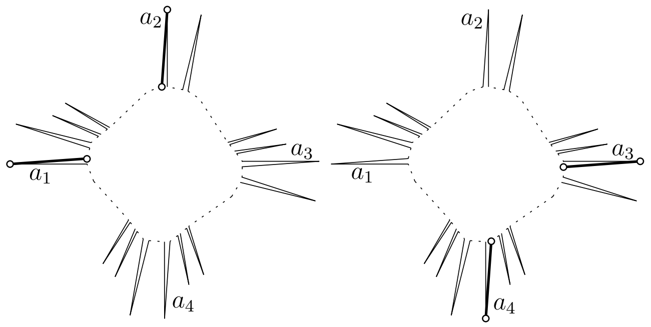


Figure 5: The two possible guardings of the four spike $\{a_i\}$ without using central edges (dotted).

Only one edge per opposing spike pair may be in any hidden edge guard set, as all four edges of an opposing spike pair see each other. Each spike has two asymmetric edges; one is able to guard the entire opposing spike pair, while the other is not. Each spike also contains a location not seem by any spike edge not in the spike's opposing spike pair. Finally, a pair of edges from a_1 and a_4 see each other, as do a pair in a_2 and a_3. So any hidden edge guard set for the opposing spike pairs $\{a_1, a_3\}$ and $\{a_2, a_4\}$ that does not include central edges consists of one of two pairs seen in Figure 5.

Now consider guarding the entire polygon. Any central edge guards the interior of at most one spike from $\{a_i\}$ or $\{b_i\}$. So one of the two spike sets $\{a_i\}$ and $\{b_i\}$ must be guarded without using central edges. Without loss of generality, assume the $\{a_i\}$ set is guarded in this way. Then one of the pairs of edges seen in Figure 5 must be in the guard set. Again without loss of generality, assume the edge pair of a_1 and a_2 are selected, as in Figure 6. Then there exist two spikes c_1 and c_2 whose edges are both seen by the guard edges in spikes a_1 and a_2, but portions of the interiors of c_1 and c_2 remain unguarded. The only edges sufficient to guard the interiors of c_1 and c_2 are the central edges e_1 and e_2. However, e_1 and e_2 each guard the interior of only one spike. Thus a portion of the interior of either c_1 or c_2 must remain unguarded, and the polygon cannot be guarded using hidden open edge guards. □

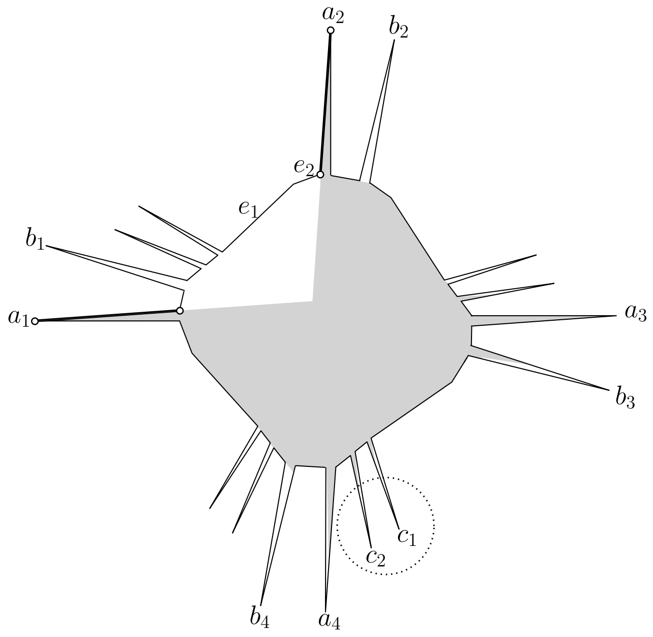


Figure 6: A incomplete but necessary set of guard edges and the region they guard. The interiors of c_1 and c_2 remain partially unguarded and cannot be guarded with a hidden open edge set.

Lemma 3 *Every orthogonal polygon admits a hidden open edge guard set.*

Omitted proofs can be found in the full version[1] of this paper.

4 Open diagonal guards

Lemma 4 *There exists a monotone and starshaped polygon that does not admit a hidden open diagonal guard set.*

[1]`http://arxiv.org/pdf/1206.1803v1`

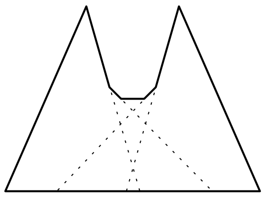


Figure 7: A monotone and starshaped polygon that does not admit a hidden open diagonal guard set.

Lemma 5 *There exists an orthogonal polygon that does not admit a hidden open diagonal guard set.*

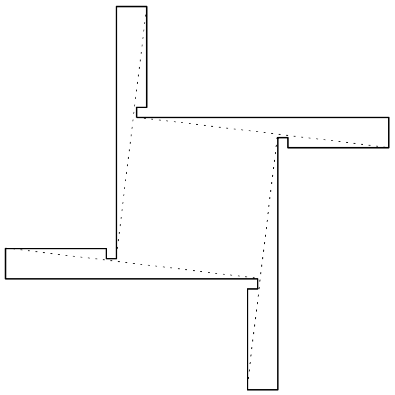


Figure 8: An orthogonal polygon that does not admit a hidden open diagonal guard set.

5 Open mobile guards

Lemma 6 *There exists a simple polygon that does not admit a hidden open mobile guard set.*

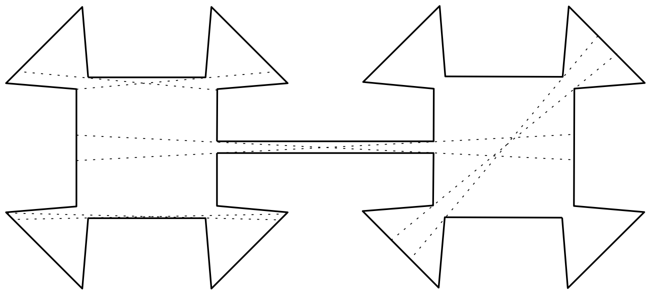


Figure 9: A simple polygon that does not admit a hidden open mobile guard set.

Observation 1 *Let g be a geodesic path between a pair of vertices p, q in a polygon P. Then the interiors of the edges g form a set of hidden open mobile guards in P.*

We refer to such a guard set for a path g as the *open mobile guard set induced by g*.

Lemma 7 *Every monotone polygon admits a hidden open mobile guard set.*

A natural approach to finding an open mobile guard set for a starshaped polygon is to look for a mobile guard that intersects the *kernel* of the polygon. Unfortunately such a guard may not exist as noted in [10] (see Figure 10).

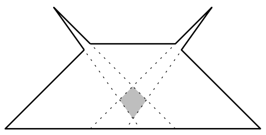


Figure 10: A starshaped polygon with no edge or diagonal intersecting its kernel (gray).

The following lemma is used in the proof of Lemma 9.

Lemma 8 *Let P be a starshaped polygon translated so that the origin lies in the kernel of P, and let v, v' be consecutive reflex vertices such that angle between the rays from v and v' through the origin (sweeping from v to v') is at most π. If a geodesic path $g \in P$ intersects both rays either before or after they intersect the origin, then g guards the subpolygon R bounded by the portions of the two rays before they intersect the origin, and the portion of ∂P from v to v'.*

Lemma 9 *Every starshaped polygon admits a hidden open mobile guard set.*

Proof. Let P be a given starshaped polygon translated so that the origin lies in the kernel of P. Consider shooting rays from each reflex vertex through the origin as seen in the left portion of Figure 11. Find a double wedge W formed by a consecutive pair of these rays such that each wedge is coincident to exactly one reflex vertex (which we call u and u') as seen in right portion of Figure 11 as a dark gray region. Such a double wedge is formed by every pair of consecutive intersections of rays along ∂P such that one intersection is the start of a ray (at a reflex vertex of P), and the other is the termination of a ray.

For every consecutive pair of reflex vertices v, v' on ∂P, the rays from v and v' through the origin lie entirely in $P - W$. Two pairs are an exception: the two pairs containing u and u' that form a pair of wedges, each containing half of the double wedge W (seen as the dark gray double wedge extended with two light gray wedges in the right portion of Figure 11). For all remaining pairs, the geodesic path from u to u' intersects both rays either before or after they have passed through the origin. Therefore, by Lemma 8, the hidden open mobile guard set induced by g sees the entire polygon except (possibly) the pair of wedges bounded by two pairs of consecutive reflex vertices adjacent to u and u'.

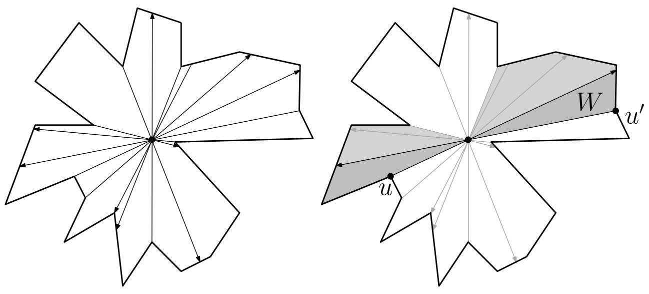


Figure 11: Left: a starshaped polygon with rays from each reflex vertex through the origin. Right: the polygon and a double wedge W (dark gray) with one reflex vertex (u or u') incident to each wedge. The light and dark gray regions together form the subpolygons possibly left unguarded by the hidden open mobile guard set induced by a geodesic path from u to u'.

It may be the case that the two remaining wedges are actually a single non-convex subpolygon with reflex vertex at the origin (see Figure 12).

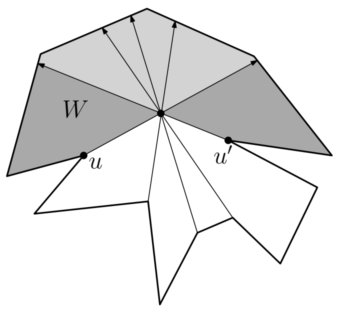


Figure 12: A polygon and double wedge W (dark gray region) where the region not necessarily guarded by the hidden open mobile guard set induced by the geodesic path from u to u' is actually a single non-convex polygon (light and dark gray regions combined) bounded by u and u'.

In this situation the subpolygon can be bisected into two convex subpolygons by a ray bisecting the reflex angle at the origin.

Recall that each convex subpolygon has a vertex u or u' in common with the geodesic's final edge (see Figure 13). If the interior angle formed by these two edges is at most π, then the subpolygon is seen by the interior of the final edge of the geodesic. If not, the geodesic can be extended to include an edge of ∂P in the subpolygon that guards the subpolygon completely.

Thus the hidden mobile guard set induced by the geodesic described guards P. □

Computing such a guard set for a polygon with n edges can be done in $O(n)$ time, as each step takes at most $O(n)$ time: 1. compute a point in the kernel of the polygon ($O(n)$ time by Lee and Preparata [8]). 2. find a separating angle θ ($O(n)$ time). 3. triangulate the polygon and find a geodesic between the reflex vertices u and u' ($O(n)$ time by Fournier and Montuno [3] and Guibas et al. [5]). 4. check whether the two remaining subpolygons are already covered by the geodesic, and extend the geodesic by an additional edge if necessary ($O(1)$ time).

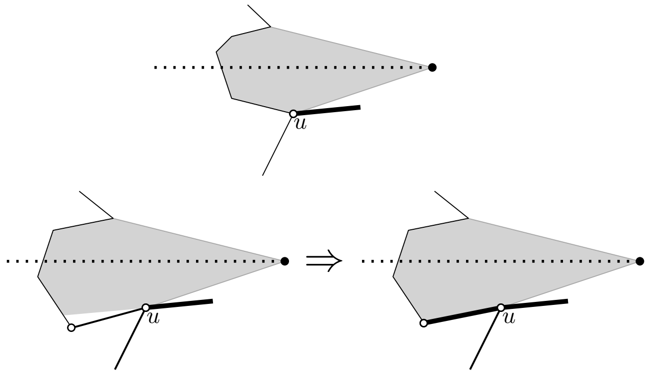


Figure 13: The two cases of guarding the remaining subpolygons. In the case shown in the upper part of the figure, the existing geodesic is sufficient to guard the wedge. In the second case, the geodesic leaves a portion of the wedge unguarded and must be extended.

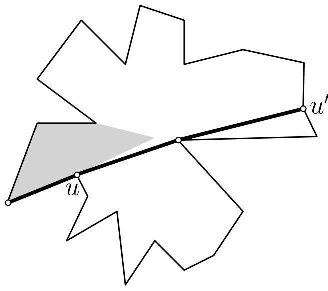


Figure 14: A polygon with a geodesic path inducing a hidden open mobile guard set for the polygon. The initial geodesic from u to u' leaves the gray region incident to u partially unguarded, so the geodesic is extended by one edge.

6 Closed edge and diagonal guards

In the next section we present orthogonal and monotone polygons that do not admit hidden closed mobile guard sets. Note that these polygons also serve as examples of polygons that do not admit hidden closed edge or hidden closed diagonal guards. For starshaped polygons no such example is known.

Lemma 10 *There exists a starshaped polygon that does not admit a hidden closed edge guard set.*

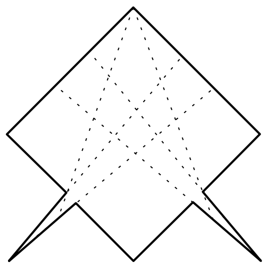


Figure 15: A starshaped polygon that does not admit a hidden closed edge guard set.

Lemma 11 *There exists a starshaped polygon polygon that does not admit a hidden closed diagonal guard set.*

7 Closed mobile guards

Lemma 12 *There exists an orthgonal polygon that does not admit a hidden closed mobile guard set.*

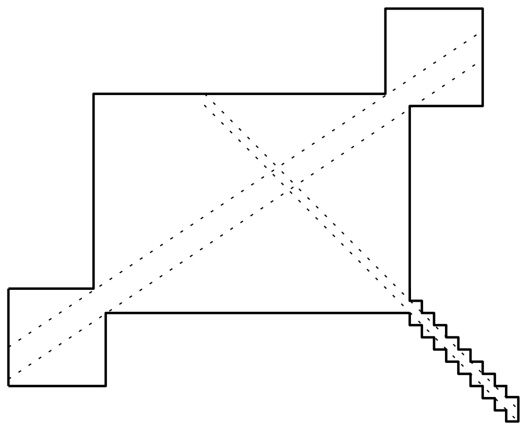


Figure 16: An orthogonal polygon that cannot be guarded using hidden closed mobile guards.

Lemma 13 *There exists a monotone polygon that does not admit a hidden closed mobile guard set.*

Conjecture 1 *Every starshaped polygon admits a hidden closed mobile guard set.*

Acknowledgements

We thank Csaba Tóth for helpful discussions and Richard Pollack, Joseph Malkevitch, John Iacono, and Bill Hall for suggesting interesting problems in this area.

References

[1] D. Avis and G. T. Toussaint, An optimal algorithm for determining the visibility of a polygon from an edge, *IEEE Trans. on Computers*, 30 (1981), 910–914.

[2] N. Benbernou, E. D. Demaine, M. L. Demaine, A. Kurdia, J. O'Rourke, G. Toussaint, J. Urrutia, G. Viglietta, Edge-guarding orthogonal polyhedra, *Proc. 23rd Canadian Conf. on Computational Geometry*, Toronto, Canada, 2011, 461–466.

[3] A. Fournier, D. Y. Montuno, Triangulating simple polygons and equivalent problems, *ACM Trans. on Graphics*, 3 (1984), 153–174.

[4] M. R. Garey, D. S. Johnson, F. P. Preparata, R. E. Tarjan, Triangulating a simple polygon, *Inform. Process. Lett.*, 7 (1978), 175–179.

[5] L. Guibas, J. Hershberger, D. Leven, M. Sharir, R. E. Tarjan, Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons, *Algorithmica*, 2 (1987), 209–233.

[6] E. Kranakis, D. Krizanc, L. Narayanan, K. Xu, Inapproximability of the perimeter defense problem, *Proc. 21st Canadian Conf. on Computational Geometry*, Vancouver, Canada, 2009, 153–156.

[7] A. Kosowski, M. Małafiejski, P. Żyliński, Cooperative mobile guards in grids, *Computational Geometry*, 37 (2006), 59–71.

[8] D. T. Lee, F. Preparata, An optimal algorithm for finding the kernel of a polygon, *J. of the ACM*, 26 (1979), 415–421.

[9] J. O'Rourke, Galleries need fewer mobile guards: a variation on Chvátal's theorem, *Geometriae Dedicata* 14 (1983), 273–283.

[10] J. O'Rourke, *Art Gallery Theorems and Algorithms*, The Intl. Series of Monographs in Comp. Sci., Oxford University Press, New York, 1987.

[11] T. Shermer, Hiding people in polygons, *Computing* 42 (1989), 109–131.

[12] T. C. Shermer, Recent results in art galleries, *Proc. of the IEEE*, 80 (1992), 1384-1399.

[13] C. D. Tóth, G. T. Toussaint, A. Winslow, Open guard edges and edge guards in simple polygons, *Proc. 23rd Canadian Conf. on Computational Geometry*, Toronto, Canada, 2011, 449–454.

[14] G. Viglietta, Searching polyhedra by rotating planes, *Intl. J. of Computational Geometry and Applications*, to appear.

The Complexity of Guarding Monotone Polygons

Erik Krohn[*] Bengt J. Nilsson[†]

Abstract

A polygon P is x-monotone if any line orthogonal to the x-axis has a simply connected intersection with P. A set G of points inside P or on the boundary of P is said to guard the polygon if every point inside P or on the boundary of P is seen by a point in G.

An interior guard can lie anywhere inside or on the boundary of the polygon. Using a reduction from Monotone 3SAT, we prove that interior guarding a monotone polygon is NP-hard. Because interior guards can be placed anywhere inside the polygon, a clever gadget is introduced that forces interior guards to be placed at very specific locations.

1 Introduction

The art gallery problem is perhaps one of the best known problems in computational geometry. It asks for the minimum number of guards to guard a space. An instance of the art gallery problem takes as input a polygon P. The polygon P is defined by a set of points $V = \{v_1, v_2, ..., v_n\}$. There are edges connecting (v_i, v_{i+1}) where $i = 1, 2, ..., n-1$. There is an edge connecting (v_n, v_1). These edges 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. We wish to find a set of points $G \subseteq P$ such that every point $p \in P$ is seen by a guard in G. We call this set G a guarding set. The optimization problem is thus defined as finding the smallest such G.

Art gallery 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 settled by Aggarwal [1] and Lee and Lin [14] independently. They showed that the problem is NP-hard for both vertex guards and interior guards. Along with being NP-complete, Brodén *et al.* and Eidenbenz [2, 7] independently prove that interior 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. Further results have shown that guarding a restricted subclass of polygons is still NP-hard [2, 15]. O'Rourke and Supowit show that several polygon decomposition problems are NP-hard [17].

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 $O(nOPT_v^2 \log^4 n)$ that produces a vertex guard cover with approximation factor $O(\log OPT_v)$ with high probability, where OPT_v is the smallest vertex guard cover for the polygon [6]. Whether a constant factor approximation can be obtained for vertex guarding a simple polygon is a longstanding and well-known open problem. Deshpande *et al.* [5] present a pseudopolynomial randomized algorithm for finding a point guard cover with approximation factor $O(\log OPT)$. King and Kirkpatrick provide an O(log log OPT)-approximation algorithm for the problem of guarding a simple polygon with guards on the perimeter in [12]. The point guarding problem seems to be much more difficult and precious little is known about it [5]. A constant factor approximation is given by Nilsson for the special case of the problem when the polygon is x-monotone [16]. Based on his result, Nilsson gives an $O(OPT^2)$-approximation algorithm for rectilinear polygons.

The approximation complexity of guarding polygons has been studied by Eidenbenz and others. Eidenbenz [8] shows that polygons with holes cannot be efficiently guarded by fewer than $\Omega(\log n)$ times the optimal number of interior or vertex guards, unless P=NP, where n is the number of vertices of the polygon.

Tight bounds for the number of guards necessary and sufficient were found by Chvátal [4]. It is sometimes necessary to place $\frac{n}{3}$ guards to guard the entire polygon. Fisk provided a simpler proof of [4] in [9] that broke up any polygon into a set of triangles and showed that this set of triangles can be 3-colored which implies that $\frac{n}{3}$ guards are sufficient for guarding a simple polygon.

[*]Department of Computer Science, University of Wisconsin - Oshkosh, Oshkosh, WI, USA. email: `krohne@uwosh.edu`

[†]Computer Science, Malmö University College, SE-205 06 Malmö, Sweden. email: `Bengt.Nilsson.TS@mah.se`

1.2 Our Contribution

Chen *et al.* [3] claim that vertex guarding a monotone polygon is NP-hard, however the details of their proof were omitted and still to be verified. Krohn and Nilsson [13] show that vertex guarding a monotone polygon is NP-hard. However, a proof showing NP-hardness for interior guards does not immediately follow from that claim. Since guards can be placed anywhere inside the polygon for interior guarding, moving a guard too far away from a vertex causes the reduction to fail. This is because too much of the polygon is seen by this guard.

Guarding a monotone polygon is very similar to the terrain guarding problem. The question of whether or not terrain guarding was NP-hard was an open problem for many years. Recently, the terrain guarding problem was shown to be NP-hard by King and Krohn [12]. Despite the similarities of guarding terrains and monotone polygons, the NP-hardness result for terrain guarding does not imply interior guarding a monotone polygon is NP-hard. In order to obtain a hardness result for interior guarding a monotone polygon, additional observations had to be made about the properties of monotone polygons. In doing so, we have developed a different reduction from Monotone 3SAT. Despite the very simple structure of a monotone polygon, we were able to create a new, intricate gadget that allows us to force guards to be placed at very specific locations.

The remainder of this paper is organized as follows. Section 2 contains the relevant section from [13] which shows vertex guarding a monotone polygon is NP-hard. Section 3 describes how to modify the reduction from Section 2 to show NP-hardness for interior guarding a monotone polygon. Section 4 provides a conclusion and future work.

2 Vertex Guarding is NP-hard

In this section, we will show that vertex guarding a monotone polygon is NP-hard. The reduction is from *Monotone 3SAT* (M3SAT) [10, page 259 (problem L02)]. An M3SAT instance $(\mathcal{X}, \mathcal{C})$ contains a set of Boolean variables, $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ and a set of clauses, $\mathcal{C} = \{c_1, c_2, ..., c_m\}$. Each clause contains three literals, $c_i = x_j \vee x_k \vee x_l$, a *positive clause*, or $c_i = \bar{x}_j \vee \bar{x}_k \vee \bar{x}_l$, a *negative clause*, for $1 \leq j, k, l \leq n$. An M3SAT instance is satisfiable if a satisfying truth assignment for $\mathcal{C}$ exists such that all clauses c_i are true.

We show that any M3SAT instance is polynomially transformable to an instance of vertex guarding a monotone polygon. We construct a monotone polygon P from the M3SAT instance such that P is guardable by K or fewer guards if and only if the M3SAT instance is satisfiable. We first present some basic gadgets to show how the polygon is constructed. We then connect these gadgets together to create a polygon.

Starting Pattern: The lower boundary of the polygon is divided into two parts, the left and the right sides. The first gadgets on the left side are the *starting patterns*. The starting patterns are shown to the left in Figure 1. *In each pattern, the bottom of the downward spike $b(x)$ is the distinguished vertex of the pattern.* This area is only seen by x and $\bar{x}$ and must be guarded by one of these two vertices. This pattern appears along the left side of the lower boundary of the monotone polygon a total of n times, one corresponding to each variable.

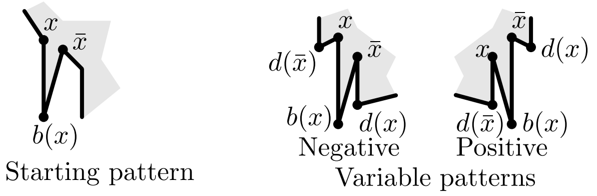


Figure 1: The different types of variable patterns.

Variable Pattern: On the left and the right side of the lower boundary we have *variable patterns* that verify the assigned truth value of each variable. This pattern is shown to the right in Figure 1. Once again, the bottom of the spike at $b(x)$ must be guarded by either x or $\bar{x}$. The pattern has additional distinguished vertices that we call *ledges* $d(x)$ and $d(\bar{x})$ that must both be seen and this is what forces the choice of guard placement at either x or $\bar{x}$.

Figure 2 shows how the starting patterns are connected to variable patterns. If we choose x_j in the starting pattern, we are forced to continuing to choose x_j in each of subsequent variable patterns. If we at some variable pattern would choose $\bar{x}_j$ instead of x_j, the ledge $d(\bar{x}_j)$ is not seen. Similarly, if we in the starting pattern choose $\bar{x}_j$, we are, by the same argument, forced to continuing to choose $\bar{x}_j$ in each of subsequent variable patterns.

Clauses: For each clause c in the boolean formula, there is a sequence of variable patterns $x_1, \ldots, x_n$ along either the left or the right side of the lower boundary and a clause pattern along the upper boundary of the polygon. On the left side of the lower boundary the variable pattern sequence corresponds to negative clauses, on the right side to positive clauses.

The clause pattern on the upper boundary consists of three vertices in an upward spike such that the top vertex of the spike is only seen by the variable patterns corresponding to the literals in the clause; see Figure 3. We denote the top vertex of the spike by c to correspond to the clause.

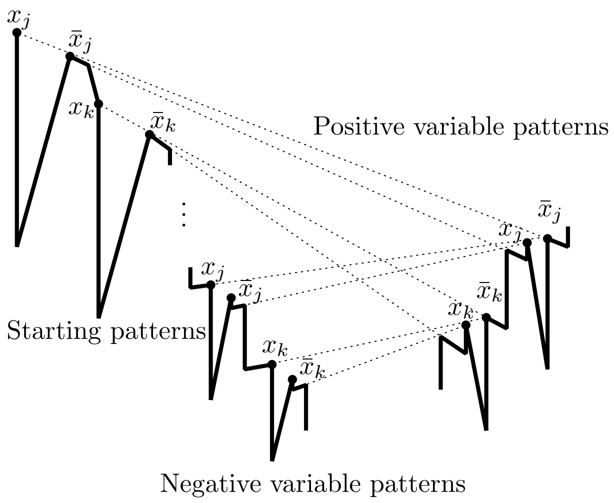


Figure 2: Variable patterns transferring logical values.

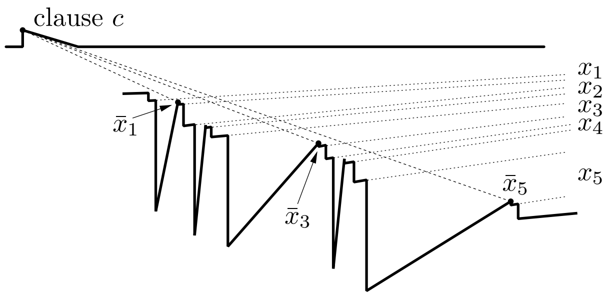


Figure 3: A variable pattern sequence with its clause spike.

We choose our truth value for each variable in the starting variable patterns. The truth values are then mirrored in turn between variable patterns on the right side, corresponding to positive clauses, and variable patterns on the left side, corresponding to negative clauses, of the lower boundary. Truth values do not change in the mirroring process since a variable x_j in clause c_i only sees the ledge $d(x_j)$ in the next variable pattern and none of the other ledges. Similarly $\bar{x}_j$ only sees ledge $d(\bar{x}_j)$ in the next variable pattern; see Figure 2.

In the example of Figure 3 the M3SAT clause corresponds to $c = \bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_5$. Hence, a vertex guard placement that corresponds to a truth assignment that makes c true, will have at least one guard on $\bar{x}_1$, $\bar{x}_3$ or $\bar{x}_5$ and can therefore see vertex c without additional guards.

We still have variables x_2 and x_4 in the clause, however none of them or their negations see the vertex c. They are simply there to transfer their truth values in case these variables are needed in later clauses.

The monotone polygon we construct consists of $4n + (6n + 4)m + 2$ vertices. Each starting variable pattern having four vertices, each variable pattern six vertices, the clause spike consists of three vertices plus one blocking vertex at the start of each clause sequence on the lower boundary and the two leftmost and rightmost points of the polygon.

Consider an M3SAT instance $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_5) \wedge (x_3 \vee x_4 \vee x_5)$. Figure 4 shows how this instance is transformed into a monotone polygon and a placement of guards corresponding to the satisfying truth assignment $x_1 = x_2 = x_4 = x_5 = \textit{false}$, $x_3 = \textit{true}$.

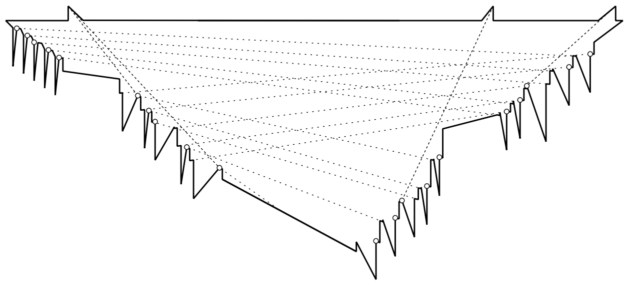


Figure 4: Example reduction of $(x_1 \vee x_2 \vee x_3) \wedge (\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_5) \wedge (x_3 \vee x_4 \vee x_5)$. Points with white centers mark the guards.

Exactly $K = n(m + 1)$ guards are required to guard the polygon since there are K bottom vertices $b(x_j)$ at downward spikes and no vertex in the polygon can see more than one such $b(x_j)$ vertex.

If the M3SAT instance is satisfiable, then we place guards at vertices in accordance to whether the variable is true or false in each of the sequences of variable patterns. Each clause vertex is seen since one of the literals in the associated clause is true and the corresponding vertex has a guard.

Suppose we have a vertex guard cover of size exactly K. Since each bottom spike $b(x_j)$ is guarded there is a guard at one of x_j, $\bar{x}_j$, or $b(x_j)$ itself. They together make up K guards so there can be no other guards. Since each clause vertex c_i is also seen, we can establish which of the guards see this vertex and deduce a satisfying truth assignment from this guard placement. We have proved the following theorem.

Theorem 1 *Finding the smallest vertex guard cover for a monotone polygon is NP-hard.*

3 Interior Guarding is NP-hard

The hardness result for vertex guarding a monotone polygon does not immediately generalize to interior guards. For example, a guard placed slightly above a variable pattern can ruin the mirroring of truth assignments because this guard would see too many ledges. The next section introduces a modified variable pattern which forces the potential guard locations to be very close to the guard locations from Section 2.

3.1 Modified Variable Pattern

The following definition is used in this section: let *VP(p)* denote the visibility polygon of P from the point p, i.e., the set of points in P that can be connected with a line segment to p without intersecting the outside of P.

Modified Variable Pattern: Similar to the variable pattern introduced in Section 2, this pattern is used to verify the assigned truth value of each variable. It is important to note that this modified pattern does not move the x or $\bar{x}$ vertices. This pattern replaces the distinguished vertex $b(x)$ at the bottom of the spike with two distinguished vertices, namely $b(x)$ and $b(\bar{x})$; see Figure 5. This pattern also introduces six new distinguished vertices which are placed directly above the original variable pattern on the top of the polygon. We will call these eight new vertices *variable distinguished vertices*. We will call the six new vertices on the top of the polygon *upper distinguished vertices*. Figure 5 shows the complete modified variable pattern. Each of the upper distinguished vertices can see at most two guards from the following set: $\{x, \bar{x}, b(x), b(\bar{x})\}$.

Lemma 2 *Two guards are necessary and sufficient to see all of the variable distinguished vertices in a modified variable pattern.*

Proof. At least two guards are needed to see all of the distinguished vertices of this modified variable pattern. $VP(1) \cap VP(6) = \emptyset$; see Figure 5. Therefore, to see all upper distinguished vertices, two guards are necessary.

Let us assume that the ledge $d(x)$ is seen from a previous variable pattern. Two guards are sufficient for seeing all of the following variable distinguished vertices: $\{1, 2, 3, 4, 5, 6, b(x), b(\bar{x})\}$ and the unseen ledge $d(\bar{x})$. A possible solution would be to place a guard at x and also at $b(x)$. x would see $\{1, 2, b(\bar{x}), d(\bar{x})\}$ and $b(x)$ would see $\{3, 4, 5, 6\}$. If we assume that $d(\bar{x})$ was seen from a previous variable pattern, then a possible solution would be to place a guard at $\bar{x}$ and $b(\bar{x})$. □

Corollary 3 *We need at least $K = 2n(m + 1)$ guards to see all of the variable distinguished vertices in P.*

All potential guard locations for the upper distinguished vertices are located inside this vertical "strip" as shown in Figure 5. In other words, no guard placed to the left of *VP(1)* and no guard placed to the right of *VP(6)* can see any of the upper distinguished vertices. Said another way, *no guard can see upper distinguished vertices in more than one modified variable pattern.*

We will now show that if all of the variable distinguished vertices are seen, 2 guards must be placed at either $(x, b(x))$ or at $(\bar{x}, b(\bar{x}))$. Consider the horizontal line L drawn in Figure 6. L is split up into several segments. The endpoints of these segments are where the edges of the visibility polygons of $1, 2, \ldots, 6$ hit L. Any guard placed on or above L will see at most 2 upper distinguished vertices. A guard placed on segment a will see vertices $\{1, 2\}$. A guard placed on segment b will see only the vertex $\{2\}$. A guard placed on segment c will see vertices $\{2, 3\}$. A guard placed on segment d will see only the vertex $\{3\}$. A guard placed on segment e will see vertices $\{3, 4\}$. The remaining segments are not named but these are the upper distinguished vertices they see in order from left to right: $\{\{4\}, \{4, 5\}, \{5\}, \{5, 6\}\}$. No guards placed above L will be able to see more upper distinguished vertices than a guard placed on L because of the monotonicity of the polygon. Since there are no obstacles, such a guard

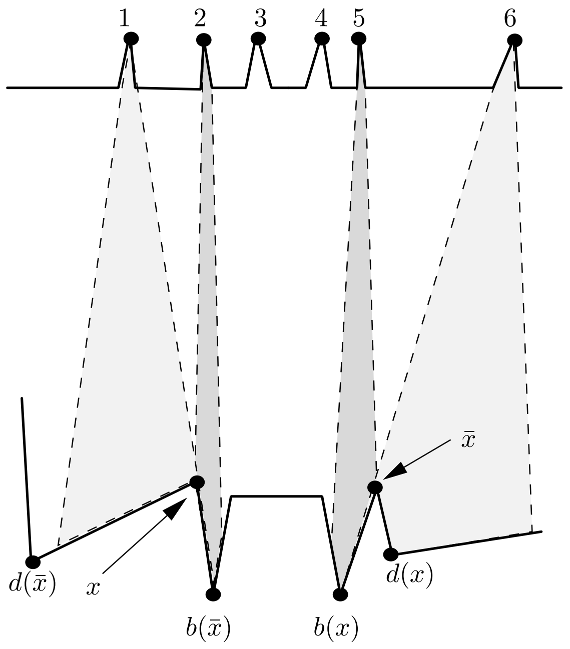


Figure 5: A complete modified variable pattern. Point 1 sees $\{x, b(\bar{x})\}$. Point 2 sees $\{x, b(\bar{x})\}$. Point 3 sees $\{b(\bar{x}), b(x)\}$. Point 4 sees $\{b(\bar{x}), b(x)\}$. Point 5 sees $\{b(x), \bar{x}\}$. Point 6 sees $\{b(x), \bar{x}\}$. The visibility polygons for points 1, 2, 5 and 6 are displayed.

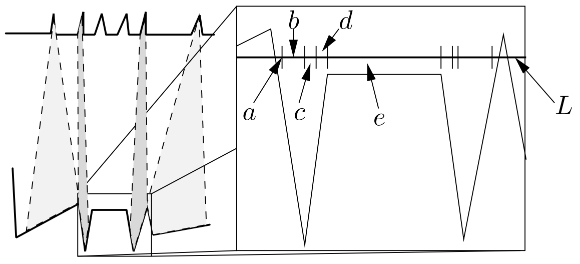


Figure 6: A horizontal line L such that no guard placed on or above L sees more than two upper variable distinguished points.

could be moved down to L without losing visibility of any of the upper distinguished vertices. It is important to note that no guard placed on L sees any of the ledges $d(x)$ or $d(\bar{x})$. A guard must be placed above L in order for the unseen ledge to be seen.

Lemma 4 *No one guard can see more than 4 upper distinguished vertices.*

Proof. We compare the visibility polygons of 1 and 2 with the visibility polygons of 5 and 6; see Figure 5. $(VP(1) \cup VP(2)) \cap (VP(5) \cup VP(6)) = \emptyset$. Any guard that sees 1 or 2 cannot see 5 or 6. For a guard to see more than 4 upper distinguished vertices, such a guard must see all but 1 of the upper distinguished vertices. This is not possible. □

Lemma 5 *For all variable distinguished vertices and one ledge from $\{d(x), d(\bar{x})\}$ of a modified variable pattern to be seen, guards must be placed at $(x, b(x))$ or $(\bar{x}, b(\bar{x}))$.*

Proof. Referring to Figure 5, let us assume that $d(\bar{x})$ is seen by a guard placed in a previous variable pattern. A guard must be placed somewhere in P to see $d(x)$. However, any guard that sees $d(x)$ must be placed above L and therefore can see at most 2 upper distinguished vertices. Because of Lemma 2, if we want to see all of the upper distinguished vertices by using only 2 guards, we are forced to place a second guard below L. To determine where such a guard must be placed, consider the *VP(d(x))*. No point in the visibility polygon of $d(x)$ sees upper variable distinguished points 1 or 2. From Lemma 4, a guard that sees 1 or 2 cannot see 5 or 6. Therefore, if we are only allowed to place 2 guards, a guard that sees $d(x)$ must see 5 and 6. This guard location must be placed at $\bar{x}$. Consider a vertical line through $\bar{x}$. A guard placed just slightly to the left of this vertical line will not see 6. A guard placed slightly to the right of this vertical line will not see 5. Draw a horizontal line through $\bar{x}$. If the guard is moved slightly above this line, neither 5 nor 6 will be seen. Therefore a guard must be placed at $\bar{x}$. Placing a guard at $\bar{x}$ leaves the following upper distinguished vertices unseen: $\{1, 2, 3, 4\}$. For these vertices to be seen, a guard must be placed below the line L. Such a guard placement will not affect the mirroring of variable truth values. The only region that sees the remaining upper variable distinguished points is $b(\bar{x})$. Similar arguments can be made when $d(x)$ has already been seen and the only solution is to place guards at x and $b(x)$. □

The introduction of this modified pattern allows us to force guards to be in certain positions. The forced positions are the same guard locations from the construction in Section 2. The original construction remains unchanged with 2 exceptions. The variable pattern is replaced with a modified variable pattern. A modified starting pattern replaces the original starting pattern. A modified starting pattern is identical to a modified variable pattern without the $d(x)$ and $d(\bar{x})$ ledges. Because there are no ledges, either $(x, b(x))$ or $(\bar{x}, b(\bar{x}))$ can be chosen. This starting choice will then affect guard locations for all future modified variable patterns.

3.2 Entire Polygon is Seen

The previous subsection showed that all distinguished vertices are seen but it does not immediately follow that the entire polygon is seen. We will make a few observations to show that the entire polygon is seen. We will break the polygon into smaller pieces and show that each of those pieces is seen by some subset of the guards already placed. It can easily be seen that every point in the interior of the polygon must fit into at least one of these categories and therefore must be seen. The numbered boxes in Figure 7 correspond to the area we are discussing in the list below.

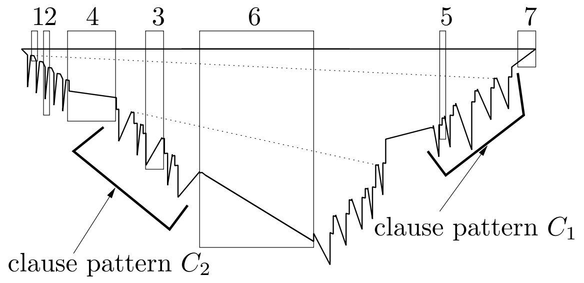


Figure 7: A simplified diagram showing different areas of the polygon.

1. All of the polygon in the vertical strip between a modified starting pattern for x_i and x_{i+1} is seen by a guard placed at one of $\{x_i, \bar{x}_i\} \in C_1$.
2. All of the polygon in a vertical strip containing a modified starting pattern for x_i is seen by either guards placed at $(x_i \in C_0, b(x_i) \in C_0, x_i \in C_1)$ or $(\bar{x}_i, b(\bar{x}_i))$.
3. A *clause pattern* is a grouping of modified variable patterns that determine whether a clause is satisfiable or not. A clause pattern always contains n modified variable patterns. Let us number clause patterns from top to bottom in the order shown in Figure 7. Consider any variable x_i in any clause C_j where $j > 0$ and j is even. The vertical strip containing the modified variable pattern is seen by either guards placed at $(x_i \in C_j, b(x_i) \in C_j, x_i \in C_{j-1})$ or $(\bar{x}_i \in C_j, b(\bar{x}_i) \in C_j, \bar{x}_i \in C_{j-1})$. Similar arguments can be made in the cases where j is odd. One should consider the initial grouping of modified starting patterns as C_0 when thinking about clause pattern C_1.

4. Consider 3 consecutive clause patterns C_{i-1}, C_i, C_{i+1}. The area of the polygon located in a vertical strip between C_{i-1} and C_{i+1} can be seen by a guard placed at either $(x_1, \bar{x}_1) \in C_i$.

5. Consider 2 consecutive modified variable patterns x_i, x_{i+1} in some clause C_i where i is odd. The vertical strip between them is seen by a guard placed at either of $(x_{i+1} \in C_{i-1}, \bar{x}_{i+1} \in C_i)$. Similar arguments can be made if i is even.

6. Consider the vertical strip between the modified variable pattern for $x_n \in C_{m-1}$ and the modified variable pattern for $x_n \in C_m$. In other words, this is the vertical strip in the "middle" of the polygon. This strip is seen by either $\bar{x}_n \in C_{m-1}$ or $x_n \in C_m$. Similar arguments can be made if C_m is to the left of C_{m-1}.

7. Lastly, consider the upper corners of the polygon. A guard placed at either $(x_1, \bar{x_1}) \in C_0$ will see both of these areas.

Using the observations in this section that show the entire polygon is seen and the modified patterns which force guards to be in specific locations along with the hardness result for vertex guarding from [13] with the new $K = 2n(m+1)$ given in Corollary 3, we have proved the following theorem.

Theorem 6 *Finding the smallest interior guard cover for a monotone polygon is NP-hard.*

4 Conclusions and Future Work

We have proved that interior guarding a monotone polygon is NP-hard. Open problems include improving the approximation bounds for monotone polygons. Since a PTAS has not yet been found for guarding a monotone polygon, an interesting open question is whether or not one exists. If a PTAS cannot be found, can guarding a monotone polygon be shown to be APX-hard? Other open problems include finding approximation algorithms for other classes of polygons and ultimately finding better approximations for guarding a simple polygon in general.

Acknowledgments

The authors would like to thank Kasturi Varadarajan and Matt Gibson for their valuable comments, discussions and suggestions.

References

[1] A. Aggarwal. *The art gallery theorem: its variations, applications and algorithmic aspects.* PhD thesis, 1984.

[2] B. Brodén, M. Hammar, and B. J. Nilsson. Guarding lines and 2-link polygons is APX-hard. In *In 13th Canadian Conf. on Computational Geometry*, pages 45–48, 2001.

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

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

[5] A. Deshpande, T. Kim, E. D. Demaine, and S. E. Sarma. A pseudopolynomial time O(log n)-approximation algorithm for art gallery problems. In F. K. H. A. Dehne, J.-R. Sack, and N. Zeh, editors, *WADS*, volume 4619 of *Lecture Notes in Computer Science*, pages 163–174. Springer, 2007.

[6] A. Efrat and S. Har-Peled. Guarding galleries and terrains. *Inf. Process. Lett.*, 100(6):238–245, 2006.

[7] S. Eidenbenz. Inapproximability results for guarding polygons without holes. In *Lecture Notes in Computer Science*, pages 427–436. Springer, 1998.

[8] S. Eidenbenz. *Inapproximability of Visibility Problems on Polygons and Terrains.* PhD thesis, 2000.

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

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[11] S. Ghosh. Approximation algorithms for art gallery problems. In *Proc. Canadian Information Processing Society Congress*, 1987.

[12] J. King and D. Kirkpatrick. Improved approximation for guarding simple galleries from the perimeter. *Discrete Comput. Geom.*, 46(2):252–269, Sept. 2011.

[13] E. Krohn and B. J. Nilsson. Approximate guarding of monotone and rectilinear polygons, 2012. To Appear in Algorithmica.

[14] D. T. Lee and A. K. Lin. Computational complexity of art gallery problems. *IEEE Trans. Inf. Theor.*, 32(2):276–282, March 1986.

[15] B. Nilsson. *Guarding Art Galleries — Methods for Mobile Guards.* PhD thesis, 1995.

[16] B. J. Nilsson. Approximate guarding of monotone and rectilinear polygons. In L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, editors, *ICALP*, volume 3580 of *Lecture Notes in Computer Science*, pages 1362–1373. Springer, 2005.

[17] J. O'Rourke and K. J. Supowit. Some NP-hard polygon decomposition problems. *IEEE Transactions on Information Theory*, 29(2):181–190, 1983.

Kinematic Joint Recognition in CAD Constraint Systems[*]

Audrey Lee-St.John[†]

Abstract

We study the **joint recognition problem**, which asks for the identification and classification of *kinematic joints* from a geometric constraint system. By laying the foundation for a rigidity-theoretic study of flexible body-and-cad frameworks, we obtain an $O(n^3)$ algorithm for identifying prismatic and revolute joints relative to a specific body and an $O(n^4)$ algorithm for finding all pair-wise joints. For a specific subset of body-and-cad frameworks, we present a combinatorial algorithm for identifying all pairs of bodies with prismatic joints in $O(n^2)$ time.

1 Introduction

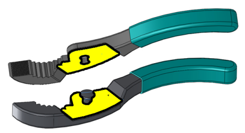
Computer-aided design (CAD) software allows engineers to design sophisticated mechanical systems by placing intuitive geometric constraints in rigid body *assemblies*. In many cases, motion of the resulting system comprises a large part of the user's **design intent**. We present a novel approach based on rigidity theory for analyzing a flexible structure constructed with CAD constraints. This comprehensive approach considers a CAD system at the global level and can recognize kinematic joints that may be implied by constraints not directly involving the two participating bodies.

Motivation. To provide engineers with meaningful feedback for verifying design intent, many CAD applications offer *motion study* tools to simulate kinematic and dynamic motion. Kinematics simulation allows for quick prototyping, as dynamics is a more computationally expensive task. Once the engineer is satisfied with the kinematics, more robust dynamic simulation can be performed before the product is physically produced.

A common approach to kinematic motion simulation of CAD systems is to perturb the system, then resolve the constraints. This can be computationally expensive as it generally reduces to solving an algebraic system of equations. An alternative approach is to directly model the kinematics, which relies on identifying well-understood mechanical joints, such as the prismatic and revolute studied here.

[*]A preliminary version of this work was presented at the FWCG '10 with Rittika Shamsuddin.

[†]Department of Computer Science, Mount Holyoke College, astjohn@mtholyoke.edu


(a) A plane-plane coincidence constraint used in the design.

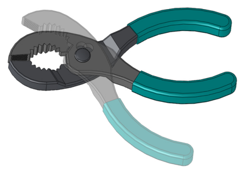

(b) Design intent includes rotation.

Figure 1: A pair of pliers, designed in SolidWorks, has a single rotational motion (i.e., a revolute joint). "Slip Joint Pliers" part from www.3dcontentcentral.com.

Contributions. In this paper, we establish a rigidity theoretic foundation for understanding *flexible* body-and-cad frameworks. The **joint recognition problem** asks for the recognition of *kinematic joints* described by a geometric constraint system. We consider 3D *body-and-cad* frameworks, which are composed of rigid bodies joined by pairwise coincidence, angular and distance constraints, first introduced in [3]. A constraint is placed between two bodies by identifying a *geometric element* (a point, line or plane) on each body and specifying the relationship between them. We address the identification of two types of kinematic joints, both allowing exactly one degree of freedom: *prismatic* (translational motion) and *revolute* (rotational motion)[*].

We present a set of examples highlighting underlying subtleties, lay the mathematical foundation for studying infinitesimal relative motions (or *twists*) of bodies and introduce key concepts for building a theory to analyze them. For motion relative to a particular body, we obtain an $O(n^3)$ algorithm for identifying kinematic joints. A naïve extension provides an $O(n^4)$ algorithm for all pairs of bodies. In the case of prismatic joints, a combinatorial approach for a subset of body-and-cad systems leads to an $O(n^2)$ algorithm.

Related work. Recognition of kinematic joints has a rich history in the CAD community [8], as engineers rely on tools for verifying the kinematic joints dictated by their designed geometric constraint system. Commercial software, such as SolidWorks, Pro/Engineer and the SimMechanics package for MatLab, include a variety of tools for recognizing joints. These proprietary techniques appear to be based on heuristics or mappings from basic constraints to joints. Such mapping

[*]In the rigidity theory literature, a *revolute* joint is called a *hinge*.

techniques require explicit constraints between the two bodies where such a joint is recognized; see, e.g., [9]. However, joints may be implied by constraints present in the global system. Recent work [1] presents a *dynamic geometric system* that uses a *filter* to check potential motions with a limited set of *predicates* (currently, this set includes "CircularMotion" and "Rocker").

The methodology in this paper relies on the foundation of body-and-cad rigidity that was presented in [3]. Our approach is similar in flavor to the *witness method* of Thierry et al. [12]. The *witness method* relies on analysis and manipulation, e.g., Gauss-Jordan elimination, of the Jacobian matrix to detect dependencies; maximal rigid components are discovered by using *references*. We refer to the Jacobian matrix as the *rigidity matrix* and *pin* a body in a way that is similar to choosing a reference. However, the references used by the witness method to detect maximal G-well-constrained components must explicitly block the "motions" of a transformation group G. An attempt to extend this approach would require a priori knowledge of the axis of motion for a proposed revolute or prismatic joint. However, our approach not only detects the presence such a kinematic joint, but additionally identifies the axis of motion. Also in contrast to the work of Thierry et al., which uses a *generic* witness for analysis, we work with the rigidity matrix of an *embedded* framework. It is only for prismatic joints in body-and-cad frameworks with certain properties that the behavior appears generic and amenable to combinatorial analysis.

Structure. Section 2 provides motivating examples, then presents preliminaries for the relevant rigidity theory. Section 3 develops the methodology for identification of prismatic and revolute joints. For a special subset of body-and-cad frameworks, Section 4 gives a purely combinatorial approach for detecting prismatic joints. Finally, Section 5 concludes with open questions.

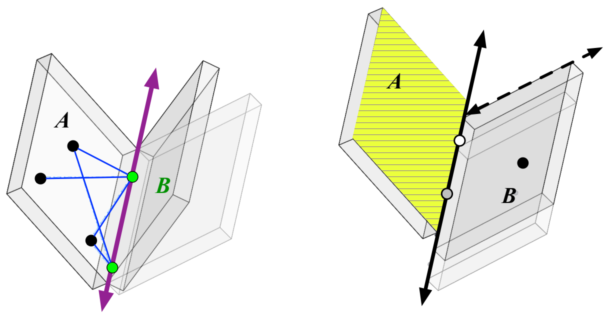
2 Background

In this section, we begin with simple examples to provide intuitions and highlight the subtleties of the **joint recognition problem**. We then provide the preliminaries necessary for describing our contributions.

2.1 Motivating examples

We present several small examples of body-and-cad frameworks that expose some of the difficulties encountered when analyzing motion and identifying kinematic joints.

Simple revolute joint. Figure 1 depicts a simple example of a 2-body system with a single revolute joint. This pair of pliers is composed of two rigid handles with (1) a plane-plane coincidence so they lie adjacent to each other, and (2) a point-point coincidence to force a center of rotation at the desired axis. Since the resulting rotational motion is clearly necessary for the design, identification of this revolute joint would allow verification of user intent. This example contains no dependencies and the rotational degrees of freedom are constrained by exactly two (primitive) angular constraints resulting from the plane-plane coincidence, so analysis seems straightforward: intuitively, we can conclude that there is one rotational degree of freedom.



(a) A revolute joint determined by 5 point-point distance constraints.

(b) A prismatic joint allows a single translational motion.

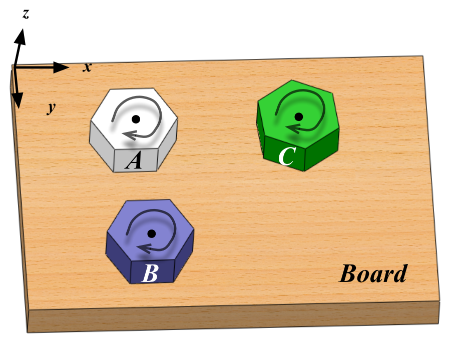
Figure 2: Two-body frameworks with revolute and prismatic joints.

Revolute joint with only blind constraints. Body-and-cad constraints can be separated into "angular" constraints (affecting only rotational degrees of freedom) and "blind" constraints (affecting either rotational or translational degrees of freedom); refer to Section 2.2. This presents a challenge as a revolute joint may be specified without the use of any angular constraints: it is known from classical rigidity theory that a "hinge" (i.e., a revolute joint) may be described by 5 bars (i.e., point-point distances) [11, 14]. Figure 2a depicts this set of bar constraints between two bodies (the three black points lie on body A); there is exactly one rotational motion about the purple (bold) line. Since point-point distance constraints are blind, there does not appear to be straightforward reasoning that would lead us to identify this revolute joint.

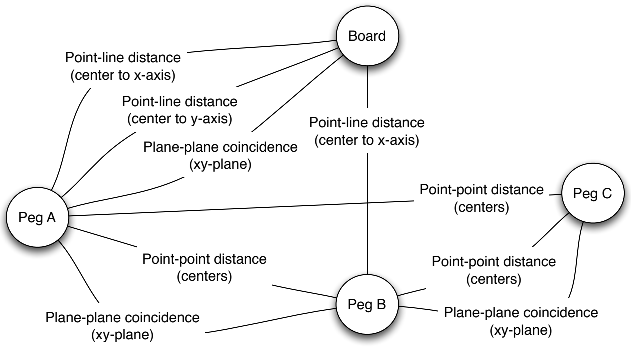
Simple prismatic joint. We return to a simple design for a prismatic joint between a pair of rigid bodies (see Figure 2b). This system is composed of (1) a line-line coincidence along the solid line, and (2) a line-plane perpendicular between the dashed line on body B and the striped plane on body A. These constraints define a prismatic joint: body B may only translate relative to body A along the solid axis. As with the first example, this system seems amenable to analysis as it has no dependencies and the rotational degrees are explicitly eliminated by exactly three angular constraints (two from the coincidence and one from the perpendicular).

Prismatic joint with only blind constraints. This next example again highlights the subtlety of blind constraints. We create an equivalent system to the previous one by using (1) a point-line coincidence between the gray point on body B and the solid line on body A, (2) a point-line coincidence between the white point on body B and the solid line on body A, and (3) a point-plane distance between the black point on body B and the yellow (striped) plane on body A. While the only motion left is a translation along the solid axis, there are no angular constraints present. Intuitively, it seems that three of the bars are somehow modeling angular constraints to eliminate the rotational degrees of freedom.

Pegboard example: implied revolute joints. In the previous examples, the constraints determining a kinematic joint explicitly involved the two bodies. The pegboard shown in Figure 3a provides an example where a revolute joint is implied by constraints not directly involving the two bodies. This system contains 4 rigid bodies: a wooden board along with three "pegs" A, B, and C. The constraints are described by the *cad graph* (formally defined in Section 2.2) in Figure 3b. The pegboard has three non-trivial degrees of freedom associated to three revolute joints: each peg can rotate relative to the board (about the vertical axis through the peg's center point). Notice the lack of constraints between peg C and the board; this revolute joint is implied by the rest of the constraints. In fact, such a design is a realistic result of the difficulties often caused by user interfaces of 3D CAD software (rotating the view to select logical surfaces can be cumbersome).



(a) A 4-body system has a single wooden board with three pegs.


Board
Point-line distance (center to x-axis)
Point-line distance (center to y-axis)
Plane-plane coincidence (xy-plane)
Point-line distance (center to x-axis)
Point-point distance (centers)
Peg C
Peg A
Point-point distance (centers)
Point-point distance (centers)
Peg B
Plane-plane coincidence (xy-plane)
Plane-plane coincidence (xy-plane)

(b) Associated cad graph denotes constraints.

Figure 3: A pegboard example with three revolute joints (one completely implied by indirect constraints).

2.2 Preliminaries

We now present background for body-and-cad rigidity, which provides a foundation for defining relative motion and classifying prismatic and revolute joints.

Body-and-cad infinitesimal rigidity theory. We work with *body-and-cad frameworks*, composed of rigid bodies with pairwise constraints from a set of 21 coincidence, angular and distance constraints. A constraint between two bodies involves a *geometric element* (a point, line or plane) rigidly affixed to each body. Since we rely on the body-and-cad rigidity theory presented by Haller et al. [3], we give a brief overview of the necessary foundations for the infinitesimal rigidity theory.

Formally, a body-and-cad framework is defined by a *cad graph* (G, c) that describes the combinatorics along with a family of 21 "length" functions describing the geometry of the structure. The cad graph (G, c) is a pair, where $G = (V, E)$ is a multigraph with $V = [1..n]$ and c an edge coloring function specifying the cad constraints. See Figure 3b for an example cad graph; we use labels to indicate the edge "colors." In this paper, we will abuse notation slightly and assume that a body-and-cad framework is given by an *embedding* from which the "length" functions can be computed.

Since we work in the infinitesimal rigidity theory, we first consider instantaneous rigid body motion in 3D. As a consequence of Chasles' Theorem (see, e.g., [10]), any instantaneous rigid body motion may be described by a *twist* (translation and rotation about a *twist axis*), which itself is represented by a 6-vector $\mathbf{s} = (\boldsymbol{\omega}, \mathbf{v})$. The 3-vector $\boldsymbol{\omega}$ describes the *angular velocity*: the direction of the twist axis and rotational speed about it. The 3-vector $\mathbf{v}$ can be used to decode the rest of the twist axis and translational speed along it.

Body-and-cad infinitesimal rigidity theory relies on expressing constraints in the Grassmann-Cayley algebra, resulting in the construction of a *rigidity matrix*. This matrix has 6 columns for each body i, corresponding to 2-tensors represented with Plücker coordinates. Since there is a mapping between the 2-tensors in the Grassmann-Cayley algebra and twists, we may interpret these 6 columns representing the degrees of freedom of body i: $\mathbf{s}_i^* = (\mathbf{v}_i, -\boldsymbol{\omega}_i)^\dagger$. The kernel of the rigidity

†The re-ordering and negation are technicalities arising from the development of the rigidity matrix. When referring to elements of the kernel, we will use $\mathbf{s}^*$; when referring to the corresponding twist, we will use $\mathbf{s}$.

matrix describes the *infinitesimal*[‡] *motion space* for a body-and-cad system.

Each body-and-cad constraint is associated to a number of *primitive* constraints (which affect at most one degree of freedom). A primitive constraint between bodies i and j is encoded by a single row in the rigidity matrix that has a vector $\mathbf{x} \in \mathbb{R}^6$ in the columns for body i and $-\mathbf{x}$ in the columns for body j; every other entry in the row is 0. A distinction is made between primitive *angular* and *blind* constraints: a primitive angular constraint may affect only a rotational degree of freedom and corresponds to a row in the rigidity matrix with zeros in the $\mathbf{v}$ columns.

Combinatorics. For a cad graph (G, c), we associate a *primitive cad graph* $H = (V, R \sqcup B)$, which assigns a vertex to each body and red (R) and black (B) edges to primitive angular and blind constraints determined by each cad constraint. Recent work [6] characterizes generic rigidity of body-and-cad frameworks without point-point coincidence constraints.

Theorem 1 *[6] Let F be a body-and-cad framework with no point-point coincidence constraints. Let $H = (V, R \sqcup B)$ be the primitive cad graph associated to F. Then F is generically minimally rigid if and only if there exists a set $B' \subseteq B$ such that $(V, R \cup B')$ and $(V, B \setminus B')$ are each the edge-disjoint union of 3 spanning trees.*

For angular constraints in isolation, *angular rigidity* for *body-and-angle* frameworks was characterized in [7], based on *sparsity counts.*

Theorem 2 *[7] A body-and-angle framework is generically minimally rigid if and only if its associated graph is $(3, 3)$-tight.*

The pebble games of [4] provide $O(n^2)$ algorithms for determining if a graph is (k, ℓ)-sparse (or tight) and for detecting (k, ℓ)-*components* ((k, ℓ)-tight subgraphs that are maximal with respect to vertices).

Relative motions. We consider *relative motions* between a pair of bodies. Let F be a body-and-cad framework, $M(F)$ its rigidity matrix and $\ker(M(F))$ the associated motion space. It must be the case that the 6-dimensional space of trivial rigid body motions is a subspace of $\ker(M(F))$. For a pair of bodies i and j, we restrict the motion space to describe relative motions between the bodies. Formally, we project $\ker(M(F))$ into $\mathbb{R}^{12}$ by a linear transformation described by the following $12 \times 6n$ matrix: the first (respectively, last) 6 rows contain the identity matrix of size 6 in the columns for body i (respectively, j) and zeros everywhere else. Then the *relative motion space* W is the resulting subspace of $\mathbb{R}^{12}$. Again, the 6-dimensional space of trivial motions must be a subspace of W. Define the *number of relative degrees of freedom* to be $\dim(W) - 6$. If this is zero, the two bodies are *relatively rigid.* A *rigid component* is a maximal set of bodies that are pairwise relatively rigid. We observe that the number of relative degrees of freedom is equal to the minimum number of rows (i.e., primitive constraints or edges in the primitive cad graph) whose addition cause i and j to fall into the same rigid component.

Intuitively, to study the non-trivial relative motions, we consider when one body is fixed and seek a description of the allowed motions of the second. To formalize this notion, we may fix body i by appending 6 rows to the rigidity matrix: the identity matrix of size 6 appears in the columns for i and zeros appear in all other entries. We denote the newly obtained *pinned* matrix by $M(F, i)$. The *non-trivial relative motion space for body i* is simply $\ker(M(F, i))$. The *non-trivial relative motion space between bodies i and j* is a projection of the kernel into $\mathbb{R}^6$. The linear transformation used is defined by a $6 \times 6n$ matrix with the identity matrix of size 6 appearing in the columns for body j and zeros everywhere else. The number of relative degrees of freedom is the dimension of this subspace of $\mathbb{R}^6$.

3 Identification of Kinematic Joints

In this paper, we are interested in non-trivial relative motion spaces of dimension 1, spanned by a single twist $\mathbf{s} \in \mathbb{R}^6$. As a consequence of the mapping between twists and 2-tensors in the Grassmann-Cayley algebra, there is a further mapping between twists that are either pure rotations or pure translations and 2-tensors that are decomposable. Decomposable 2-tensors are those that satisfy the Plücker relation, i.e., those that lie on the Grassmannian. We choose the same convention as [3] for the Plücker coordinates, so that the Plücker relation is satisfied for a vector $\mathbf{s} = (\boldsymbol{\omega}, \mathbf{v})$ if $\langle -\boldsymbol{\omega}, \mathbf{v} \rangle = 0$, where the angle brackets denote the dot product. These 6-vectors lie on the Klein quadric and describe lines in 3-dimensional projective space. Furthermore, if $\boldsymbol{\omega} = \mathbf{0}$, the twist has only a translation $\mathbf{v}$ (encoding a prismatic joint). Otherwise, the twist corresponds to pure rotation about the axis (encoding a revolute joint), and the axis of rotation is described by $\mathbf{s}$, the Plücker coordinates of the line. See, e.g., [10, 13], for reference.

We can now describe the **algorithm for identifying infinitesimal prismatic and revolute joints**. To find joints relative to body i in a body-and-cad framework F:

1. Construct the pinned rigidity matrix $M(F, i)$.

2. Compute its kernel $\ker(M(F, i))$.

3. For each body $j \neq i$:

[‡]For brevity, we will omit "infinitesimal" for the remainder of the paper.

(a) Restrict to body j by finding a basis for the non-trivial relative motion space between i and j.

(b) Compute the number of relative degrees of freedom (the dimension of the space).

(c) If there is > 1 degree of freedom, continue to the next body.

(d) Otherwise, consider the single basis vector $\mathbf{s}^* \in \mathbb{R}^6$.

(e) Check if $\mathbf{s} = (\boldsymbol{\omega}, \mathbf{v})$ satisfies the Plücker relation. If not, continue to the next body.

(f) If $\boldsymbol{\omega} = \mathbf{0}$, output *prismatic joint between i and j* with translation $\mathbf{v}$; otherwise, output *revolute joint between i and j* with axis of rotation described via Plücker coordinates $\mathbf{s}$.

Let m be the number of rows (primitive constraints) and n the number of bodies. The dominating factor is step 2 (computing the kernel), which requires $O(\min(n, m)nm)$ time; if we assume a linear number of constraints (e.g., if there are no dependencies), the algorithm has time complexity $O(n^3)$. By executing the algorithm for all $i \in [1..n]$, we obtain an algorithm for identifying all pairs of infinitesimal prismatic and revolute joints that runs in $O(n^4)$ time.

Since the analysis is done at the infinitesimal level, this is not a characterization (i.e., the method may incorrectly identify a relative kinematic joint), but can be used as a filter to identify potential pairs of bodies with prismatic or revolute joints.

4 Combinatorial Identification of Prismatic Joints

We now give a combinatorial algorithm for identifying prismatic joints in a subset of body-and-cad frameworks. We consider frameworks that have no point-point coincidence constraints, dependencies or non-trivial (involving more than one body) rigid components. This allows us to find a combinatorial condition for characterizing when two bodies have a single relative degree of freedom, leading to an algorithm for finding candidate pairs that may have a kinematic joint.

Lemma 3 *For a body-and-cad framework with no point-point coincidence constraints, dependent constraints or non-trivial rigid components, a pair of bodies i and j generically have one relative degree of freedom if and only if they lie in a common* $(6, 7)$*-component of* $H = (V, R \sqcup B)$*, the associated primitive cad graph.*

Proof. By [2], a graph is $(6, 7)$-tight if and only if the addition of any edge results in the edge-disjoint union of 6 spanning trees. As a consequence of Theorem 1, $(6, 7)$-components of H are equivalent to subgraphs such that the addition of any edge results in a rigid component. Since the number of relative degrees of freedom between two bodies is the minimum number of edges whose addition results in their being in the same rigid component, i and j have one relative degree of freedom if and only if they lie in a common $(6, 7)$-component. □

To find prismatic joints, we begin by defining *angular-rigid components*. As with Section 2.2, we restrict the motion space to consider only rotational degrees of freedom. For a framework F with rigidity matrix $M(F)$ and motion space $\ker(M(F))$, the *angular motion space* is the space obtained by projecting the kernel into $\mathbb{R}^{3n}$ (intuitively retaining only the rotational coordinates). Formally, we use the linear transformation described by the $3n \times 6n$ matrix with a set of 3 rows for each body i containing the identity matrix of size 3 in the columns for $-\boldsymbol{\omega}_i$ and zeros elsewhere. Instead of rigid body motions, the 3-dimensional space of (trivial) rotations is contained in the angular motion space. The concepts of *relative angular motion space*, *relative angular degrees of freedom*, *relatively angular-rigid* and *non-trivial relative angular motion space* follow analogously. An *angular-rigid component* is a maximal set of bodies that are pairwise relatively angular-rigid.

Lemma 4 *In a body-and-cad framework, a pair of bodies i and j with one relative degree of freedom share a prismatic joint if and only if they lie in a common angular-rigid component.*

Proof. Let F be a body-and-cad framework; let bodies i and j have one relative degree of freedom and denote by W their relative motion space (i.e., $dim(W) = 7$).

Bodies i and j lie in a common angular-rigid component if and only if W_R, their relative angular motion space, has dimension 3. Now consider fixing body i; let W' be the non-trivial relative motion space for j and W'_R the non-trivial relative angular motion space. Then W' must have dimension 1, defined by a single basis vector $\mathbf{s}^* = (\mathbf{v}, -\boldsymbol{\omega})$. W_R has dimension 3 if and only if W'_R has dimension 0 if and only if $\boldsymbol{\omega} = 0$. Thus, bodies i and j share a prismatic joint if and only if they lie in a common angular-rigid component. □

For a subset of body-and-cad frameworks, the detection of (generic) angular-rigid components becomes a combinatorial problem. If F is a framework with rigidity matrix $M(F)$ and primitive cad graph $H = (V, R \sqcup B)$, let $M_R(F)$ be the submatrix determined by the $3n$ columns corresponding to the rotational degrees of freedom $\boldsymbol{\omega}_i$ and the rows associated to the red edges R. We define a body-and-cad framework to be *angular-distinct* if the kernel of $M_R(F)$ is the same as the angular motion space of F. Then, as a consequence of Theorem 2, angular-rigid components in an angular-distinct body-and-cad framework are equivalent to $(3, 3)$-tight components in $H_R = (V, R)$.

We now present a combinatorial **algorithm for detecting prismatic joints** for a body-and-cad framework F satisfying the conditions: (A) F contains no point-point coincidence constraints, (B) F contains no dependent constraints or non-trivial rigid components, and (C) F is angular-distinct.

1. Play the $(6, 7)$-pebble game on H to detect $(6, 7)$-components.
2. Play the $(3, 3)$-pebble game on $H_R = (V, R)$ to detect $(3, 3)$-components.
3. Find all pairs of bodies i and j that share a $(6, 7)$-component and a $(3, 3)$-component; output *prismatic joint between bodies i and j*.

Condition (A) allows the application of Theorem 1. Since dependencies or rigid components result in subgraphs that are not $(6, 7)$-sparse, Condition (B) ensures that we find all pairs of bodies with one relative degree of freedom. Without Condition (C), we will still correctly output prismatic joints, but may not find all.

The pebble game algorithms run in $O(n^2)$ time, so Steps 1 and 2 each take quadratic time. In Step 3, the union pair-find data structure developed for rigid components [5] allows us to query if two bodies share a component in constant time and quadratic space. Thus, the entire algorithm has $O(n^2)$ time complexity.

5 Conclusions and Future Work

We initiated the study of understanding flexible body-and-cad frameworks and relative motions from a rigidity-theoretic perspective. This led to the development of an $O(n^3)$ algorithm for detecting infinitesimal prismatic and revolute joints relative to a fixed body and an $O(n^4)$ algorithm for finding all pair-wise kinematic joints. Based on standard linear algebra techniques, this method outputs the type of kinematic joint (revolute or prismatic) as well as the axis of motion.

For the special case of prismatic joints in a restricted set of body-and-cad structures, we gave a purely combinatorial algorithm with $O(n^2)$ complexity, indicating that prismatic joints are more amenable to detection algorithms. This may be due to their correspondence to a 2-dimensional plane in the Klein quadric, whereas revolute joints correspond to all other points on the quadric.

Future work. The combinatorial algorithm for prismatic joints motivates the need for an efficient algorithm for body-and-cad rigidity (including detecting dependencies and rigid components). Further investigation of angular-distinct systems or the transformation (or identification) of blind constraints to angular constraints may elucidate the special treatment of angular constraints with respect to rotational degrees of freedom. More generally, we anticipate that the foundations presented here will allow us to understand motions beyond infinitesimal prismatic and revolute joints.

Acknowledgements. We are grateful to reviewers for insightful suggestions and observations and to Rittika Shamsuddin for participating in the initial work.

References

[1] M. Freixas, R. Joan-Arinyo, and A. Soto-Riera. A constraint-based dynamic geometry system. *Comput. Aided Des.*, 42(2):151–161, 2010.

[2] R. Haas. Characterizations of arboricity of graphs. *Ars Combinatorica*, 63:2002.

[3] K. Haller, A. Lee-St.John, M. Sitharam, I. Streinu, and N. White. Body-and-cad geometric constraint systems. *Computational Geometry: Theory and Applications*, 2010. In press. http://dx.doi.org/10.1016/j.comgeo.2010.06.003.

[4] A. Lee and I. Streinu. Pebble game algorithms and sparse graphs. *Discrete Mathematics*, 308(8):1425–1437, 2008.

[5] A. Lee, I. Streinu, and L. Theran. Finding and maintaining rigid components. In *Proceedings of the 17th Canadian Conference of Computational Geometry*, Windsor, Ontario, 2005.

[6] A. Lee-St.John and J. Sidman. Combinatorics and the rigidity of cad systems. Accepted to SPM '12: Symposium of Solid and Physical Modeling, 2012.

[7] A. Lee-St.John and I. Streinu. Angular rigidity in 3d: combinatorial characterizations and algorithms. In *Proceedings of the 21st Canadian Conference on Computational Geometry*, pages 67–70, 2009.

[8] K. Lyons, V. Rajan, and R. Sreerangam. Representations and methodologies for assembly modeling. *National Institute of Standards and Technology,Gaithersburg, MD*, 6059, 1997.

[9] O. E. Ruiz. Geometric constraint subsets and subgraphs in the analysis of assemblies and mechanisms. *Ingenieria y Ciencia (Engineering and Science)*, 2(3):103–137.

[10] J. M. Selig. *Geometric Fundamentals of Robotics*. Monographs in Computer Science series. Springer, New York, 2nd edition, 2005.

[11] T.-S. Tay. Rigidity of multi-graphs. I. Linking rigid bodies in n-space. *Combinatorial Theory Series*, B(26):95–112, 1984.

[12] S. E. B. Thierry, P. Schreck, D. Michelucci, C. Fünfzig, and J.-D. Génevaux. Extensions of the witness method to characterize under-, over- and well-constrained geometric constraint systems. *Computer-Aided Design*, 43(10):1234–1249, 2011.

[13] N. White. Grassmann-Cayley algebra and robotics. *Journal of Intelligent and Robotics Systems*, 11:91–107, 1994.

[14] W. Whiteley. The union of matroids and the rigidity of frameworks. *SIAM Journal Discrete Mathematics*, 1(2):237–255, May 1988.

Computing Motorcycle Graphs Based on Kinetic Triangulations[*]

Willi Mann[†] Martin Held[†] Stefan Huber[‡]

Abstract

We present an efficient algorithm for computing generalized motorcycle graphs, in which motorcycles are allowed to emerge after time zero. Our algorithm applies kinetic triangulations inside of the convex hull of the input, while a plane sweep is used outside of it. Its worst-case complexity is $O((n+f)\log n)$, where $f \in O(n^3)$ denotes the number of flip events that occur in the kinetic triangulation. Outside of the convex hull it runs in $O(n \log n)$ time. In order to reduce the number of flip events we investigate the use of Steiner triangulations. We prove the existence of Steiner triangulations that eliminate all flip events and discuss heuristics for approximating such a Steiner triangulation.

Extensive experiments with our C++ implementation run on thousands of datasets of various characteristics demonstrate a runtime of $c \cdot 10^{-6} \cdot n \log n$ seconds, with $c \leq 4$ for virtually all of our datasets. This constitutes a significant practical improvement over the motorcycle code Moca [Huber&Held 2011], which runs in $O(n \log n)$ time only if the motorcycles are distributed uniformly enough. In particular, our experiments yielded $f \leq 5n$ flip events for all but very few datasets.

1 Introduction

1.1 Motivation and Definitions

A *motorcycle* is a point that moves with constant velocity along a straight-line ray. Consider n motorcycles $m_1, \ldots, m_n$, each of them having a start point $p_i \in \mathbb{R}^2$ and a velocity $v_i \in \mathbb{R}^2$, with $1 \leq i \leq n$. (We assume that at most a constant number of motorcycles share one start point.) The *track* of motorcycle m_i is defined by the ray $\{p_i + t \cdot v_i \, : \, t \geq 0\}$. While a motorcycle moves it leaves a *trace* behind. A motorcycle *crashes* when it reaches the trace of another motorcycle: It stops driving but its trace remains. A motorcycle *escapes* if it never crashes. The motorcycle graph $\mathcal{M}(m_1, \ldots, m_n)$ is defined as the arrangement of all motorcycle traces.

Motorcycle graphs were introduced by Eppstein and Erickson [4] when investigating straight skeletons [1]. The basic motivation behind the definition of the motorcycle graph was to extract the essential sub-problem of computing straight skeletons and to cast it into a separate problem. In fact, it turns out that motorcycle graphs and straight skeletons share a strong relationship: (i) motorcycle graphs can be used to give a non-procedural characterization of straight skeletons [9, 2], (ii) the straight-skeleton algorithm by Huber and Held [9] and the algorithm by Cheng and Vigneron [2] are based on motorcycle graphs, and (iii) the $\mathcal{P}$-completeness of straight skeletons of polygons with holes follows from the $\mathcal{P}$-completeness of motorcycle graphs [7, 4]. In particular, the currently fastest straight-skeleton code Bone [9] employs the motorcycle graph in a preprocessing step.

In this paper, we present an algorithm for computing the motorcycle graph $\mathcal{M}(G)$ that is induced by a planar straight-line graph (PSLG) G. This requires a generalization of the original motorcycle graph, see Fig. 1.

First, we consider rigid *walls* formed by straight-line segments. If a motorcycle meets a wall then it crashes, too. Secondly, if two or more motorcycles $m_1, \ldots, m_k$ crash simultaneously into each other at the point p such that the traces of $m_1, \ldots, m_k$ lie in a half-plane whose boundary contains p then a new motorcycle m is launched from p in the complementary half-plane. In other words, a local disc at p is tessellated into convex slices by the traces of $m, m_1, \ldots, m_k$. To sum up, in generalized motorcycle graphs a motorcycle is specified by a start point, a velocity and a start time. A formal definition of $\mathcal{M}(G)$ involves concepts of straight skeletons, see [9] for further

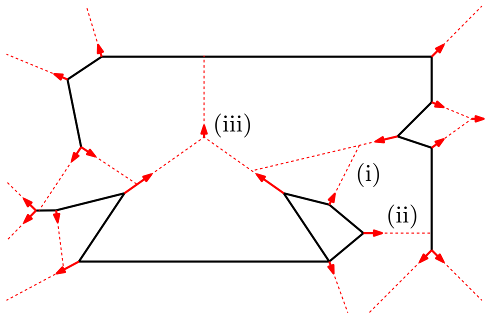


Figure 1: A generalized motorcycle graph. The motorcycles' velocities are depicted by (red) arrows. A motorcycle may crash against (i) another trace or (ii) a wall. (iii) A Motorcycle may be launched after two or more motorcycles crashed into each other.

[*]Work supported by Austrian FWF Grant L367-N15.

[†]FB Computerwissenschaften, Universität Salzburg, A–5020 Salzburg, Austria, {wmann,held}@cosy.sbg.ac.at

[‡]FB Mathematik, Universität Salzburg, A–5020 Salzburg, Austria, shuber@cosy.sbg.ac.at

details. For the reader to be able to follow this paper it suffices to note that not all motorcycles are known a priori, and that the motorcycles may crash into a total of $O(n)$ walls.

1.2 Prior Work

For the original setting of the motorcycle graph problem, the algorithm by Cheng and Vigneron [2] achieves the best worst-case complexity: it runs in $O(n\sqrt{n}\log n)$ time. In a preprocessing, they compute a $1/\sqrt{n}$-cutting on the supporting lines of the motorcycle tracks. However, this requires to know all motorcycles in advance and hence their algorithm is not applicable to generalized motorcycle graphs.

The algorithm by Eppstein and Erickson [4] is applicable to generalized motorcycle graphs. It employs various efficient closest-pair data structures in a hierarchical fashion. By a clever trade-off between time and space they achieve an $O(n^{17/11+\epsilon})$ time and space complexity. However, their algorithm is far too complex for an actual implementation.

A fairly simple recent algorithm by Huber and Held [8] uses a $\sqrt{n}\times\sqrt{n}$-grid to speed up computation. If the motorcycles are distributed uniformly enough then a motorcycle crosses only $O(1)$ grid cells on average, which leads to an $O(n\log n)$ runtime. The resulting motorcycle-graph code Moca has become to be known as the fastest implementation. While Moca works nicely for most datasets, it requires up to $O(n^2\sqrt{n}\log n)$ time for some contrived inputs, and $O(n^2\log n)$ time for densely sampled convex bodies.

1.3 Our Contribution

In Sec. 2 we present a novel motorcycle-graph algorithm for the computation of $\mathcal{M}(G)$ that consists of two phases. The first phase computes $\mathcal{M}(G)$ within the convex hull $CH(G)$ of the walls and the start points of all motorcycles. It is based on a kinetic triangulation, akin to [1]. Its worst-case time complexity is $O((n+f)\log n)$, where $f\in O(n^3)$ denotes the number of flip events that occur in the kinetic triangulation. The second phase uses a plane-sweep algorithm to compute $\mathcal{M}(G)$ outside of $CH(G)$ in time $O(n\log n)$. Thus, in the worst case the total complexity is $O(n^3\log n)$. However, no input is known that causes a runtime of more than $O(n^2\log n)$.

As the time complexity strongly depends on the number f of flip events, we investigate the use of Steiner triangulations for reducing f. In fact, we prove that Steiner triangulations exist for which no flip event occurs and for which our algorithm would take $O(n\log n)$ time. This motivates the search for practical heuristics to approximate such a Steiner triangulation.

We implemented our algorithm in C++ and report on implementational and numerical aspects. Extensive benchmarks on several thousand datasets clearly demonstrate an $O(n\log n)$ runtime. In particular, our experiments yielded $f\leq 5n$ flip events for virtually all datasets. Additional experiments showed that our heuristics reduce the number of flip events by 20% on average. As our algorithm does not rely on a roughly uniform distribution of the motorcycles this constitutes a major practical improvement compared to the algorithm that drives Moca.

2 Computing the Generalized Motorcycle Graph

2.1 Computation Inside of Convex Hull $CH(G)$

To compute $\mathcal{M}(G)$ within $CH(G)$ we need to determine which motorcycle crashes into which trace or wall. (Motorcycles which start on the boundary of $CH(G)$ and do not move inwards are considered in the second phase of our algorithm, see Sec. 2.2.) The basic idea is to use a kinetic triangulation such that every crash event is indicated by the collapse of a triangle in the triangulation. This approach is motivated by the straight-skeleton algorithm by Aichholzer and Aurenhammer [1]. Thus, we start by computing the constrained Delaunay triangulation T within $CH(G)$, where the start points of the initial motorcycles and the endpoints of the walls form the vertices and the walls form the constrained diagonals of T.

In the next step, we insert at each start point p_i of an initially present motorcycle a duplicate vertex q_i, which represents the moving motorcycle. Thus, q_i will move away from p_i according to the speed vector v_i. (We get a function linear in t for the movement of q_i.) Initially, $q_i := p_i$. We call q_i a moving triangulation vertex. We also regard it as one of k moving triangle vertices if k triangles are incident at q_i. (This distinction will be useful in the complexity analysis, see Sec. 2.3.)

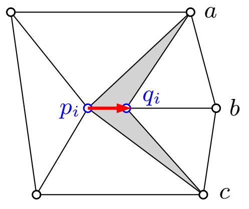


Figure 2: Illustration of initial split

In general, q_i will move into the interior of precisely one triangle $\Delta(p_i,b,a)$, and we replace $\Delta(p_i,b,a)$ by the two degenerate triangles $\Delta(p_i,q_i,a)$ and $\Delta(p_i,b,q_i)$, and by $\Delta(q_i,b,a)$. (All triangle vertices are always kept in counter-clockwise (CCW) order.) If q_i moves along the edge (p_i,b) of the non-degenerate triangles $\Delta(p_i,b,a)$ and $\Delta(p_i,c,b)$, see Fig. 2, then we replace these two triangles by the two degenerate triangles $\Delta(p_i,q_i,a)$ and $\Delta(p_i,c,q_i)$, and by $\Delta(q_i,b,a)$ and $\Delta(q_i,c,b)$. Similarly if both vertices of an edge correspond to start points of motorcycles and, thus, degenerate triangles are already present. In any case, this initial split of all moving triangle vertices from their start points results in the generation of only a constant number of new triangles per vertex.

We note that we may deviate from these strict rules as long as the topology is not violated and no wall is altered. For instance, if q_i of Fig. 2 would move into the interior of $\Delta(p_i, b, a)$ but rather close to the edge (p_i, b) then we could still use the split depicted if (p_i, b) is no wall, thus avoiding to split off the sliver triangle $\Delta(p_i, b, q_i)$.

We start the event processing after all moving triangulation vertices have been split from their start vertices. A priority queue maintains a list of collapsing triangles as events, sorted by their collapse time. The three main types of events are flip event, crash event, and stop event. We discuss these events below. After all motorcycles have stopped moving, no further triangulation vertex moves and no triangle collapses. Thus, at that point in time the priority queue is empty and we can continue with the second part of the algorithm, see Sec. 2.2.

A *flip event* occurs when the vertex a of the triangle $\Delta(a, b, c)$ ends up on the edge (b, c) which is not a wall, motorcycle trace or edge of $CH(G)$. Within the quadrilateral formed with its neighbor $\Delta(b, d, c)$ on the other side of the edge (b, c), the diagonal is flipped such that the triangles $\Delta(a, d, c)$ and $\Delta(b, d, a)$ are generated, cf. Fig. 3. As no triangulation vertex changes its speed, all that remains to do is to update the priority queue by rescheduling the collapse events of the two triangles.

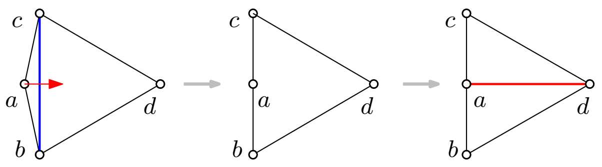


Figure 3: Handling of flip event.

A *crash event* occurs when a motorcycle, i.e., a moving triangulation vertex, reaches the trace of another motorcycle or a wall. A crash events manifests itself as a triangle collapse, which is handled similar to a flip event, except that the moving vertex needs to be halted. Of course, halting a moving vertex requires to re-schedule all triangles attached to this vertex. As a special case we get a vertex collapse if two vertices of a triangle become coincident. Vertex collapses lead to the removal of the edge between two vertices and their incident triangles, and the two vertices are fused to one. This can be seen as the reverse of the initial split, recall Fig. 2.

A *stop event* occurs when a motorcycle reaches $CH(G)$. The handling of this event is very similar in concept to crash events. The only difference is that stopped motorcycles are awakened again in the second part of the algorithm.

2.2 Computation Outside of Convex Hull $CH(G)$

We apply a generalized plane sweep, whose front is given by increasingly larger copies of $CH(G)$. This can be seen as a generalization of the approach sketched by Erickson [5] for motorcycles whose speed vectors do not span more than 180°. As the front advances the common vertex of two neighboring edges of $CH(G)$ moves along their outwards angular bisector. Hence, the exterior of $CH(G)$ is partitioned into individual sweep regions by the bisectors, with one region per edge of $CH(G)$.

All motorcycles that start on $CH(G)$ and move away from it or that were stopped during the first phase, when reaching $CH(G)$, are stored in cyclic order in a doubly-linked circular list. This list represents the front of the sweep. We do not need to maintain the front as a search tree since the only case where a new motorcycle needs to be inserted into the front after the initial set-up happens at a joint crash position of two or more motorcycles; in this case we have handles to the position, and do not need to search for the correct insert position.

During the sweep only one type of event needs to be handled: A *crash event* occurs when a motorcycle track intersects the motorcycle track of a neighboring motorcycle, where "neighboring" is defined relative to the sorted cyclic order of motorcycles in the front of the sweep. The motorcycle that is second at the intersection position is stopped. If two or more motorcycles meet in the same intersection position p at the same time, they are all stopped and a new motorcycle that moves further away from $CH(G)$ is inserted at p. (Recall that a local disc at p is tessellated into convex slices by the traces of the motorcycles according to our definition of $\mathcal{M}(G)$.)

The priority queue is sorted in increasing order of the distances of the event positions to $CH(G)$. Hence, all events are handled in the order as they occur relative to the front of the sweep, which is not necessarily the chronological order. In order to compute the distance to $CH(G)$ we use bisection on $CH(G)$ for determining the sweep region that contains the event position.

2.3 Complexity Analysis

Complexity of computation inside of $CH(G)$. The initial constrained Delaunay triangulation within $CH(G)$ contains $O(n)$ triangles and can be computed in $O(n \log n)$ time [3]. For each of the n initial motorcycles we create two new degenerate triangles in the initial split. Afterwards, one new motorcycle is only created if at least two motorcycles have crashed. Hence, the overall number of motorcycles (resp. moving triangulation vertices) and the number of triangles created at the start times of all motorcycles are in $O(n)$.

For each initial motorcycle we have to determine the triangle(s) that the motorcycle runs into. Since at most a constant number of motorcycles is allowed to share a starting point, we can determine all those triangles in $O(n)$ time by means of brute-force searches around each start vertex. And since the events for only a constant number of triangles need to be stored in the priority queue, the priority queue after all initial splits can be set up in $O(n \log n)$ time.

While flip events do not affect the number of moving triangulation vertices, every flip may increase the number of moving triangle vertices by two: Assume that a and d of Fig. 3 are moving triangle vertices, while b and c do not move. Since after the flip a, d are counted as moving triangle vertices for each triangle on either side of (a, d), the number of moving triangle vertices has increased by two. Hence, f edge flips increase the number of moving triangle vertices by at most $2f$.

If a crash or stop event occurs for a moving triangulation vertex q then we have to re-schedule all triangles incident at q. Since we crash or stop a moving triangulation vertex at most once, the overall number of triangles that need to be re-scheduled equals the overall number of moving triangle vertices, which is in $O(n + f)$. Thus, the complexity of updating the priority queue for handling all crash and stop events is $O((n + f) \log n)$, which also models the worst-case complexity of the entire first phase of our algorithm.

Complexity of computation outside of $CH(G)$. The distance of one event position from $CH(G)$ can be determined in $O(\log n)$ time. The priority queue that stores the events defined by neighboring motorcycles is initialized in $O(n \log n)$ time. The handling of a crash event takes $O(k \log n)$ time, where k denotes the number of motorcycles stopped. Since each crash reduces the number of active motorcycles by at least one, the complexity of handling all crash events is $O(n \log n)$. Summarizing, the total complexity of the second phase of the algorithm is $O(n \log n)$.

One may wonder whether this complexity bound is tight. Consider n motorcycles which have their start points on the x-axis and whose speed vectors have positive y-coordinates. That is, all motorcycles move upwards in the direction of the positive y-axis. If the start points are given in sorted order relative to their x-coordinates then our plane-sweep algorithm requires $O(n \log n)$ time to compute the motorcycle graph. But can one do better? Note that if their sorted order is unknown then a $\Omega(n \log n)$ bound can be shown by a reduction to sorting: Assume that n distinct natural numbers $c_1, \ldots, c_n$ are given. Then we define for each c_i a motorcycle m_i starting at $(c_i, 0)$, with velocity $(-1, 2^{-c_i})$. This guarantees that each motorcycle crashes into the motorcycle starting left to it, except for the left-most motorcycle, which escapes. Hence, we can determine the sorted order of $c_1, \ldots, c_n$ in $O(n)$ time from $\mathcal{M}(m_1, \ldots, m_n)$.

Overall Runtime Complexity. The worst-case complexity is $O((n + f) \log n)$, where f denotes the number of flip events. A flip event requires the area of a triangle to become zero. As all moving triangulation vertices move with constant speeds along rays (until they stop), the signed area of a triangle can be expressed as a quadratic polynomial in time and, hence, a single triangle with three moving vertices can collapse at most at two single points in time. Similarly if one or two vertices have been stopped. Hence, by an argument similar to [1, Lem. 5], we get a trivial $O(n^3)$ bound on the number of flip events. This gives $O(n^3 \log n)$ as total worst-case complexity. However, note that we are not aware of any input that leads to $\omega(n^2)$ flip events. (But one can design inputs that exhibit $\Theta(n^2)$ flip events.)

3 Heuristics for Reducing the Number of Flip Events

It is known that a PSLG G and its motorcycle graph $\mathcal{M}(G)$ partition the plane into a set of convex polygons. Suppose that we overlay $\mathcal{M}(G)$ and G, and triangulate the resulting convex polygons arbitrarily. The edges of $\mathcal{M}(G)$ are called Steiner tracks, and their final points are called Steiner points. Then each motorcycle would move along a triangulation edge, and because no other triangulation edge crosses its track, its movement does not require any edge to be flipped to let it pass through the triangulation. Triangulation edges never leave the convex polygon they were created in because they are always stopped when their incident vertices hit a Steiner point. Thus, for this triangulation our algorithm would be free of flip events.

Of course, the crash positions of the motorcycles are not known to us. But we can try to approximate a portion of the unknown Steiner tracks, in an attempt to reduce the number of flip events by enabling motorcycles to move along triangulation edges.

If Steiner points are present in the triangulation then we handle a point collapse between a moving triangulation vertex (motorcycle) and a Steiner point as follows: We stop the motorcycle at the Steiner point, which is handled like any other vertex collapse, and restart it with the same method as used for the initial split.

In our first heuristic we exploit the average track length of n motorcycles that start from within the unit square, as established in [8]: for each motorcycle we insert a line segment (in the direction of its movement) of length $c/\sqrt{n}$ as Steiner track, for some constant $c > 0$. The second heuristic inserts Steiner tracks of unlimited length for $c \cdot \sqrt{n}$ randomly chosen motorcycles. In both heuristics a Steiner track is terminated at the first intersection if it crosses a wall or intersects $CH(G)$. A intersection between Steiner tracks is resolved by adding an additional Steiner point at the intersection. Due to our choice of the Steiner tracks one may assume for both heuristics that at most $O(n)$ points of intersection need to be added to the input as Steiner points.

Stopping Steiner tracks at walls and $CH(G)$ is done by running a plane sweep twice, once top-down, once bottom-up. Walls and the convex hull segments are inserted as normal line segments, but each motorcycle is only inserted in the phase that fits its direction. When a motorcycle first intersects a wall or convex hull segment, it is removed and a Steiner point is placed at the intersection. A third plane sweep is done to resolve intersections between Steiner tracks, also adding Steiner points at the intersections.

4 Implementational Issues

4.1 Simultaneous and Out-of-Order Events

A standard problem of any algorithm that uses a kinetic data structure is the reliable computation of the times when the structure changes. This problem is known as “root sorting”, i.e., determining which root of which polynomial occurs first. If root sorting is not guaranteed to be exact, e.g., due to the use of floating-point arithmetic, then some form of out-of-order processing of events is required in order to achieve reliability.

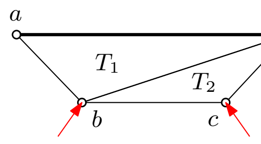


Figure 4: Simultaneous collapse of two triangles.

We note that our algorithm has to cope with a second problem in addition to the handling of out-of-order events: So far we have ignored the fact that events can occur simultaneously for real-world data. Consider Fig. 4, and suppose that the two vertices a and d are non-moving vertices, while the vertices b and c represent moving motorcycles that meet the edge (a, d) at the same time. Further assume that (a, d) is a normal triangulation edge, rather than a wall or motorcycle trace that would stop the motorcycles b and c. As the collapse times of T_1 and T_2 are identical, it is a matter of chance which flip event is processed first. If the event associated with T_2 is handled first then we flip the diagonal (b, d) to the diagonal (a, c), resulting in the triangles $\Delta(a, b, c)$ and $\Delta(a, c, d)$. If we now happen to choose triangle $\Delta(a, b, c)$ as next triangle to process then that diagonal will just flip back, and we have ended up in a loop. We emphasize that this problem occurs both on floating-point and exact arithmetic.

In order to handle simultaneous and out-of-order events we employ a strategy described in [10]. Roughly, a history of all events processed so far allows to detect a loop. If a loop is encountered then all events of the loop are considered to occur exactly at the same time and replaced by a set of events that guarantee progress.

4.2 Numerical Aspects

As noted, the computation of the collapse times is a challenging problem when using a floating point arithmetic. While the movement of each motorcycle is described by a linear function in time t, the signed area of a triangle with two moving vertices becomes a quadratic function in t.

Since we keep the vertices of all triangles in CCW order, the signed area of a triangles always is positive until the triangle collapses. Note, however, that in the case of a vertex collapse the area of a triangle may be positive again after the collapse time plus some positive epsilon. Fortunately, the degree of the polynomial which describes a vertex collapse can be reduced: We note that the time of a vertex collapse can also be calculated by a linear function, as it corresponds to the minimum of a function modeling the distance of two points moving with constant speeds along two lines. Similarly, all other events where a motorcycle is stopped involve triangles where only one vertex, namely the motorcycle being stopped, is moving. So the collapse times of events that stop motorcycles can be obtained by solving linear equations.

5 Experimental Results

We implemented our algorithm and both heuristics for reducing the number of flip events in C++. Shewchuk’s Triangle [11] is employed for computing the initial constrained Delaunay triangulation.

The following tests were conducted on a Intel Core i7 X980 CPU clocked at 3.33 GHz, with Ubuntu 10.04.4 LTS and GCC version 4.4.3. The memory usage was limited to 4.5 GB, and the runtime on each file was limited to 15 minutes by means of the `ulimit` command. We computed generalized motorcycle graphs for both real-world and contrived data of different characteristics. In order to avoid unreliable timings and other idiosyncrasies of small datasets, we only analyze test runs that involved at least 1 000 motorcycles, resulting in a few thousand tests covered by our experiments.

The left plot of Fig. 5 shows the runtimes of our code divided by $n \log n$, where n denotes the number of vertices. The plot shows clearly that the runtime (in seconds) can be modeled by the function $c \cdot 10^{-6} \cdot n \log n$, with $c \leq 4$ for virtually all of our inputs.

This runtime behavior suggests a linear number of flip events, which is confirmed by our tests. The right plot of Fig. 5 shows the number f of flip events divided by n. As can be seen, we get $2n$ flip events on average and $0.8n$ to $5n$ flip events for virtually all inputs. The maximum number of flip events recorded was $39n$ for an input with roughly $n = 60\,000$ motorcycles.

A closer inspection of the test results reveals that an increased runtime of our code is caused by either an abnormal runtime of Triangle [11], which is used to compute the initial constrained Delaunay triangulation, or by an increased number of flip events, or by a combination of both. For instance, for most inputs Triangle consumes about 5–20% of the total runtime, but we witnessed in-

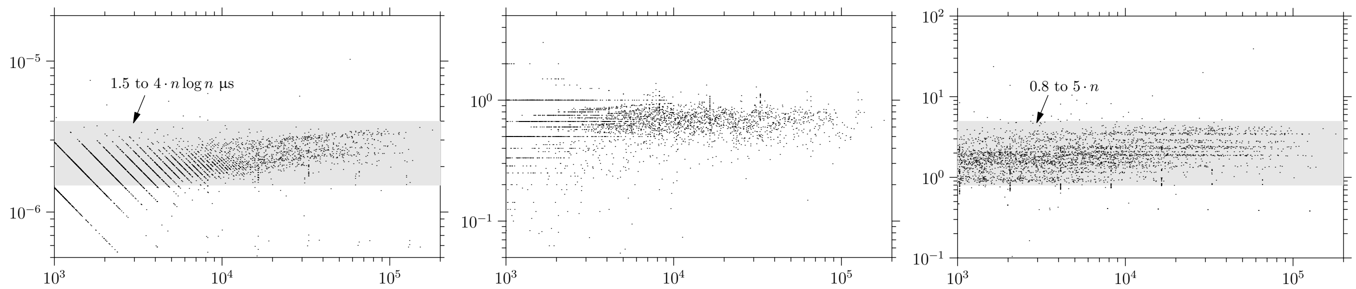


Figure 5: Experimental results for our code without Steiner tracks: In all three plots the x-axis corresponds to the number n of vertices. The left plot shows the runtimes divided by $n \log n$. The middle plot shows the ratio of the runtimes of our code divided by the runtimes of Moca, and the right plot shows the number of flip events divided by n.

puts for which it consumed 85% of the total runtime.

The fact that our code truly runs in $O(n \log n)$ time for most data is also confirmed by a comparison with Moca [8]. The middle plot of Fig. 5 shows the runtimes of our code divided by the runtimes of Moca. On average, our code needs about 32% less runtime than Moca. It is rarely slower than Moca, being at most twice as slow. However, our code is substantially faster for those inputs which cause Moca to consume $O(n^2 \log n)$ time.

Our heuristics for reducing the number of flip events turned out to be a mixed blessing. While a combination of both heuristics did indeed manage to reduce the number of flip events by about 20%, the preprocessing necessary for computing the intersections among the Steiner tracks and with the walls caused the runtime to increase. Apparently, performing plane sweeps is significantly more costly than what our heuristics manage to save by reducing the number of flips.

We also tested our code with the MPFR library [6] for multiple-precision computations, and witnessed an average slow-down by a factor of 25 for an MPFR precision of 212. (Plots are omitted due to lack of space.)

6 Conclusion

We developed and implemented a triangulation-based algorithm for the computation of generalized motorcycle graphs. While its theoretical worst-case time complexity is worse than prior art, our experiments demonstrate that it runs in $O(n \log n)$ time for virtually all inputs. Our new algorithm is an improvement over Moca as it clearly outperforms Moca in our runtime tests and its runtime does not depend on a sufficiently uniform distribution of motorcycles. Our experiments also show that our algorithm requires only $O(n)$ flip events in practice, and that this number can be reduced by the use of Steiner tracks. It remains an interesting problem to come up with more sophisticated methods to place Steiner tracks. After all, if the number of flip events could deterministically be reduced to $O(n)$ then our algorithm would run in optimal $O(n \log n)$ worst-case time.

References

[1] O. Aichholzer and F. Aurenhammer. Straight skeletons for general polygonal figures in the plane. In A. Samoilenko, editor, *Voronoi's Impact on Modern Science, Book 2*, pages 7–21. Institute of Mathematics of the National Academy of Sciences of Ukraine, Kiev, Ukraine, 1998.

[2] S.-W. Cheng and A. Vigneron. Motorcycle graphs and straight skeletons. *Algorithmica*, 47:159–182, Feb 2007.

[3] L. Chew. Constrained Delaunay triangulations. *Algorithmica*, 4(1):97–108, 1989.

[4] D. Eppstein and J. Erickson. Raising roofs, crashing cycles, and playing pool: applications of a data structure for finding pairwise interactions. *Discrete Comput. Geom.*, 22(4):569–592, 1999.

[5] J. Erickson. Crashing motorcycles efficiently. `http://compgeom.cs.uiuc.edu/~jeffe/open/cycles.html`.

[6] GNU. The GNU MPFR library. `http://www.mpfr.org/`.

[7] S. Huber and M. Held. Approximating a motorcycle graph by a straight skeleton. In *Proc. 23rd Canad. Conf. Comput. Geom. (CCCG 2011)*, pages 261–266, Toronto, Canada, Aug 2011.

[8] S. Huber and M. Held. Motorcycle graphs: stochastic properties motivate an efficient yet simple implementation. *ACM J. Experimental Algorithmics*, 16:1.3:1.1–1.3:1.17, May 2011.

[9] S. Huber and M. Held. Theoretical and practical results on straight skeletons of planar straight-line graphs. In *Proc. 27th Annu. ACM Sympos. Comput. Geom.*, pages 171–178, Paris, France, June 2011.

[10] P. Palfrader, M. Held, and S. Huber. On computing straight skeletons by means of kinetic triangulations. In *Proc. ESA'12*, to appear Sep 2012.

[11] J. Shewchuk. Triangle: engineering a 2D quality mesh generator and Delaunay triangulator. In *1st ACM Workshop Appl. Comput. Geom.*, pages 124–133, Philadelphia, PA, USA, May 1996.

Variable Radii Poisson-Disk Sampling

Scott A. Mitchell[*] Alexander Rand[†] Mohamed S. Ebeida[‡] Chandrajit Bajaj[§]

Abstract

We introduce three natural and well-defined generalizations of maximal Poisson-disk sampling. The first is to decouple the disk-free (inhibition) radius from the maximality (coverage) radius. Selecting a smaller inhibition radius than the coverage radius yields samples which mix advantages of Poisson-disk and uniform-random samplings. The second generalization yields hierarchical samplings, by scaling inhibition and coverage radii by an abstract parameter, e.g. time. The third generalization is to allow the radii to vary spatially, according to a formally characterized sizing function. We state bounds on edge lengths and angles in a Delaunay triangulation of the points, dependent on the ratio of inhibition to coverage radii, or the sizing function's Lipschitz constant. Hierarchical samplings have distributions similar to those created directly.

1 Maximal Poisson-disk Sampling

A *sampling* is a set of ordered points taken from a domain at random. Each point is the center of a disk that precludes additional points inside it, but points are otherwise chosen uniformly. The sampling is maximal if the entire domain is covered by disks. Together these define maximal Poisson-disk sampling (MPS).

More formally, a sampling $X = (\mathbf{x}_i)_{i=1}^n$, $\mathbf{x}_i \in \Omega$ satisfies the *inhibition* or *empty disk* property if

$$\forall i < j \leq n, |\mathbf{x}_i - \mathbf{x}_j| \geq r. \tag{1}$$

The set of *uncovered points* is defined to be

$$S(X) = \{\mathbf{y} \in \Omega : |\mathbf{y} - \mathbf{x}_i| \geq r, \, i = 1..n\}. \tag{2}$$

A sampling X is *maximal* if $S(X)$ is empty:

$$S(X) = \emptyset. \tag{3}$$

Given a non-maximal sampling, the next sample is *bias-free* if the probability of selecting it from any uncovered subregion is proportional to the subregion's area, i.e.,

$$\forall A \subset S(X) : P(\mathbf{x}_{n+1} \in A \,|\, X) = \frac{|A|}{|S(X)|}. \tag{4}$$

[*]Sandia National Laboratories, samitch@sandia.gov

[†]Institute for Computational Engineering and Sciences, The University of Texas at Austin

[‡]Sandia National Laboratories

[§]Dept. of Computer Science and Institute for Computational Engineering and Sciences, The University of Texas at Austin

We generalize these equations: decoupling the radii in the empty disk and uncovered equations; scaling the radii for a hierarchy of denser samplings; and varying the radii spatially by a sizing function.

The purpose of this short paper is to introduce these generalizations in a mathematically consistent way. Examples illustrate the properties of the resulting output distributions. For simplicity our language is two-dimensional, e.g. "disks" instead of "spheres," but the definitions are general dimensional. Also for simplicity, we consider only periodic (or free-boundary) domains. These domains are used in some applications: computer graphics texture synthesis and mesh generation of material grains.

2 Motivation and Previous Work

An MPS sampling is a separated-yet-dense point set: points are not too close together and lie throughout the entire domain. This is an efficient way to distribute a fixed budget of points.

In mesh generation, separated-yet-dense points yield Delaunay triangulations (DT) with provable quality bounds [4, 9, 19]. Delaunay Refinement (DR) [20] introduces points to improve DT triangle quality and a separated-yet-dense point set follows. Variations of DR provide adaptivity and sizing control [16]. DR is usually deterministic; although regions of acceptable points have been characterized [12, 13], and one may select from regions randomly to improve tetrahedron quality [5], randomized point positions are not a traditional *requirement.* However, random meshes are of independent interest for certain applications; e.g. in some fracture mechanics methods, cracks propagate only along mesh edges. Meshes from MPS produce more physically realistic cracks [1, 2, 8, 7]. Ensembles of MPS meshes can model natural material strength variations.

In a sphere packing no two disks overlap. If the disk radii satisfy a Lipschitz condition then a quality mesh results [19]. As in MPS and in reverse to DR, algorithms add disks until the packing is (nearly) maximal, and a good-quality DT follows. A fixed-r MPS sampling is a sphere packing: halve the disk radius r so no disks overlap. We define four new spatial variations for MPS, however none are equivalent to maximal sphere packings. Conflicts are defined by disks containing each other's centers; for unequal radii this is not equivalent to non-overlapping 1/2-radii disks. Also, we achieve a maximal distribution following a characterized statistical process.

MPS is popular for computer graphics [15] for texture synthesis because the distribution avoids repeating patterns of distances between points which produce visible artifacts. Fixed radius disks are traditional, but not suitable in all situations.

In real-time games and data exploration [17] with level-of-detail adaptivity, renderings use a finer sampling as the camera zooms in. Switching between discrete sets of samples is common, but has the potential to introduce visible artifacts or scene jumps [23]. Our definitions enable smoothly increasing density in time. Spatially varying samplings are useful for objects with varying curvature and lighting [3, 14]. Curvature and solution gradients motivate spatially-varying finite element meshes, and incremental adaptivity is preferred over mesh replacement.

Varying density sampling is popular in Graphics but often the algorithms are heuristic, and the requirements not well understood. This paper seeks to provide some formal guidance. For example, the spatially-varying sampling algorithm of Bowers et al. [3] uses a datastructure that holds all the nearby points whose Poisson-disks might conflict with a new point. This datastructure sometimes overflows in practice. We show that this is the fault of the input and not their algorithm: the **bigger-disks** criteria in Section 5 shows that a sizing function with Lipschitz constant $L < 1/2$ is necessary to bound the number of nearby points.

Classic dart throwing [6] generates samples and rejects those inside prior disks. The probability of generating an acceptable sample becomes vanishingly small, so maximality is not reached. After many rejected samples McCool and Flume [18] reduce the radii of disks, either locally or globally, to make room for more samples. An adaptive MPS variation [23] for deforming point clouds coarsens to remove points that are too close together, and refines to re-achieve maximality. For coarsening the disk-free and maximal criteria hold approximately, subject to a tolerance band. In Section 3 we effectively tune this tolerance band by the ratio of the two radii, and scale the radii continuously in Section 4.

3 Different inhibition and coverage radii

Here we relax the condition that the coverage and inhibition radii are equal. We focus on a particular relaxation that proves useful for generating hierarchical point sets, and flatter FFT radial power spectra.

Let $R_f \leq R_c$ denote the inhibition and coverage radii, respectively. The *empty disk* property is

$$\forall i < j \leq n, |\mathbf{x}_i - \mathbf{x}_j| \geq R_f. \tag{5}$$

The set of *free* points is defined to be

$$S(X) = \{\mathbf{y} \in \Omega : |\mathbf{y} - \mathbf{x}_i| \geq R_f,\ i = 1..n\}. \tag{6}$$

The set of *uncovered* points is defined to be

$$U(X) = \{\mathbf{y} \in \Omega : |\mathbf{y} - \mathbf{x}_i| \geq R_c,\ i = 1..n\}. \tag{7}$$

The sampling is *maximal* if $U(X)$ is empty,

$$U(X) = \emptyset. \tag{8}$$

For this variation to be useful and different than the single radius case, we sample from S, but restrict to points that are close enough to U to reduce it:

$$T(X) = S(X) \cap \{U(X) + R_c\}. \tag{9}$$

The *bias-free* process selects from $T(X)$ uniformly.

This variation is useful to add randomness to initial and continuously parameterized hierarchical samples. Samplings will likely have points that could be removed and still meet the coverage condition (Equation 8). There are more extra points the smaller R_f is compared to R_c. This process provides samplings that are less uniform, i.e., with greater variation in inter-sample distances, than classical MPS. In particular, as the ratio of inhibition and coverage radii grows, the rings in the FFT spectrum of the output are reduced. For a modest ratio, $R_c/R_f = 2$, the radial power oscillations are barely perceptible: the resulting FFT spectrum is much closer to a uniform-random distribution, except for the low frequency component. See Figures 4–10 for examples.

3.1 Edge Length and DT Angle Bounds

We consider a Delaunay triangulation (DT) of our point cloud. The inhibition radius bounds the shortest edge length. The coverage radius bounds the largest empty Delaunay circumcircle. The longest edge length is at most the diameter of that circle. To summarize:

Proposition 1 *$|e| \in [R_f, 2R]$ and $R \leq R_c$, where R is the radius of a Delaunay circumcircle.*

The Central Angle Theorem provides a relation between the smallest angle α, the shortest edge length $|e|$, and the circumradius R of a triangle. (This has been used to provide quality bounds for separated-yet-dense points since DR's inception [4].)

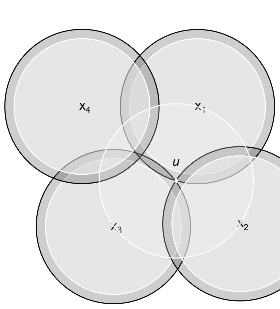
Proposition 2 *$\sin \alpha \geq |e|/2R$.*

For example, in a DT of a point set with $R_c = R_f$, we have $\alpha > 30°$. If $R_c = 2R_f$, then $\alpha > 14.4°$.

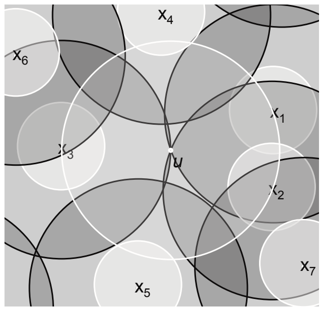
4 Hierarchical Sampling

4.1 Parameterized radii

Consider a maximal sampling, from either a single disk radius or different inhibition and coverage radii. We scale these radii by t; e.g., t could be time. For $t \in (0, 1]$ we have $r_f(t) = tR_f$ and $r_c(t) = tR_c$.



(a) $r_c \approx r_f$



(b) $r_c \ll r_f$.

Figure 1: Possible T shapes for two radii when t is reduced to uncover a single point $u = U$. The circumcircle of $\triangle \mathbf{x}_1\mathbf{x}_2\mathbf{x}_3$ has center u and radius r_c. T is the part of this circumcircle outside the light r_f disks at x_i.

4.2 Continuous Decrease Refinement

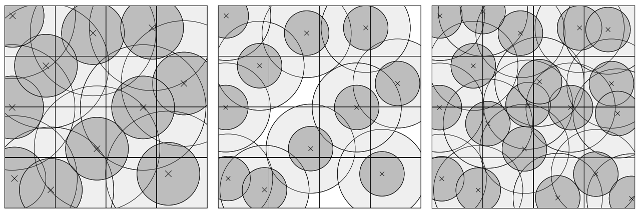
Consider decreasing t continuously from 1 to 0. The sampling becomes non-maximal for some t^* when $U(X) \neq \emptyset$; recall Equation 8. To simplify the discussion assume distinct Delaunay circumradii so the largest one is unique; then at t^* we have that $U(X)$ grows by a single point, a single Voronoi vertex u. A new sample is needed. If $r_f = r_c$ then there is only one place to put the sample, at u, so the process is deterministic. Otherwise, we insert a random point from the set T of free points which will reduce the size of the uncovered set. See Figure 1 for example T shapes. Efficiently selecting a new sample can be done by sampling from a geometric outer approximation to T and resampling if necessary [10, 11].

In 2d periodic or infinite domains, we observe that u is the circumcenter of a non-obtuse triangle, which lies inside it. For obtuse triangles, the Delaunay triangle sharing its longest edge has a larger circumsphere, so its center would be uncovered for a smaller t.

DR can be implemented with a priority queue, prioritizing the circumcenters of Delaunay triangles by decreasing radii. A new sample creates new triangles and destroys some old ones, so the queue must be updated. This is essentially the generic Delaunay refinement algorithm with a largest-first queue priority for inserting circumcenters. DR makes no restrictions on the circumcenter insertion order, and the Triangle code [22] takes the opposite approach: processing the smallest triangles first. The main difference is that when an event occurs, we insert a nearby random point, but DR inserts the point itself (or an off-center, etc.).

4.3 Discrete Decrease Refinement

Consider decreasing t in discrete jumps. For a new value of t, the sampling may be non-maximal, and the same algorithm that generated the initial sampling can be continued to achieve maximality. Figure 2 shows completing a sampling after a jump. Some new samples are inside the light covered region, but, nonetheless, each of their r_c disks reduced the white uncovered area when it was introduced.



(a) $t = 0.8$ end (b) $t = 0.6$ start (c) $t = 0.6$ end

Figure 2: A step in a discrete hierarchy of samplings.

5 Spatially Varying Radii

We aim to produce spatially varying point density according to a sizing function $r(\mathbf{x}) : \Omega \to (0, \infty)$. A sample satisfies the *empty disk* property, vs. (1), if

$$\forall i < j \leq n, |\mathbf{x}_i - \mathbf{x}_j| \geq f(\mathbf{x}_i, \mathbf{x}_j), \tag{10}$$

and the set of *uncovered points*, vs. (2), is

$$S(X) = \{\mathbf{y} \in \Omega : |\mathbf{y} - \mathbf{x}_i| \geq f(\mathbf{x}_i, \mathbf{y}),\ i = 1..n\}. \tag{11}$$

Here $f(\mathbf{x}_i, \mathbf{y})$ is a function of $r(\cdot)$ evaluated at a previously accepted sample $\mathbf{x}_i$ and a later candidate sample y. A candidate is accepted if $|\mathbf{x} - \mathbf{y}| \geq f(\mathbf{x}, \mathbf{y})\, \forall \mathbf{x} \in X$ so far. We have four variations:

$f(\mathbf{x}, \mathbf{y}) := r(\mathbf{x})$	**Prior-disks**,
$f(\mathbf{x}, \mathbf{y}) := r(\mathbf{y})$	**Current-disks**,
$f(\mathbf{x}, \mathbf{y}) := \max(r(\mathbf{x}), r(\mathbf{y}))$	**Bigger-disks**,
$f(\mathbf{x}, \mathbf{y}) := \min(r(\mathbf{x}), r(\mathbf{y}))$	**Smaller-disks**.

(Sphere packings use a sum-of-disks sizing function, $f(\mathbf{x}, \mathbf{y}) = r(\mathbf{x}) + r(\mathbf{y})$.) The f are equivalent for a fixed radius r, but are all distinct for spatially-varying r. Each approach has certain advantages in terms of simplicity, output size, DT quality, and how quickly the sizing function may vary. See Table 1 for a summary, below for proofs for one case, and the extended version of this paper for the other cases.

A variation of Ebeida et al. [10] can efficiently produce a maximal sampling using a flat-quadtree to capture

Method	Distance Function	Order Independent	Full Coverage	Conflict Free	Edge Min	Edge Max	Sin Angle Min	Max L
Prior	$r(\mathbf{x})$	no	no	no	$1/(1+L)$	$2/(1-2L)$	$(1-2L)/2$	$1/2$
Current	$r(\mathbf{y})$	no	no	no	$1/(1+L)$	$2/(1-L)$	$(1-L)/2$	1
Bigger	$\max(r(\mathbf{x}), r(\mathbf{y}))$	yes	no	yes	1	$2/(1-2L)$	$(1-2L)/2$	$1/2$
Smaller	$\min(r(\mathbf{x}), r(\mathbf{y}))$	yes	yes	no	$1/(1+L)$	$2/(1-L)$	$(1-L)/2$	1

Table 1: Summary of results for spatially varying radii. Points closer than f conflict. Symmetric f provide order independence: any sampling with the order of samples permuted still satisfies the empty disk property. Full coverage means that every point of the domain is inside some sample's r disk. Conflict free means that no sample is inside another sample's r disk. Edge max and min bound the lengths of an edge containing $\mathbf{x}$ in a Delaunay triangulation of X, as a factor of $r(\mathbf{x})$. The Lipschitz constant must be less than max L to bound the maximum DT edge length and minimum DT angle.

the uncovered area. Implementing the conflict condition and coverage checks is simpler for some variations.

There is a limit to how quickly $r(\cdot)$ is allowed to vary. We require that r is L-Lipschitz, i.e., for all $\mathbf{x}, \mathbf{y} \in \Omega$, $|r(\mathbf{x}) - r(\mathbf{y})| \leq L\,|\mathbf{x} - \mathbf{y}|$ for some constant L. The lengths of DT edges at $\mathbf{x}$ depend not only on $r(\mathbf{x})$ but also on $r(\mathbf{y})$, which can be bounded using L. Some approaches require $L < 1$, others $L < 1/2$. The quality guarantees disappear as L approaches the upper limit. As L approaches zero the quality guarantees smoothly approach those in the uniform case.

Bias-free An alternative to uniform-random is to weight the uncovered set by the local sizing function, i.e., the desired output density. In dimension d,

$$w(S) = \int_S \frac{1}{r(\mathbf{x})^d}\,\mathrm{d}\mathbf{x}, \text{ and}$$
$$\forall A \subset S(X) : P(\mathbf{x}_{n+1} \in A \,|\, X) = \frac{w(A)}{w(S(X))}. \qquad (12)$$

While we have not implemented it, one could approximate Equation 12 from values at quadtree corners.

Prior-disk Output Guarantees We justify the edge-length and angle guarantees in Table 1 for **prior-disks**. The proofs for the other criteria are similar and are given in the extended version of this paper.

Proposition 3 *If X satisfies the empty disk property, then for all i,j,* $|\boldsymbol{x}_i - \boldsymbol{x}_j| \geq \frac{r(\boldsymbol{x}_i)}{1+L}$.

Proof. If $i < j$, the empty-disk definition implies $|\mathbf{x}_i - \mathbf{x}_j| \geq r(\mathbf{x}_i)$. Otherwise,

$$r(\mathbf{x}_i) \leq r(\mathbf{x}_j) + L\,|\mathbf{x}_i - \mathbf{x}_j| \leq |\mathbf{x}_i - \mathbf{x}_j| + L\,|\mathbf{x}_i - \mathbf{x}_j|$$

by the Lipschitz property and the fact that $\mathbf{x}_i$ satisfies the empty-disk property when it is inserted. □

Proposition 4 *If X is maximal and T is a resulting Delaunay triangle, then the circumradius* $R_T \leq \min\left(\frac{r(\boldsymbol{y})}{1-L}, \frac{r(\boldsymbol{x})}{1-2L}\right)$ *where $\boldsymbol{y}$ is the circumcenter and $\boldsymbol{x}$ is any triangle vertex.*

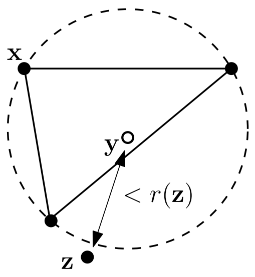


Figure 3: Notation for proofs of circumradii bounds in the Delaunay triangulation of a maximal sampling.

Proof. Since X is maximal, $|\mathbf{z} - \mathbf{y}| \leq r(\mathbf{z})$ for some sample $\mathbf{z} \in X$, where $\mathbf{z}$ is not required to be a vertex of T; see Figure 3. The Lipschitz property gives

$$|\mathbf{z} - \mathbf{y}| \leq r(\mathbf{z}) \leq r(\mathbf{y}) + L\,|\mathbf{z} - \mathbf{y}|\,.$$

Rearranging gives $R_T \leq |\mathbf{z} - \mathbf{y}| \leq \frac{r(\mathbf{y})}{1-L}$. Applying the Lipschitz property again gives,

$$R_T = |\mathbf{x} - \mathbf{y}| \leq |\mathbf{z} - \mathbf{y}| \leq \frac{r(\mathbf{y})}{1-L} \leq \frac{r(\mathbf{x}) + L\,|\mathbf{x} - \mathbf{y}|}{1-L}.$$

Rearranging again completes the proof. □

Corollary 5 *If X is maximal,* $|\boldsymbol{x}_i - \boldsymbol{x}_j| \leq \frac{2r(\boldsymbol{x}_i)}{1-2L}$.

Lemma 6 *Suppose X is a maximal sample satisfying the empty disk property. Then all the angles in the Delaunay triangulation are at least* $\arcsin\left(\frac{1-2L}{2}\right)$.

Proof. Let α be an angle in the Delaunay triangulation of X and let $\mathbf{x}$ be the vertex on the edge opposite of α which was inserted *first*. This opposite edge has length at least $r(\mathbf{x})$. Propositions 2 and 4 give $\sin\alpha \geq \frac{r(\mathbf{x})}{2r(\mathbf{x})/(1-2L)} = \frac{1-2L}{2}$. □

6 Experimental Results

We consider the spectra of distributions generated with the different methods, but similar coverage/inhibition radii. Spectra are analyzed using the Point Set Analysis [21] tool, which generates standardized diagrams, aiding direct comparison. The first panel is the point set. The second panel is the FFT spectrum of the point set with the DC component removed. The third panel is the radial mean power, which measures the average variation of the second panel's rings' magnitudes.

Figure 4 is for uniform MPS. In the FFT spectrum we see the typical dark central disk surrounded by alternating light and dark rings decreasing in magnitude. Figures 5 and 8 show the PSA results for point clouds generated using different inhibition and coverage radii. The FFT ringing artifacts are dramatically reduced, as is the size of the central disk. Comparing Figures 4 & 5 to 6 & 7 shows that there is little difference in the spectra whether a sampling is generated in a discrete hierarchy over t or directly. Using a large coverage radius yields significantly fewer samples, as seen in Figures 8 and 9.

Figure 11 shows sampling results using the same pseudo-random number sequence over all four spatially varying radii strategies. The experimental results match the theory: the smaller-disk construction yields a larger minimum angle. Figure 12 shows our resampling of a stippled image [14, 24].

7 Conclusions

We provide simple definitions for separated-yet-dense random samplings, which are amenable to simple algorithms for generating provable quality point sets and meshes. Intermediate triangulations and Delaunay circumspheres are not needed. Of our spatial variations, the smaller-disks approach has the weakest requirements and provides the best quality, but generates the most points. The prior-disks method is the easiest to implement, as it is a minor change to existing MPS algorithms. However, it has the most restrictions on the input and provides the weakest output guarantees.

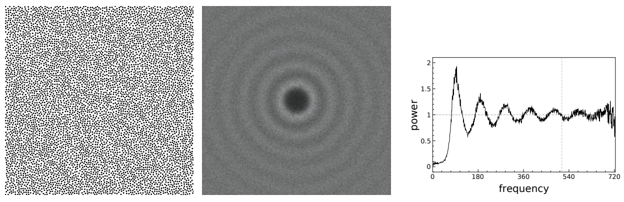
Acknowledgements

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. The research of the authors from University of Texas at Austin was supported in part by NSF grant CNS-0540033 and NIH grant R01-EB004873.

References

[1] J. Bishop. Simulating the pervasive fracture of materials and structures using randomly close packed Voronoi tessellations. *Comput. Mech.*, 44:455–471, 2009.

[2] J. E. Bolander and S. Saito. Fracture analyses using spring networks with random geometry. *Eng. Fracture Mech.*, 61(5-6):569–591, 1998.



(a) Point Set (b) FFT Spectrum (c) Radial Power

Figure 4: Uniform MPS with $r = 0.01$ (6656 points).

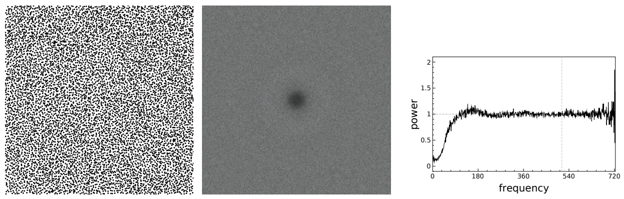


Figure 5: Two-radii, $r_c = 2r_f = 0.01$ (8566 points).

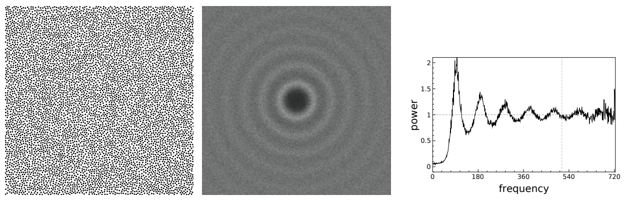


Figure 6: Final sampling in a ten step discrete hierarchy terminating with $r_c = r_f = 0.01$ (6727 points).

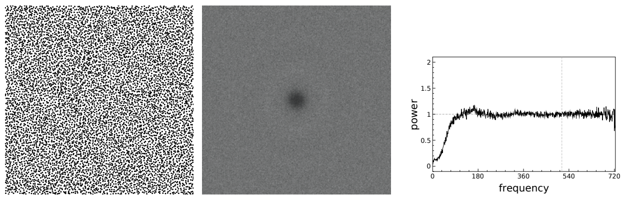


Figure 7: Ten steps to $r_c = 2r_f = 0.01$ (8432 points).

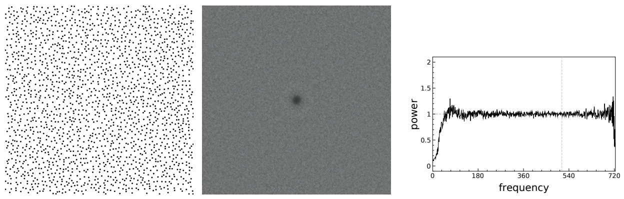


Figure 8: Two-radii, $\frac{r_c}{2} = r_f = 0.01$ (2010 points).

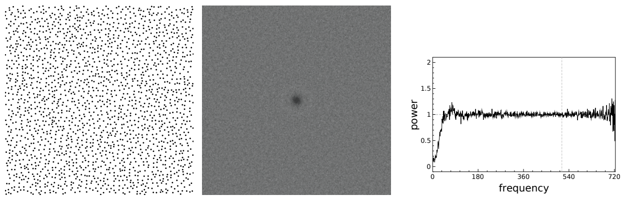


Figure 9: Ten steps to $\frac{r_c}{2} = r_f = 0.01$ (2006 points).

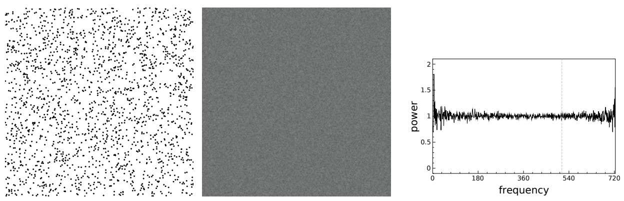
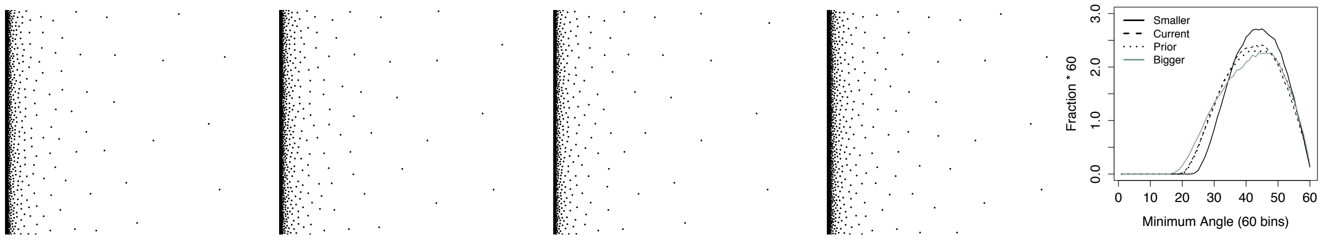


Figure 10: Uniform-random sampling (2010 points).



(a) prior-disks (b) current-disks (c) bigger-disks (d) smaller-disks (e) angles

Figure 11: Experimental results for spatially varying radii samplings across conflict criteria. Left, sampling the linear-ramp function, $r(x, y) = 0.001 + 0.3x$. Right, typical DT angle histograms for an r with $L \approx 0.37$.

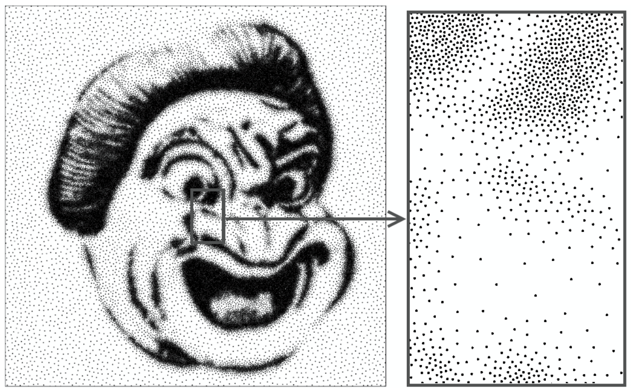


Figure 12: A point cloud was scanned, grayscaled, smoothed for L, then resampled. (Wei, Kopf et al.)

[3] J. Bowers, R. Wang, L.-Y. Wei, and D. Maletz. Parallel Poisson disk sampling with spectrum analysis on surfaces. *ACM Trans. Graphics*, 29:166:1–166:10, 2010.

[4] L. P. Chew. Guaranteed-quality triangular meshes. Technical Report 89-983, Department of Computer Science, Cornell University, 1989.

[5] L. P. Chew. Guaranteed-quality Delaunay meshing in 3D. In *Proc. 13th Symp. Comput. Geom.*, pages 391–393, 1997.

[6] R. Cook. Stochastic sampling in computer graphics. *ACM Trans. Graphics*, 5(1):51–72, 1986.

[7] M. S. Ebeida, P. M. Knupp, V. J. Leung, J. E. Bishop, and M. J. Martinez. Mesh generation for modeling and simulation of carbon sequestration process. In *Proc. DOE Scientific Discovery through Advanced Computing (SciDAC) conference*, July 2011.

[8] M. S. Ebeida and S. A. Mitchell. Uniform random Voronoi meshes. In *Proc. 20th Int. Meshing Roundtable*, pages 258–275, 2011.

[9] M. S. Ebeida, S. A. Mitchell, A. A. Davidson, A. Patney, P. M. Knupp, and J. D. Owens. Efficient and good Delaunay meshes from random points. *Comput. Aided Des.*, 43(11):1506–1515, 2011.

[10] M. S. Ebeida, S. A. Mitchell, A. Patney, A. A. Davidson, and J. D. Owens. A simple algorithm for maximal Poisson-disk sampling in high dimensions. *Comput. Graphics Forum*, 31(2):tbd, 2012.

[11] M. S. Ebeida, A. Patney, S. A. Mitchell, A. Davidson, P. M. Knupp, and J. D. Owens. Efficient maximal Poisson-disk sampling. *ACM Trans. Graphics*, 30(4):49:1–49:12, 2011.

[12] H. Erten and A. Üngör. Quality triangulations with locally optimal Steiner points. *SIAM J. Sci. Comput.*, 31:2103, 2009.

[13] P. Foteinos, A. Chernikov, and N. Chrisochoides. Fully generalized 2D constrained Delaunay mesh refinement. *SIAM J. Sci. Comput.*, 32:2659–2686, 2010.

[14] J. Kopf, D. Cohen-Or, O. Deussen, and D. Lischinski. Recursive Wang tiles for real-time blue noise. *ACM Trans. Graphics*, 25(3):509–518, 2006.

[15] A. Lagae and P. Dutré. A comparison of methods for generating Poisson disk distributions. *Comput. Graphics Forum*, 27(1):114–129, 2008.

[16] X.-Y. Li, S.-H. Teng, and A. Üngör. Simultaneous refinement and coarsening: Adaptive meshing with moving boundaries. In *Proc. 7th Int. Meshing Roundtable*, pages 201–210, 1998.

[17] P. Ljung. Adaptive sampling in single pass, GPU-based raycasting of multiresolution volumes. In *Eurographics*, pages 39–46, 2006.

[18] M. McCool and E. Fiume. Hierarchical Poisson disk sampling distributions. In *Graphics Interface*, pages 94–105, 1992.

[19] G. L. Miller, D. Talmor, S.-H. Teng, N. J. Walkington, and H. Wang. Control volume meshes using sphere packing: Generation, refinement and coarsening. In *Proc. 5th Int. Meshing Roundtable*, pages 47–61, 1996.

[20] J. Ruppert. A Delaunay refinement algorithm for quality 2-dimensional mesh generation. *J. Algorithms*, 18(3):548–585, 1995.

[21] T. Schlömer. PSA point set analysis. Version 0.2.2, `http://code.google.com/p/psa/`, 2011.

[22] J. R. Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Comput. Geom.*, 22(1–3):86–95, 2002.

[23] D. Vanderhaeghe, P. Barla, J. Thollot, and F. Sillion. Dynamic point distribution for stroke-based rendering. In *Rendering Techniques*, pages 139–146, 2007.

[24] L.-Y. Wei. Parallel Poisson disk sampling. *ACM Trans. Graphics*, 27(3):20:1–20:9, Aug. 2008.

Cannons at Sparrows[*]

Günter M. Ziegler[†]

The story told in this lecture starts with an innocuous little geometry problem (one that Erdős would have liked), posed in a September 2006 blog entry by R. Nandakumar, an engineer from Calcutta, India: "Can you cut every polygon into a prescribed number of convex pieces that have equal area and equal perimeter?" This little problem is a "sparrow", tantalizing, not as easy as one could perhaps expect, and *Recreational Mathematics*: of no practical use.

I will sketch, however, how this little problem connects to very serious mathematics, including Computational Geometry: For the modelling of this problem we employ insights from a key area of *Applied Mathematics*, the Theory of Optimal Transportation, which leads to weighted Voronoi diagrams with prescribed areas. This will set up the stage for application of a major tool from *Very Pure Mathematics*, known as Equivariant Obstruction Theory. This is a "cannon", and we'll have fun with shooting it at the sparrow.

On the way to a solution, combinatorial properties of the permutahedron turn out to be essential. These will, at the end of the story, lead us back to India, with some time travel 100 years into the past: For the last step in our (partial) solution of the sparrows problem we need a simple divisibility property for the numbers in Pascal?s triangle, which was first observed by Balak Ram, in Madras 1909.

But even if the existence problem is solved, the Computational Geometry problem is not: If the solution exists, how do you find one? This problem will be left to you. Instead, I will comment on the strained relationship between cannons and sparrows, and to this avail quote a poem by Hans Magnus Enzensberger.

[*]Erdős Memorial Lecture
[†]Freie Universität Berlin

A Note on Interference in Random Networks

Luc Devroye[*] Pat Morin[†]

Abstract

The (maximum receiver-centric) interference of a geometric graph (von Rickenbach *et al.* (2005)) is studied. It is shown that, with high probability, the following results hold for a set, V, of n points independently and uniformly distributed in the unit d-cube, for constant dimension d: (1) there exists a connected graph with vertex set V that has interference $O((\log n)^{1/3})$; (2) no connected graph with vertex set V has interference $o((\log n)^{1/4})$; and (3) the minimum spanning tree of V has interference $\Theta((\log n)^{1/2})$.

1 Introduction

Von Rickenbach *et al.* [8, 9] introduce the notion of (maximum receiver-centric) interference in wireless networks and argue that topology-control algorithms for wireless networks should explicitly take this parameter into account. Indeed, they show that the minimum spanning tree, which seems a natural choice to reduce interference, can be very bad; there exists a set of node locations in which the minimum spanning tree of the nodes produces a network with maximum interference that is linear in the number, n, of nodes, but a more carefully chosen network has constant maximum interference, independent of n. These results are, however, *worst-case*; the set of node locations that achieve this are very carefully chosen. In particular, the ratio of the distance between the furthest and closest pair of nodes is exponential in the number of nodes.

The current paper continues the study of maximum interference, but in a model that is closer to a typical case. In particular, we consider what happens when the nodes are distributed uniformly, and independently, in the unit square. This distribution assumption can be used to approximately model the unorganized nature of ad-hoc networks and is commonly used in simulations of such networks [10]. Additionally, some types of sensor networks, especially with military applications, are specifically designed to be deployed by randomly placing (scattering) them in the deployment area. This distribution assumption models these applications very well.

[*]School of Computer Science, McGill University, lucdevroye@gmail.com

[†]School of Computer Science, Carleton University, morin@scs.carleton.ca

Our results show that the maximum interference, in this case, is very far from the worst-case. In particular, for points independently and uniformly distributed in the unit square, the maximum interference of the minimum spanning tree grows only like the square root of the logarithm of the number of nodes. That is, the maximum interference is *not even logarithmic* in the number of nodes. Furthermore, a more carefully chosen network topology can reduce the maximum interference further still, to the cubed root of the logarithm of n.

1.1 The Model

Let $V = \{x_1, \ldots, x_n\}$ be a set of n points in $\mathbb{R}^d$ and let $G = (V, E)$ be a simple undirected graph with vertex set V. The graph G defines a set, $B(G)$, of closed balls $B_1, \ldots, B_n$, where B_i has center x_i and radius

$$r_i = \max\{\|x_i x_j\| : x_i x_j \in E\} \ .$$

(Here, and throughout, $\|xy\|$ denotes the Euclidean distance between points x and y.) In words, B_i is just large enough to enclose all of x_i's neighbours in G. The *(maximum receiver-centric) interference* at a point, x, is the number of these balls that contain x, i.e.,

$$I(x, G) = |\{B \in B(G) : x \in B\}| \ .$$

The *(maximum receiver-centric) interference* of G is the maximum interference at any vertex of G, i.e.,

$$I(G) = \max\{I(x, G) : x \in V\} \ .$$

Figure 1 shows an example of a geometric graph G and the balls $B(G)$. Each node, x, is labelled with $I(x, G)$.

One of the goals of network design is to build, given V, a connected graph $G = (V, E)$ such that $I(G)$ is minimized. Thus, it is natural to consider interference as a property of the given point set, V, defined as

$$I(V) = \min\{I(G) : G = (V, E) \text{ is connected}\} \ .$$

A *minimum spanning tree* of V is a connected graph, $MST(V)$, of minimum total edge length. Minimum spanning trees are a natural choice for low-interference graphs. The purpose of the current paper is to prove the following results (here, and throughout, the phrase *with high probability* means with probability that approaches 1 as $n \to \infty$):

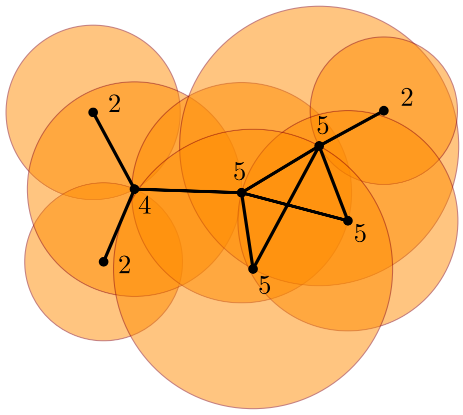


Figure 1: A geometric graph G with $I(G) = 5$.

Theorem 1. *Let V be a set of n points independently and uniformly distributed in $[0, 1]^d$. With high probability,*

1. $I(MST(V)) \in O((\log n)^{1/2})$*;*
2. $I(V) \in O((\log n)^{1/3})$*, for* $d \in \{1, 2\}$*; and*
3. $I(V) \in O((\log n)^{1/3}(\log \log n)^{1/2})$*, for* $d \geq 3$.

Theorem 2. *Let V be a set of n points independently and uniformly distributed in $[0, 1]^d$. With high probability,*

1. $I(MST(V)) \in \Omega((\log n)^{1/2})$
2. $I(V) \in \Omega((\log n)^{1/4})$.

1.2 Related Work

This section surveys previous work on the problem of bounding the interference of worst-case and random point sets. A summary of the results described in this section is given in Figure 2. In the statements of all results in this section, $|V| = n$.

The definition of interference used in this paper was introduced by von Rickenbach *et al.* [8] who proved upper and lower bounds on the interference of one dimensional point sets:

Theorem 4 (von Rickenbach *et al.* 2005)**.** *For any $d \geq 1$, there exists $V \subset \mathbb{R}^d$ such that $I(V) \in \Omega(n^{1/2})$.*

The point set, V, in this lower-bound consists of any sequence of points $x_1, \ldots, x_n$, all on a line, such that $\|x_{i+1}x_i\| \leq (1/2)\|x_i x_{i-1}\|$, for all $i \in \{2, \ldots, n-1\}$. That is, the gaps between consecutive points decrease exponentially.

This lower bound is matched by an upper-bound:

Theorem 5 (von Rickenbach *et al.* 2005)**.** *For all $V \subset \mathbb{R}$, $I(V) \in O(n^{1/2})$.*

The upper bound in Theorem 5 is obtained by selecting $n^{1/2}$ vertices to act as *hubs*, connecting the hubs into any connected network and then having each of the remaining nodes connect to its nearest hub. This idea was extended to two and higher dimensions by Halldórsson and Tokuyama [3], by using a special type of $(n^{-1/2})$-net as the set of hubs:

Theorem 6 (Halldórsson and Tokuyama 2008)**.** *For all $V \subset \mathbb{R}^d$,*

1. $I(V) \in O(n^{1/2})$ *for* $d = 2$*; and*
2. $I(V) \in O((n \log n)^{1/2})$*, for* $d \geq 3$.

Several authors have shown that the interference of a point set is related to the (logarithm of) the ratio between the longest and shortest distance defined by the point set. In particular, different versions of the following theorem have been proven by Halldórsson and Tokuyama [3]; Khabbazian, Durocher, and Haghnegahdar [4]; and Maheshwari, Smid, and Zeh [6]:

Theorem 7 (Halldórsson and Tokuyama 2008; Khabbazian, Durocher, and Haghnegahdar 2011; Maheshwari, Smid, and Zeh 2011)**.** *For any constant $d \geq 1$ and for all $V \subset \mathbb{R}^d$, $I(V) = O(\log D)$, where $D = \max\{\|xy\| : \{x, y\} \subseteq V\}/\min\{\|xy\| : \{x, y\} \subseteq V\}$.*

At least two of the proofs of Theorem 7 proceed by showing that $I(MST(V)) = O(\log D)$. A strengthening of this theorem is that the numerator in the definition of D can be replaced with the length of the longest edge in $MST(V)$ [4, 6].

Theorem 7 suggests that point sets with very high interference are unlikely to occur in practice. This intuition is born out by the results of Kranakis *et al.* [5], who show that high interference is unlikely to occur in random point sets in one dimension:

Theorem 8 (Kranakis *et al.* 2010)**.** *Let V be a set of n points independently and uniformly distributed in $[0, 1]$. Then, with high probability, $I(MST(V)) \in \Theta((\log n)^{1/2})$.*

Note that, in this one-dimensional case, the minimum spanning tree, $MST(V)$, is simply a path that connects the points of V in order, from left to right. Taken together, Part 1 of Theorems 1 and 2 generalize Theorem 8 to arbitrary constant dimensions $d \geq 1$.

In higher dimensions, Khabbazian, Durocher, and Haghnegahdar [4] use their version of Theorem 7 to show that minimum spanning trees of random point sets have at most logarithmic interference.

Theorem 9 (Khabbazian, Durocher, and Haghnegahdar 2011)**.** *Let V be a set of n points independently and uniformly distributed in $[0, 1]^d$. Then, with high probability, $I(MST(V)) \in O(\log n)$.*

Ref.	Dimension	Statement
[8]	$d \geq 1$	there exists V s.t. $I(V) \in \Omega(n^{1/2})$
[8]	$d = 1$	for all V, $I(V) \in O(n^{1/2})$
[3]	$d = 2$	for all V, $I(V) \in O(n^{1/2})$
[3]	$d \geq 3$	for all V, $I(V) \in O((n \log n)^{1/2})$
[5]	$d = 1$	for V i.u.d. in $[0,1]$, $I(MST(V)) \in \Theta((\log n)^{1/2})$ w.h.p.
[4]	$d \geq 2$	for V i.u.d. in $[0,1]^d$, $I(MST(V)) \in O(\log n)$ w.h.p.
Here	$d \geq 1$	for V i.u.d. in $[0,1]^d$, $I(MST(V)) \in \Theta((\log n)^{1/2})$ w.h.p.
[5, 8]	$d = 1$	for V i.u.d. in $[0,1]$, $I(V) \in \Omega((\log n)^{1/4})$ w.h.p.
Here	$d \geq 1$	for V i.u.d. in $[0,1]^d$, $I(V) \in \Omega((\log n)^{1/4})$ w.h.p.
Here	$d \in \{1,2\}$	for V i.u.d. in $[0,1]^d$, $I(V) \in O((\log n)^{1/3})$ w.h.p.
Here	$d \geq 3$	for V i.u.d. in $[0,1]^d$, $I(V) \in O((\log n)^{1/3}(\log\log n)^{1/2})$ w.h.p.

Figure 2: Previous and new results on interference in geometric networks.

Part 1 of Theorem 1 improves the upper bound in Theorem 9 to $O((\log n)^{1/2})$ and Part 1 of Theorem 2 gives a matching lower bound.

The second parts of Theorems 1 and 2 show that minimum spanning trees do not minimize interference, even for random point sets. For random point sets, one can construct networks with interference $O((\log n)^{1/3})$ and the best networks have interference in $\Omega((\log n)^{1/4})$.

The remainder of this paper is devoted to proving Theorems 1 and 2. For ease of exposition, we only present these proofs for the case $d = 2$ though they generalize, in a straightforward way, to arbitrary (constant) dimensions. Due to space constraints, some proofs are omitted from this version of the paper. All proofs can be found in the preprint version [2].

2 Proof of the Upper Bounds (Theorem 1)

In this section, we prove Theorem 1. However, before we do this, we state a slightly modified version of Theorem 7 that is needed in our proof.

Lemma 1. *Let $V \subset \mathbb{R}^d$, let $r > 0$, and let $MST^r(V)$ denote the subgraph of $MST(V)$ containing only the edges whose length is in $(r, 2r]$. Then $I(MST^r(V)) \in O(1)$.*

Proof. (This proof is similar to the proof of Lemma 3 in Ref. [6].) Let x be any point in $\mathbb{R}^d$ and let B the set of all balls in $B(MST^r(V))$ that contain x so that, by definition $I(x, MST^r(V)) = |B|$. All the centers of balls in B are contained in a ball of radius $2r$ centered at x. Therefore, a simple packing argument implies that there exists a ball, b, of radius $r/2$ that contains at least $|B|/5^d$ centers of balls in B. (5^d is the volume of a ball of radius $5r/2$ divided by the volume of a ball of radius $r/2$.) The center of each of these ball is the endpoint of an edge of length at most $2r$. The other endpoints of these edges are all contained in a ball of radius $5r/2$ centered around b. The same packing argument shows that we can find a ball of radius $r/2$ that contains at least $|B|/(5 \cdot 6)^d$ of these other endpoints.

We claim that this implies that $|B|/30^d < 2$ (so $|B| < 2 \cdot 30^d$). Otherwise, $MST(V)$ contains two edges, $x_i x_j$ and $x_k x_\ell$, each of length greater than r and such that $\|x_i x_k\| \leq r$ and $\|x_j x_\ell\| \leq r$. But this contradicts the minimality of $MST(V)$, since one could replace $x_i x_j$ with one of $x_i x_k$ or $x_j x_\ell$ and obtain a spanning tree of smaller total edge length. We conclude that $|S_i| < 2 \cdot 30^d$, and this completes the proof. $\square$

Note that Lemma 1 implies Theorem 7, since it implies that we can partition the edges of $MST(V)$ into $\lceil \log_2 D \rceil$ classes, based on length, and each class will contain only a constant number of edges.

We are ready to prove Parts 2 and 3 of Theorem 1. The sketch of the proof is as follows: We partition $[0,1]^d$ into equal cubes of volume $1/nt$, for some parameter t to be chosen later. Using Chernoff's bounds, we show that each cube contains $O((\log n)^{2/3})$ points so that the points within each cube can be connected, using the results of Halldórsson and Tokuyama, with maximum interference $O((\log n)^{1/3})$. Next, the cubes are connected to other cubes by selecting one point in each cube and connecting these selected points with a minimum spanning tree. Lemma 1 is then used to show that this minimum spanning tree has maximum interference $O((\log n)^{1/3})$. Without further ado, we present:

Proof of Theorem 1, Parts 2 and 3. Partition $[0,1]^2$ into square *cells* of area $1/nt$ for some value t to be specified later. Let N_i denote the number of points that are contained in the ith cell. Then N_i is binomial with mean $\mu = 1/t$. Recall Chernoff's Bounds [1] on the tails of binomial random variables:

$$\Pr\{N_i \geq (1+\delta)\mu\} \leq \left(\frac{e^\delta}{(1+\delta)^{1+\delta}}\right)^\mu .$$

In our setting, we have,

$$\begin{aligned}\Pr\{N_i \geq k\} &= \Pr\{N_i \geq kt\mu\} \\ &\leq \left(\frac{e^{kt}}{(kt)^{kt}}\right)^{1/t} \\ &= \frac{e^k}{(kt)^k} \\ &\leq \frac{1}{t^k} \qquad \text{for } k \geq e \\ &\leq \frac{1}{n^{c+2}} \ ,\end{aligned}$$

for $t = 2^{(\log n)^{1/3}}$ and $k = (c+2)(\log n)^{2/3}$.

Note that the number of cells is no more than $nt \leq n^2$, for sufficiently large n. Therefore, by the union bound, the probability that there exists any cell containing more than k points is at most n^{-c}.

Within each non-empty cell, we apply Theorem 6 to connect the vertices in the ith cell into a connected graph G_i with $I(G_i) = O(\sqrt{N_i})$.[1] In fact, a somewhat stronger result holds, namely that $\max\{I(x, G_i) : x \in \mathbb{R}^2\} = O(\sqrt{N_i})$. Notice that each edge in G_i has length at most $\sqrt{2/nt}$. Stated another way, in $\bigcup_i G_i$, any point, x, receives interference only from cells within distance $\sqrt{2/nt}$ of the cell containing x. There are only 25 such cells, so

$$\max\left\{I\left(x, \bigcup_i G_i\right) : x \in \mathbb{R}^2\right\} = O(\sqrt{k}) = O((\log n)^{1/3})$$

with high probability.

Thus far, the points within each cell are connected to each other and the maximum interference, over all points in $\mathbb{R}^2$, is $O(\sqrt{k})$. To connect the cells to each other, we select one point from each non-empty cell and connect these using a minimum spanning tree, T. What remains is to show that the additional interference caused by the addition of the edges in T does not exceed $O((\log n)^{1/3})$.

Suppose that $I(x, T) = r$, for some point $x \in \mathbb{R}^2$. There are at most 9 vertices in T whose distance to x is less than $1/\sqrt{nt}$. Therefore, by Lemma 1, T must contain an edge of length at least $c2^r/\sqrt{nt}$, for some constant $c > 1$.

A well-known property of minimum spanning trees is that, for any edge x_ix_j in T, the open ball with diameter x_ix_j does not contain any vertices of T. In our setting, this means that there is an open ball, B, of radius $c2^r/2\sqrt{nt}$ such that every cell contained in B contains no point of V. Inside of B is another empty ball B' of radius $c2^r/(2\sqrt{nt}) - \sqrt{2/nt}$ whose center is also the center of some cell.

[1]This is where the discrepancy between Parts 2 and 3 of the theorem occurs. For $d \geq 3$, Theorem 6 only guarantees $I(G_i) = O(\sqrt{N_i}\log N_i)$.

At least one quarter of the area of B' is contained in $[0,1]^2$, so the number of cells completely contained in B' is at least $\pi c^2 2^{2r}/16 - O(2^r/\sqrt{nt})$. By decreasing c slightly, and only considering r larger than a sufficiently large constant, r_0, we can simplify this number of cells to $\pi c^{2r}/16$.

For a fixed ball B', the probability that the $c\pi 2^{2r}/16$ cells defined by B' are empty of points in V is at most

$$\begin{aligned}p &\leq (1 - c\pi 2^{2r}/16nt)^n \\ &\leq \exp(-c\pi 2^{2r}/16t) \\ &\leq 1/n^{2+c'} \ ,\end{aligned}$$

for $r \geq \log(16/c\pi) + \log t + \log(2+c') + \log\ln n$. By the union bound, the probability that there exists any such B' is at most $pnt \leq 1/n^{c'}$. Since we can choose $r \in O(\log t + \log\log n) = O((\log n)^{1/3})$, this completes the proof. □

The proof of Part 1 of Theorem 1 is just a matter of reusing the ideas from the previous proof of Parts 2 and 3.

Proof of Theorem 1, Part 1. Let x be any point in $\mathbb{R}^2$. We partition the balls in $B(MST(V))$ that contain x into three sets:

1. the set B_0 of balls having area at most $1/nt$;
2. the set B_1 of balls having area in the range $[1/nt, (c\log n)/n]$; and
3. the set B_2 of balls having area greater than $(c\log n)/n$.

In this proof, the parameter $t = 2^{(\log n)^{1/2}}$.

The set B_0 consists of points contained in a ball of area $1/nt$ centered at x. Exactly the same argument used in the first part of the previous proof shows that, with high probability, every such ball contains $O((\log n)^{1/2})$ points, so

$$|B_0| \in O((\log n)^{1/2}) \ .$$

The set B_1 consists of balls whose radii are in the range $[\sqrt{1/\pi nt}, \sqrt{(c\log n)/\pi n}]$. Lemma 1 shows that the number of these balls is

$$\begin{aligned}|B_1| &\in O\left(\log\left(\frac{\sqrt{(c\log n)/\pi n}}{\sqrt{1/\pi nt}}\right)\right) \\ &= O(\log\log n + \log t) \\ &= O((\log n)^{1/2}) \ .\end{aligned}$$

Finally, any edge in the set B_2 implies the existence of an empty ball, with center in $[0,1]^2$, having area $c\log n/n$. The second part of the previous proof shows that the probability that such a ball exists is $O(n^{-c})$. Therefore, with high probability,

$$|B_2| = 0 \ .$$ □

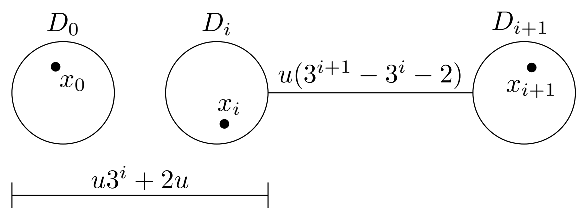


Figure 4: The ball centered at x_i that contains x_{i+1} also contains x_0.

3 Proof of The Lower Bounds (Theorem 2)

In this section, we prove the lower bounds in Theorem 2. We define a *Zeno configuration* as follows (see Figure 3): A Zeno configuration of size k, centered at a point, x, is defined by a set of $k+1$ balls. The construction starts with disjoint balls $D_0,\ldots,D_{k-1}$, each having radius u. The ball D_0 is centered at x. The center of D_i, $i\in\{1,\ldots,k-1\}$ is at $x+(u3^i,0)$. A final large ball, D, of radius $r=u3^k$ is centered at x and contains all other balls. A Zeno configuration occurs at location x in a point set V when D contains exactly k points of V and these occur with exactly one point in each ball D_i.

The following lemma shows that a Zeno configuration in V causes high interference in $MST(V)$.

Lemma 2. *If V contains a Zeno configuration of size k, $I(MST(V))\geq k-1$.*

Proof. Let x_i, $i\in\{0,\ldots,k-1\}$, denote the point of V contained in D_i. Note that, for $i\in\{1,\ldots,k-1\}$ the closest point to x_i in V is x_{i-1}. Since $MST(V)$ contains the nearest-neighbour graph, this implies that $MST(V)$ contains the edges x_ix_{i+1} for all $i\in\{0,\ldots,k-2\}$. See Figure 4 for what follows. We claim that, for all $i\in\{0,\ldots,k-2\}$, the ball B_i centered at x_i that contains x_{i+1} also contains x_0. This is clearly true for $i=0$ and $i=1$. Next, note that

$$\|x_ix_0\|\leq u(3^i+2)\ .$$

On the other hand, for $i\geq 2$,

$$\|x_ix_{i+1}\|\geq u(3^{i+1}-3^i-2)=2u3^i-2u\geq u(3^i+7)>\|x_ix_0\|\ .$$

Therefore, $I(x_0,MST(V))\geq k-1$. □

The next lemma shows that a Zeno configuration causes high interference on any connected graph on vertex set V.

Lemma 3. *If V contains a Zeno configuration of size k, then $I(V)\geq\sqrt{k-1}$.*

Proof. Let G be any connected graph on V. Using the same notation as in the proof of Lemma 2, call a vertex, x_i, a *big one* if x_i is adjacent to any vertex x_j, with $j>i$, or x_i is adjacent to any vertex x not in D. The proof of Lemma 2 shows that every big one contributes to the interference at x_0. Therefore, if the Zeno configuration contains $\sqrt{k-1}$ or more big ones, then $I(x_0,G)\geq\sqrt{k-1}$ and there is nothing left to prove. Otherwise, note that each of $x_0,\ldots,x_{k-2}$ is either a big one or adjacent to a big one. Therefore, there must be a big one, x_i, with degree at least $\sqrt{k-1}-1$, so $I(x_i,G)\geq\sqrt{k-1}$. □

To prove Theorem 2, all that remains is to show a Zeno configuration of size $\Omega((\log n)^{1/2})$ occurs in V with high probability. We omit this proof due to space constraints.

4 Discussion

Summary. This paper gives new bounds on the maximum interference for graphs defined by points randomly distributed $[0,1]^d$. Minimum spanning trees have interference $\Theta((\log n)^{1/2})$, but better graphs exist; a strategy based on bucketing yields a graph with interference $O((\log n)^{1/3})$. No graph on such a point set has interference $o((\log n)^{1/4})$.

Open Problem. An obvious open problem is that of closing the gap between the upper bound of $O((\log n)^{1/3})$ and the lower bound of $\Omega((\log n)^{1/4}$. One strategy to achieve this would be to prove the following conjecture, which has nothing to do with probability theory:

Conjecture 1. *For any $V\subset\mathbb{R}^d$, $I(V)=O(\sqrt{I(MST(V))})$.*

A weaker version of this conjecture is due to Halldórsson and Tokuyama [3], who conjecture that $I(V)=O(\sqrt{\log D})$ where D is the ratio of the lengths of the longest and the shortest edges of $MST(V)$.

Unit Disk Graphs. Several of the references consider interference in the *unit disk graph model*, in which the graph G is constrained to use edges of maximum length $r(n)$. It is straightforward to verify that all of the proofs in this paper continue to hold in this model, when $r(n)\in\Omega(\sqrt{(\log n)/n})$. This is not an unreasonable condition; for i.u.d. points in $[0,1]^d$, it is known that $r(n)\in\Omega(\sqrt{(\log n)/n})$ is a necessary condition to be able to form a connected graph G [7].

Locally Computable Graphs. Khabbazian, Durocher, and Haghnegahdar [4] give a local algorithm, called LocalRadiusReduction, that is run at the nodes of a communication graph, $G=(V,E)$, and that reduces the number of edges of G. The resulting graph G' comes from a class of graphs that they denote as $\mathcal{T}(V)$. The

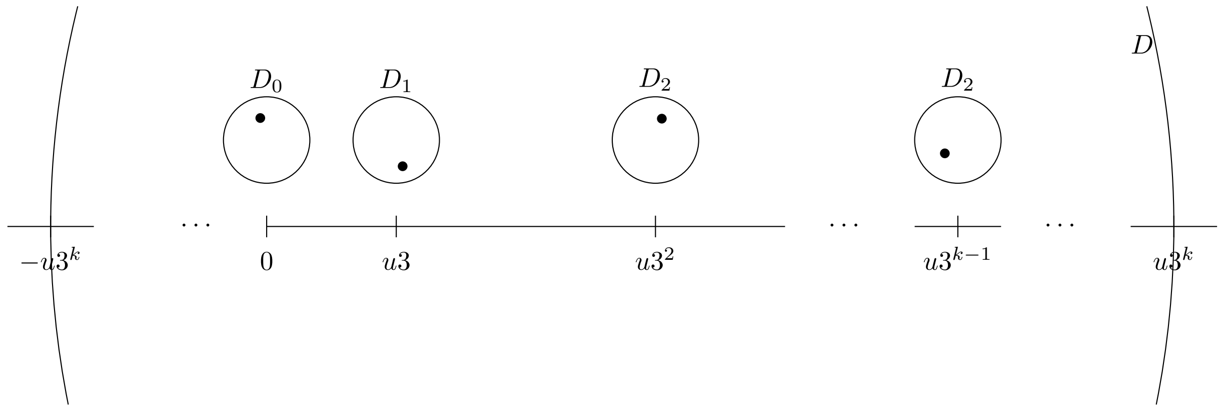


Figure 3: A Zeno configuration of size k.

class $\mathcal{T}(V)$ includes the minimum spanning tree of V and the graphs in this class share many of the same properties as the minimum spanning tree. In particular, the following result can be obtained by using the proof of Theorem 1 Part 1 and properties of the family $\mathcal{T}(V)$ [4, Theorem 3].

Theorem 3. *Let V be a set of n independently and uniformly distributed points in $[0,1]^d$ and let G be any graph in $\mathcal{T}(V)$. With high probability, $I(G) = O((\log n)^{1/2} + \log(\ell\sqrt{n}))$, where ℓ is the length of the longest edge in G.*

In particular, Theorem 3 implies that running the LocalRadiusReduction algorithm at the nodes of a unit disk graph with unit $r(n) \in O(2^{\sqrt{\log n}}/\sqrt{n})$ yields a connected graph with maximum interference $O((\log n)^{1/2})$.

Acknowledgement

The research in this paper was started at the workshop on *Models of Sparse Graphs and Network Algorithms (12w5004)*, hosted at the Banff International Research Station (BIRS), February 5–10, 2012. The authors are grateful to the other workshop organizers, Nicolas Broutin and Gábor Lugosi, the other participants, and the staff at BIRS, for providing a stimulating research environment.

References

[1] H. Chernoff. A measure of the asymptotic efficient of tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–507, 1952.

[2] L. Devroye and P. Morin. A note on interference in random networks. *CoRR*, abs/1202.5945, 2012.

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

[4] M. Khabbazian, S. Durocher, and A. Haghnegahdar. Bounding interference in wireless ad hoc networks with nodes in random position. *CoRR*, abs/1111.6689, 2011.

[5] 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.

[6] A. Maheshwari, M. Smid, and N. Zeh. Low-interference networks in metric spaces with bounded doubling dimension. *Information Processing Letters*, 111(23–24):1120–1123, 2011.

[7] M. D. Penrose. The longest edge of the random minimal spanning tree. *The Annals of Applied Probability*, 7(2):340–361, 1997.

[8] P. von Rickenbach, S. Schmid, R. Wattenhofer, and A. Zollinger. A robust interference model for wireless ad-hoc networks. In *IPDPS*. IEEE Computer Society, 2005.

[9] P. von Rickenbach, R. Wattenhofer, and A. Zollinger. Algorithmic models of interference in wireless ad hoc and sensor networks. *IEEE/ACM Transactions on Networking*, 17(1):172–185, 2009.

[10] G. Mao X. Ta and B. D. O. Anderson. On the phase transition width of K-connectivity in wireless multi-hop networks. *IEEE Transactions on Mobile Computing*, 8(7):936–949, 2009. To appear.

On Farthest-Point Information in Networks[*]

Prosenjit Bose[†] Jean-Lou De Carufel[†] Carsten Grimm[†‡] Anil Maheshwari[†] Michiel Smid[†]

Abstract

Consider the continuum of points along the edges of a network, an embedded undirected graph with positive edge weights. Distance between these points can be measured as shortest path distance along the edges of the network. We introduce two new concepts to capture farthest-point information in this metric space. The first, eccentricity diagrams, are used to encode the distance towards farthest points for any point on the network compactly. With this, we can solve the minimum eccentricity feed-link problem, i.e., the problem to extend a network by one new point minimizing the largest network distance towards the new point. The second, network farthest-point diagrams, provide an implicit description of the sets of farthest points. A network farthest-point diagram is, in principle, a compressed farthest-point network Voronoi link diagram generated by the entire continuum of uncountably many points on the network at hand. We provide construction algorithms for data structures that allow for queries for the distance to farthest points as well as their location from any point on a network in optimal time. Thus, we establish first bounds on construction times and storage requirements of such data structures.

1 Introduction

The topic of this article was inspired by the following network extension problem introduced by Aronov et. al. [2]. We are given a network of roads and the position of a site, e.g., a hospital, that is not on the network, yet. The site needs to be connected to the existing roads with a new one, referred to as a *feed-link*. Aronov et. al. [2] seek a feed-link that minimizes the largest ratio between the distance to the site via the roads versus the Euclidean distance from any location on the roads. This ratio signifies the largest detour one may take to the site by traveling along the roads as opposed to flying directly to it. When this detour, also referred to as *dilation*, is minimized, the distances via the network resemble the straight line distances as best as possible. An illustration is shown in Figure 1. Grüne [7] provides a summary of dilation and its properties. Notice that all positions on the network are taken into account to evaluate a feed-link and that the feed-link might be connected to any location along the roads. In this sense the dilation is a generalization of the *stretch factor* [11].

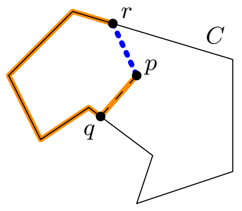


Figure 1: A polygonal cycle C with a point p in it that is connected to C via a feed-link at q. The dilation of point r on the cycle is the ratio between the highlighted path (orange) via the roads to p versus the Euclidean distance between p and r (blue, dotted).

Depending on the application at hand, one might consider other measures. For instance, if the site is a hospital, one might seek to optimize emergency unit response times [5]. Assume an accident occurs along any of the roads, then it is desirable to ensure that the time an emergency crew needs to drive from the hospital to the accident is as small as possible or below a certain critical threshold. Therefore, we seek to minimize the largest road-wise distance to the hospital. The set of farthest locations from the site is precisely the same as that for the meeting point of the feed-link with an existing road. Hence, determining how the set of farthest points and the road-wise distances to them change along the existing roads turns out to be helpful to solve this variant of the feed-link problem.

In this article, we will solve the latter for arbitrary networks of roads using novel data structures that support queries for farthest-point information.

1.1 Problem Definition

A *network* is a straight-line embedding of a simple, finite, connected, and undirected graph $G = (V, E)$, where V is a set of points in $\mathbb{R}^2$, and E is a set of segments whose endpoints are in V. Each edge e has a positive weight $w_e > 0$. A point $p \in \mathbb{R}^2$ is on G, denoted by $p \in G$, if p is on some edge of G. A

[*]Research supported in part by FQRNT and NSERC.

[†]School of Computer Science, Carleton University, `jit@scs.carleton.ca, jdecaruf@cg.scs.carleton.ca, carsten.grimm@ovgu.de, anil@scs.carleton.ca, michiel@scs.carleton.ca`

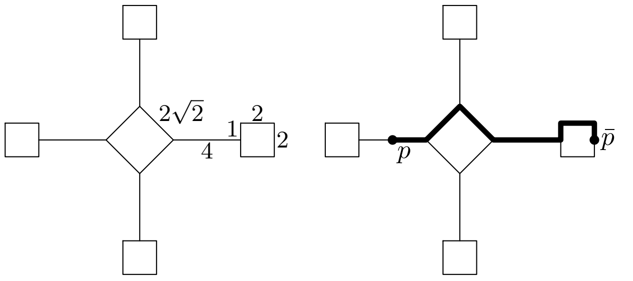
[‡]Institut für Simulation und Graphik, Fakultät für Informatik, Otto-von-Guericke-Universität Magdeburg

point p on an edge $uv \in E$ subdivides uv such that $|up| = \lambda\,|uv|$ and $|pv| = (1-\lambda)\,|uv|$ for some $\lambda \in [0,1]$. We choose the weights of the resulting sub-edges up and pv according to the fraction λ, viz., $w_{up} := \lambda w_{uv}$ and $w_{pv} := (1-\lambda)w_{uv}$. A point can only be on one edge. Thus, if a point happens to lie on the proper intersection of two edges, the point can only be associated with one edge. Consider the weighted shortest path distance $d_G\colon V \times V \to [0,\infty)$ between vertices of G with respect to the edge weights w_e, $e \in E$. This can be extended to arbitrary points p and q on G by considering them to be vertices for the sake of evaluating $d_G(p,q)$ [2, 7]. We refer to this as the *network distance* on G. The following definition generalizes a term that is usually introduced with respect to distances between vertices [9, pp. 35–36].

Definition 1 (Eccentricity) *Let G be a network (refer to Figure 2). For a point p on G, the largest network distance towards p is the* eccentricity *of p with respect to G and it is denoted by $\mathrm{ecc}_G(p)$, i.e.,*

$$\mathrm{ecc}_G(p) := \max_{q \in G} d_G(p,q).$$

The point q on G is eccentric *to p if it is a farthest point from p with respect to the network distance, i.e., if $d_G(p,q) = \mathrm{ecc}_G(p)$*[1].



(a) A network G. (b) A point p on G and a point $\bar{p}$ eccentric to p.

Figure 2: A network G (a) with edge weights as indicated. A point p with its non-vertex eccentric point $\bar{p}$ is shown (b). Here we have $\mathrm{ecc}(p) = 10 + 4\sqrt{2}$ achieved on the highlighted path (black).

Our goal is to design algorithms and data structures for a given network G in order to answer the following types of queries. For any point p on G.

1. What is the eccentricity of p?
2. Which points on G are farthest from p with respect to the network distance?
3. Let uv be an edge such that $p \in uv$. Which points r on uv have the same farthest points as p?

[1]In the remainder of this article we will omit the subscript indicating the underlying network G in all of the above notation when it is clear from the context.

1.2 Related Work

The relation between points p on a network G and their farthest points $\bar{p}$ on G can be expressed in terms of existing notions as follows. It can be stated as the *farthest-point Voronoi diagram* on the metric space $(G, d(\cdot,\cdot))$ where all of the uncountable infinitely many points on G are considered to be *sites* or *generators* of the diagram. Usually *Voronoi diagrams* are computed with respect to a finite set of $n \in \mathbb{N}$ sites. The *farthest-point Voronoi diagram* is a special case of the *k-th nearest neighbor Voronoi diagram* with $k = n$. Even though Voronoi diagrams on networks have been studied before, e.g., [3, 5, 8, 13], they were defined with respect to a finite set of generators. A survey of various notions of Voronoi diagrams, including some for networks, can be found in [12]. Refer to [13] for generalized variants of network Voronoi diagrams and further references.

Information about the eccentricity of points along the edges of a network is also useful in contexts other than the stated feed-link problem. For instance, in the *continuous absolute* 1*-center problem* from location analysis [4, 14] we seek a point with minimum eccentricity in a network. Furthermore, a point of maximum eccentricity and one of its farthest points form a pair of diametral points. Recent surveys of existing related notions and results can be found in [10, 14].

2 Eccentricity Diagrams

We seek a concise representation of the mapping from the points on a network G to their eccentricity value. Frank [4] seeks a point with minimum eccentricity on G. He finds it by determining the smallest among the minimal eccentricity values on each edge uv. To obtain these values, Frank [4] computes the eccentricity of points on edge uv as a function as follows. Let $\phi_{uv}^{st}\colon [0,1] \to [0,\infty)$ be the mapping such that

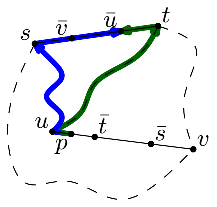
$$[0,1] \ni \lambda \overset{\phi_{uv}^{st}}{\longmapsto} \max_{q \in st} d((1-\lambda)u + \lambda v, q).$$

Consider a point p on edge uv with $p = (1-\lambda)u + \lambda v$, $\lambda \in [0,1]$. The value $\phi_{uv}^{st}(\lambda)$ is the largest network distance from p to any point on edge st. We obtain the eccentricity function for the points p on uv by building the upper envelope of the functions ϕ_{uv}^{e} for all edges e of the network, since

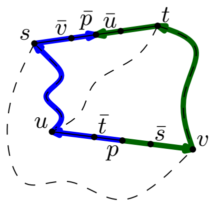
$$\mathrm{ecc}(p) = \max_{q \in G} d(p,q) = \max_{e \in E} \max_{q \in e} d(p,q).$$

The shape of the functions is described in Lemma 2 and depicted in Figure 3.

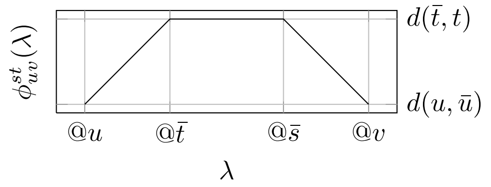
Lemma 2 ([4]) *Let uv and st be edges of a network G. Then the function ϕ_{uv}^{st} is piece-wise linear with slopes $+w_{uv}$, 0, and $-w_{uv}$ in this order.*



(a) Case $p \in u\bar{t}$.



(b) Case $p \in \bar{t}\bar{s}$.



(c) The function ϕ_{uv}^{st}.

Figure 3: The function ϕ_{uv}^{st} for two edges uv an st consist of three linear segments. The point $\bar{u}$ (respectively $\bar{v}$) is the farthest point from u (respectively v) on st. Likewise, the point $\bar{t}$ (respectively $\bar{s}$) is the farthest point from t (respectively s) on uv. Shortest paths attaining the network distance $\phi_{uv}^{st}(p)$ form p to the farthest point $\bar{p}$ from p on st are shown in (a) and (b).

As a consequence, the eccentricity along an edge uv of a network with m edges is the upper envelope of m piece-wise linear functions as in Lemma 2. The domain of all these functions is $[0, 1]$. We can compute this upper envelope using a divide-and-conquer approach described by Agarwal and Sharir [1, Section 2.3].

Lemma 3 *Let uv be an edge of a network G. Any pair of functions ϕ_{uv}^{e} and $\phi_{uv}^{e'}$ for edges e and e' of G intersect at most twice disregarding overlaps.*

Theorem 4 ([1]) *Let $\mathcal{F}$ be a set of k continuous, totally defined functions with a common domain whose graphs intersect in at most two points. The sequence of functions along the upper envelope of $\mathcal{F}$ can be obtained in $\mathcal{O}(k \log(k))$ time and has length at most $2k - 1$.*

With Lemmas 2 and 3 we can use Theorem 4 to estimate the size and construction time of the upper envelope of the functions ϕ_{uv}^{st}.

Corollary 5 *Let uv be an edge of a network G with m edges. The eccentricity on uv is a piece-wise linear and continuous function, consisting of at most $6m - 3$ line segments. It can be computed in $\mathcal{O}(m \log(m))$ time.*

Due to its piece-wise linearity, we can describe the eccentricity completely by stating the value of the eccentricity at the endpoints of each linear segment. That is for any point p in the segment ab with linear eccentricity and with $p = (1 - \lambda)a + \lambda b$, we have

$$\operatorname{ecc}(p) = (1 - \lambda) \operatorname{ecc}(a) + \lambda \operatorname{ecc}(b).$$

This leads us to the following notion.

Definition 6 (Eccentricity Diagram) *Let G be a network. Consider the subdivisions G' of G with*

$$\operatorname{ecc}(a + \lambda(b - a)) = (1 - \lambda) \operatorname{ecc}(a) + \lambda \operatorname{ecc}(b),$$

for each edge ab of G' and each $\lambda \in [0, 1]$. Among these we call the one with the least number of vertices the eccentricity diagram *of G and denote it by $\mathcal{ED}(G)$.*

The eccentricity diagram of a network is well-defined and unique, as it can be obtained by subdividing each edge uv at the endpoints of the line segments of the eccentricity function on uv. By Corollary 5, this yields a finite subdivision with the minimum number of additional vertices. An example is shown in Figure 5. As the computation of the upper envelope is performed on each edge, we have the following corollary.

Corollary 7 *The eccentricity diagram of a network with m edges has size $\mathcal{O}(m^2)$ and can be constructed in $\mathcal{O}(m^2 \log(m))$ time, provided the shortest path information between any pair of vertices is known a-priori.*

Next we establish that the size bound stated in Corollary 7 is tight for planar networks. In the full version of this paper we establish a lower bound of $\Omega(nm)$ for general networks with n vertices and m edges.

Lemma 8 *For all $n \in \mathbb{N}$, there exists a (planar) network G with n vertices that has an eccentricity diagram $\mathcal{ED}(G)$ of size $\Omega(n^2)$.*

Proof. Consider the network G depicted in Figure 4 for $k > 2$ and a value of ϵ with $0 < \epsilon < \frac{3}{2(k-2)}$. Each of the k edges $u_i v_i$, $i = 1, \ldots, k$, is subdivided into k sub-edges in the eccentricity diagram of G on $u_i v_i$ by $k - 1$ additional vertices. Thus, we have at least $k(k - 1) \in \Omega(n^2)$ additional vertices in total, as the network has $n = 4k$ vertices. $\square$

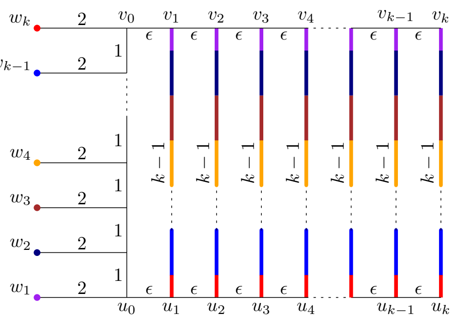
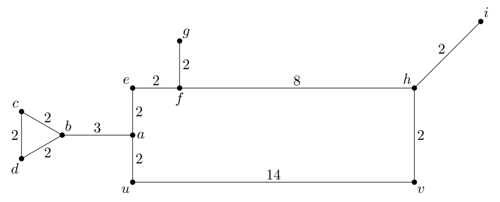
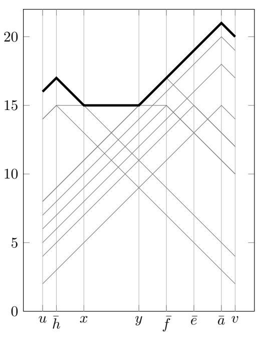


Figure 4: A network whose number of vertices in the eccentricity diagram is quadratic in the number of vertices n in the network itself. Along the edges $u_i v_i$ for $i = 1, \ldots, k$, the farthest point among $w_1, \ldots, w_k$ is indicated in the corresponding colour.



(a) A network.



(b) The functions ϕ_{uv}^{st} and their upper envelope.

Figure 5: An example for the brute-force method applied to the edge uv of the network in (a). The functions ϕ_{uv}^{st} representing the edge-to-edge distances are shown in (b) together with their upper envelope, representing the eccentricity along uv. The vertices $\bar{h}$, x, y, and $\bar{a}$ must be added to uv, to obtain the subdivision of uv in the eccentricity diagram $\mathcal{ED}(G)$ of G.

Assume we are given the eccentricity diagram $\mathcal{ED}(G)$ of a network G as well as the eccentricity values $\operatorname{ecc}(v)$ of all vertices v of $\mathcal{ED}(G)$. Then we can answer queries for the eccentricity value $\operatorname{ecc}(p)$ of a point p on an edge uv using the piecewise linearity of $\operatorname{ecc}(\cdot)$ on uv, where

$$\operatorname{ecc}(p) = \left(1 - \frac{w_{ap}}{w_{ab}}\right)\operatorname{ecc}(a) + \frac{w_{ap}}{w_{ab}}\operatorname{ecc}(b),$$

and ab is the sub-edge of uv in $\mathcal{ED}(G)$ containing p. We assume that we are given the edge uv of the original network G containing p when conducting such a query. The sub-edge ab of uv can be found in $\mathcal{O}(\log(n))$ time using binary search as there are at most $6m-3 \in \mathcal{O}(n^2)$ additional vertices on uv in $\mathcal{ED}(G)$ by Corollary 5. The above yields in combination with Corollary 7 and Lemma 8 the following theorem.

Theorem 9 *Given a network G with n vertices and m edges. There is a data structure that can be used to determine the eccentricity value* $\operatorname{ecc}(p)$ *of any point p on G in $\mathcal{O}(\log(n))$ time, provided that the edge uv of G containing p is given. This data structure can be constructed in $\mathcal{O}(m^2 \log(n))$ time, provided that the network distances between all vertices of G are known. The size of this data structure is at most $\mathcal{O}(m^2)$ in general and can be at least $\Omega(n^2)$ for certain planar networks.*

3 Network Farthest-Point Diagrams

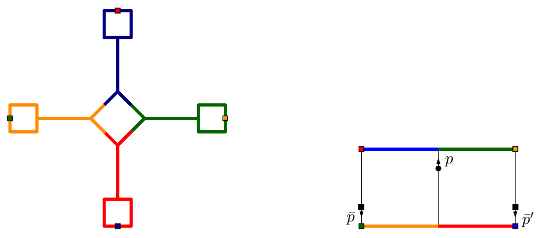
In addition to computing the distance towards farthest points, we are also interested in their location. That is we seek to query for the set of farthest points from any point g on a network G. This suggests the introduction of a continuous version of a farthest-point Voronoi diagram on the metric space formed by the edges of a network and the corresponding network distance.

Definition 10 *Let G be a network. Consider the set*

$$\mathcal{V}_{\textit{far-net}}(g) := \{p \in G \colon \forall g' \in G \colon d(p, g') \leq d(p, g)\},$$

of points $p \in G$ whose network distance $d(p, g)$ to a point g on G is largest among the network distances to all other points g' on G. We call $\mathcal{V}_{\textit{far-net}}(g)$ the farthest-point network Voronoi link cell *of g. We obtain the* farthest-point network Voronoi link diagram *of G by adding a new vertex to G for each boundary point of the non-empty farthest-point network Voronoi link cells, i.e., at all points of the set $\bigcup_{g \in G} \partial\mathcal{V}_{\textit{far-net}}(g)$. If the latter set is finite, we say that the diagram is* finite.

The existing notions of Voronoi diagrams are determined by a finite set of reference points. For instance the *farthest-point Voronoi diagram* [12, Section 3.3] subdivides the plane into regions such that the points in the interior of any region have one common unique farthest point among a finite set of points in $\mathbb{R}^2$. Likewise, the *network Voronoi link diagram* on G [12, Section 3.8] subdivides a network into parts, such that the points in the interior of each part are closest to a common subset of a finite set of points on G. However, for the queries described in Section 1.1, the situation is different. First, we are oblivious of which points g on G are considered farthest points, i.e., satisfy $\mathcal{V}_{\text{far-net}}(g) \neq \emptyset$, when creating the farthest-point network Voronoi link diagram. Thus, the set of reference points is to be determined as opposed to given a-priori. Secondly, this set of reference points may be infinite, as depicted in Figure 6. Therefore, known methods to determine Voronoi diagrams do not necessarily apply here. Moreover, we need to find a way to deal with infinite farthest-point network Voronoi link diagrams. If a finite number of vertices is added, the farthest-point network Voronoi link diagram is a subdivision of G. In that case, it is considered a network itself. Otherwise, it is an *infinite network*, i.e., a network with infinitely many vertices and degenerate edges that may have an empty interior and identical endpoints. In the finite case, it would be sufficient to store the set of farthest points at each vertex and each edge of the farthest-point network Voronoi link diagram to give a full description of the location of eccentric points. However, this is impossible to do explicitly in the infinite case. Next, we will investigate the latter in order to obtain a finite representation of the same information.



(a) Finite diagram. (b) Infinite diagram.

Figure 6: The farthest-point network Voronoi link diagrams for two networks. Parts that have a common farthest point (square) are indicated in colour. In the finite case (a), the network is subdivided into regions with a fixed farthest point. We have a different behaviour on the vertical edges (black) of (b). When the point p is moved upwards, its two farthest points $\bar{p}$ and $\bar{p}'$ move downwards accordingly. No two points on this edge have a common farthest point.

Theorem 11 *Let G be a network. The farthest-point network Voronoi link diagram of G is infinite if and only if there exists an edge ab of the eccentricity diagram $\mathcal{ED}(G)$ of G such that the eccentricity is constant on ab, i.e., we have $\operatorname{ecc}(a) = \operatorname{ecc}(p)$ for all $p \in ab$.*

We distinguish two types of phenomena on the edges of the eccentricity diagram. On edges uv with non-constant eccentricity, the farthest points are stationary in the sense that uv can be subdivided into finitely many sub-edges without any change of the farthest-point set in their interior. On edges with constant eccentricity however, each point has its own set of farthest points distinct from that of any of the uncountably many other points on it. Nonetheless, we can subdivide edges that exhibit the latter behavior into finitely many portions, such that the farthest points on each portion are contained in a common set of edges. This simplification yields a finite representation of the farthest-point network Voronoi link diagram defined as follows.

Definition 12 (Farthest-Point Diagram) *Let G be a network. Consider the subdivisions G' of the eccentricity diagram $\mathcal{ED}(G)$ of G such that each edge uv of G' is of one of the following types.*

(i) The eccentricity on uv is non-constant, and all points in the interior of uv have the same set of farthest points in G.

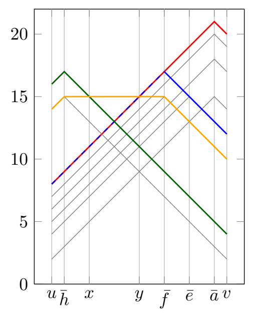
(ii) The eccentricity on uv is constant, and all points in the interior of uv have the same set of edges of G containing their farthest points in G.

Among these subdivisions we call the one with the least number of additional vertices the network farthest-point diagram *of G and denote it by $\mathcal{FD}(G)$.*

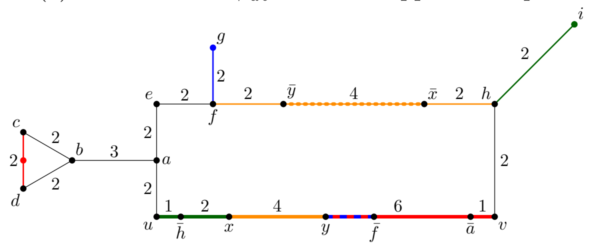
The network farthest-point diagram can be obtained in the same manner as the eccentricity diagram. During the construction of the upper envelope, we keep track of the edges e whose functions ϕ^{e}_{uv} contribute to this envelope. See Figure 7 for an example.

For each edge of the network farthest-point diagram we can store the set of farthest points or the set of edges containing them depending on the type of the edge. Each of these sets consist of at most m elements. With this data we can answer queries for the set of farthest points of a point p on a given edge uv of G as follows. First, we identify the sub-edge ab of uv in $\mathcal{FD}(G)$ containing p using binary search. If a set of farthest points is stored with ab, we return this set. Otherwise, we store the set of edges containing the farthest points of p with ab. In that case we use the distances between all vertices of G to obtain the locations of the farthest points from p in constant time per point.

Theorem 13 [2] *Given a network G with n vertices and m edges. There is a data structure that can be used to determine the set of farthest points of any point p on G in $\mathcal{O}(\log(n)+k)$ time, when given the edge uv containing p, where k is the size of the output. This data structure has a construction time and size of $\mathcal{O}(m^3)$.*



(a) The functions ϕ^{st}_{uv} and thier upper envelope.



(b) The network farthest-point diagram on uv indicated with colours. Farthest points are located at the dot(ted segments) of matching colour.

Figure 7: An example for determining the network farthest-point diagram for the network G from Figure 5. The upper envelope (a) of the functions ϕ^{st}_{uv} reveals which edges contain farthest points (b).

[2]In the full version of this paper, we show how to obtain this result with a construction time and size bound of $\mathcal{O}(m^2 \log(n))$.

4 Solving a Feed-Link Problem

Now we demonstrate how to solve the feed-link problem stated in the introduction with the aid of the data structure from Theorem 9. We begin with a formal definition of the former. Here we assume all edge weights, including that of any possible feed-link, to be equal to the Euclidean length of the corresponding line segment. A network with this property is referred to as *geometric*. Further, if we introduce the feed-link pq to a point q on G, we denote the resulting network by $G + pq$ and refer to q as the *anchor* of p in $G + pq$.

Definition 14 *Let G be a geometric network. Further, let p be a point in the plane that is not on G. We call the problem of determining a point q on G such that the eccentricity of p with respect to $G + pq$ is smallest the* minimum eccentricity feed-link problem*.*

Lemma 15 *Let uv be an edge of the eccentricity diagram of G. If the eccentricity is increasing on uv from u to v, then u is the optimal anchor on uv. Otherwise, the closest point to p on uv is the optimal anchor on uv.*

The (globally) optimal anchor on G is found by scanning through all edges of the eccentricity diagram of G and determining the (locally) optimal anchor on each of them. In case there are restrictions for the position of the anchor point, we only use the part of the eccentricity diagram for the allowed anchor points. For example, one could require that the extended network $G + pq$ should be planar. Then only the points on G that are visible from p may be anchors.

5 Future Work

The construction algorithms for the data structures in Theorem 9 and 13 work for any type of network, yet they suffer from slow running times and the need to know all vertex-to-vertex distances in the network. The following improvements [6] upon these results are beyond the scope of this extended abstract and will be the matter of future publications. For cactus networks, we can obtain data structures with the same query times as in Theorem 9 and 13 but with storage requirement and construction time of $\mathcal{O}(n)$. Moreover, for planar networks, we can construct a data structure for a designated face in $\mathcal{O}(n \log(n))$ time. Neither of these results require precomputed vertex-to-vertex distances. For more details refer to Grimm [6].

References

[1] P. K. Agarwal and M. Sharir. Davenport-Schinzel sequences and their geometric applications. In *Handbook of computational geometry*, pages 1–47. North-Holland, 2000.

[2] B. Aronov, K. Buchin, M. Buchin, B. M. P. Jansen, T. de Jong, M. J. van Kreveld, M. Löffler, J. Luo, R. I. Silveira, and B. Speckmann. Connect the dot: Computing feed-links for network extension. *Journal of Spatial Information Science*, 3(1):3–31, 2011.

[3] M. Erwig. The graph Voronoi diagram with applications. *Networks*, 36(3):156–163, 2000.

[4] H. Frank. A note on a graph theoretic game of Hakimi's. *Operations Research*, 15(3):567–570, 1967.

[5] T. Furuta, A. Suzuki, and K. Inakawa. The k-th nearest network Voronoi diagram and its application to districting problem of ambulance systems, 2005.

[6] C. Grimm. Eccentricity diagrams. Diplomarbeit (Master's thesis), Otto-von-Guericke-Universität Magdeburg, Magdeburg, 2012.

[7] A. Grüne. *Geometric Dilation and Halving Distance*. Dissertation (PhD thesis), Rheinische Friedrich-Wilhelms-Universität Bonn, Bonn, 2006.

[8] S. L. Hakimi, M. Labbé, and E. F. Schmeichel. The Voronoi partition of a network and its implications in location theory. *INFORMS Journal on Computing*, 4(4):412–417, 1992.

[9] F. Harary. *Graph theory*. Narosa/Addison-Wesley, indian student edition edition, 1989.

[10] R. K. Kincaid. Exploiting structure: Location problems on trees and treelike graphs. In *Foundations of Location Analysis*, volume 155 of *International Series in Operations Research & Management Science*, pages 315–334. Springer US, 2011.

[11] G. Narasimhan and M. Smid. *Geometric spanner networks*. Cambridge University Press, 2007.

[12] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial tessellations: concepts and applications of Voronoi diagrams*. Wiley Series in Probability and Statistics. John Wiley & Sons Ltd., Chichester, 2nd edition, 2000.

[13] A. Okabe, T. Satoh, T. Furuta, A. Suzuki, and K. Okano. Generalized network Voronoi diagrams: Concepts, computational methods, and applications. *International Journal of Geographical Information Science*, 22(9):965–994, 2008.

[14] B. Ç. Tansel. Discrete center problems. In *Foundations of Location Analysis*, volume 155 of *International Series in Operations Research & Management Science*, pages 79–106. Springer US, 2011.

Tight Linear Lower Memory Bound for Local Routing in Planar Digraphs

Maia Fraser*

Abstract

Local *geometric* routing algorithms with logarithmic memory are in widespread use in modern MANETs (mobile ad hoc networks). Formally, these are algorithms executed by an agent traveling from node to node in a geometric graph, using only *local geometric information* at each node, leaving no traces and carrying $O(\log n)$ bits of memory. For the case of undirected graphs, theoretical and practical aspects of such algorithms have been extensively developed since Face Routing (FR) was proposed in 1999 for planar embedded graphs. By contrast, the corresponding problem in geometric digraphs has been relatively little studied. In CCCG'08, the author and co-authors showed a lower bound of $\Omega(n)$ bits on the memory of local geometric routing algorithms for planar embedded digraphs.

The purpose of the present paper is to show this lower bound is tight under two models: either node identifiers (possibly coordinates) are known to come from an $O(n)$ size space and no marks may be left, or else $O(\log n)$ bits suffice to identify a node (as assumed in FR) and pebbles may be left. We describe an analog to FR which under either model guarantees delivery in polynomial time in strongly connected planar embedded digraphs. In the first model, memory of $O(n)$ bits is transported, in the second $O(\log n)$ bits are transported and $O(n)$ pebbles are left. By contrast, for non-geometric algorithms of the second model a tight lower memory bound of $\Omega(n \log n)$ bits follows from work of Ilcinkas and Fraigniaud with an exponential runtime algorithm. Our work thus provides a first example confirming that geometry simplifies routing in digraphs.

1 Introduction

Face Routing (FR), proposed in a CCCG'99 paper by Kranakis, Singh and Urrutia, was the first routing algorithm to use geometric and purely local information. It guarantees delivery in time $O(n)$ for planar embedded (undirected) graphs of size n, while transporting only $O(\log n)$ bits memory[1]. It opened a new era in routing at a time when both wireless ad hoc networks (where global connectivity information is not available) and global positioning devices (which provide real-time position information) were becoming commonplace. The idea is simple: to get from node s to node t the algorithm walks one-by-one around the faces which meet the segment st, until t is reached. While only applying to graphs with no edge-crossings, the simplicity and robustness of the algorithm made it canonical and it was modified and extended in many ways so that today most practically occurring MANETs may be handled by some derivative of FR, and it remains at the base of some of the most commonly used algorithms in this class today.

By contrast very little is known for the case of geometric *directed graphs* (digraphs). In certain special cases of planar embedded digraphs, local geometric routing algorithms with logarithmic memory are known to exist: in Eulerian graphs (in which in- and out-degrees coincide and are constant) and in outerplanar graphs (in which a single face contains all vertices) [2]. But these are very restricted classes and even within the class of planar embedded digraphs there is no hope of a logarithmic memory algorithm like FR: in CCCG'08 we showed [5] there exist planar embedded digraphs in which correct local geometric routing *requires* $\Omega(n)$ *bits memory.* In that paper, we considered only algorithms which do not leave traces, however our proof can be adapted to cover algorithms which do leave traces, in which case the *required memory* is the combined number of pebbles and transported bits. We give the complete argument in the full version of this paper. The point is that our proof of the lower bound in [5] was based on a simulator of local geometry for a special class, $\mathcal{C}$, of graphs, defined by a Kolmogorov random bit string x, and this simulator can be made to reproduce pebbles as well.

The aim of the present paper is theoretical[2]. We show the linear lower memory bound is tight under two models: either node identifiers (possibly coordinates) come from an $O(n)$ size space and no marks are left or else $O(\log n)$ bits identify a node (as assumed in FR) and marks (pebbles) may be left. We describe an analog of FR which correctly routes in all strongly connected planar embedded digraphs under these models. In the

*Department of Computer Science, University of Chicago, maia@cs.uchicago.edu

[1]the minimal memory for a routing algorithm, if it by definition carries at least the destination ID.

[2]We concluded in [5] that uni-directional links should be avoided in MANETs. In fact, there is another practical issue. Unless we allow multicasting, node v will not send to node u unless it knows of u's existence and given a uni-directional arc from v to u this is impossible to achieve directly by communication between the two nodes: nodes will not be locally aware of their downstream neighbours.

first model, memory of $O(n)$ bits is transported and no marks are left; in the second, $O(\log n)$ bits are transported and $O(n)$ pebbles are left.

We remark that a significant amount of theoretical work does exist on routing in *non-geometric* digraphs under other models. In particular the problem of directed *st*-connectivity is usually posed assuming a JAG model (see for example the survey article [1]). JAG's are automata working in a team and able to teleport (jump) to teammates. This is very different from our setting. An instance of our agent may not be teleported but must be *transmitted* and this is by definition only possible to direct neighbours. Our second model, however, is the geometric analog of a model considered by Ilcinkas and Fraignaud in [3]. They assume an agent which cannot jump but is able to leave pebbles at nodes and transport some memory. They show such an agent needs $\Omega(n \log d)$ bits of memory to explore a digraph with maximum out-degree d but *no vertex labeling*, even if it can use a linear amount of pebbles. Since no geometric information is present, by the adversary argument this is also a lower bound on routing (the destination would be marked instead of specified by ID). They give such an algorithm with exponential runtime. By contrast, our algorithm which has access to geometry (and hence node ID's) can successfully route in polynomial time leaving $O(n)$ pebbles and transporting $O(\log n)$ bits.

2 Directed Face Routing Algorithm

Let G be a directed graph of size n embedded in the plane. The main strategy of the algorithm is the same as that of FR:

Input: source s and destination t
Procedure:

1. Traverse anti-clockwise the face which is entered by the segment st at s.
2. If t is visited then STOP.
3. Else if a node s' such that $d(s',t) < d(s,t)$ is visited then $s \leftarrow s'$.
4. Go to step 1.

The process of traversing a face, however, is much more arduous than in the undirected case.

3 Results

Assuming either of the two models defined above, we will show:

Theorem 1 *There is a local geometric algorithm,* Directed Face Traversal*, which uses $O(n)$ bits memory and traverses a given face F of a strongly connected planar embedded digraph G.*

This then implies:

Corollary 2 *Directed FR is a local geometric algorithm transporting $O(n)$ bits memory which guarantees delivery in any strongly connected planar embedded digraph G.*

Indeed, using Directed Face Traversal, Directed FR is guaranteed to reach t by the usual argument: since G is embedded in the plane, when the segment st enters a face F it must meet the boundary of F again either to exit F or to arrive at t; thus, one of the conditions in steps 2. and 3. is guaranteed to hold and either we will stop at t or the next iteration begins with a strictly reduced distance $d(s,t)$. Moreover, this local geometric algorithm transports only $O(\log n)$ bits memory of its own (to record coordinates of s and t) besides the memory of Directed Face Traversal so the over all memory requirement remains $O(n)$ bits.

Observation 1 *For simplicity, we will describe the algorithm in terms of the second model - leaving $O(1)$ pebbles per node. In the case of the first model*[3]*, we may associate $O(1)$ bits to each node using an array of total size $O(n)$ bits, and so by transporting this memory we can simulate the algorithm written for the second model.*

The rest of this paper is devoted to describing Directed Face Traversal and proving Theorem 1.

4 Terminology

First we fix some standard terminology for embedded digraphs. By *edge* we mean an edge of the undirected graph G' obtained by forgetting the directions of arcs of G. By *face* we mean a connected component of the complement of G' in the plane. A *walk* will refer to a walk of G'; i.e., a sequence of vertices $v_0, v_i, \ldots, v_k$ such that either $v_{i-1}v_i$ or v_iv_{i-1} is an arc of G for each $i : 1 \leq i \leq k$. The walk is said to be *directed* if arc orientation is respected, i.e. each $v_{i-1}v_i$ is actually an arc of G. A walk is closed if $v_0 = v_k$. The vertices of a walk may in principle coincide; we say the walk is *simple* if they do not. The fact the digraph G is *strongly connected* means that for any two nodes, s and t, there exists a directed walk starting at s and ending at t. Every face has a unique *boundary walk* starting at a vertex v on the boundary of the face, namely the closed walk consisting of the nodes

[3]We make two remarks. First, the assumption of $O(\log n)$ identifiers is also made in standard FR when coordinates are used as identifiers. Indeed, if nodes were arranged on the real line with exponentially increasing spacing (e.g. distance 2^k between k'th and $k+1$'st nodes) then coordinates with $\Omega(n)$ bits would be needed in order to distinguish nodes; such an example is usually ruled out. Second, if either node ID's or coordinate pairs come naturally from key spaces which are large they could be hashed first; we assume we start with (possibly virtual) node ID's which are integers in a range $[0, Cn]$ for some constant C.

which are encountered when following the face boundary in its canonical boundary orientation, starting from v. This orientation is *to the left* for an observer standing within the face. Boundary walks are in general not directed walks.

Since G is embedded there is a natural cyclical ordering of edges at each vertex: draw a small circle around v, travel around it in the clockwise direction and cyclically order the edges incident at v by the cyclical order in which they are encountered. Suppose uv is an incoming arc at v then we denote by $\text{succ}(uv)$ the next outgoing arc in this cyclical ordering. Given a set M of marked vertices one may analogously define $\text{succ}_M(uv)$ as the next outgoing arc whose head belongs to M; we assume succ_M returns NULL if there are no such outgoing arcs.

This allows one to define a walk starting from a given arc a by iteratively taking $\text{succ}(a)$. If a walk $v_0, \ldots, v_k$ is thus defined by succ_M where M is the vertex set of the walk, we say the walk is *left-free*, since the condition is equivalent to there being no edges of the walk incident on the left side of the walk. Boundary walks are always left-free (since there are no edges at all between uv and vw)[4]. Simple walks are certainly left-free.

We now introduce two notions which will simplify our discussion of face traversal.

Definition 1 *A closed walk $v_0, v_1, \ldots, v_k = v_0$ is said to* **self-cross** *if some edge $v_{i-1}v_i$ is incident at $v_k = v_i$ and lies strictly to one side of the walk at v_k, while v_iv_{i+1} lies strictly to the other. When a closed walk β is not self-crossing we define $\mathcal{C}(\beta)$ to be the collection of* **faces it encloses***: those faces from which a curve may be traced which first meets β on the left and not the right side of β.*

This is illustrated in Figure 1.

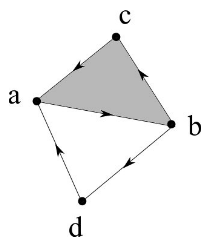


Figure 1: The shaded face is enclosed by the walk a, b, c, a. On the other hand, the walk a, b, d, a encloses a collection of two faces: the shaded one and the outer face.

Suppose β is non self-crossing and F is a face *not* in $\mathcal{C}(\beta)$ but *adjacent* to an edge $v_{i-1}v_i$ of β. It lies necessarily on the right of β and when one replaces the sub-walk $v_{i-1}v_i$ of β with the remainder of the boundary walk of F (also a walk from v_{i-1} to v_i) one obtains a new non self-crossing walk β' which encloses one more face than did β. For example, in Figure 1, the unshaded triangle is a face not enclosed by a, b, c, a, and so using it we can produce the walk a, d, b, c, a which encloses both triangular faces.

[4]For this to hold, it is important that the cyclical ordering be done in clockwise fashion when one considers boundary walks with canonical boundary orientation.

Definition 2 *Given a directed walk α define its* **outer shell** *shell(α) to be the (non self-crossing) closed directed walk which includes α as a sub-walk and encloses a minimal number of faces.*

To see this is well-defined, suppose there are two non-identical closed directed walks $u, v, w_1, \ldots, w_k = u$ and $u, v, w'_1, \ldots, w'_k = u$ both enclosing a minimal number M of faces. Starting from v let the last vertex where they coincide be w_ℓ, so $w_i = w'_i$ for $i \leq \ell$ and $w_{\ell+1} \neq w'_{\ell+1}$. Now let $w_m = w'_p$ be the first vertices which coincide for $m, p > \ell$. We then have two directed walks from $w_\ell = w'_\ell$ to $w_m = w'_p$ which meet only at their endpoints: $w_\ell, w_{\ell+1}, \ldots, w_m$ and $w'_\ell, w'_{\ell+1}, \ldots, w'_p$. By replacing one sub-walk with the other we can obtain a closed directed walk which includes u, v and encloses fewer than M faces, a contradiction.

5 Directed Face Traversal Algorithm

Idea The idea of this algorithm is to traverse the face boundary β until an opposing arc a is encountered, say at vertex v, and then to traverse the outer shell σ of the arc a (or the outer shell of a longer directed walk ending in a). This shell traversal cannot in general be done in one iteration, rather our algorithm will on the i'th iteration produce a non self-crossing directed closed walk δ_i such that $\mathcal{C}(\delta_i) \subset \mathcal{C}(\sigma)$ and such that $\mathcal{C}(\delta_i) \subsetneq \mathcal{C}(\delta_{i+1})$. Eventually therefore we must have $\delta = \sigma$. Suppose this walk starts and ends at the head of a. We now use, as an exit from this area, the outgoing arc uw whose tail (on σ) most closely preceeds v (in the order on σ) such that there are no outgoing arcs on the right of σ between uw at u and the incoming arc at v and thus we know that this part of σ (which includes a) is part of the face boundary β which we seek. And then we continue. This process must terminate since we always increase the visited portion of β.

5.1 Internal memory

Under the first model, the algorithm stores three binary arrays of size $O(n)$:

$$\text{INNER[]},\ \text{OUTER[]},\ \text{and MID[]},$$

all initialized to zero. Under the second model it uses pebbles of types INNER, OUTER, MID. In both models, it also records three node identifiers: JOIN, NEW_EXIT, and EXIT, each of size $O(\log n)$ as well as the coordinates of s and t. We will assume there is a simple logspace subroutine which verifies if a given edge crosses the segment st.

In addition, Directed Face Traversal records its state which is one of the following: WALK_BDRY, BACKTRACK, FIND_EXIT, GO_EXIT. Finally there is a Boolean flag TENTATIVE, initially set to false.

5.2 State diagram

The state diagram for the algorithm is cyclical:

$$\begin{aligned} \text{WALK\_BDRY} &\rightarrow \text{BACKTRACK} \\ &\rightarrow \text{FIND\_EXIT} \\ &\rightarrow \text{GO\_EXIT} \quad \rightarrow \text{WALK\_BDRY} \end{aligned}$$

5.3 Invariant

In the full version of this paper we prove that the following property is an invariant, always true upon entering a new state.

Path property:

1. The nodes marked OUTER form a *directed* simple walk, τ, from s to EXIT.
2. The nodes marked INNER form a left-free walk from s to JOIN which passes through EXIT. We denote by η the sub-walk up to EXIT and by λ the sub-walk after EXIT.
3. The nodes marked INNER and MID can be used to form a *directed* walk from EXIT to s (take MID when INNER cannot be taken).
4. τ concatenated with $-\eta$ (which goes from EXIT to s) is non self-crossing and all nodes marked MID belong to faces enclosed by this concatenated curve.

This is illustrated in Figure 2.

Explanation Borrowing the notation used to describe the *Idea* of the algorithm at the start of Section 5, the point of the Path property is that it allows the current δ_i to be defined by pebbles; namely, δ_i is 'almost' traversed by starting at s and following τ then $-\eta$ back to s. The reason for the word 'almost' is that such a walk will in general not be a directed walk: although τ is directed so that one can indeed follow it in this direction, $-\eta$ may not be. The small walks marked MID are detours which allow one to get back to s from EXIT (i.e. each piece of MID by-passes an unsuitably directed part of $-\eta$). Thus a more accurate statement is that by following OUTER till EXIT and then INNER+MID back to s one traverses δ_i.

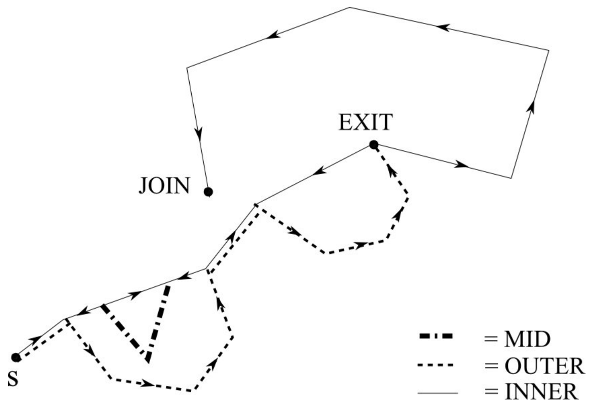


Figure 2: Path property

In establishing the Path property, showing pebbles form a walk relies on the following fact.

Lemma 3 *Any left-free walk β may be uniquely followed (from some vertex u onward) by a memory-less (local) geometric algorithm as long as all its vertices are marked by pebbles, no adjacent non-β vertices are marked, and the algorithm is given a starting arc at u.*

This holds because if we start at a pebble-marked node u and are given the first arc uv to follow, then we may iteratively follow the path defined by succ_M where M is the set of pebble-marked vertices.

Now, assume we have established the Path property as an invariant. We will *upon entry into each state* be able to follow τ, $-\eta$, or λ . In our pseudocode we write `follow` to indicate we are following one of these walks appealing to these principles. We now describe the main loop of Directed Face Traversal.

5.4 Main loop

We describe the algorithm in **high-level pseudocode**, suitable for proving correctness. In the interests of clarity we will refer to the curves η, τ and λ from the Path property, as well as a vertex called PRIOR which is defined below. **All of these walks as well as PRIOR can however be determined locally.** The extended version of this paper includes an appendix where lower-level pseudocode is given for the algorithm. That code does not refer to η, τ, λ and PRIOR but implements the same actions as specified below.

It remains to define the high-level variable PRIOR. Assuming the path property, the walk along INNER from EXIT to JOIN meets either:

1. the walk INNER only, and on the left side of that walk, in a section bypassed by MID (a) or not (b).

2. the walk OUTER only, and on the right side of that walk
3. a vertex of both walks, either on the left or right side (cases (a) and (b) resp.).

In all cases, we define PRIOR to be the vertex belonging to *both walks* which most closely preceeds (or equals) the vertex[5] JOIN (in the walk that was met).

This is illustrated in Figure 3. The convention is the same as in Figure 2: the dotted directed walk represents OUTER and it ends at EXIT. After that point the walk along INNER continues from EXIT to JOIN, where it meets either INNER, or OUTER, or both. This is indicated by a ray with its arrow pointing to the place of return. The possible cases are illustrated and labeled as above: 1ab, 2, 3ab.

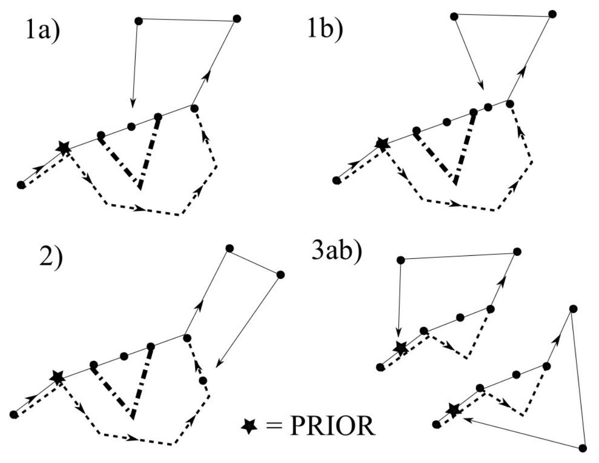


Figure 3: PRIOR is shown for the various cases

We now make two remarks. They concern respectively the setting of PRIOR (in the state BACKTRACK) and the use of PRIOR (in the state FIND_EXIT).

Observation 2 *It is possible for the agent to deduce which case has occurred by walking provisionally to the left once JOIN occurs. A case by case analysis shows the logic involved. The details are given in the extended version of this paper. The key point is that the orientation of arcs along INNER between intersections with OUTER or MID is prescribed and so contradictions must eventually arise if the provisional assumption is wrong. We assume that PRIOR can be found, i.e. the agent can walk to this vertex and determine that it is the vertex PRIOR. Moreover, at that moment it knows which way τ is oriented.*

Observation 3 *Once the agent is at PRIOR, by walking along τ and then $-\eta$ (using detours provided by MID), always keeping track of the last outgoing arc on the right until PRIOR is reached again, the location of a new exit can be found. By strong connectivity some outgoing arc must exist and this method of choosing it will ensure there are no outgoing arcs between it and PRIOR on the walk just described.*

[5]In the first case, where JOIN is a repeated vertex of INNER, we refer to its first occurrence.

The actions to perform in the main loop depend on the state. If at any point the following implicit exit condition becomes valid the main loop is exited and the algorithm places the agent at whichever endpoint u or v is closer to t and returns.

Exit condition: the arc uv crosses the line segment st.
Initialization: Set EXIT and JOIN to s and let $a = sv$ be the first arc which exits s to the right of the line segment st.
Main loop:

- WALK_BDRY:
  1. Use succ() starting along the arc a (at EXIT) to build a left-free walk, marking all its vertices with INNER. Stop when this walk first meets an existing INNER or OUTER vertex and let JOIN be the intersection vertex.
  2. Change state to BACKTRACK.
- BACKTRACK:
  1. `Follow` INNER+MID or OUTER and thereby determine and reach the vertex PRIOR (see Observation 2).
  2. Change state to FIND_EXIT.
- FIND_EXIT:
  1. Walk along τ then $-\eta$, always recording the last outgoing arc encountered on the right (see Observation 3). Let NEW_EXIT be the vertex where this outgoing arc is found (it will be PRIOR itself if no later outgoing arc was found).
  2. Change state to GO_EXIT.
- GO_EXIT:
  1. Return to NEW_EXIT correcting all markers so the Path property will be preserved[6] upon setting EXIT $\leftarrow$ NEW_EXIT. Set EXIT $\leftarrow$ NEW_EXIT and also JOIN $\leftarrow$ NEW_EXIT. Set a to succ(b) at EXIT, where b is the outgoing arc of the new $-\eta$ at EXIT.
  2. Change state to WALK_BDRY.

[6]We want to move EXIT to a new vertex, NEW_EXIT, along the combined walk τ with $-\eta$ and thus we will be changing the breakpoint between τ and $-\eta$. To retain the Path property the pebbles for INNER, MID and OUTER must be adjusted accordingly. Details are given in the extended version of this paper.

5.5 Correctness

Proof. Let F be the face determined by sv from the initialization. We will prove that Directed Face Traversal traverses F. We assume that the Path property holds upon entry to all states (the proof of this invariance is given in the extended version).

Consider the first v_{i-1}, v_i on the boundary walk β of F which is *not an arc*. Then $v_i v_{i-1}$ is an arc. We claim the agent will eventually reach v_i. Let α be a maximal directed sub-walk of $-\beta$ which ends with v_i, v_{i-1}. Let $\sigma = \text{shell}(\alpha)$ be the outer shell of α.

Let μ be the concatentation of τ with $-\eta$. We prove by induction (on the iteration), that at the start of the main loop we have $\mathcal{C}(\mu) \subsetneq \mathcal{C}(\sigma)$ as long as the agent has not yet arrived v_i. To see this, assume μ is as at the start of the k'th iteration and it satisfies this condition. We claim its new value at the start of the $k+1$'st iteration will also do so. We consider the new λ which will be formed during the WALK_BDRY state. Suppose the agent will not arrive at v_i by the start of the $k+1$'st iteration. Let $F_1, F_2, F_3, \ldots$ be the faces encountered on the left when following the new λ and $a_1, a_2, a_3, \ldots$ the corresponding arcs which are incoming to λ on the left. If the first vertex of this walk after EXIT does *not* belong to a face in $\mathcal{C}(\sigma)$ then since $\mathcal{C}(\mu) \subset \mathcal{C}(\sigma)$, the walk μ must coincide with σ at EXIT. But following μ back to v_{i-1} we do not meet any outgoing arcs on the right. Thus μ and σ must coincide all the way back to v_{i-1} and so μ passes through v_i, a contradiction. Thus the first vertex past EXIT on the new λ does indeed belong to a face in $\mathcal{C}(\sigma)$. And so σ encloses F_1. But given the orientation of a_1, this implies σ encloses the outer shell of a_1 and hence encloses F_2, and so on. This continues until we reach the end of the new λ (at the new JOIN) because on the left side of λ there are only incoming arcs. Thus at the start of the $k+1$'st iteration we will have $\mathcal{C}(\mu) \subset \mathcal{C}(\sigma)$ and this inclusion must be strict, otherwise μ would equal σ and the agent would return to v_i.

Moreover, we have also shown that $\mathcal{C}(\mu)$ increases on each such iteration. This cannot go on forever so eventually the agent must reach v_i. This means it will advance along β past a_1. We repeat the above argument using (in place of τ and η) τ_v and η_v, the sub-walks of τ and η starting at v, where v is the next value of EXIT after the agent has returned to v_i (we know EXIT belongs to both walks). This process continues and since the agent thus advances by at least one edge along β every time, eventually it must traverse all of β. □

6 Conclusions

We have shown that the lower linear bound on memory for local routing algorithms that was proved in [5] is tight. Our algorithm mimics Face Routing but involves a much more involved face traversal procedure. Nevertheless the linear memory requirement of the algorithm is a significant reduction in the $\Omega(n \log n)$ memory required for local routing in *non-geometric* digraphs proven by Ilcinkas and Fraigniaud [3], where they also provide algorithms with exponential runtime or else polynomial runtime and higher memory. Our geometric algorithm's worst case runtime (hop-length) is polynomial. Indeed on a given iteration of Directed Face Traversal, each arc may be traversed at most $O(1)$ times and there are $O(n)$ iterations for a given face (since the node EXIT will be different on each) and $O(n)$ faces in total.

7 Acknowledgements

The author thanks the referees for their very useful comments and suggestions, in particular for seeking clarification on the models used and prompting Figure 1.

References

[1] G. Barnes, J. Edmonds. Time-space lower bounds for directed st-connectivity on graph automata models. *SIAM J. on Computing*, 27(4):1190–1202,1998

[2] E. Chavez, S. Dobrev, E. Kranakis, J. Opatrny, L. Stacho, J. Urrutia. Route discovery with constant memory in oriented planar geometric networks. *Networks*, 48(1):7-15, 2006.

[3] P. Fraigniaud and D. Ilcinkas. Digraphs exploration with little memory. *21st Symposium on Theoretical Aspects of Computer Science (STACS04)*, LNCS 2296:246-257, 2004.

[4] E. Kranakis, H. Singh, J. Urrutia. Compass routing in geometric graphs. *11th Canadian Conference on Computational Geometry (CCCG'99)*, 51-54, 1999.

[5] M. Fraser, E. Kranakis, J. Urrutia. Memory requirements for local geometric routing and traversal in digraphs. *20th Canadian Conference on Computational Geometry (CCCG'08)*, 2008.

Packing Trominoes is NP-Complete, #P-Complete and ASP-Complete

Takashi Horiyama* Takehiro Ito† Keita Nakatsuka‡ Akira Suzuki† Ryuhei Uehara§

Abstract

We study the computational complexity of packing puzzles of identical polyominoes. Packing dominoes (i.e., 1×2 rectangles) into grid polygons can be solved in polynomial time by reducing to a bipartite matching problem. On the other hand, packing 2×2 squares is known to be NP-complete. In this paper, we fill the gap between dominoes and 2×2 squares, that is, we consider the packing puzzles of trominoes. Note that there exist only two shapes of trominoes: L-shape and I-shape. We show that their packing problems are both NP-complete. Our reductions are carefully designed so that we can also prove #P-completeness and ASP-completeness of the counting and the another-solution-problem variants, respectively.

1 Introduction

Since Golomb introduced the notion of polyominoes, many puzzles have been considered and solved for polyominoes [5]. Most puzzles are classified into two groups: the sliding-block puzzles and the packing puzzles (see Figure 1(a) and Figure 1(b), respectively). The computational complexity of solving sliding-block puzzles was a long standing open problem since Gardner introduced the problem in 1964 [4]. In 2005, it was settled by Hearn and Demaine [6]: generalized sliding-block puzzles are PSPACE-complete even if all blocks are of size 1×2. On the other hand, sliding-block puzzles are polynomial-time solvable if all blocks are of size 1×1 (see [7] for further details). Thus, in this sense, we have no gap for sliding-block puzzles.

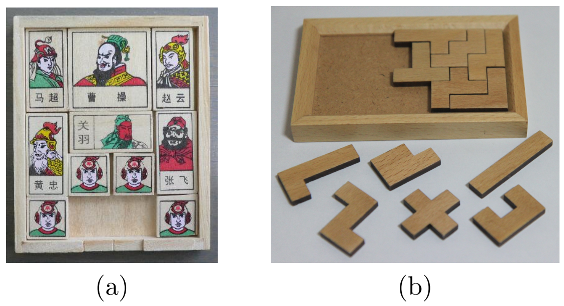


Figure 1: (a) A sliding-block puzzle and (b) a packing puzzle.

In this paper, we consider packing puzzles, and we aim to fill this kind of gap. We consider the problem of so-called identical packing into the two dimensional plane: Given k identical polyominoes and a polygon P, determine whether the k polyominoes can be placed in P without overlap or not. We note that P may contain holes. Packing dominoes (i.e., 1×2 rectangles) into a polyomino P (i.e., a polygon made by connecting unit squares) can be solved in polynomial time by reducing to a bipartite matching problem. (See [8] for a bipartite matching algorithm.) On the other hand, packing 2×2 squares into a polyomino is known to be NP-complete [2, 3]. That is, there exist gaps between dominoes and 2×2 squares: Can you pack trominoes into a polyomino in polynomial time?

Here, a tromino is a (connected) polygon with three unit squares. As shown in Figure 2, trominoes have only two possible types of shapes: L-trominoes and I-trominoes. We consider the corresponding two problems, called 3L-PACKING and 3I-PACKING. In 3L-PACKING, an integer k and a polyomino P are given. P is given as a set of pairs of integers, corresponding to the positions (e.g., the centers) of unit squares in P. The 3L-PACKING problem is to determine whether k L-trominoes can be placed in P without overlap or not. 3I-PACKING is defined in the same manner, so as to pack I-trominoes.

We prove both problems are NP-complete by giving reductions from one-in-three 3SAT. The reductions are carefully designed so that each (valid) packing of trominoes has one-to-one correspondence with a (valid) solution of the original instance of one-in-three 3SAT. Therefore, our reductions also imply the results for the counting variant and the another-solution-problem variant of the tromino packing problems. Sim-

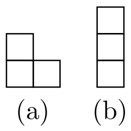


Figure 2: (a) An L-tromino and (b) an I-tromino.

*Information Technology Center, Saitama University, horiyama@al.ics.saitama-u.ac.jp
†Graduate School of Information Sciences, Tohoku University, {takehiro, a.suzuki}@ecei.tohoku.ac.jp
‡Faculty of Engineering, Saitama University, nakatsuka@al.ics.saitama-u.ac.jp
§School of Information Science, JAIST, uehara@jaist.ac.jp

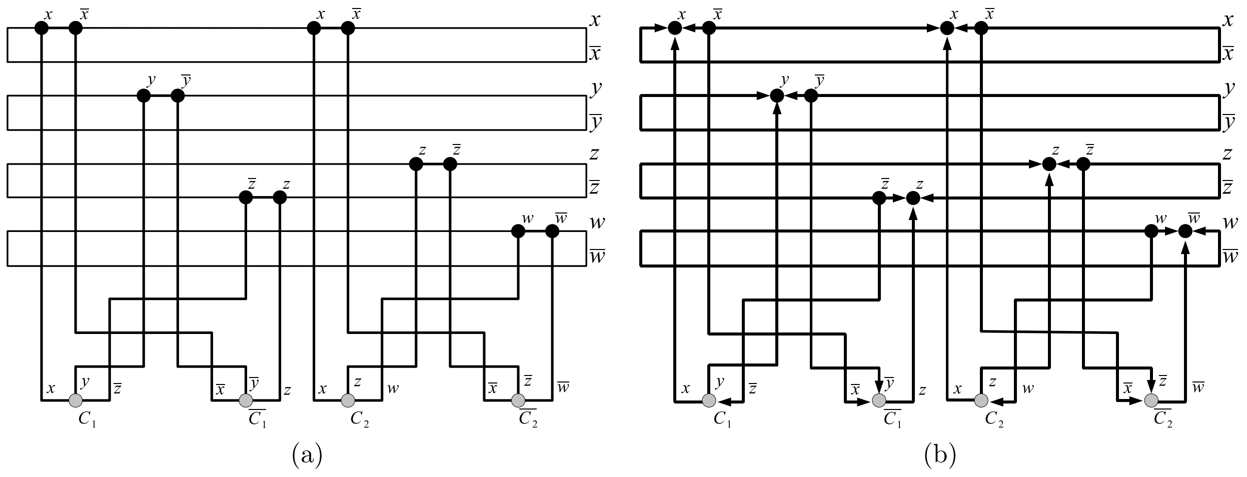


Figure 3: (a) Instance G_φ of 1-in-3 GO, and (b) its valid orientation.

ilar to other ASP-complete problems [10], the another-solution-problem variant is defined as follows: Given an instance (k, P) of 3L-PACKING or 3I-PACKING and its (valid) solution s, find a solution s' of (k, P) other than s. The counting variants and the another-solution-problem variants for the 3L-PACKING and 3I-PACKING problems are #P-complete and ASP-complete, respectively.

2 Reduction

To prove the NP-completeness of 3L-PACKING and 3I-PACKING, we introduce a graph orientation problem, called the one-in-three graph orientation problem (or 1-in-3 GO in short), and give reductions from one-in-three 3SAT to 3L-PACKING and 3I-PACKING via 1-in-3 GO.

2.1 Reduction to 1-in-3 Graph Orientation Problem

We first give a polynomial-time reduction from one-in-three 3SAT to 1-in-3 GO.

In one-in-three 3SAT, we are given a 3-CNF φ consisting of m clauses with n variables, where each clause C_j contains three literals (variables or their negations). One-in-three 3SAT is to determine whether there is a satisfying assignment to the variables so that each clause in φ has exactly one true literal. For example, given $\varphi = (x \vee y \vee \overline{z})(x \vee z \vee w)$, we have a satisfying assignment $(x, y, z, w) = (\text{False}, \text{False}, \text{False}, \text{True})$.

1-in-3 GO is defined as follows:

Definition 1 1-*in*-3 GO (*One-in-three Graph Orientation Problem*).

An undirected 3*-regular graph* $G = (V, E)$ *is given, where* V *can be partitioned into three* (*disjoint*) *node-subsets* V_ℓ, V_c *and* V_n *consisting of literal nodes, clause nodes, and negated-clause nodes, respectively. The objective is to determine whether we can assign a direction to each edge so that* (1) *every literal node in* V_ℓ *has in-degree* 0 *or* 3; (2) *every clause node in* V_c *has exactly one inbound edge, i.e., in-degree* 1; (3) *every negated-clause node in* V_n *has exactly one outbound edge, i.e., out-degree* 1.

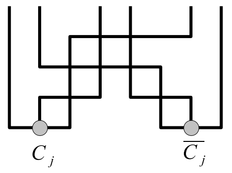


Figure 4: A clause node and its corresponding negated-clause node.

Figure 3(a) illustrates the undirected graph G_φ corresponding to an instance $\varphi = (x \vee y \vee \overline{z})(x \vee z \vee w)$. Note that a node in G_φ is depicted by a (black or gray) circle. The upper half of G_φ consists of n cycles, each of which corresponds to a variable in φ. Each cycle consists of some pairs of literal nodes, and the number of pairs equals to the number of occurrences of the variable in φ. If a pair of literal nodes is located in the upper half of its belonging cycle, the left (resp., right) node of the pair is labeled with a positive (resp., negative) literal. Otherwise, the left (resp., right) node is labeled with a negative (resp., positive) literal. The lower half of G_φ consists of m gadgets given in Figure 4, where the nodes labeled with C_j are clause nodes, and those labeled with $\overline{C_j}$ are negated-clause nodes. If clause C_j contains three literals ℓ_1, ℓ_2 and ℓ_3, the clause node labeled with C_j has exactly three edges connecting with literal nodes labeled with ℓ_1, ℓ_2 and ℓ_3. A negated-clause node labeled with $\overline{C_j}$ has exactly three edges connecting with literal nodes labeled with the negated literals $\overline{\ell_1}$, $\overline{\ell_2}$ and $\overline{\ell_3}$.

Therefore, G_φ contains $8m$ nodes and $12m$ edges in

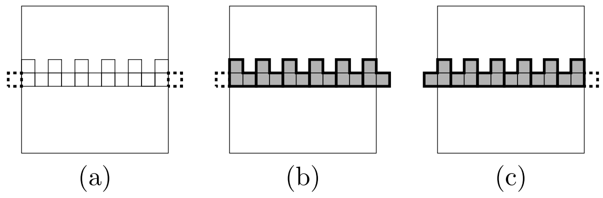


Figure 5: A line gadget and its two ways of packing.

total. We can draw G_φ within the grid of width $O(m)$ and height $O(n)$. Thus, we can obtain G_φ in $O(mn)$ time.

Figure 3(b) illustrates a valid orientation of the graph G_φ in Figure 3(a). Note that the two vertical edges emanating from a pair of literal nodes have different orientations, which implies that one is assigned True and another is assigned False. Also note that the literal nodes labeled with the same literal have the same vertical orientation, i.e., there is a consistency on the assignment to the variables. If a positive literal node has a downward (resp., upward) edge, its corresponding variable in φ is assigned True (resp., False). Restriction (2) on clause node labeled with C_j guarantees that exactly one literal in C_j is assigned True.

Thus, valid orientations to G_φ of 1-in-3 GO have one-to-one correspondence with satisfying assignments to φ of one-in-three 3SAT. We can easily check 1-in-3 GO is in NP, in #P and in ASP. Since one-in-three 3SAT is NP-complete [9], #P-complete [1] and ASP-complete [10], we have the following theorem.

Theorem 1 1-*in*-3 GO *is NP-complete, #P-complete and ASP-complete.*

2.2 Reduction to 3L-PACKING

Now, we give a polynomial-time reduction from 1-in-3 GO to 3L-PACKING. An intuitive correspondence between the two problems can be observed in Figure 5. We use the gadget in Figure 5(a) as a *line gadget*. We can place trominoes on both of the solid and dotted unit squares. If we pack as many L-trominoes as possible into this gadget, we have only two ways of packings as illustrated in Figures 5(b) and (c). We regard the line gadget in Figure 5(a) as an edge, and the ways of packings (b) and (c) as two corresponding orientations of the edge: Packing (b) corresponds to the orientation from left to right; in contrast, packing (c) corresponds to the orientation from right to left. The dotted unit square covered (resp., not covered) by a tromino represents that the orientation is outbound (resp., inbound).

If we need to bend an edge, we use a *corner gadget* in Figure 6(a). Similarly to the case of a line gadget, if we pack as many L-trominoes as possible into a corner gadget, we have only two ways of packings. The dotted unit square covered by a tromino represents that the orientation is outbound. By combining gadgets as in Figure 6(b), we can propagate the orientations of edges.

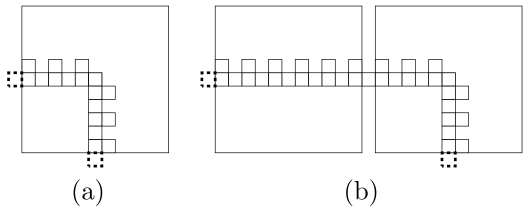


Figure 6: (a) A corner gadget, and (b) a combination of line and corner gadgets.

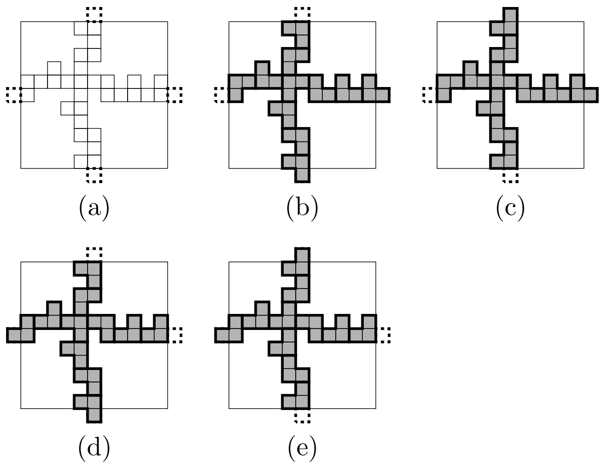
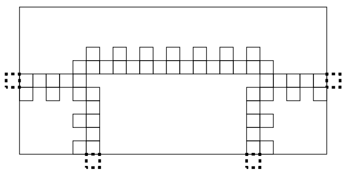


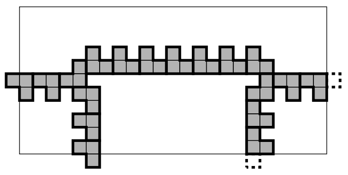
Figure 7: A cross gadget and its four ways of packings.

We respectively replace the crossing points of edges, pairs of literal nodes, clause nodes, and negated-clause nodes by cross, duplicator, clause, negated-clause gadgets, which are defined later. All but the duplicator gadget are of the same size. (More precisely, the normal size is 11×11.) The size of a duplicator gadget equals to that of a combination of two normal-size gadgets, since we replace two literal nodes by one duplicator gadget. As a result of the replacement, we can obtain a patchwork of the gadgets as a polyomino of 3L-PACKING.

A *cross gadget* is given in Figure 7(a). There are four ways, as illustrated in Figure 7(b)–(e), to cover the crossing unit square in Figure 7(a) by an L-tromino. Note that the left and right dotted unit squares always represent the same orientation; so do the upper and bottom dotted unit squares. In contrast, the orientations of vertical and horizontal directions are independent.

A pair of literal nodes is replaced by a *duplicator gadget* given in Figure 8(a). As mentioned before, the width of a duplicator gadget equals to that of a combination of two normal-size gadgets. Apart from the cross gadgets, if we pack as many L-trominoes as possible into a duplicator gadget, we have only two ways of packings. The two dotted unit squares in the left half of the gadget have the same orientation (inbound or outbound); so do the two dotted unit squares in the right half.


(a)


(b)

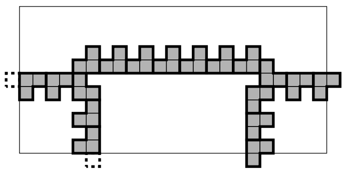
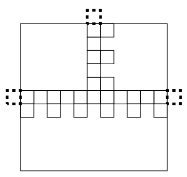
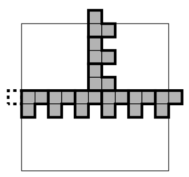
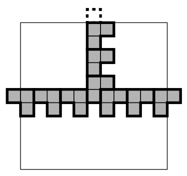

(c)

Figure 8: A duplicator gadget and its two ways of packings.


(a)


(b)


(c)

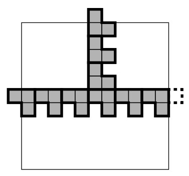

(d)

Figure 9: A clause gadget and its three ways of packings.

A clause node is replaced by a *clause gadget* given in Figure 9(a). We have three ways of packing. In every packing, there are exactly one inbound orientation and exactly two outbound orientations. We can regard this gadget as a three-forked road. The tromino covering the center of the three-forked road indicates which edge has inbound orientation. A negated-clause node is replaced by a *negated-clause gadget* given in Figure 10. Similar to the case of a clause gadget, we have three ways of packing. In every packing, however, there are exactly one outbound orientation and exactly two inbound orientations.

Since the drawing of graph G_φ is of size $O(mn)$, we use at most $O(mn)$ gadgets for replacing the elements in G_φ. All gadgets consist of constant number of unit squares, which implies a polynomial-time construction of P for the reduction.

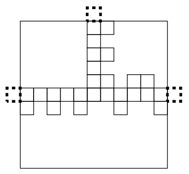


Figure 10: A negated-clause gadget.

We set the 'magic number' k of 3L-PACKING as $|P|/3$, where $|P|$ denotes the number of unit squares in P. This enforces that no unit squares in P remain uncovered. All of the above explained packings satisfy this constraint. Moreover, there is no other way of packing. Thus, valid packings to P of 3L-PACKING have one-to-one correspondence with valid orientations to G_φ of 1-in-3 GO. We can easily check 3L-PACKING is in NP, in #P and in ASP, since P is given as a set of the coordinates of all unit squares in P. By Theorem 1, we have the following theorem.

Theorem 2 3L-PACKING *is NP-complete, #P-complete and ASP-complete.*

2.3 Reduction to 3I-PACKING

By a similar argument as above, we give a polynomial-time reduction from 1-in-3 GO to 3I-PACKING. We use the gadgets in Figure 11(a)–(f) as line, corner, cross, duplicator, clause and negated-clause gadgets, respectively. The gadgets in Figure 11(a), (b) and (e) are straightforward for packing. We note that a crank in a line gadget (Figure 11(a)) guarantees exactly two ways of packings, which correspond to two orientations. These non-trivial gadgets work as follows.

(c) Cross gadget. In general, the cross gadgets are the most difficult part to design in these kinds of reductions, as mentioned in [7]. Such situation also occurs in 3I-PACKING as you can observe from Figure 11(c) and 12. We first observe that each of the left and right squares drawn in dotted line has two ways of packings since it is close to the corner as in the line gadget in Figure 11(a). The top and bottom dotted squares also have two ways since they are close to a three-forked road in the gadget. Next, the gadget (or the huge square) contains 127 squares. Thus, the gadget is covered by 43 trominoes, and two squares remain. The remaining squares can cover two out of the four dotted squares. Carefully tracing all possible packings, we can check that both of the top and bottom dotted squares cannot be covered at the same time, and both of the right and left dotted squares cannot be covered at the same time. Therefore, we only have four valid packings shown in Figure 12.

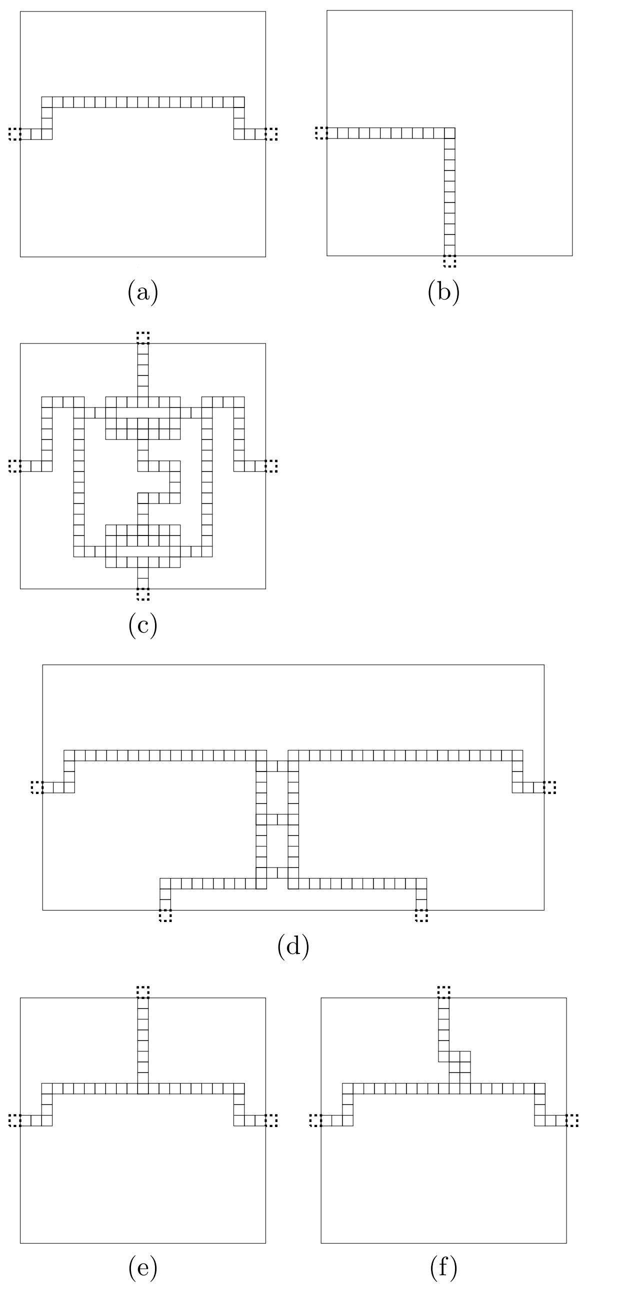


Figure 11: Gadgets for the reduction to 3I-packing.

(d) Duplicator gadget. At the center of the gadget in Figure 11(d), there are three bridges of length 4 joining the right and left parts. Each of the bridges can be packed in two possible ways by putting a tromino left or right. As in the cross gadget, since the number of the squares in the gadget is 106, we can place two squares at the positions of the dotted squares. use two squares at the dotted squares. The corners close to dotted squares force the configuration of trominoes. Contrary to the case of the cross gadget, these conditions allow us to pack I-trominoes in only two ways as shown in Figure 13.

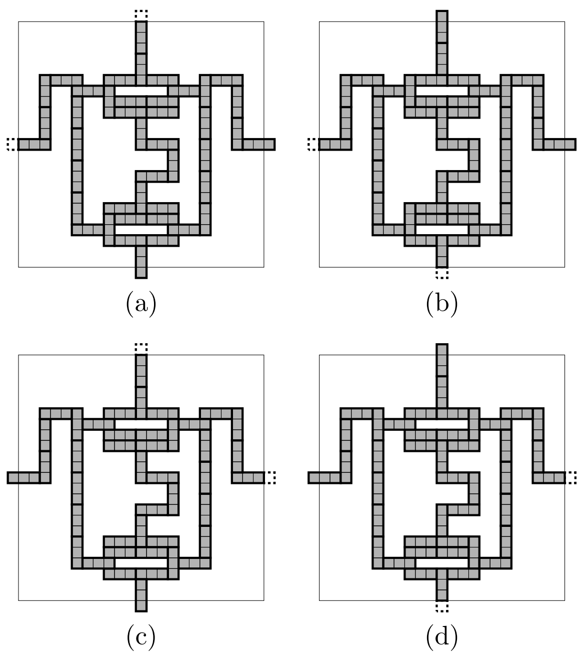


Figure 12: Four ways of packings for a cross gadget.

(f) Negated-clause gadget. In the negated-clause gadget given in Figure 11(f), the center rectangle of size 2×4 is critical. We can pack two trominoes in four ways here, but if they take different heights, we cannot pack trominoes in the horizontal segment between the left and right corners. Therefore, there are only two ways to pack the trominoes into the rectangle. This gives us three valid packings as illustrated in Figure 14.

The two ways of packings for a line gadget correspond to the orientations of the edge. A clause gadget has three ways of packings such that exactly one of the three dotted unit squares gives an inbound orientation. In contrast, a negated-clause gadget has three ways of packings such that exactly one of the three dotted unit squares gives an outbound orientation. Therefore, valid packings for P in 3I-packing have one-to-one correspondence with valid orientations of G_φ in 1-in-3 GO. We thus have the following theorem.

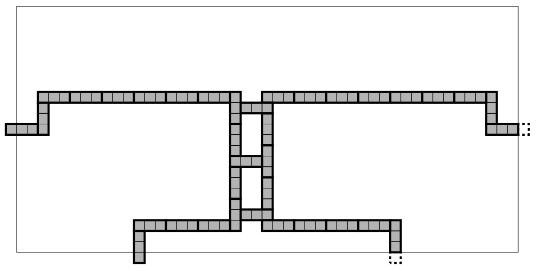
Theorem 3 3I-packing *is NP-complete, #P-complete and ASP-complete.*

3 Concluding Remarks

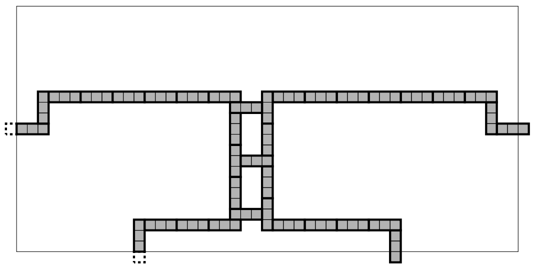
We have studied the computational complexity of packing problems with L-trominoes and I-trominoes. Our results fill the complexity gap between packing dominoes (i.e., 1×2 rectangles) and packing 2×2 squares.

We can extend our results to the following problems.

(1) Tromino-Packing (i.e., a mixture variant of packing trominoes): Given an integer k and a polyomino P,

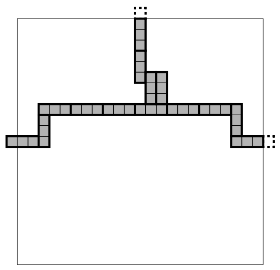


(a)

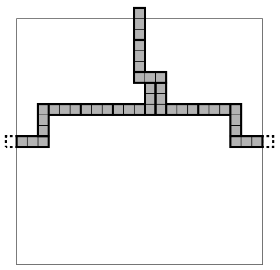


(b)

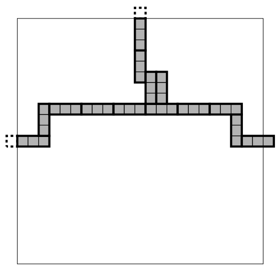
Figure 13: Two ways of packings for a duplicator gadget.



(a)



(b)



(c)

Figure 14: Three ways of packings for a negated-clause gadget.

determine whether k trominoes can be placed into P without overlap. Note that we can use both L-trominoes and I-trominoes in this variant. By constructing new gadgets, similar arguments establish that this variant is NP-complete, #P-complete and ASP-complete.

(2) 3L-Cover, 3I-Cover and Tromino-Cover: Given an integer k and a set S of points in the plane, determine whether k trominoes can cover all points in S. In 3L-COVER and 3I-COVER, we are only allowed to use identical L-trominoes and I-trominoes, respectively. In TROMINO-COVER, we are allowed to use both shapes. Trominoes are allowed to mutually overlap each other.

(3) 3L-Unique-Cover, 3I-Unique-Cover and Tromino-Unique-Cover: Given an integer k and a set S of points in the plane, determine whether k trominoes can uniquely cover all points in S, that is, no overlap of trominoes is allowed.

By converting the gadgets in this paper to the positions of points (and regarding the set of points as the set S), we can easily see the correspondence between the packing and the covering problems. Therefore, all these variants in (2) and (3) are NP-complete, #P-complete and ASP-complete.

References

[1] N. Creignou and M. Hermann, Complexity of generalized satisfiability counting problems. *Information and Computation*, 125, pp. 1–12, 1996.

[2] D. El-Khechen, M. Dulieu, J. Iacono and N. van Omme. Packing 2 × 2 unit squares into grid polygons is NP-complete. *Proc. 21st Canadian Conf. on Comput. Geom.*, pp. 33–36, 2009.

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

[4] M. Gardner. The hypnotic fascination of sliding-block puzzles. *Scientific American*, 210:122–130, 1964. (Also Chapter 7 of *Martin Gardner's Sixth Book of Mathematical Diversions*, University of Chicago Press, Chicago, 1984.)

[5] S. Golomb. *Polyominoes* (2nd edition). Princeton University Press, 1994.

[6] R. A. Hearn and E. D. Demaine. PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation. *Theoretical Computer Science*, 343(1-2):72–96, 2005.

[7] R. A. Hearn and E. D. Demaine. *Games, Puzzles, and Computation.* A K Peters Ltd., 2009.

[8] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Computing*, 2(4):225–231, 1973.

[9] T. J. Schaefer. The complexity of satisfiability problems. *Proc. 10th Ann. ACM Symp. on Theory of Computing*, pp. 216–226, 1978.

[10] T. Yato and T. Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E86-A(5):1052–1060, 2003.

Tiling Polyhedra with Tetrahedra

Andras Bezdek[*] Braxton Carrigan[†]

Abstract

When solving an algorithmic problem involving a polyhedron in $\mathbb{R}^3$, it is common to start by partitioning the given polyhedron into simplier ones. The most common process is called triangulation and it refers to partitioning a polyhedron into tetrahedra in a face-to-face manner. In this paper instead of triangulations we will consider tilings by tetrahedra. In a tiling the tetrahedra are not required to be attached to each other along common faces. We will construct several polyhedra which can not be triangulated but can be tiled by tetrahedra. We will also revisit a nontriangulatable polyhedron of Rambau and a give a new proof for his theorem. Finally we will identify new families of non-tilable, and thus non-triangulable polyhedra.

1 Introduction and Definitions

One of the fundamental approaches found in computational geometry is to break a region into smaller or simpler pieces. What is simple depends on the application. The process of partitioning a closed region into triangles has been abstracted to higher dimensions, yet still bears the name triangulation. One of the classical applications of triangulation is the art gallery theorem which states the fewest number of guards needed to guard a two dimensional polygonal region.

In this paper we will be concerned with the triangulation of polyhedra, in particular identifying polyhedra which cannot be triangulated. We will give five known examples of non-triangulable polyhedra and provide another example to justify a more general type of partitioning which we call tiling by tetrahedra. We will use the general partition to revisit the analogue of example 5, providing a shorter proof. In doing so we will show another family of polyhedra which cannot be tiled by tetrahedra and thus is non-triangulable. Finally we will introduce more families of non-tilable polyhedra and pose a generalization to this family.

Definition 1 *A* ***triangulation*** *of a polytope $P \in \mathbb{R}^d$ is a collection of d-simplices that satisfies the following two properties:*

1. The union of all these simplices equals P. (Union Property)
2. Any pair of these simplices intersect in a common face (possibly empty). (Intersection Property)

[*]Department of Mathematics and Statistics, Auburn University, `bezdean@auburn.edu`

[†]Department of Mathematics, Southern Connecticut State University, `bac0004@auburn.edu`

In this paper, we restrict ourselves to partitions where the vertices of each tetrahedron is a subset of the vertex set of P. For further information on triangulation, we suggest the texts [2], [3], and [5].

We wish to introduce the concept of *tiling by tetrahedra*, which weakens the intersection property of *triangulation*.

Definition 2 *A* ***tiling by tetrahedra*** *of a polyhedron* ***P*** *is a collection of tetrahedra, all of whose vertices are vertices of* ***P****, that satisfies the following two properties:*
1. The union of all these tetrahedra equals ***P****. (Union Property)*
2. The intersection of any two tetrahedra (possibly empty) is a subset of a plane. (Intersection Property)

Remark 1 *Figure 1 is an example of a tiling of the cube which is not a triangulation.*

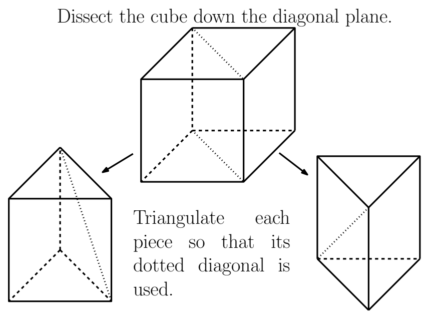

Dissect the cube down the diagonal plane.
Triangulate each piece so that its dotted diagonal is used.

Figure 1: Tiling a cube

2 Non-Triangulable Polyhedra

It was first shown in 1911 by Lennes [4] that not all three-dimensional polyhedra are triangulable. We will provide eight other known examples of non-triangulable polyhedra in this section.

Example 1 (Schönhardt)
A frequently quoted and simple example was given by Schönhardt [10] in 1927. Schönhardt made a non-convex

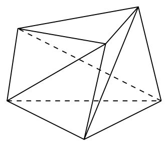


Figure 2: Schönhardt's twisted triangular prism

twisted triangular prism (Figure 2) by rotating the top face of a triangular prism so that a set of cyclic diagonals became edges with interior dihedral angles greater than 180^o.

Claim: Schönhardt's Twisted Triangular Prism cannot be triangulated.

Proof. Every diagonal of the polyhedron lies outside the polyhedron. Therefore any tetrahedron containing four vertices of the twisted triangular prism will contain at least one edge lying outside the polyhedron. □

Example 2 (Bagemihl)

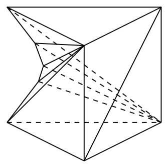


Figure 3: Bagemihl's generalization

In 1948, Bagemihl [1] modified Schönhardt's idea to construct a nonconvex polyhedron on $n \geq 6$ vertices by replacing one of the twisted vertical edges with a concave curve and placing $n-6$ vertices along the curve so that the interior dihedral angles of the edges to these vertices are greater than 180^o.

Claim: Bagemihl's Generalization cannot be triangulated.

Proof. If a triangulation exists, then the top triangular face must be a face of some tetrahedron. For every vertex v, not on the top face, there is a diagonal from v to some vertex on the top face which lies outside the polyhedron. Therefore there is no tetrahedron from the vertex set which has the top face as a boundary lying inside the polyhedron. □

Example 3 (Ruppert and Seidel)
Another method of creating non-triangulable polyhedra with large number of vertices was presented by Ruppert and Seidel [9]. They attached a copy of a non-triangulable polyhedron to another polyhedron. Figure 4 shows a polyhedron where a copy of Schönhardt's non-convex twisted triangular prism, called a niche, is attached to a face of a cube along a base of the twisted triangular prism.

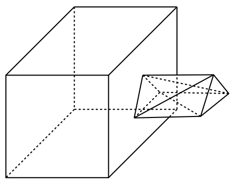


Figure 4: Attaching a niche to a cube

Claim: If a niche is attached properly, the union of the polyhedron and the niche cannot be triangulated.

Proof. It can be arranged that the vertices of the Schöhardt prism which do not lie on the face of the cube do not see any vertex of the cube. Since each diagonal to the non-attached base of the triangular prism lies outside the polyhedron, then there must exist a tetrahedron contained inside the non-convex twisted triangular prism. We know from Example 1 this is not possible, so no set of tetrahedra triangulates the union. □

Example 4 (Thurston et al.)

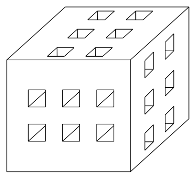


Figure 5: Thurston polyhedron

Figure 5 was attributed to Thurston by Paterson and Yao [7], where 18 non-intersecting square prisms, six from each pair of parallel faces, are removed from the cube. It is important to note that this polyhedron was independently discovered by several people including W. Kuperberg, Holden, and Seidel.

Claim: Thurston's polyhedron cannot be triangulated.

Proof. A point in a polyhedron "sees" another point in the polyhedron if the line segment between the two points is contained inside the polyhedron. We observe that each point of a tetrahedron can see each of the tetrahedron's vertices. If a polyhedron contains a point which does not see at least four non-coplanar vertices of the polyhedron, then it cannot be contained in a tetrahedron from the triangulation. In Thurston's polyhedron, the center of the cube does not see any vertex of the polyhedron, so it is obviously not in the interior of a tetrahedron of a triangulation. □

Example 5: (Rambau)

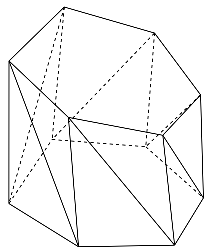


Figure 6: Twisted prism S_{C_6}

Rambau [8] provided another generalization of the Schönhardt twisted triangular prism. To construct the Nonconvex Twisted Prism we will first define a right prism over a convex polygon with n vertices, C_n. Label the vertices of C_n clockwise as $v_1, v_2, ..., v_n$. He defines the *right prism over* C_n as $P_{C_n} = \text{conv}\{(C_n \times \{0\}) \cup (C_n \times \{1\})\}$.

Now pick a point O in the interior of C_n and rotate C_n clockwise about O by ϵ, and label the vertices of $C_n(\epsilon)$, $v_1(\epsilon), v_2(\epsilon), ..., v_n(\epsilon)$, corresponding to the vertices of C_n.The **convex twisted prism over** C_n is $P_{C_n}(\epsilon) = \text{conv}\{(C_n \times \{0\}) \cup (C_n(\epsilon) \times \{1\})\}$.
The **non-convex twisted prism over** C_n (Figure 6) is $S_{C_n} = P_{C_n}(\epsilon)$ - conv$\{(v_i,0),(v_{i+1},0),(v_i(\epsilon),1),(v_{i+1}(\epsilon),1)\}$, for all $i \in (1, n)$ taken modulo n.

In [8] Rambau proves:

Theorem 1 *For all $n \geq 3$, no prism P_{C_n} admits a triangulation without new vertices that uses the cyclic diagonals $\{(v_i,0),\ (v_{i+1},1)\}$.*

Which implies

Corollary 2 *For all $n \geq 3$ and all sufficiently small $\epsilon > 0$, the non-convex twisted prism S_{C_n} cannot be triangulated without new vertices.*

The proof of Theorem 1 is too long to discuss here, but we will provide a shorter proof in the following section for Corollary 2.

3 Tilling by Tetrahedra

Notice that Rambau's results do not imply that S_{C_n} cannot be tiled with tetrahedra. Rambau uses Theorem 1 to conclude that no triangulation of S_{C_n} exist, but Figure 1 clearly shows that a tiling by tetrahedra exists for P_{C_4}, which is not a triangulation. Furthermore, this shows that there exists such a tiling which uses the cyclic diagonals of the cube. We prove that:

Theorem 3 *There exist a polyhedron which is not triangulable, but can be tiled by tetrahedra.*

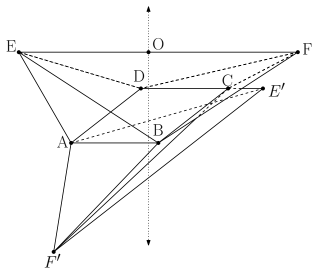


Figure 7: A non-triangulable polyhedron which can be tiled with tetrahedra

Proof. Example 6 will provide this result.
Example 6
Start with a horizontal unit square Q. Let A, B, C and D be the vertices of Q in counterclockwise order when we look down at the square from above. Choose the point O over Q at unit distance from its vertices. Next add to this arrangement a segment EF, whose midpoint is O, has length 4, and which is parallel to AB (assume E is closer to A than to B). Rotate this segment clockwise (i.e. opposite to the order of the vertices A, B, C and D) around the vertical line through O by a small angle ϵ. Let P be a non-convex polyhedron bounded by Q and by six triangles EAB, EBF, BFC, CDF, EFC, and EDA.

Finally let P' be the image of P under the reflection around the plane of Q followed by a 90° rotation around the vertical line containing O. Label the images of E and F as E' and F' respectively.

First notice that P is triangulable as it is the union of the tetrahedra $EABD, EBDF$ and $DBCF$. Since the same holds for P' we have that the union of P and P' can be tiled by tetrahedra.

Next we show that the union of P and P' is not triangulable. Since neither E nor F can see the vertices E' and F', we have that any triangulation of the union is the union of triangulations of P and P'. The polyhedron P was constructed so that the dihedral angles corresponding to the edges EB and FD are concave, therefore the diagonals AF and EC lie outside of P. It is easy to see that the triangles ABC and ACD cannot be faces of disjoint tetrahedra contained in P, thus diagonal BD must be an edge of at least one tetrahedron in any triangulation of P. A similar argument applied for P' gives that the diagonal AC is an edge of at least one tetrahedron in any triangulation of P'. Thus the union of P and P' is not triangulable. $\square$

Observation 1 *A non-triangulable polyhedron is tilable only if it contains at least four coplanar vertices where no three are incident with a common face. (We wish to thank one of the referees for this helpful observation)*

Since S_{C_n} does not contain 4 coplanar points, for sufficiently small ϵ, where no three are incident with a common face, Remark 1 implies that no tiling exists. However we wish to provide a shorter proof than that of Theorem 1 and provide a more general family of non-tilable polyhedra.

We will look at possible tetrahedra contained inside a polyhedron and determine if any two interior tetrahedra intersect. If two tetrahedra intersect in more than a plane, we can conclude that both tetrahedra cannot be in the tiling.

Lemma 4 *Let two tetrahedra T_O and T_B share an edge e and contain two coplanar faces t_O and t_B respectively on a plane P. If there exists a plane $Q \neq P$ containing e such that the fourth vertex O of T_O is in the open halfplane bounded by Q containing t_B, and such that the fourth vertex B of T_B is in the open halfplane bounded by Q containing t_O, then $T_O \cap T_B$ is a polyhedron, hence the interiors of T_O and T_B are not disjoint.*

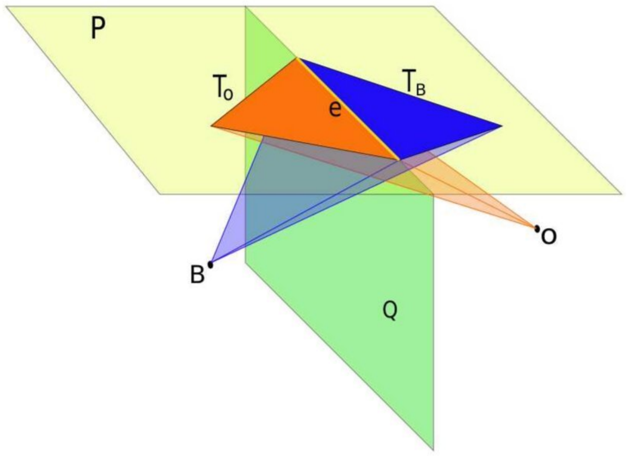


Figure 8: Edge sharing tetrahedra which cross

Lemma 4 can simply be proven by noticing that the interior dihedral angle of T_O, created by the faces t_O and $\{E, O\}$, and the interior dihedral angle of T_B, created by the faces t_B and $\{E, B\}$, sum to greater than 180^o.

Remark 2 *We will use Lemma 4 to say T_O and T_B cannot both be tetrahedra of a tiling by tetrahedra.*

Since each face of a tetrahedron $t \in T$ is a triangle, we say T induces a surface triangulation. Rambau used this observation in proving Corollary 2 by using Theorem 1. We will also use this observation to determine which tetrahedron contains a particular surface triangle as a face.

Definition 3 *An ear of a 2-dimensional triangulation of a polygon P is a triangle with exactly two of its edges belonging to P.*

Theorem 5 *(Meisters [6]) For $n > 3$, every triangulation of a polygon has at least 2 ears.*

It is common to view each triangulation as a tree by letting each triangle be represented by a dual vertex where two dual vertices are adjacent if the corresponding triangles share an edge. In this dual tree each ear is a leaf. We will borrow the terminology of pruning a leaf, to prune ears of a triangulation.

Definition 4 *An ear, E, is pruned by deleting the ear from the triangulation, leaving the edge which was not an edge of P as an edge of $P' = P - E$. In doing so, we delete a vertex of the polygon.*

Example 7:

Define an infinite set of polyhedra B_{C_n} (Figure 10) as follows:

Let the bottom base be a convex polygon on n vertices, C_n, with vertices labeled clockwise as $b_1, b_2, ..., b_n$. Define l_i to be the line containing edge $\overline{b_i b_{i+1}}$ (indices taken modulo n). Now we will call the closed area bounded by the lines l_i, l_{i-1}, and l_{i-2}, which contains $\overline{b_{i-1} b_i}$ but does not contain C_n, region R_i (Figure 9). (Region R_i may be infinite if l_i and l_{i-2} are parallel or intersect on the same side of l_{i-1} as the polygon.)
Now define the upper base as the convex polygon $U_n = \text{conv}\{u_i, u_2, ..., u_n\}$, where $u_i \in R_i$.

Let $B'_{C_n} = \text{conv}\{(C_n \times \{0\}) \cup (U_n \times \{1\})\}$, so that $B_{C_n} = B'_{C_n} - \text{conv}\{(b_i,0),(b_{i+1},0),(u_{i+1},1),(u_{i+2},1)\}$, for all $i \in \{1, 2, \ldots, n\}$ taken modulo n.

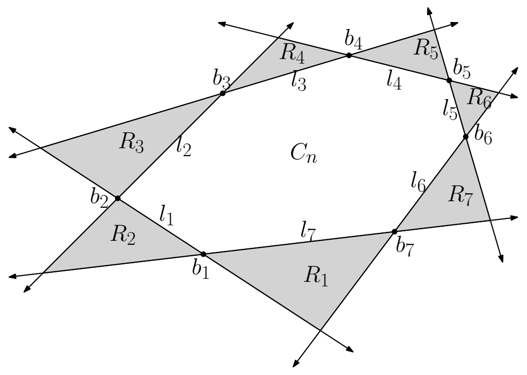


Figure 9: All regions R_i for C_7

Theorem 6 *The non-convex polyhedron B_{C_n} cannot be tiled with tetrahedra.*

Proof. Assume a set of simplices (tetrahedra) S tiles B_{C_n}. The tiling by S induces a triangulation of $(U_n \times$

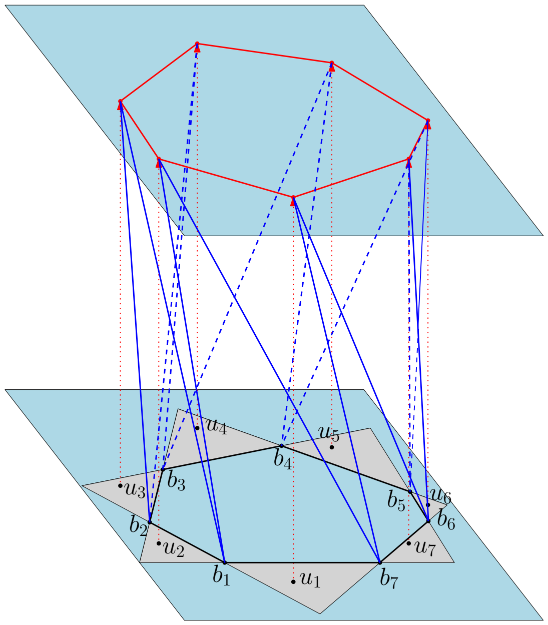


Figure 10: B_{C_7}

$\{1\}$), which we will call T. Now, for every $t \in T$ there exists exactly one $s \in S$ such that t is a face of s. Obviously, the fourth vertex of s must be a vertex of $(C_n \times \{0\})$.

Define a *sub-polygon* to be the convex hull of a subset of the vertices of a polygon. Let P be the set of sub-polygons of U_n such that every edge of a sub-polygon $p \in P$ is an edge of some $t \in T$.

Let e be an edge of p and let t be a triangle of T inside p, having e as an edge. We will say p is *separating* if every point b_i in the open halfplane bounded by the line containing e which does not contain p, is not in a tetrahedron of S with t.

Let $P' \subseteq P$ so that every $p' \in P'$ is separating. P' is not empty since U_n is a separating sub-polygon.

Let a minimal separating sub-polygon be the sub-polygon with the fewest vertices.

Let $m \in P'$ be a minimal separating sub-polygon with n vertices. If $n > 3$, then there is a $t \in T$ which is an ear of m. Let d be the edge of t which is not an edge of m. Observe that there exists a triangle $t' \in T$ which has d as an edge and is contained in m.

Remark 3 *The construction of U_n yields the property that the line containing the diagonal $\overline{u_i u_j}$ (for $i < j$) bounds two open halfplanes such that the halfplane containing the vertices u_k for $i < k < j$ also contains the vertices b_m for $i \le m < j$ and no other vertices from the polygon C_n.*

Let Q be the plane through d, perpendicular to the plane containing U_n. Since m is separating, we can conclude by Lemma 4 that t' cannot be in a tetrahedron with any $(b_i, 0)$ where b_i is in the open halfplane, bounded by the line containing d, containing t. Therefore we can prune t so that $m - t$ is a separating sub-polygon with fewer vertices than m. Thus m is not a minimal separating sub-polygon. Therefore we can conclude that the minimal separating sub-polygon is a triangle.

So there is a $t = \{u_x, u_y, u_z\} \in T$ which is a minimal separating sub-polygon. Since t is separating, for every b_i outside of t, t is not in a tetrahedron with $(b_i, 0)$. By Remark 3, the only vertices which can exist inside t are b_x, b_y, or b_z, but the segments $\overline{(b_i, 0)(u_i, 1)}$ lie outside the polyhedron by the construction of B_{C_n}. Thus there exists a surface triangle which is not the face of a tetrahedron of S. Therefore there is no set of tetrahedra which tiles B_{C_n}. $\square$

A closer look at the proof yields that Remark 3 is the only observation necessary of U_n for the proof. Thus we will define a particular alteration, A_{C_n}, of a prism.

Let C_n be the same convex polygon defined in B_{C_n}. Let $A_n = \text{conv}\{a_1, a_2, ..., a_n\}$, where the line containing the diagonal $\overline{a_i a_j}$ (for $i < j$) bounds two open halfplanes such that the halfplane containing the vertices a_k for $i < k < j$ also contains the vertices b_m for $i \le m < j$ from the polygon C_n. Let $A'_{C_n} = \text{conv}\{(C_n \times \{0\}) \cup (A_n \times \{1\})\}$. The *non-convex altered prism over C_n* is $A_{C_n} = A'_{C_n} -$ conv$\{(b_i,0),(b_{i+1},0),(a_i,1),(a_{i+1},1)\}$, for all $i \in (1, n)$ taken modulo n.

Corollary 7 *For all $n \ge 3$ the non-convex altered prism A_{C_n} cannot be tiled by tetrahedra, hence it also cannot be triangulated.*

Remark 4 *Upon close inspection, it is easy to see that there is a convex polygon C_n where no rotational center yields the observations made in Remark 3. Such an example is provided on the coordinate plane in Figure 11.*

We note that, for small rotations, if the center of rotation lies on a point with a positive or 0 x-coordinate, then the diagonal $\overline{(-1,1)(-3,1-\epsilon)}$ will not satisfy Remark 3. Similarly, if the center of rotation lies on a point with a negative or 0 x-coordinate, then the diagonal $\overline{(1,-1)(3,-1+\epsilon)}$ will not satisfy Remark 3.

Theorem 8 *For all $n \ge 3$ and all sufficiently small $\epsilon > 0$, the non-convex twisted prism S_{C_n} cannot be tiled by tetrahedra without new vertices.*

Proof. It suffices to show that for any C_n, there exists a sufficiently small ϵ such that for any diagonal $\overline{(v_i, 1)(v_j, 1)}$ (for $i < j$) of $C_n(\epsilon)$ there is a plane Q containing the diagonal $\overline{(v_i, 1)(v_j, 1)}$ which bounds two open halfspaces such that the halfspace containing the

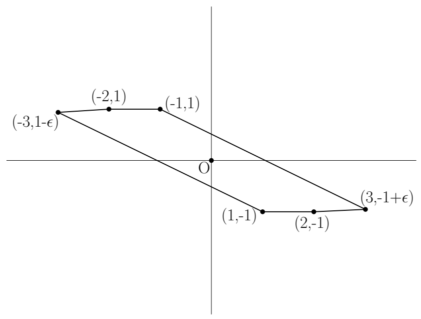


Figure 11: No rotational center

vertices $(v_k, 1)$ for $i < k < j$ also contains the vertices $(v_m, 0)$ for $i \leq m < j$ and no other vertices from the polygon $C_n \times \{0\}$. When constructing $C_n(\epsilon)$ we must consider the planes through each diagonal. Now, for any rotational center O, there is some angle of rotation α_{ij} where the diagonal $\overline{v_i(\alpha_{ij})v_j(\alpha_{ij})}$ lies on a line parallel to the diagonal $\overline{v_{i-1}v_{j-1}}$. Thus, for every S_{C_n} where $0 < \epsilon_{ij} < \alpha_{ij}$ there exists a plane Q satisfying the conditions of Lemma 4 for the diagonal $\overline{(v_i, 1)(v_j, 1)}$. It follows that if we let $\alpha = \min\{\alpha_{ij} | i, j \in (1, 2, 3, ..., n)\}$, then for every ϵ, $0 < \epsilon < \alpha$, S_{C_n} cannot be tiled by tetrahedra. $\square$

Example 8: (Nonconvex Twisted Dodecahedron)

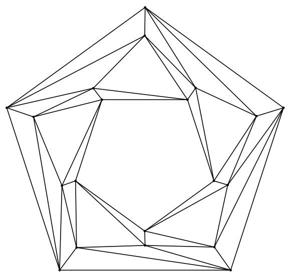


Figure 12: Planar representation of $DH(\epsilon)$

Let two parallel faces of a regular dodecahedron be the Bottom and Top faces. Let l be the line through the center of the two parallel faces. Rotate the Bottom face about l counterclockwise by an angle $\beta \leq \epsilon$, and the Top face about l clockwise by an angle $\tau \leq \epsilon$. The nonconvex twisted dodecahedron $DH(\epsilon)$ (Figure 12) is obtained by taking the convex hull of the 20 points and then removing the convex hull of each set of five points which was the face of the dodecahedron, with the exception of the Top and Bottom faces.

Theorem 9 *For sufficiently small ϵ the nonconvex twisted dodecahedron cannot be tiled by tetrahedra.*

A generalization of Lemma 4 and the argument used for Theorem 6 suffice to show Theorem 9 to be true. Furthermore we believe the previous known techniques would not be able to show this family of polyhedra is non-triangulable.

4 Open Problem

The result for $DH(\epsilon)$ motivates a generalization, as Schönhardt's twisted triangular prism motivated Rambau's generalization.

Notice that the position of the Top and Bottom faces of the regular dodecahedron is the same as the right pentagonal anti-prism. A n-gonal pentaprism PP_n is bounded by two congruent regular n-gonal bases in the same position as the right n-sided anti-prism and $2n$ pentagonal lateral faces, one adjacent to each edge of a base. If δ is the interior dihedral angle of the right n-sided anti-prism, then let the angle between a base and a lateral pentagon be $\delta < \alpha < 180$. A non-convex twisted n-gonal pentaprism $PP_n(\epsilon)$ is created by twisting the Top and Bottom faces of PP_n as in $DH(\epsilon)$.

Remark 5 *$DH = PP_5$ for $\epsilon = \arccos(\frac{-1}{\sqrt{5}})$.*

We leave the reader with this open problem. Is the non-convex twisted PP_n non-tilable by tetrahedra for all $n > 3$?

References

[1] F. Bagemihl, On indecomposable polyhedra, *American Math Monthly* **55**, (1948), 411-413.

[2] J.A. De Loera, J. Rambau, and F. Santos, *Triangulations: structures for algorithms and applications*, Algorithms and Computation in Mathematics, Vol. 25, Springer, (2010).

[3] S. Devadoss and J. O'Rourke, *Discrete and Computational Geometry*, Princeton University Press, (2011).

[4] N.J. Lennes, Theorems on the simple finite polygon and polyhedron, *American Journal of Mathematics* **33**, (1911), 37-62.

[5] J. O'Rourke, *Computational Geometry in C*, ed. 2, Cambridge University Press, (1998).

[6] G.H. Meisters, Polygons have ears, *American Mathematical Monthly*, June/July 1975,(648-651).

[7] M. S. Paterson and F. F. Yao, Binary partitions with applications to hidden-surface removal and solid modeling, *Proc. 5th ACM Symp. Comp. Geom.* (1989) 23-32.

[8] J. Rambau, On a generalization of Schönhardt's polyhedron, *Combinatorial and Computational Geometry* **52**, (2005), 501-516.

[9] J. Ruppert and R. Seidel, On the difficulty of triangulating three-dimensional nonconvex polyhedra, *Discrete Computaional Geometry* **7** issue 3, (1992), 227-253.

[10] E. Schönhardt, Über die Zerlegung von Dreieckspolyedern in Tetraeder, *Mathematics Annalen* **89**, (1927), 309-312.

Point-Set Embedding in Three Dimensions [*]

Henk Meijer[†] Stephen Wismath[‡]

Abstract

Given a graph G with n vertices and m edges, and a set P of n points on a three-dimensional integer grid, the 3D Point-Set Embeddability problem is to determine a (three-dimensional) crossing-free drawing of G with vertices located at P and with edges drawn as poly-lines with bend-points at integer grid points. We solve a variant of the problem in which the points of P lie on a plane. The resulting drawing lies in a bounding box of reasonable volume and uses at most $O(\log m)$ bends per edge.

If a particular point-set P is not specified, we show that the graph G can be drawn crossing-free with at most $O(\log m)$ bends per edge in a volume bounded by $O((n+m)\log m)$. Our construction is asymptotically similar to previously known drawings, however avoids a possibly non-polynomial preprocessing step.

1 Introduction

The two-dimensional (2D) graph drawing literature is extensive. Drawing graphs in three dimensions (3D) has been considered by several authors under a variety of models. One natural model is to draw vertices as points at integer-valued grid points in a 3D Cartesian coordinate system and represent edges as straight line segments between adjacent vertices, with no pair of edges intersecting.

Cohen, Eades, Lin and Ruskey [4] showed that it is possible to draw *any* graph in this model, and indeed the complete graph K_n is drawable within a bounding box of volume $\Theta(n^3)$. Restricted classes of graphs may however be drawn in smaller asymptotic volume. For example, Calamonieri and Sterbini [3] showed that all 2-, 3-, and 4-colourable graphs can be drawn in $O(n^2)$ volume. Pach, Thiele and Tóth [17] showed a volume bound of $\Theta(n^2)$ for r-colourable graphs, where r is a constant. Dujmović, Morin and Wood [9] investigated the connection of bounded tree-width to 3D layouts. For outerplanar graphs, Felsner, Liotta and Wismath [13] described a 3D drawing in $\Theta(n)$ volume. Establishing tight volume bounds for the class of planar graphs remains an open problem. An upper bound of $O(n^{1.5})$ was established by Dujmović and Wood [10]. Recently, di Battista, Frati and Pach [8] improved the volume bound for planar graphs to $O(n\log^{16} n)$.

In two-dimensional graph drawing, the effect of allowing bends in edges has been well studied. For example, Kaufmann and Wiese [15] showed that all planar graphs can be drawn with only two bends per edge and all vertices located on a straight line.

The consequences of allowing bends in 3D has received less attention. The model considered here draws both vertices and bend points of edges at integer grid points. A simple one-bend construction achieving a volume of $O(n\cdot m)$ uses two skew lines – one for the vertices and one for a single bend-point on each edge. Bose, Czyzowicz, Morin, and Wood [1], showed that the number of edges in a graph provides an asymptotic lower bound on the volume regardless of the number of bends permitted, thus establishing $\Omega(n^2)$ as the lower bound on the volume for K_n. This lower bound was explicitly achieved by Dyck, Joevenazzo, Nickle, Wilsdon and Wismath [12] who presented a construction with at most two bends per edge. The upper bound is also a consequence of a more general result of Dujmović and Wood [11]. Morin and Wood [16], presented a one-bend drawing of K_n that achieves $O(n^3/\log^2 n)$ volume and in [5] the gap between this result and the $\Omega(n^2)$ lower bound was narrowed to achieve a one-bend drawing with volume $O(n^{2.5})$.

It is also interesting to consider the volume of classes of graphs when bends are allowed. Dujmović and Wood [11] showed that in general, a volume of $O(n+m\log q)$ is achievable with $O(\log q)$ bends per edge, where q is the queue number of the graph and thus $q \leq n$. A recent result of di Battista et al. [8] on the queue number for planar graphs thus implies a volume of $O(n\log\log n)$ with $O(\log\log n)$ bends per edge for planar graphs. Both of these results implicitly rely on bounding the queue number of the given graph and obtaining an initial ordering of the vertices that achieves the queue layout. It is known that determining the queue number of a graph is in general NP-Complete [14].

In this paper, we describe a three-dimensional drawing technique that is asymptotically competitive with previous volume bounds for some classes of graphs, including planar graphs, at the expense of a relatively large number of bends per edge. For planar graphs, the

[*]Supported in part by the Natural Sciences and Engineering Research Council of Canada. (NSERC).

[†]Roosevelt Academy, Middelburg, the Netherlands h.meijer@roac.nl

[‡]Dept. of Mathematics & Computer Science, U. of Lethbridge, Canada. wismath@uleth.ca

volume of our drawing is bounded by $O(n \log n)$ with at most $O(\log n)$ bends per edge. Previous results were primarily existential, whereas our technique is constructive and does not rely on computing a queue layout of the graph. Our construction achieves the stated bounds independent of the vertex ordering.

Furthermore, our construction can be extended to resolve an interesting three-dimensional point-set embeddability problem. For example, we show that if P is a set of n points given on a plane and in a bounding box of size W by H, then any graph with n vertices and m edges can be drawn crossing-free on P within a bounding box of dimensions $\max(W, m) \times (H+3) \times (2+\log m)$ and with at most $O(\log m)$ bends per edge.

1.1 Point-Set Embedding

Embedding a graph onto a specified point-set in 2D has been considered in various models. We formulate a variant which includes consideration of the resulting area and number of bends per edge.

2DPSE: Given a planar graph G with n vertices, $V = \{v_1, v_2, \ldots v_n\}$, and given a set of n distinct points $P = \{p_1, p_2, \ldots p_n\}$ each with integer coordinates in the plane, can G be drawn crossing-free on P with v_i at p_i and with a number of bends polynomial in n and in an area polynomial in n and the dimension of P?

If the bijection is relaxed so that each vertex v_i is mapped to any point p_j, the embedding is said to be *without mapping*, and if a specific bijection is provided, the embedding is said to be *with mapping*.

We now formulate a version of the 2DPSE problem for three dimensions which also constrains the bends and volume of the resulting drawing.

3DPSE: Given a graph G with n vertices, $V = \{v_1, v_2, \ldots v_n\}$, and given a set of n distinct points $P = \{p_1, p_2, \ldots p_n\}$ each with integer coordinates in three dimensions, can G be drawn crossing-free on P with v_i at p_i and with a number of bends polynomial in n and in an volume polynomial in n and the dimension of P?

This general problem remains open. The existence version of the problem, ignoring bend and volume constraints is resolved in section 4 where we also present a version of the 3DPSE problem that is solvable via a modification of the construction we present in section 2. But here we first review the relevant results from 2D.

In two dimensions, Cabello [2] showed that determining whether there exists a straight-line drawing of planar graph G on P *without mapping* is NP-hard. Pach and Wenger [18] proved that for the *with mapping* version, $O(n^2)$ bends may be required. The Kaufmann and Wiese [15] result establishes that two bends are always sufficient for the *without mapping* version of the problem, however the bend-points that are computed are not required to have integer coordinates and the resulting area appears to be inherently exponential. Di Giacomo et al. [6] investigated a version of the point-set embeddability problem in which some of the edges of the graph are specified to be straight-line segments. They showed that some edges may then require $O(2^n)$ bends. These two results thus motivate the 3DPSE problem in which both the bends and volume are constrained.

2 Definitions and Preliminaries

Given an undirected graph G of n vertices and m edges, a 3D grid drawing of G maps each vertex to a distinct point of $\mathbb{Z}^3$, and each edge of G to a polyline between its associated endpoints. The *bend points* of each edge are also located at distinct integer grid points. No pair of polylines representing edges is permitted to cross except at common endpoints.

The *volume* of such a drawing is typically defined in terms of a smallest bounding box containing the drawing and with sides orthogonal to one of the coordinate axes. If such a box B has width w, length l and height h, then we refer to the *dimensions* of B as $(w+1) \times (l+1) \times (h+1)$ and define the volume of B as $(w+1) \cdot (l+1) \cdot (h+1)$.

The concept of *track drawing* has been used by several authors with slightly different definitions. Here we follow the notation of [7]. Let $G = (V, E)$ be an undirected graph. A *t-track assignment* of G consists of a partition of V into t sets $V_0, \ldots V_{t-1}$, called *tracks* and a total order $\leq_i$ for each set V_i. An *overlap* on track t_i occurs if there is an edge (u, w) and a vertex v with $u <_i v <_i w$. An *X-crossing* occurs if there are two edges (u, v) and (w, z) with $u, w \in V_i$, $v, z \in V_j$, $u <_i w$ and $z <_j v$. A t-track assignment with no overlaps and no X-crossing is called a *t-track layout*.

In a *subdivision* of a graph G, at least one edge (u, v) of G is replaced by a path $u, d_1, d_2, \ldots d_k, v$, with $k \geq 1$. The internal vertices on the path are called *division vertices*.

For a specific ordering of the vertices of a graph, a subset E' forms a *queue* if for each edge $(v_i, v_l) \in E'$ there is no edge $(v_j, v_k) \in E'$ with $i < j < k < l$. I.e. a FIFO invariant holds and no pair of edges *nest*. If, for a specific vertex ordering, the edges of G can be partitioned into q queues, then the partition is called a *queue layout* of G. The *queue number* of a graph is the minimum over all vertex orderings of the minimum cardinality queue layout. Determining the queue number of a graph is in general NP-Hard, however many properties of queue layouts are known – see [10] for an overview of relevant results.

2.1 Perfect Matching Layouts

The technique developed in section 3 places all vertices collinearly and with edges arranged to avoid intersec-

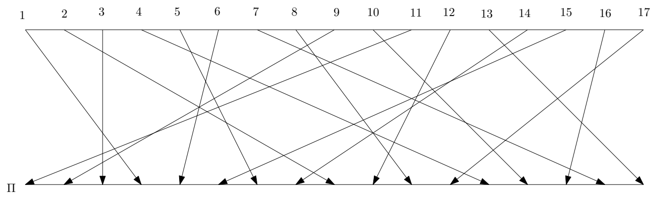


Figure 1: A perfect matching

tions. A critical device used in our construction involves a track layout of a (subdivision of a) perfect matching. See Fig. 1. We redraw the matching on a sequence of tracks to eliminate any X-crossings. Such a track layout can then be drawn in 3D without edge crossings. Our technique is similar in spirit to that used in circuit design, for example the radix-k butterfly layout for FFTs.

Lemma 1 *Let $m = 2^k$, where k is an integer. A perfect matching between two sets of m points can be drawn on $3k + 1$ tracks and with at most $2k$ bends per edge and no X-crossings.*

Proof. Assume the perfect matching is defined by a permutation $\pi(i)$ for $0 \le i < m$. On each track there are potentially points numbered $0, 1, 2, ..., m - 1$ There are $3k$ tracks.

Construction: Every point i on track 0 must be connected (via a polyline) to point $\pi(i)$ on track $3k$. Divide the points on track $3k$ in two equal sized intervals of size $m/2$, and into 4 equal sized intervals of size $m/4$, etc. If $\pi(i)$ is in the first/second interval of size $m/2$ of track $3k$, we place i on the first/second interval of size $m/2$ of track 3. If $\pi(i)$ is in the first/second/third/fourth interval of size $m/4$ of track $3k$, we place i on the first/second/third/fourth interval of size $m/4$ of track 6. In general if $\pi(i)$ is in the h-th interval of size $m/(2^j)$ of track $3k$, we place i on the h-th interval of size $m/(2^j)$ track $3j$.

The above construction yields a track layout of a subdivision of the given matching. Furthermore, the resulting track layout contains no X-crossings. The proof is by induction on k.

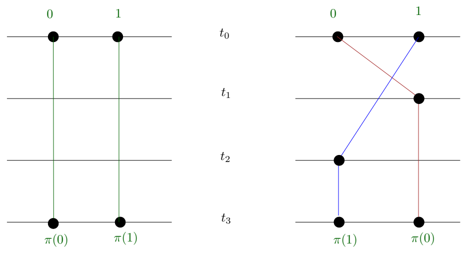


Figure 2: Base case – the two possible matches.

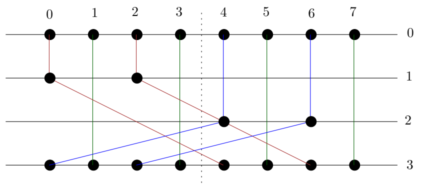


Figure 3: Inductive case. Bend-points shown in black. A simple 3D layout places tracks 0 and 3 in the plane $Z = 0$, track 1 in $Z = -1$, and track 2 in $Z = +1$ which thus eliminates crossings.

Base case, $k = 1$: See Fig. 2. If $\pi(0) = 0$, we connect points 0 and 1 on track 0 to points 0 and 1 on track 3. If $\pi(0) = 1$, we connect point 0 on track 0 via a point on track 1 to point $\pi(0)$ on track 3. We connect point 1 on track 0 via a point on track 2 to point $\pi(1)$ on track 3. Thus the potential X-crossing formed by 0, $\pi(0)$ and 1, $\pi(1)$ is removed.

Inductive case, $k > 1$: See Fig. 3. If ($i < m/2$ and $\pi(i) < m/2$) or ($i \ge m/2$ and $\pi(i) \ge m/2$), we connect point i on track 0 to point i on track 3. If ($i < m/2$ and $\pi(i) \ge m/2$) we connect point i on track 0 to point i on track 1. We connect the points in track 1 to the remaining points in the second half on track 3 so that the points have the same order on both tracks. If ($i \ge m/2$ and $\pi(i) < m/2$) we connect point i on track 0 to point i on track 2. We connect the points in track 2 to the remaining points in the first half on track 3 so that the points have the same order on both tracks. There are no crossings involved within each of these three cases, since the same order is maintained within each case. A crossing occurring between two different cases is not an X-crossing since each case involves different tracks – an intersection can only occur between a pair of edges involving two of tracks 0,3, tracks 1,3, and tracks 2,3. Tracks 1, 2 and 3 contain at most 2 subdivision vertices per edge which correspond to bend-points.

We can now recursively connect the 2^{k-1} points in the first half of track 3 to the first half of track $3k$, using the first halves of tracks $4, 5, 6, ..., 3k$ and at most $2(k - 1)$ bends per edge. And we can recursively connect 2^{k-1} points in the second half of track 3 to the second half of track $3k$, using the second halves of tracks $4, 5, 6, ..., 3k$ and with at most $2(k - 1)$ bends per edge. □

This matching construction will be used in section 3 but it is instructive to note a simple three-dimensional drawing with all points on three parallel planes. Since there are no X-crossings in the resulting layout, a 3D drawing is easily constructed by placing each track t_i as follows:

- if $i = 0 (\text{mod } 3)$ then track t_i is in the plane $Z = 0$
- if $i = 1 (\text{mod } 3)$ then track t_i is in the plane $Z = -1$
- if $i = 2 (\text{mod } 3)$ then track t_i is in the plane $Z = +1$

3 Three-Dimensional Drawing of a Given Graph

Given an arbitrary graph with n vertices and m edges, we outline a technique to draw the graph in three dimensions. As a first step, a track layout of (a subdivision of) the graph is produced, and subsequently a three dimensional drawing is produced.

Each vertex of the graph is placed on track t_0 with coordinates for v_i at $(i, 0)$. A matching is now created based on the edges. For each edge (i, j), $i < j$, a point is placed on track t_1 at consecutive integer x values. The order of the edges is lexicographic – all edges for vertex v_i come before those of v_{i+1}, and the point for edge (i, j) comes before $(i, j + 1)$. Thus there are m points on track t_1; each point representing edge (i, j) is joined by a line segment to v_i on track t_0. Label the points 1,...m. Fig. 4 shows this preliminary step for K_6 but with a modified labeling.

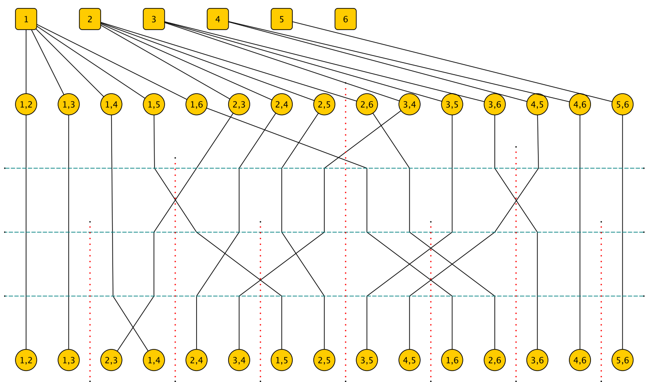


Figure 4: The matching for the 15 edges of K_6. The intervals are 8, 4, and 2. Vertices are labeled 1,2,...6 and the label for edge (i, j) is denoted as i, j. X-crossings in the matching are removed as in Lemma 2.1. Edges from the lowest track to the vertices are not displayed.

Similarly, on track t_*, m points are located as follows. For each edge (i, j) with $i < j$, points for v_j come before those of v_{j+1} and those from v_i are before those of v_{i+1}. The labels for the points on track t_* are determined from the associated edge point on track t_1; if edge (i, j) has label α on track t_1, then it maintains that label on track t_*. Each edge point on track t_* associated with an edge (i, j) is joined by a line segment to v_j on track t_0. The resulting perfect matching between the edge points on the two tracks t_1 and t_* can be processed as in the previous section. The track drawing thus constructed has $3 \log m + 2$ tracks and no X-crossings and no overlaps. The width of the drawing is $\max(n, m)$.

One method to convert the final track drawing into a three-dimensional drawing is to use the technique described in [7] which would automatically yield a drawing of volume $O((n + m) \log^3 m)$, however a more compact drawing can be achieved as follows.

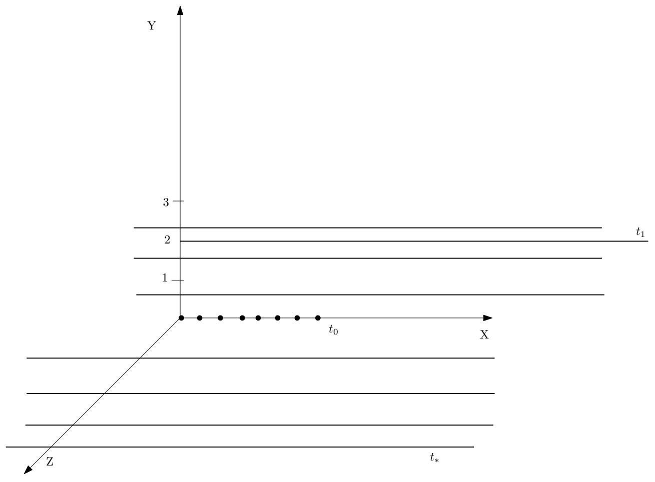


Figure 5: Sketch of track construction – only the first and last sets of tracks for the j groups are shown.

Place track t_0 along the x-axis ($y = 0, z = 0$). Place track t_1 parallel to t_0 but at $y = 2, z = 0$ and then each group j of 3 tracks for $0 \leq j < \log m$ at: $y = 1, z = j+1$; $y = 3, z = j + 1$; $y = 2, z = j + 1$.

The track t_* is placed at $y = 1, z = \log m + 1$ and connected to each point on the final track which is at $y = 2, z = \log m$. Finally, each point on track t_* is joined to the associated vertex on track t_0. Fig. 5 sketches the layout of the tracks.

The volume is $O((n+m) \log m)$. Each point produced in the perfect matching construction may contribute a bend point and there are at most $2 \log m$ such points. The construction outlined above can be summarized in the following theorem.

Theorem 2 *An arbitrary graph with n vertices and m edges can be drawn crossing-free in three dimensions in volume $O((m + n) \log m)$ with at most $O(\log m)$ bends per edge.*

For planar graphs $m \leq 3n - 6$. Hence the following corollary is immediate.

Corollary 3 *A planar graph with n vertices can be drawn crossing-free in three dimensions with a volume of $O(n \log n)$, and with at most $O(\log n)$ bends per edge.*

3.1 Modified Construction

We now modify the previous construction to obtain a drawing with asymptotically similar volume but slightly improved bend complexity.

The previous construction has dimensions $m \times 4 \times (1+\log m)$ and indeed is contained in an infinite wedge-shaped region defined by t_0, and the two half-planes through t_0,t_1 and t_0,t_*. We now partition the edges of G into groups of cardinality n, and draw each group in a separate wedge. Let $z = \lceil m/n \rceil$. Arbitrarily partition the edges of E into $E_1, \cdots, E_z$. Each set of edges E_i is drawn in a separate (infinite) wedge bounded by t_0, and the two half-planes containing t_0 and the tracks t_1^i at $y = 1$, $z = i(1 + \log n)$ and t_*^i at $y = 1$, $z = i(1+\log n)+\log n$. The intersection of these wedges is exactly the track at t_0 containing the vertices of G. The volume of the resulting drawing is $O(n \cdot z \log n)$ which is $O(m \log n)$, and there are $O(\log n)$ bends. Note that these results match asymptotically those of Dujmović, and Wood [10], however our construction is independent of the vertex ordering, whereas their construction requires knowledge of a queue layout.

4 Three Dimensional Point-Set Embedding

Our first result is that the existence version of 3DPSE is always solvable if the volume of the drawing is unconstrained.

Theorem 4 *The complete graph K_n can be drawn crossing-free on any set of n distinct grid points in 3D, with at most 3 bends per edge.*

The proof is existential and omitted in this version. Clearly, if no bends are allowed, K_n may be undrawable on the specified point-set.

Since the 3DPSE problem remains open when the bends and volume are constrained, it is natural to consider constraints on the point-set in this context. We now consider a three dimensional version of the point-set embeddability problem *with mapping.*

3DPSE$_p$: Given a graph G with n vertices, $V = \{v_1, v_2, \ldots v_n\}$, and given a set of n distinct points $P = \{p_1, p_2, \ldots p_n\}$ each with integer coordinates in the XY plane, can G be drawn crossing-free on P with v_i at p_i and with a polynomial number of bends (with integer coordinates) and in a polynomial volume?

Remark: Since our construction does not rely on properties of the graph, we assume the *with mapping* version of the problem. Our solution trivially solves the *without mapping* version of the problem by creating an arbitrary mapping.

Theorem 5 *Let G be an arbitrary graph with n vertices and m edges and let P be a set of n points each with integer coordinates in the XY plane in a bounding box of size $W \times H$, with $W \geq H$. Then G can be drawn crossing-free on P within a bounding box of dimensions $max(W, m) \times (H + 3) \times (2 + \log m)$ and with at most $O(\log m)$ bends per edge.*

Proof. Without loss of generality, we assume the points are labelled in order by X-coordinate, and then by Y-coordinate in the case of points with equal X-coordinate. We assume $X(p_0) = 0$ and $\min_i Y(p_i) = 0$. Then, $W = X(p_n)$ and $H = \max_i Y(p_i)$. See Fig. 6.

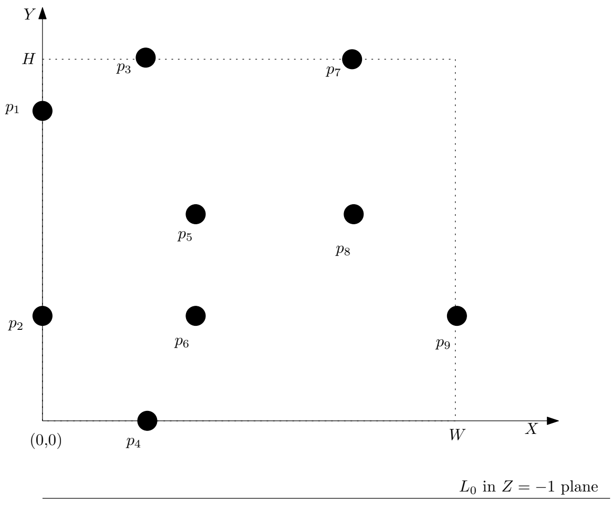


Figure 6: Points specified in the $Z = 0$ plane and projection of the line L_0

The drawing technique described in section 3 can now be almost directly applied, using two lines that host the required matching of the edge points. We construct a line L_0 from (0,-2,-1) to $(m - 1$,-2,-1). The m edges of G are ordered lexicographically. That is, edge (v_i, v_j) precedes edge (v_i, v_{j+1}), and if $i < j$ then all edges from v_i precede those of v_j. More precisely, the following pseudocode specifies the ordering as drawn.

```
e:=0;
for i:=1 to n-1
  for j:= i+1 to n
    if $(v_i, v_j) \in E(G)$ then
    { join $p_i$ to (e,-2,-1);
    e++; }
```

The line L' from (0,-1, $\log m$) to $(m - 1, -1, \log m)$ provides the matching, but for convenience a parallel line L'' is used from $(0,0,1+\log m)$ to $(m-1, 0, 1+\log m)$ and joined to the vertex points as follows.

```
e:=0;
for j:=2 to n
  for i:= 1 to j-1
    if $(v_i, v_j) \in E(G)$ then
    { join $p_j$ to (e,0,1+$\log m$);
    e++; }
```

Each point $(i$,0,$1+\log m)$ is joined to the corresponding point $(i, -1, \log m)$ on L'. The matching construction from L_0 to L' then lies between the planes $Y = -1$ and $Y = -3$.

We now prove there are no crossings in the graph as drawn. Consider the line segments between L_0 and the points P. A pair of segments that crossed would necessarily have two endpoints on L_0, one from each segment. Consider the set of all planes containing L_0 and intersecting a pair of points p_i and p_j. These two points must have the same Y-coordinate, in which case their edges cannot cross since all edges from p_i come strictly before any edges from p_j.

A similar argument holds for the segments between L'' and P. Finally, no segments in the matching intersect the segments in the previous two cases since they are separated in space by a plane. $\square$

A simple consequence of our construction is that K_n can be drawn with the vertices located in an $\sqrt{n} \times \sqrt{n}$ 2D grid and with a volume of $m \cdot \sqrt{n} \cdot \log m$. Since $m = n(n-2)/2$, this volume is $O(n^{2.5} \log n)$.

Similarly, any planar graph can be drawn with vertices in an $\sqrt{n} \times \sqrt{n}$ 2D grid and with a volume of $O(n \log n)$. The aspect ratio of such a drawing is superior to previously published drawings, thus partially addressing the open problem presented in [8].

5 Conclusions and Open Problems

This paper presented a constructive technique to draw arbitrary graphs in three dimensions with low volume but with a non-constant number of bends. In particular for planar graphs the construction results in a volume of $O(n \log n)$ with $O(\log n)$ bends per edge. It remains an open problem to determine if planar graphs can be drawn in $O(n)$ volume with any number of bends.

Our solution to the 3DPSE$_p$ problem requires $O(\log m)$ bends per edge. Can the number of bends per edge be reduced to a constant while preserving a reasonable volume bound? The general 3D point-set embeddability problem in which the specified point-set is not constrained to a plane remains as an interesting open problem if the volume must be constrained.

Acknowledgments: A python implementation of the construction described in section 3 was written by Ian Stewart and Fei Wang and has been incorporated as a plugin to the `GLuskap` software package for drawing and editing graphs in 3D. See `http://www.cs.uleth.ca/~wismath/bends3d` for links and additional pictures.

References

[1] P. Bose, J. Czyzowicz, P. Morin, D. R. Wood. The maximum number of edges in a three-dimensional grid-drawing, *JGAA*, 8(1):21–26, 2004.

[2] S. Cabello. Planar embeddability of the vertices of a graph using a fixed point set is NP-Hard. *JGAA* 10 (2006) 353–363.

[3] T. Calamoneri and A. Sterbini. 3D straight-line grid drawing of 4-colorable graphs, Information Processing Letters 63(2):97–102, 1997.

[4] R. F. Cohen, P. Eades, T. Lin, and F. Ruskey. Three-dimensional graph drawing, *Algorithmica*, 17:199–208, 1997.

[5] O. Devillers, H. Everett, S. Lazard, M. Pentcheva, S. Wismath. Drawing K_n in three dimensions with one bend per edge, *JGAA* 2006, Vol. 10 (2), pp. 287–295.

[6] E. Di Giacomo, W. Didimo, G. Liotta, H. Meijer, S. Wismath. Constrained point-set embeddability of planar graphs. *IJCGA* Vol. 20, (5), pp. 577-600, 2010.

[7] E. Di Giacomo, G. Liotta, H. Meijer, S. Wismath. Volume requirements of 3D upward drawings, *Discrete Mathematics* 309, pp. 1824-1837, 2009

[8] G. Di Battista, F. Frati, J. Pach. On the queue number of planar graphs, *FOCS 2010*, 365-374

[9] V. Dujmović, P. Morin, and D. R. Wood. Layout of graphs with bounded tree-width, *SIAM J. of Computing*, 34(3)553-579, 2005.

[10] V. Dujmović, and D. R. Wood. Three-dimensional grid drawings with sub-quadratic volume, (GD '03), *LNCS* 2912:190–201, Springer-Verlag, 2004.

[11] V. Dujmović, and D. R. Wood. Stacks, queues and tracks: layouts of graph subdivisions, *Discrete Math and Theoretical Computer Science*, 7:155-202, 2005.

[12] B. Dyck, J. Joevenazzo, E. Nickle, J. Wilsdon, and S. Wismath. Drawing K_n in 3D with 2 bends per edge, U. of Lethbridge Tech Rep #CS-01-04: 2–7, Jan 2004.

[13] S. Felsner, G. Liotta, S. Wismath. Straight-line drawings on restricted integer grids in two and three dimensions, *JGAA*, 7(4):363–398, 2003.

[14] L. Heath and A. Rosenberg. Graph layouts using queues, SIAM Journal on Computing, 21(5):927–958, 1992.

[15] M. Kaufmann and R. Wiese. Embedding vertices at points: few bends suffice for planar graphs, *JGAA*, 6(1):115–129, 2002.

[16] P. Morin and D. R. Wood. Three-dimensional 1-bend graph drawings, *JGAA*, 8(3), 2004.

[17] J. Pach, T. Thiele, and G. Tóth. Three-dimensional grid drawings of graphs, (GD '97), *LNCS*, 1353:47–51, Springer-Verlag, 1997.

[18] J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. *Graph and Combinatorics* 17, 2001, 717–728.

Approximating Majority Depth

Dan Chen[*] Pat Morin[†]

Abstract

We consider the problem of approximating the majority depth (Liu and Singh, 1993) of a point q with respect to an n-point set, S, by random sampling. At the heart of this problem is a data structures question: How can we preprocess a set of n lines so that we can quickly test whether a randomly selected vertex in the arrangement of these lines is above or below the median level. We describe a Monte-Carlo data structure for this problem that can be constructed in $O(n \log n)$ time, can answer queries $O((\log n)^{4/3})$ expected time, and answers correctly with high probability.

1 Introduction

A *data depth measure* quantifies the centrality of an individual (a point) with respect to a population (a point set). Depth measures are an important part of multivariate statistics and many have been defined, include Tukey depth [17], Oja depth [13], simplicial depth [10], majority depth [11], and zonoid depth [8]. For an overview of data depth from a statistician's point of view, refer to the survey by Small [15]. For a computational geometer's point of view refer to Aloupis' survey [1].

In this paper, we focus on the bivariate majority depth measure. Let S be a set of n points in $\mathbb{R}^2$. For a pair $x, y \in S$, the *major side* of x, y is the union of the (at most 2) closed halfplanes having both x and y on their boundary that contain at least $n/2$ points of S. The *majority depth* [11, 14] of a point, q, with respect to S, is defined as the number of pairs $x, y \in S$ that have q in their major side.

Under the usual projective duality [9], the set S becomes a set, S^*, of lines; pairs of points in S become vertices in the arrangement, $A(S^*)$, of S^*; and q becomes a line, q^*. The *median-level* of $A(S^*)$ is the closure of the set of points on lines in S that have exactly $\lfloor n/2 \rfloor$ lines of S above them. Then the majority depth of q with respect to S is equal to the number of vertices, x, in $A(S^*)$ such that

1. x is above q^* and x is above the median level; or
2. x is below q^* and x is below the median level.

Chen and Morin [5] present an algorithm for computing majority depth that works in the dual. Their algorithm works by computing the median level, computing the intersections of q^* with the median level, and using fast inversion counting to determine the number, t, of vertices of the arrangement sandwiched between q^* and the median level. The majority depth of q is then equal to $\binom{n}{2} - t$. The running time of this algorithm is within a logarithmic factor of m, the complexity of the median level.

The maximum complexity of the median level of n lines has been the subject of intense study since it was first posed. The current best upper bound is $O(n^{4/3})$, due to Dey [7] and the current best lower bound is $2^{\Omega(\sqrt{\log n})}$, due to Tóth [16]. The median level can be computed in time $O(\min\{m \log n, n^{4/3}\})$ [2, 3]. Thus, the worst-case running time of Chen and Morin's majority depth algorithm is $\omega(n(\log n)^c)$ for any constant c, but no worse than $O(n^{4/3} \log n)$.

It seems difficult for any algorithm that computes the exact majority depth of a point to avoid (at least implicitly) computing the median level of $A(S^*)$. This motivates approximation by random sampling. In particular, one can use the simple technique of sampling vertices of $A(S^*)$ and checking whether

1. each sample lies above or below q^*; and
2. each sample lies above or below the median level of S^*.

In the primal, this is equivalent to taking random pairs of points in S and checking, for each such pair, (x, y), if, (1) the closed upper halfplane, h_{xy}, with x and y on its boundary, contains q and (2) if h_{xy} contains $n/2$ or more points of S.

The former test takes constant time but the latter test leads to a data structuring problem: Preprocess the set S^* so that one can quickly test, for any query point, x, whether x is above or below the median level of $A(S^*)$. (Equivalently, does a query halfplane, h, contain $n/2$ or more points of S.) We know of two immediate solutions to this problem. The first solution is to compute the median level explicitly, in $O(\min\{m \log n, n^{4/3}\})$ time, after which any query can be answered in $O(\log n)$ time by binary search on the x-coordinate of x. The second solution is to construct a half-space range counting

[*]School of Computer Science, Carleton University, dchen4@connect.carleton.ca

[†]School of Computer Science, Carleton University, morin@scs.carleton.ca

structure—a partition tree—in $O(n \log n)$ time that can count the number of points of S in h_{xy} in $O(n^{1/2})$ time [4].

The first solution is not terribly good, since Chen and Morin's algorithm shows that computing the *exact* majority depth of q can be done in time that is within a logarithmic factor of m, the complexity of the median level. (Though if the goal is to preprocess in order to approximate the majority depth for many different points, then this method may be the right choice.)

In this paper, we show that the second solution can be improved considerably, at least when the application is approximating majority depth. In particular, we show that when the query point is a randomly chosen vertex of the arrangement $A(S^*)$, a careful combination of partition trees [4] and ε-approximations [12] can be used to answer queries in $O((\log n)^{4/3})$ expected time. This faster query time means that we can use more random samples which leads to a more accurate approximation.

The remainder of this paper is organized as follows. In Section 2 we review results on range counting and ε-approximations and show how they can be used for approximate range counting. In Section 3 we show how these approximate range counting results can be used to quickly answer queries about whether a random vertex of S^* is above or below the median level of S^*. In Section 4 we briefly mention how all of this applies to the problem of approximating majority depth. Finally, Section 5 concludes with an open problem.

2 Approximate Range Counting

In this section, we consider the problem of approximate range counting. That is, we study algorithms to preprocess S so that, given a closed halfplane h and an integer $i \geq 0$, we can quickly return an integer $r_i(h, S)$ such that

$$||h \cap S| - r_i(h, S)| \leq i \enspace .$$

This data structure is such that queries are faster when the allowable error, i, is larger.

There are no new results in this section. Rather it is a review of two relevant results on range searching and ϵ-approximations that are closely related, but separated by nearly 20 years. The reason we do this is that, without a guide, it can take some effort to gather and assemble the pieces; some of the proofs are existential, some are stated in terms of discrepancy theory, and some are stated in terms of VC-dimension. The reader who already knows all this, or is uninterested in learning it, should skip directly to Lemma 2.

The first tools we need come from a recent result of Chan on optimal partition trees and their application to exact halfspace range counting [4, Theorems 3.2 and 5.3, with help from Theorem 5.2]:

Theorem 1. *Let S be a set of n points in $\mathbb{R}^2$ and let $N \geq n$ be an integer. There exists a data structure that can preprocess S in $O(n \log N)$ expected time so that, with probability at least $1-1/N$, for any query halfplane, h, the data structure can return $|h \cap S|$ in $O(n^{1/2})$ time.*

We say that a halfplane, h, *crosses* a set, X, of points if h neither contains X nor is disjoint from X. The partition tree of Theorem 1 is actually a *binary space partition tree.* Each internal node, u, is a subset of $\mathbb{R}^2$ and the two children of a node are the subsets of u obtained by cutting u with a line. Each leaf, w, in this tree has $|w \cap S| \leq 1$. The $O(n^{1/2})$ query time is obtained by designing this tree so that, with probability at least $1 - 1/N$, there are only $O(n^{1/2})$ nodes crossed by any halfplane.

For a geometric graph $G = (S, E)$, the *crossing number* of G is the maximum, over all halfplanes, h, of the number of edges $uw \in E$ such that h crosses $\{u, w\}$. From Theorem 1 it is easy to derive a spanning tree of S with crossing number $O(n^{1/2})$ using a bottom-up algorithm: Perform a post-order traversal of the partition tree. When processing a node u with children v and w, add an edge to the tree that joins an arbitrary point in $v \cap S$ to an arbitrary point in $w \cap S$. Since a halfplane cannot cross any edge unless it also crosses the node at which the edge was created, this yields the following result [4, Corollary 7.1]:

Theorem 2. *For any n point set, S, and any $N \geq n$, it is possible to compute, in $O(n \log N)$ expected time, a spanning tree, T, of S that, with probability at least $1 - 1/N$, has crossing number $O(n^{1/2})$.*

A spanning tree is not quite what is needed for what follows. Rather, we require a matching of size $\lfloor n/2 \rfloor$. To obtain this, we first convert the tree, T, from Theorem 2 into a path by creating a path, P, that contains the vertices of T in the order they are discovered by a depth-first traversal. It is easy to verify that the crossing number of P is at most twice the crossing number of T. Next, we take every second edge of P to obtain a matching:

Corollary 1. *For any n point set, S, and any $N \geq n$, it is possible to compute, in $O(n \log N)$ expected time, a matching, M, of S of size $\lfloor n/2 \rfloor$ that, with probability at least $1 - 1/N$ has crossing number $O(n^{1/2})$.*

The following argument is due to Matoušek, Welzl and Wernsich [12, Lemma 2.5]. Assume, for simplicity, that n is even and let $S' \subset S$ be obtained by taking exactly one endpoint from each edge in the matching M obtained by Corollary 1. Consider some halfplane h, and let M_h^I be the subset of the edges of M contained in h and let M_h^C be the subset of edges crossed by h. Then

$$|h \cap S| = 2|M_h^I| + |M_h^C| \enspace .$$

In particular,

$$|h \cap S| - |M_h^C| \le 2|h \cap S'| \le |h \cap S| + |M_h^C|$$

Since $|M_h^C| \in O(n^{1/2})$, this is good news in terms of approximate range counting; the set S' is half the size of S, but $2|h \cap S'|$ gives estimate of $|h \cap S|$ that is off by only $O(n^{1/2})$. Next we show that this can be improved considerably with almost no effort.

Rather than choose an arbitrary endpoint of each edge in M to take part in S', we choose each one of the two endpoints at random (and independently of the other $n/2 - 1$ choices). Then, each edge in M_h^C has probability $1/2$ of contributing a point to $h \cap S'$ and each edge in M_h^I contributes exactly one point to $h \cap S'$. Therefore,

$$\mathrm{E}[2|h \cap S'|] = 2\left(1|M_h^I| + \frac{1}{2}|M_h^C|\right) = |h \cap S| \ .$$

That is, $2|h \cap S'|$ is an unbiased estimator of $|h \cap S|$. Even better: the error of this estimator is (2 times) a binomial$(|M_h^C|, 1/2)$ random variable, with $|M_h^C| \in O(n^{1/2})$. Using standard results on the concentration of binomial random variables (i.e., Chernoff Bounds [6]), we immediately obtain:

$$\Pr\{|2|h \cap S'| - |h \cap S|| \ge cn^{1/4}(\log N)^{1/2}\} \le 1/N \ ,$$

for some constant $c > 0$. That is, with probability $1 - 1/N$, $2|h \cap S'|$ estimates $|h \cap S|$ to within an error of $O(n^{1/4}(\log N)^{1/2})$. Putting everything together, we obtain:

Lemma 1. *For any n point set, S, and any $N \ge n$, it is possible to compute, in $O(n \log N)$ expected time, a subset S' of S of size $\lceil n/2 \rceil$ such that, with probability at least $1 - 1/N$, for every halfplane h,*

$$|2|h \cap S'| - |h \cap S|| \in O(n^{1/4}(\log N)^{1/2}) \ .$$

What follows is another argument by Matoušek, Welzl and Wernisch [12, Lemma 2.2]. By repeatedly applying Lemma 1, we obtain a sequence of $O(\log n)$ sets $S_0 \supset S_1 \cdots \supset S_r$, $S_0 = S$ and $|S_j| = \lceil n/2^j \rceil$. For $j \ge 1$, the set S_j can be computed from S_{j-1} in $O(2^{-j} n \log N)$ time and has the property that, with probability at least $1 - 1/N$, for every halfplane h,

$$\left|2^j |h \cap S_j| - |h \cap S|\right| \in O(2^{3j/4} n^{1/4} (\log N)^{1/2}) \ . \quad (1)$$

At this point, we have come full circle. We store each of the sets $S_0, \ldots, S_r$ in an optimal partition tree (Theorem 1) so that we can do range counting queries on each set S_i in $O(|S_i|^{1/2})$ time. This (finally) gives the result we need on approximate range counting:

Lemma 2. *Given any set S of n points in $\mathbb{R}^2$ and any $N \ge n$, there exists a data structure that can be constructed in $O(n \log N)$ expected time and, with probability at least $1 - 1/N$, can, for any halfspace h and any integer $i \in \{0, \ldots, n\}$, return a number $r_i(h, S)$ such that*

$$||h \cap S| - r_i(h, S)| \le i \ .$$

Such a query takes $O(\min\{n^{1/2}, (n/i)^{2/3}(\log N)^{1/3}\})$ expected time.

Proof. The data structure is a sequence of optimal partition trees on the sets $S_0, \ldots, S_r$. All of these structures can be computed in $O(n \log N)$ time, since $|S_0| = n$ and the size of each subsequent set decreases by a factor of 2.

To answer a query, (h, i), we proceed as follows: If $i \le n^{1/4}$, then we perform exact range counting on the set $S_0 = S$ in $O(n^{1/2})$ time to return the value $|h \cap S|$. Otherwise, we perform range counting on the set S_j where j is the largest value that satisfies

$$C 2^{3j/4} n^{1/4} (\log N)^{1/2} \le i \ ,$$

where the constant C depends on the constant in the big-Oh notation in (1). This means $|S_j| = O((n/i)^{4/3}(\log N)^{2/3}))$ and the query takes expected time

$$O(|S_j|^{1/2}) = O((n/i)^{2/3}(\log N)^{1/3}) \ ,$$

as required. □

Our main application of Lemma 2 is a test that checks whether a halfspace, h, contains $n/2$ or more points of S.

Lemma 3. *Given any set S of n points in $\mathbb{R}^2$ and any $N \ge n$, there exists a data structure that can be constructed in $O(n \log N)$ expected time and, with probability at least $1 - 1/N$, can, for any halfspace h determine if $|h \cap S| \ge n/2$. Such a query takes expected time*

$$Q(i) = \begin{cases} O(n^{1/2}) & \text{for } 0 \le i \le n^{1/4} \\ O((n/i)^{2/3}(\log N)^{1/3}) & \text{otherwise,} \end{cases}$$

where $i = ||h \cap S| - n/2|$.

Proof. As preprocessing, we construct the data structure of Lemma 2. To perform a query, we perform a sequence of queries (h, i_j), for $j = 0, 1, 2, \ldots$, where $i_j = n/2^j$. The jth such query takes $O(2^{2j/3}(\log N)^{1/3})$ time and the question, “is $|h \cap S| \ge n/2$?” is resolved once $n/2^j < i/2$. Since the cost of successive queries is exponentially increasing, this final query takes time $O(\min\{n^{1/2}, (n/i)^{2/3}(\log N)^{1/3}\})$ and dominates the total query time. □

3 Side of Median Level Testing

We are now ready to tackle the main problem that comes up in trying to estimate majority depth by random sampling: Given a range pair of points $x, y \in S$, determine if there are more than $n/2$ points in the upper halfspace, h_{xy}, whose boundary is the line through x and y. In this section, though, it will be more natural to work in the dual setting. Here the question becomes: Given a random vertex, x, of $A(S^*)$, determine whether x is above or below the median level of S^*. The data structure in Lemma 3 answers these queries in time $O((n/i)^{2/3}(\log N)^{1/3})$ when the vertex x is on the $n/2 - i$ or $n/2 + i$ level.

Before proving our main theorem, we recall a result of Dey [7, Theorem 4.2] about the maximum complexity of a sequence of levels.

Lemma 4. *Let L be any set of n lines and let s be the number of vertices of $A(L)$ that are on levels $k, k+1, \ldots, k+j$. Then, $s \in O(nk^{1/3}j^{2/3})$.*

We are interested in the special case of Lemma 4 where $k = n/2 - i$ and $j = 2i$:

Corollary 2. *Let L be any set of n lines. Then, for any $i \in \{1, \ldots, n/2\}$ the maximum total number of vertices of $A(L)$ whose level is in $\{n/2 - i, \ldots, n/2 + i\}$ is $O(n^{4/3}i^{2/3})$.*

Corollary 2 is useful because it gives bounds on the distribution of the level of a randomly chosen vertex of $A(S^*)$.

Theorem 3. *Given any set, L, of n lines and any $c > 0$, there exists a data structure that can test if a point x is above or below the median level of L. For any constant, c, this structure can be made to have the following properties:*

1. *It can be constructed in $O(n \log n)$ expected time and uses $O(n)$ space;*
2. *with probability at least $1 - n^{-c}$, it answers correctly for all possible queries; and*
3. *when given a random vertex of $A(L)$ as a query, the expected query time is $O((\log n)^{4/3})$.*

Proof. The data structure is, of course, the data structure of Lemma 3 with $N = n^c$. Let n_i be the number of vertices of $A(L)$ on levels $n/2 - i$ and $n/2 + i$. Then the expected query time of this data structure is at most

$$F(n_0, \ldots, n_{n/2}) = \frac{1}{\binom{n}{2}} \sum_{i=0}^{n/2} n_i Q(i) \ , \tag{2}$$

where, for sufficiently large n, $Q(i)$ is upper-bounded by

$$Q(i) \le \begin{cases} \beta n^{1/2} & \text{if } 0 \le i \le n^{1/4} \\ \beta (n/i)^{2/3} (\log N)^{1/3} & \text{otherwise.} \end{cases}$$

for some constant $\beta > 0$. Our goal, therefore, is to upper-bound $F(n_0, \ldots, n_{n/2})$ subject to Dey's constraints (Lemma 4):

$$\sum_{i=0}^{j} n_i \le \gamma n^{4/3} j^{2/3}$$

for some constant $\gamma > 0$ and all $j \in \{0, \ldots, n/2\}$.

Working in our favour is that $Q(i) \ge Q(i')$ for all $i \le i'$. This implies that, to obtain an upper bound on $F(n_0, \ldots, n_{n/2})$, we can set

$$\sum_{i=0}^{j} n_i = \begin{cases} \gamma n^{4/3} & \text{if } j = 0 \\ \gamma n^{4/3} j^{2/3} & \text{otherwise} \end{cases} \tag{3}$$

for all $j \in \{0, \ldots, n/2\}$. To see why this is so, suppose we have a sequence $S = n_0, \ldots, n_{n/2}$ that satisfies Dey's constraints but for which $\sum_{i=0}^{j} n_i < \gamma n^{4/3} j^{2/3}$ for some index j. If $j = n/2$ then we can obviously increase the value of n_j, still satisfy Dey's constraints and increase the value of $F(n_0, \ldots, n_j)$. Otherwise ($j \in \{0, \ldots, n/2 - 1\}$), the sequence

$$S' = n_0, \ldots, n_j + \delta, n_{j+1} - \delta, \ldots, n_{n/2} \ ,$$

where $\Delta = \gamma n^{4/3} j^{2/3} - \sum_{i=0}^{j} n_i$, also satisfies Dey's constraints. Furthermore,

$$F(S') - F(S) = \Delta Q(j) - \Delta Q(j+1) \ge 0 \ ,$$

so $F(S') \ge F(S)$. Repeatedly applying this type of modification (or using induction) shows that the sequence $S = n_0, \ldots, n_{n/2}$ that satisifies (3) is a sequence that maximizes $F(S)$.

Finally, we can bound the sequence that satisfies (3) by differentiating $\gamma n^{4/3} j^{2/3}$ with respect to j. This yields $n_i \in O(n^{4/3}/i^{1/3})$ for all $i \in \{1, \ldots, n/2\}$. Plugging this back into (2) yields

$$\begin{aligned} & F(n_0, \ldots, n_{n/2}) && (4) \\ & \le \frac{1}{\binom{n}{2}} \left(O(n^{4/3} Q(0)) + \sum_{i=1}^{n/2} O(n^{4/3} Q(i)/i^{1/3}) \right) \\ & \le o(1) \\ & \qquad + \frac{1}{\binom{n}{2}} \sum_{i=1}^{n^{1/4}} O(n^{4/3} n^{1/2}/i^{1/3}) && (5) \\ & \qquad + \frac{1}{\binom{n}{2}} \sum_{i=n^{1/4}+1}^{n/2} O(n^{4/3}(n/i)^{2/3}(\log N)^{1/3}/i^{1/3}) \\ & && (6) \end{aligned}$$

Recall that $\int_1^n i^{-1/3}\, \mathrm{d}i = \frac{3}{2}(n^{2/3} - 1)$. Using this integral to bound the sum in (5) allows us to just squeak by:

$$\begin{aligned}(5) &= \frac{1}{\binom{n}{2}}\sum_{i=1}^{n^{1/4}} O(n^{4/3}n^{1/2}/i^{1/3}) \\ &= \frac{1}{\binom{n}{2}}O(n^{4/3}n^{1/2}(n^{1/4})^{2/3}) \quad \text{(bounding by integral)} \\ &= O(1)\end{aligned}$$

We are not so lucky with the sum in (6), which ends up being harmonic:

$$\begin{aligned}(6) &= \frac{1}{\binom{n}{2}}\sum_{i=n^{1/4}+1}^{n/2} O(n^{4/3}(n/i)^{2/3}(\log N)^{1/3}/i^{1/3}) \\ &= \sum_{i=n^{1/4}+1}^{n/2} O((\log N)^{1/3}/i) \\ &= O((\log n)(\log N)^{1/3}) \quad (\text{since } \textstyle\sum_{i=1}^{n} 1/i = O(\log n)) \\ &= O((\log n)^{4/3}) \ ,\end{aligned}$$

since $N = n^c$ and c is constant. To summarize, the expected running time of the query algorithm is at most

$$F(n_0, \dots, n_{n/2}) \le o(1) + (5) + (6) = O((\log n)^{4/3}) \ . \ \square$$

4 Estimating Majority Depth

Finally, we return to our application, namely estimating majority depth.

Theorem 4. *Given a set S of n points in $\mathbb{R}^2$ and any constant $c > 0$, there exists a data structure that can preprocess S in $O(n \log n)$ expected time such that, for any point q, the data structure can compute, in $O(r(\log n)^{4/3})$ expected time, a value $d'(q, S)$ such that*

$$\Pr\left\{\frac{|d'(q,S) - d(q,S)|}{d(q,S)} \ge \varepsilon\right\} \le \exp\left(-\Omega\left(\varepsilon^2 rp\right)\right) + n^{-c} \ ,$$

where $d(q, S)$ is the majority depth of p with respect to S and $p = d(q, S)/\binom{n}{2}$ is the normalized majority depth of q.

Proof. The data structure is the one described in Theorem 3. Let $p = d(q, S)/\binom{n}{2}$. Select r random vertices of $A(S^*)$ (by taking random pairs of lines in S^*) and, for each sample, test if it contributes to $d(q, S)$. This yields a count $r' \le r$ where

$$\mathrm{E}[r'] = rp \ .$$

We then return the value $d'(q, S) = (r'/r)\binom{n}{2}$, so that $\mathrm{E}[d'(q, S)] = d(q, S)$, as required.

To prove the error bound, we use the fact that r' is a binomial(p, r) random variable. Applying Chernoff Bounds [6] on r' yields:

$$\Pr\{|r' - rp| \ge \varepsilon rp\} \le \exp(-\Omega(\varepsilon^2 rp)) \ .$$

Finally, the algorithm may fail not because of badly chosen samples, but rather, because the data structure of Theorem 3 fails. The probability that this happens is at most n^{-c}. Therefore, the overall result follows from the union bound. $\square$

5 Conclusions

Although the estimation of majority depth is the original motivation for studying this problem, the underlying question of the tradeoffs involved in preprocessing for testing whether a point is above or below the median level seems a fundamental question that is still far from answered. In particular, we have no good answer to the following question:

Open Problem 1. What is the fastest linear-space data structure for testing if an arbitrary query point is above or below the median level of a set of n lines?

To the best of our knowledge, the current state of the art is partition trees, which can only answer these queries in $O(n^{1/2})$ time.

References

[1] G. Aloupis. Geometric measures of data depth. In R.Liu, R.Serfling, and D.Souvaine, editors, *Data Depth: Robust Multivariate Analysis, Computational Geometry and Applications*, volume 72 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 147–158. American Mathematical Society, 2006.

[2] G. S. Brodal and R. Jacob. Dynamic planar convex hull. In *FOCS*, pages 617–626. IEEE Computer Society, 2002.

[3] T. M. Chan. Remarks on k-level algorithms in the plane. Manuscript, 1999.

[4] T. M. Chan. Optimal partition trees. *Discrete & Computational Geometry*, 47(4):661–690, 2012.

[5] D. Chen and P. Morin. Algorithms for bivariate majority depth. In *Proceedings of the 23rd Canadian Conference on Computational Geometry (CCCG'11)*, pages 425–430, 2011.

[6] H. Chernoff. A measure of the asymptotic efficient of tests of a hypothesis based on the sum of observations. *Annals of Mathematical Statistics*, 23:493–507, 1952.

[7] T. K. Dey. Improved bounds for planar k-sets and related problems. *Discrete & Computational Geometry*, 19(3):373–382, 1998.

[8] R. Dyckerhoff, G. Koshevoy, and K. Mosler. Zonoid data depth: Theory and computation. In A. Prat, editor, *COMPSTAT 1996 - Proceedings in Computational*

Statistics, pages 235–240. Physica-Verlag, Heidelberg, August 1996.

[9] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Heidelberg, Germany, 1997.

[10] R. Liu. On a notion of data depth based on random simplices. *Annals of Statistics*, 18(1):405–414, 1990.

[11] R. Liu and K. Singh. A quality index based on data depth and multivariate rank tests. *Journal of the American Statistical Association*, 88(421):252–260, 1993.

[12] J. Matoušek, E. Welzl, and L. Wernisch. Discrepancy and approximations for bounded VC-dimension. *Combinatorica*, 13:455–466, 1993.

[13] H. Oja. Descriptive statistics for multivariate distributions. *Statistics and Probability Letters*, 1(6):327–332, 1983.

[14] K. Singh. A notion of majority depth. Technical report, Department of Statistics, Rutgers University, 1991.

[15] C. Small. A survey of multidimensional medians. *International Statistical Review*, 58(3):263–277, 1990.

[16] G. Tóth. Point sets with many k-sets. In *Symposium on Computational Geometry*, pages 37–42, 2000.

[17] J. W. Tukey. Mathematics and the picturing of data. In Ralph D. James, editor, *Proceedings of the International Congress of Mathematicians*, volume 2, pages 523–531, Vancouver Canada, August 1974.

Flexible Crystal Frameworks

Ciprian S. Borcea[1]*   Ileana Streinu[2]*

Abstract

Building upon and complementing recent results on the rigidity theory of periodic bar-and-joint frameworks, this paper studies tetrahedral structures modeled on specific crystalline materials: quartz, cristobalite and tridymite. The general theory predicts at least three infinitesimal degrees-of-freedom. Here, we investigate the actual deformations of these structures. We show that quartz and cristobalite have smooth three-dimensional configuration spaces, but ideal high tridymite is a singular configuration with a six-dimensional tangent space. The topology around this singularity is explicitly described.

1 Introduction

Motivated by questions arising in mathematical crystallography and computational materials science, we present in this paper specific geometric applications of the general theory of rigidity and flexibility for periodic frameworks developed in our recent papers [1, 4, 5, 3].

Molecules as mechanical frameworks. At a certain approximation level, many molecules can be modeled as mechanical frameworks, with rigid, fixed-length bonds between particular pairs of atoms and fixed-angles between particular adjacent bonds. This opens the possibility of using techniques from rigidity theory in molecular flexibility analysis. Considerations related to *framework flexibility* appear already in the early 20th century structural investigations based on X-ray crystallography, see e.g. [6, 8, 16]. Geometric models of deforming frameworks have been used in studying *displacive phase transitions* in materials [7].

Rigidity and flexibility of crystalline materials. Most of the molecular flexibility studies rely on computationally intensive physics-based simulations or simplified, kinematics-based methods [20, 10]. For large molecules, and especially for crystalline materials, these approaches are not only prohibitively expensive but also numerically imprecise. Faster approaches for degree-of-freedom counting and rigid component calculations are known for mechanical frameworks characterized by theorems of Maxwell-Laman type. For the infinite, periodic structures relevant to crystallography, an adequate *generic* rigidity theoretical formulation has been proposed only recently [1], leading to a combinatorial treatment [4, 5] and efficient algorithms.

Generic and non-generic frameworks. Maxwell-Laman theorems are rare and difficult to obtain (see [9, 17] and the references given there), yet they are the starting point of any research on computationally tractable rigidity and flexibility studies of mechanical structures. They provide generic, *combinatorial characterizations* in graph-theoretical terms, for those structures which are minimally rigid for *almost all possible geometric realizations.* For a measure zero set of non-generic situations, such a theorem will not hold. The theory also predicts flexibility and counts infinitesimal degrees of freedom in generic situations. While deciding rigidity and flexibility for generic frameworks is a tractable problem in most of the situations where Maxwell-Laman theorems have been found (see [14]), the non-generic cases remain elusive.

Results. In this paper we apply our theory of periodic bar-and-joint frameworks to tetrahedral crystal structures modeled on quartz, cristobalite and tridymite. We obtain topological descriptions of their configuration spaces.

2 Generic rigidity for Periodic frameworks

To put in perspective the present results, we remind the reader the classical combinatorial characterization of rigidity, in terms of graph sparsity. Then, we give a brief overview of our recent theorems, characterizing generic periodic rigidity in arbitrary dimensions and the flexibility of frameworks made from vertex-sharing simplices, as well as the algorithmic implications.

Maxwell-Laman sparsity conditions. The theory of finite frameworks goes back almost 150 years to Maxwell [15], who identified a *sparsity condition* to be necessary for minimal rigidity of bar-and-joint frameworks: in dimension d, for any subset of $d \leq n' \leq |V|$ vertices, the underlying graph (with a node for each joint and an edge for each bar) should span at most $dn' - \binom{d+1}{2}$ edges, with equality for the whole set of $n = |V|$ vertices. The sufficiency of this condition for generic frameworks in dimension two was proven over 100 years later (Laman's

*Research of the authors funded by NSF and DARPA. 1. Department of Mathematics, Rider University, Lawrenceville, New Jersey, USA. borcea@rider.edu; 2. Computer Science Department, Smith College, Northampton, Massachusetts, USA. streinu@cs.smith.edu

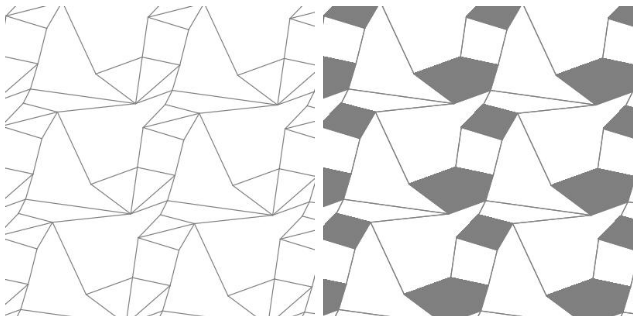


Figure 1: Left: A 2D periodic bar-and-joint framework containing smaller rigid components. Right: the same framework, viewed as a body-and-bar framework; the rigid components form the bodies, and the remaining bars connect distinct bodies.

theorem [12]), and is known to fail in higher dimensions. The problem of completing the combinatorial characterization for bar-and-joint frameworks in arbitrary dimensions remains an elusive open question. However, some restricted classes of finite frameworks have been shown to have similar Maxwell-Laman counting characterizations: body-and-bar and body-and-hinge frameworks [18, 19], and panel-and-hinge frameworks [11].

Maxwell-Laman sparsity for periodic frameworks. The connection pattern of a periodic bar-and-joint framework determines an infinite graph $G = (V, E)$. Periodicity, or more precisely d-periodicity (where d represents the dimension of the ambient space in which the graph is realized geometrically) requires a free Abelian automorphism subgroup $\Gamma \subset Aut(G)$ of rank d. We work under the assumption that the quotient graph G/Γ has a finite number n of vertex orbits and a finite number m of edge orbits. The problem of characterizing periodic frameworks is substantially different from the finite case, and the generic periodic bar-and-joint frameworks have been characterized in all dimensions by a Maxwell-sparsity condition *on the quotient graph*. However, a quotient graph may correspond to several periodic frameworks, called “liftings”. The following result gives, therefore, a necessary condition for rigidity which is also sufficient in almost all the situations (i.e. except for a measure-zero set of possibilities).

Theorem 1 *([4]) Let (G, Γ) be a d-periodic graph. Let n and m denote the number of vertices, respectively edges of the graph G modulo the periodicity group Γ.*

If (G, Γ) is minimally rigid, then $m = dn + \binom{d}{2}$ and the quotient graph G/Γ contains a subgraph with $dn - d$ edges on the n vertices, which is $(dn - d)$-sparse.

Conversely, if $m = dn + \binom{d}{2}$ and the quotient graph G/Γ contains a subgraph with $dn-d$ edges on the n vertices, which is $(dn - d)$-sparse, then a generic lifting of the edges yields a minimally rigid d-periodic graph, that is, a generic quotient equivalent of (G, Γ) is minimally rigid.

As a consequence, there are efficient (pebble game) algorithms for deciding generic periodic bar-and-joint rigidity [13].

As we said, a quotient graph may correspond to several periodic frameworks. Distinguishing among them the truly rigid ones remains a difficult problem. Sometimes, substructures in the infinite graph can be identified from the outset as being rigid; see Fig. 1 for an example in 2D. The quotient graph loses this information. We overcome this problem if we work with more specific types of frameworks, such as periodic body-and-bar, body-and-hinge, body-and-pin, etc. They all appear as special cases where *additional algebraic dependencies* are present. Such cases are not guaranteed to be generic *a priori*, and even getting a necessary sparsity condition (something that was trivial in the finite case) is not easy. A recent result along these lines is [5], which covers many situations occurring in molecular frameworks (body-and-bar, body-and-hinge, mixed plate-and-bar), but not the vertex-sharing polyhedra of this paper. Indeed, characterizing generic body-and-pin structures remains an open question, in both the finite and periodic case.

We have further proven that:

Theorem 2 *([1]) A periodic framework in R^d consisting in vertex-sharing simplices has at least $\binom{d}{2}$ infinitesimal degrees-of-freedom (flexes). However, rigid examples can be constructed.*

For the structures studied in this paper, all of which are vertex-sharing tetrahedra, this theorem implies the existence of at least 3 infinitesimal flexes. However, the existence of infinitesimal flexes does not imply a configuration space of dimension three; in fact, in [1] we construct an explicit rigid example.

With these preliminaries in mind, one can see that our results (presented next) confirm the predictions of the generic theory, but do not follow directly from these theorems; indeed, they require different proof techniques, analyzing the entire configuration space (via appropriate parametrizations), rather than just the infinitesimal behavior.

3 The quartz framework

The ideal quartz structure considered here is built from congruent regular tetrahedra. Quartz is made from two types of atoms, oxygen and silicon. Oxygen atoms correspond to the vertices of the tetrahedra. Each oxygen is shared by two tetrahedra and allows for their relative rotations. Silicon atoms are placed at the centers of the tetrahedra, and are rigidly attached to the oxygens. We aim at examining all the geometric configurations of the periodic framework, without concern for self-collision or any other prohibition of a physical nature.

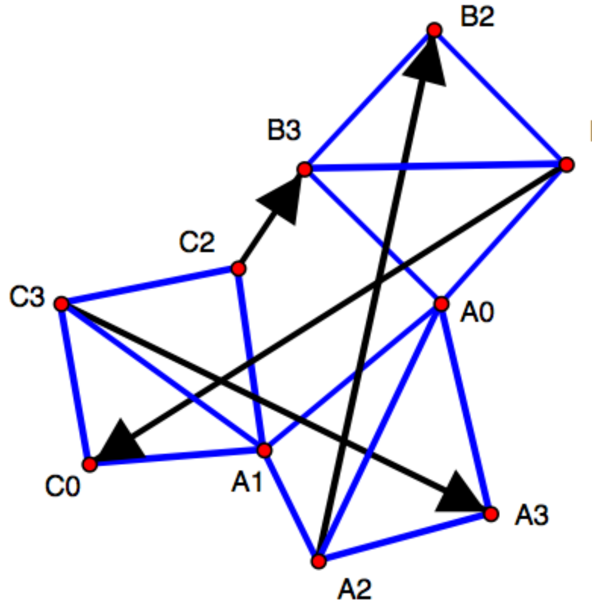


Figure 2: A fragment of the tetrahedral framework of quartz. The periodicity lattice is generated by the four black vectors, which must maintain a zero sum under deformation. The full (infinite) framework is obtained by translating the depicted tetrahedra with all periods (black arrows).

We rely on the notation described in Figure 2. Equivalence under Euclidean motions is eliminated by assuming the tetrahedron marked $A_0A_1A_2A_3$ as fixed. Since all edges maintain their length, the positions of the two tetrahedra which share the vertices A_0 and A_1 are completely described by two orthogonal transformations R_0, respectively R_1 as follows: R_0 fixes A_0 and takes A_i to B_i, $i \neq 0$, while R_1 fixes A_1 and takes A_j to C_j, $j \neq 1$. Figure 2, by depicting only the 'visible' edges, implies that both R_0 and R_1 are orientation reversing, that is, as orthogonal matrices $-R_0, -R_1 \in SO(3)$.

If we denote the edge vectors $A_i - A_0$ by $e_i, i = 1, 2, 3$, we have:

$$B_3 - C_2 = R_0 e_3 - (e_1 + R_1(e_2 - e_1))$$

$$A_3 - C_3 = e_3 - (e_1 + R_1(e_3 - e_1))$$

$$B_2 - A_2 = R_0 e_2 - e_2$$

$$C_0 - B_1 = e_1 - R_1 e_1 - R_0 e_1$$

It follows that the dependency condition of a zero sum for these four generators of the periodicity lattice takes the form

$$R_1(e_1 - e_2 - e_3) - R_0(e_1 - e_2 - e_3) = e_1 + e_2 - e_3 \quad (1)$$

Under our regularity assumptions, the three vectors $R_1(e_1 - e_2 - e_3)$, $R_0(e_1 - e_2 - e_3)$ and $(e_1 + e_2 - e_3)$ have the same length and form an equilateral triangle. This restricts $R_0(e_1 - e_2 - e_3)$ to the circle on the sphere of radius $||e_1 - e_2 - e_3||$ (which corresponde with an angle of $2\pi/3$ with $e_1 + e_2 - e_3$). Thus, $-R_0 \in SO(3)$ is constrained to a surface, which is differentiably a two-torus $(S^1)^2$.

For each choice of $-R_0$ on this torus, $R_1(e_1 - e_2 - e_3)$ is determined by (1), hence $-R_1$ is restricted to a circle S^1 in $SO(3)$. We summarize these calculations as:

Theorem 3 *The deformation space of the ideal quartz framework is given by a three dimensional torus $(S^1)^3$ minus the degenerate cases when the span of the four vectors is less than three dimensional.*

4 The cristobalite framework

The 'ideal β cristobalite' structure is illustrated in Figures 3 and 4. The periodicity group of the framework is given by all the translational symmetries of the ideal crystal framework. As a result, there are $n = 4$ orbits of vertices and $m = 12$ orbits of edges.

Adopting the notations of Figure 3, we may assume the tetrahedron $Os_1s_2s_3$ as fixed and parametrize the possible positions of the other tetrahedon by a rotation around the origin O. We obtain that:

Theorem 4 *The deformation space of the ideal high cristobalite framework is naturally parametrized by the open set in $SO(3)$ where the depicted generators remain linearly independent.*

5 The tridymite framework

The tetrahedral framework (G, Γ) of tridymite is depicted in Figure 5. We consider the ideal case made of regular tetrahedra. The quotient graph has $|V/\Gamma| = 8$ and $|E/\Gamma| = 24$. All deformations can be described by three orthogonal transformations (matrices) R_0, R_1, R_2 acting with centers at $O, O1$ and respectively $O2$. With O as the origin and the tetrahedron $OD_1E_1O_1$ assumed fixed, we denote:

$$O_1 = f_0, \quad D_1 = f_1 \;\text{ and }\; E_1 = f_2$$

Then, our orthogonal transformations are determined by the following relations:

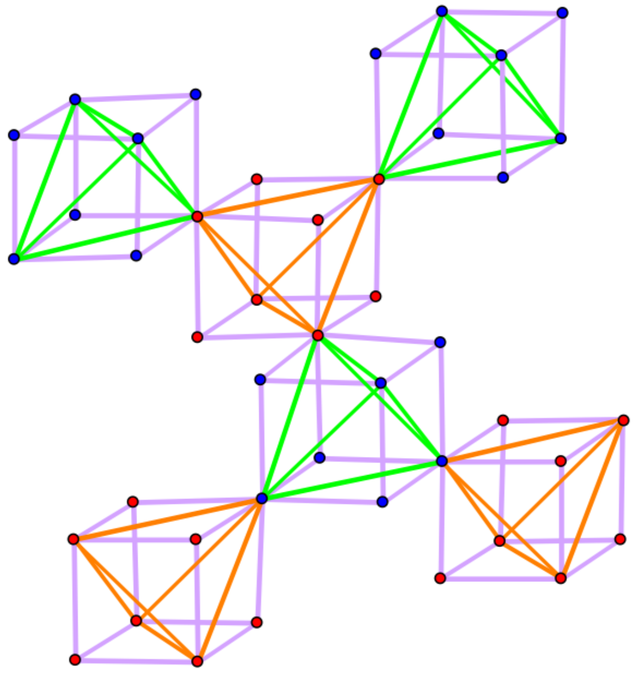


Figure 3: The ideal cristobalite framework (aristotype). The framework is made of vertex sharing regular tetrahedra. Cubes are traced only for suggestive purposes regarding symmetry and periodicity. See also Figure 4.

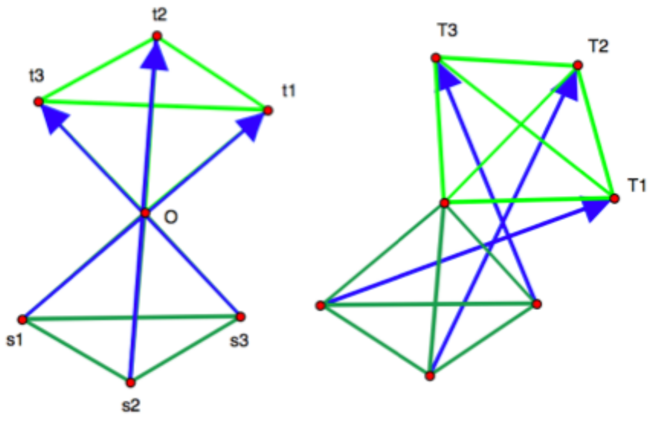


Figure 4: Deforming the ideal cristobalite framework. The periodicity lattice is generated by the three vectors $\gamma_i = t_i - s_i$ which vary as the framework deforms.

$$O_2 = R_0 f_0, \quad D_2 = R_0 f_1 \quad \text{and} \quad E_2 = R_0 f_2$$

$$A_1 = f_0 + R_1(f_1 - f_0)$$

$$B_1 = f_0 + R_1(f_2 - f_0)$$

$$C_1 = f_0 - R_1 f_0$$

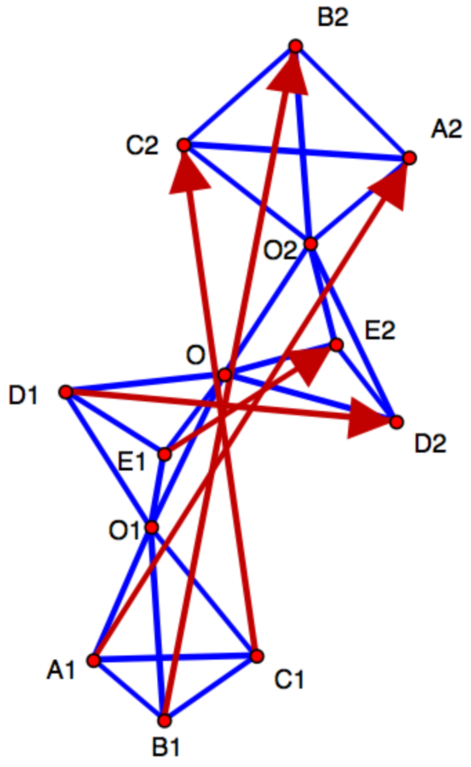


Figure 5: The tetrahedral framework of tridymite. The periodicity lattice is generated by the marked vectors, subject to the relations $(C_2 - C_1) + (D_2 - D_1) = (A_2 - A_1)$ and $(C_2 - C_1) + (E_2 - E_1) = (B_2 - B_1)$.

and

$$A_2 = R_0 f_0 + R_2 R_0 (f_1 - f_0)$$

$$B_2 = R_0 f_0 + R_2 R_0 (f_2 - f_0)$$

$$C_2 = R_0 f_0 - R_2 R_0 f_0$$

As a result, the two linear dependence relations between the six depicted periods take the form:

$$(I - R_0 - R_1 + R_2 R_0) f_i = 0, \quad i = 1, 2 \tag{2}$$

where I denotes the identity. We note that the ideal high tridymite structure (the *aristotype*) corresponds to $R_0 = -I$ and $R_1 = R_2$ the reflection in the plane $span(f_1, f_2)$.

We now describe the deformation space in a neighborhood of this high tridymite structure. We put $-R_0 = Q$, $R_1 = Q_1$ and $-R_2 R_0 = Q_2$, so that (2) becomes:

$$I + Q = Q_1 + Q_2 \quad \text{on} \quad span(f_1, f_2) \tag{3}$$

with $Q, -Q_1, -Q_2 \in SO(3)$. Since the orthogonal transformations Q, Q_1, Q_2 are completely determined by their values on two vectors e_1, e_2 of a Cartesian frame with $span(e_1, e_2) = span(f_1, f_2)$, we have to solve the system:

$$e_i + Qe_i = Q_1e_i + Q_2e_i \quad i = 1,2 \qquad (4)$$

where we assume $Q \in SO(3)$ given in a neighborhood of the identity transformation, and look for solutions Q_1, Q_2.

This system may be interpreted in terms of *spherical four-bar mechanisms* in the following way. All the vectors implicated in (4) are unit vectors and can be depicted as points on the unit sphere S^2. For a given Q, we mark by M_i the midpoint of the spherical geodesic segment $[e_i, Qe_i]$ and trace the circle with center M_i and diameter $[e_i, Qe_i]$. This is illustrated in Figure 6.

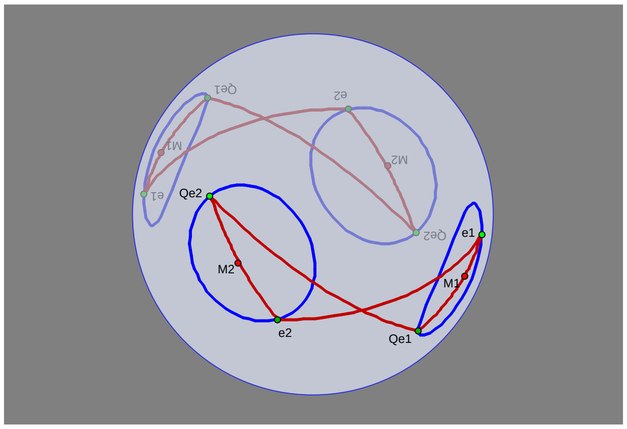


Figure 6: The spherical four-bar mechanism associated to the system (4).

It is an elementary observation that any solution Q_1e_i and Q_2e_i determines diameters of the corresponding circles for $i = 1,2$, with the two geodesic arcs $[Q_ke_1, Q_ke_2]$, like $[e_1, e_2]$ and $[Qe_1, Qe_2]$, of length $\pi/2$. Thus, the two spherical quadrilaterals with vertices at e_1, Qe_1, Qe_2, e_2 and respectively Q_1e_1, Q_2e_1, Q_2e_2, Q_1e_2 are two configurations of the same four-bar mechanism and moreover, the distance between the midpoints of the opposite edges represented by diameters is the same.

It follows from the theory of the spherical four-bar mechanism that, for a generic Q near the identity of $SO(3)$, the abstract configuration space is made of two loops which correspond by reflecting the corresponding realizations. Each loop component has two configurations with the prescibed distance $[M_1M_2]$. Thus, there are four configurations with the prescribed distance.

We observe that if we replace Q_1 by Q_2 and Q_2 by Q_1 in the labeling of the vertices of a realization, the orientation is reversed, hence the configuration belongs to the other component. Thus, the two obvious solutions of (4), namely:

$$Q_1e_i = e_i, \; Q_2e_i = Qe_i$$

and

$$Q_1e_i = Qe_i, \; Q_2e_i = e_i, \quad i = 1,2$$

correspond to configurations belonging to different loop components, as do the remaining two, which are also paired by relabeling. This discussion shows that all four solutions are obtained from the quadrilateral e_1, Qe_1, Qe_2, e_2 and its reflection in the geodesic $[M_1, M_2]$, by the two relabelings with Q_1 and Q_2 possible in each case.

In Figure 7 we have depicted the quadrilateral e_1, Qe_1, Qe_2, e_2 as $A_1B_1B_2A_2$, with reflection in $[M_1M_2]$ marked as rA_1, rB_1, rB_2, rA_2. Then,, the solutions $(Q_1e_1, Q_1e_2, Q_2e_1, Q_2e_2)$ of the system (4) are the following four solutions: (A_1, A_2, B_1, B_2), (B_1, B_2, A_1, A_2), (rA_1, rA_2, rB_1, rB_2) and (rB_1, rB_2, rA_1, rA_2).

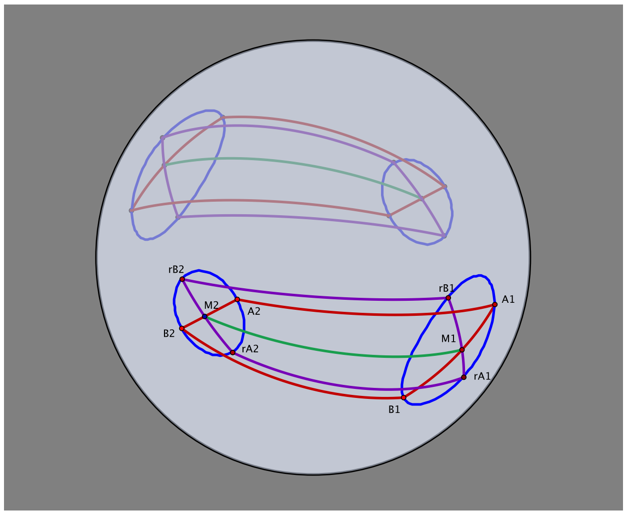


Figure 7: Spherical four-bar mechanism and reflection in $[M_1, M_2]$.

We summarize this result as:

Theorem 5 *The deformation space of the tridymite framework is singular in a neighbourhood of the aristotype and can be represented as a ramified covering with four sheets of a three-dimensional domain. There is a natural $Z_2 \times Z_2$ action on this covering which fixes the aristotype framework.*

Indeed, the two involutions, inverting the labeling and reflecting in $[M_1, M_2]$, commute and give a $Z_2 \times Z_2$. action on the covering. The dimension of the tangent space at the aristotype framework is computed from the linear version of (4) and is six.

6 Conclusions

Periodic (crystalline) materials, either occurring in nature or man-made, have been studied with experimental tools (such as X-ray crystallography) for over 100 years, and yet important phenomena related to their flexibility properties remain largely uncharted. In this paper, we studied the flexibility of three important families of periodic structures. These frameworks are flexible, and descriptions of their configuration spaces were explicitly given. A version of this paper has been posted on the arxiv [2].

References

[1] C. S. Borcea and I. Streinu. Periodic frameworks and flexibility. *Proceedings of the Royal Society A 8*, 466(2121):2633–2649, September 2010.

[2] C. S. Borcea and I. Streinu. Deformations of crystal frameworks. *arxiv:1110.4661*, 2011.

[3] C. S. Borcea and I. Streinu. Frameworks with crystallographic symmetry. *arXiv:1110.4662*, 2011.

[4] C. S. Borcea and I. Streinu. Minimally rigid periodic graphs. *Bulletin of the London Mathematical Society*, 43:1093–1103, 2011. doi:10.1112/blms/bdr044.

[5] C. S. Borcea, I. Streinu, and S. Tanigawa. Periodic body-and-bar frameworks. In *Proc. 28th Symp. Computational Geometry (SoCG'12), to appear*, June 2012.

[6] W. L. Bragg and R. E. Gibbs. The structure of α and β quartz. *Proceedings of the Royal Society A: mathematical, physical and engineering sciences*, 109(751):405–427, 1925.

[7] M. T. Dove. Theory of displacive phase transitions in minerals. *American Mineralogist*, 82:213–244, 1997.

[8] R. E. Gibbs. The polymorphism of silicon dioxide and the structure of tridymite. *Proceedings of the Royal Society A: mathematical, physical and engineering sciences*, 113:351–368, 1926.

[9] J. Graver, B. Servatius, and H. Servatius. *Combinatorial rigidity*. Graduate Studies in Mathematics. American Mathematical Society, 1993.

[10] V. Kapko, M. M. J. Treacy, M. F. Thorpe, and S. Guest. On the collapse of locally isostatic networks. *Proceedings of the Royal Society A: mathematical, physical and engineering sciences*, 465:3517–3530, 2009.

[11] N. Katoh and S. Tanigawa. A proof of the molecular conjecture. *Discrete and Computational Geometry*, 45(4):647–700, 2011.

[12] G. Laman. On graphs and rigidity of plane skeletal structures. *Journal of Engineering Mathematics*, 4:331–340, 1970.

[13] A. Lee and I. Streinu. Pebble game algorithms and sparse graphs. *Discrete Mathematics*, 308(8):1425–1437, April 2008.

[14] A. Lee, I. Streinu, and L. Theran. Analyzing rigidity with pebble games. In *SOCG '08: Proceedings of the twenty-fourth annual symposium on Computational geometry*, pages 226–227, New York, NY, USA, 2008. ACM.

[15] J. C. Maxwell. On the calculation of the equilibrium and stiffness of frames. *Philosophical Magazine*, 27:294–299, 1864.

[16] L. Pauling. The structure of some sodium and calcium aluminosilicates. *Proceedings of the National Academy of Sciences*, 16(7):453–459, 1930.

[17] I. Streinu and L. Theran. Slider-pinning rigidity: a Maxwell-Laman-type theorem. *Discrete and Computational Geometry*, 44(4):812–834, September 2010. Published on line, 8 Sep. 2010.

[18] T.-S. Tay. Rigidity of multigraphs I: linking rigid bodies in n-space. *Journal of Combinatorial Theory, Series B*, 36:95–112, 1984.

[19] T.-S. Tay. Linking $(n-2)$-dimensional panels in n-space II: $(n-2,2)$-frameworks and body and hinge structures. *Graphs and Combinatorics*, 5:245–273, 1989.

[20] S. A. Wells, M. T. Dove, and M. G. Tucker. Finding best-fit polyhedral rotations with geometric algebra. *Journal of Physics Condensed Matter*, 14:4567–4584, May 2002.

Characterizing Delaunay Graphs via Fixed Point Theorem

Tomomi Matsui* Yuichiro Miyamoto†

Abstract

This paper discusses a problem for determining whether a given plane graph is a Delaunay graph, i.e., whether it is topologically equivalent to a Delaunay triangulation. There exists a theorem which characterizes Delaunay graphs and yields a polynomial time algorithm for the problem only by solving a certain linear inequality system. The theorem was proved by Rivin based on arguments of hyperbolic geometry. Independently, Hiroshima, Miyamoto and Sugihara gave another proof of the theorem based on primitive arguments on Euclidean geometry. Unfortunately, the existing proofs of the theorem are rather difficult or long. In this paper, we give a simple proof of the theorem characterizing Delaunay graphs, which is based on the fixed point theorem.

1 Introduction

The two-dimensional Delaunay triangulation and its dual, the Voronoi diagram, are fundamental concepts in computational geometry, and have many practical applications such as interpolation and mesh generation [1, 3, 8]. It is also important to recognize Delaunay triangulations. The recognition problem can be divided into two types: geometric and combinatorial. The geometric problem is to judge whether a given drawing is a Delaunay triangulation. The combinatorial problem, which is discussed in this paper, determines whether a given embedded graph is topologically equivalent to a Delaunay triangulation. The combinatorial problem is important not only theoretically but also practically because it is closely related to the design of a topologically consistent algorithm for constructing the Delaunay/Voronoi diagram in finite-precision arithmetic [7, 11, 12].

Hodgson et al. [6] characterized the convex polyhedra that can be inscribed in a sphere, and constructed a polynomial time algorithm for judging whether a given graph is realizable as a convex polyhedron with all the vertices on a common sphere. On the basis of this characterization, Rivin [9, 10] reduced the recognition problem on the Delaunay graph to a certain linear programming problem, and thus gave a polynomial time algorithm. His proof was based on sophisticated arguments about hyperbolic geometry, and hence is not easy to understand. Almost the same time Hiroshima et al. independently found the same algorithm [5]. Their proof is simple in the sense that it is based on primitive arguments on Euclidean geometry, but the proof is long and intricate.

In this paper, we give a simple short proof of the theorem characterizing Delaunay graphs by employing the fixed point theorem. After making preparations in Section 2, we give our main result (a simple proof) in Section 3.

*Department of Information and System Engineering, Chuo University, matsui@ise.chuo-u.ac.jp

†Department of Information and Communication Sciences, Sophia University, miyamoto@sophia.ac.jp.

2 Preliminaries

2.1 Delaunay Graph

First, we briefly review the notion of a Delaunay triangulation. Given a set of mutually distinct points $P \subseteq \mathbb{R}^2$, a *Delaunay triangulation* of P is commonly defined as a triangulation of P satisfying the property that the circumcircle of each inner cell (triangle) contains no point of P in its interior. A Delaunay triangulation of P is also known as the planar dual of the *Voronoi diagram* of P. A Delaunay triangulation is called *non-degenerate* if and only if it satisfies the conditions that no three vertices on the outermost cell are collinear, and no four vertices lie on a common circle that circumscribes an inner cell.

Next, we give a definition of *combinatorial triangulation*. Let G be an undirected graph with vertex set V and edge set E. We assume that G is connected and plane graph (planar graph embedded in the 2-dimensional plane) without selfloops and parallel edges. The outermost cell is unbounded while the other cells, called *inner cells*, are bounded. We also assume that all the inner cells are bounded by exactly three edges. For each inner cell, we associate a directed 3-cycle which is obtained from an undirected 3-cycle of G forming the boundary of the cell by directing edges counterclockwise. Let C be the set of all the directed 3-cycles corresponding to all the inner cells of G. A combinatorial triangulation is defined by a triplet (V, E, C). In the rest of this paper, we write $G = (V, E, C)$ and concentrate our attention on the topological structure of G; we do not care about the actual positions at which the vertices are placed. When a given undirected graph, which is defined by vertex set V and edge set E, is

2-connected, we say that a combinatorial triangulation (V, E, C) is *2-connected*.

In the following, we introduce some notions related to a problem for judging whether a given combinatorial triangulation is obtained from a Delaunay triangulation. Given a combinatorial triangulation $G = (V, E, C)$, we seek an injection $\psi : V \to \mathbb{R}^2$ satisfying that the set of points $\{\psi(v) \in \mathbb{R}^2 \mid v \in V\}$ and the set of line segments between pairs in $\{\{\psi(u), \psi(v)\} \mid \{u, v\} \in E\}$ define a Delaunay triangulation. A map ψ satisfying the above conditions is called a *Delaunay realization*, if it exists. When a combinatorial triangulation G has a Delaunay realization, we say that G is a *Delaunay graph*. In particular, if a corresponding Delaunay triangulation is non-degenerate, then G is called a *non-degenerate Delaunay graph*.

2.2 Characterizing Delaunay Graphs

In this subsection, we briefly review an inequality system which characterizes (non-degenerate) Delaunay graphs. Given a combinatorial triangulation $G = (V, E, C)$, we denote the elements of C by $c_0, c_1, \dots, c_{|C|-1}$. For each cycle $c_i \in C$, we introduce three variables $x_{3i+1}, x_{3i+2}, x_{3i+3}$ assigned to three vertices in c_i. In the rest of this paper, we interpret these variables as angles in degrees at the corresponding corner of a triangle defined by cycle c_i. So, let us call these variables *angle variables*. There are $3|C|$ angle variables. We denotes the index set of angle variables by $J := \{1, 2, \dots, 3|C|\}$. For example, a combinatorial triangulation shown in Figure 1 has nine angle variables $x_1, x_2, \dots, x_9$.

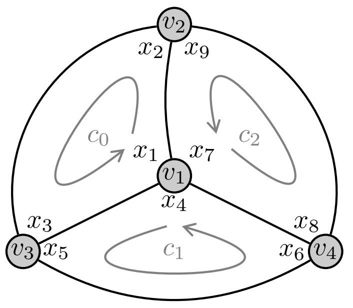


Figure 1: Combinatorial triangulation and angle variables

A vertex of a combinatorial triangulation G is called an *outer vertex* if it is on the boundary of the outermost cell, and an *inner vertex* otherwise. Similarly, an edge of G is called an *outer edge* if it is on the boundary of the outermost cell, and an *inner edge* otherwise. In the rest of this paper, we denote a set of outer vertices and a set of inner vertices by V^{outer} and V^{inner} respectively.

If a given combinatorial triangulation $G = (V, E, C)$ is a Delaunay graph, a corresponding vector of angle variables, defined by a Delaunay realization, satisfies the following conditions.

C1 For each cycle in C, the sum of the associated three angle variables is equal to 180.

C2 For each inner vertex, the sum of all the associated angle variables is equal to 360.

C3 For each outer vertex, the sum of all the associated angle variables is at most 180.

C4 For each inner edge, the sum of the associated pair of the facing angle variables (i.e., the angle variables corresponding to the vertices that are on the same cycle as, but are not incident to, the inner edge) is at most 180.

C5 Each angle variable is positive.

For example, if we consider the combinatorial triangulation in Figure 1, the above conditions give the following linear inequality system;

(C1) defined by c_0 :	$x_1 + x_2 + x_3 = 180,$
(C1) defined by c_1 :	$x_4 + x_5 + x_6 = 180,$
(C1) defined by c_2 :	$x_7 + x_8 + x_9 = 180,$
(C2) defined by v_1 :	$x_1 + x_4 + x_7 = 360,$
(C3) defined by v_2 :	$x_2 + x_9 \leq 180,$
(C3) defined by v_3 :	$x_3 + x_5 \leq 180,$
(C3) defined by v_4 :	$x_6 + x_8 \leq 180,$
(C4) defined by $\{v_1, v_2\}$:	$x_3 + x_8 \leq 180,$
(C4) defined by $\{v_1, v_3\}$:	$x_2 + x_6 \leq 180,$
(C4) defined by $\{v_1, v_4\}$:	$x_5 + x_9 \leq 180,$
(C5) :	$x_1, x_2, \dots, x_9 > 0.$

Figure 2 gives an example of a Delaunay realization of the combinatorial triangulation in Figure 1.

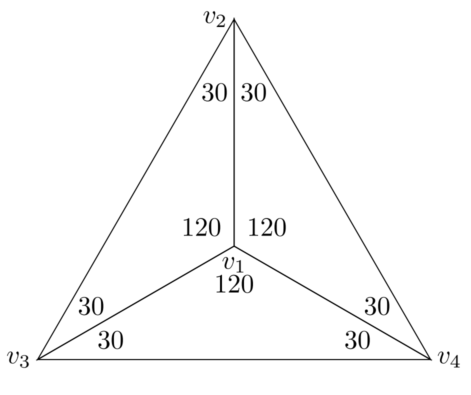


Figure 2: Realized Delaunay triangulation.

Unfortunately, the values of the angle variables satisfying all the conditions C1–C5 do not necessarily correspond to a Delaunay triangulation. For example, the combinatorial triangulation in Figure 1 has a vector of angle variables defined by

$$x_1 = x_4 = x_7 = 120,\ x_2 = x_5 = x_8 = 32,$$
$$x_3 = x_6 = x_9 = 28,$$

which satisfies C1–C5, but it does not correspond to any triangulations. If we try to draw the diagram using these angle values, we come across an inconsistency as shown in Figure 3. In order to avoid this inconsistency, we still need other conditions described below.

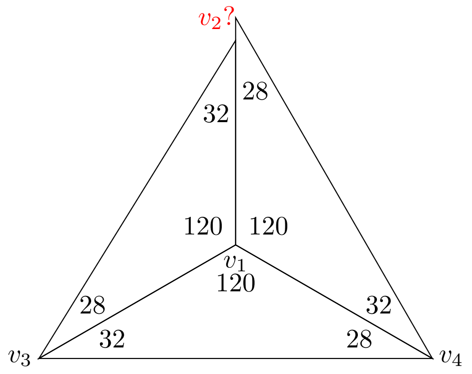


Figure 3: Angle variables satisfying C1–C5.

Let $c \in C$ be an inner cell with three vertices v_α, v_β, v_γ, and x_i, x_j, x_k be three angle variables corresponding to the three vertices, respectively. We say that x_j is *cc-facing* (meaning "facing counterclockwise") around v_α and x_k is *c-facing* (meaning "facing clockwise") around v_α. In Figure 4, for example, x_2, x_5, x_8 are cc-facing around v_1 while x_3, x_9, x_6 are c-facing around v_1.

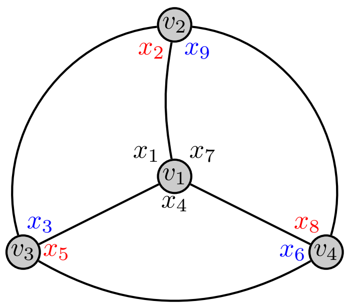


Figure 4: x_3, x_6, x_9 are c-facing and x_2, x_5, x_8 are cc-facing around v_1.

For any inner vertex $v \in V^{\text{inner}}$, let $X_v^{\text{CC}} \subseteq J$ be indices of cc-facing angle variables around v, and $X_v^{\text{C}} \subseteq J$ be indices of c-facing angle variables around v. Furthermore, we introduce a function

$$F_v(\boldsymbol{x}) := \frac{\prod_{j \in X_v^{\text{CC}}} \sin x_j}{\prod_{j \in X_v^{\text{C}}} \sin x_j}, \tag{1}$$

where $\boldsymbol{x} \in \mathbb{R}^J$ is a vector of angle variables (in degrees). We only consider angle variables satisfying $0 < x_j < 180$ $(\forall j \in J)$, and hence we get $0 < F_v(\boldsymbol{x}) < \infty$.

Now we describe a necessary and sufficient condition that a combinatorial triangulation becomes a Delaunay graph.

Theorem 1 ([5]) *A 2-connected combinatorial triangulation $G = (V, E, C)$ is a Delaunay graph if and only if the set of conditions C1–C6 is satisfiable, where*

C6 *$F_v(\boldsymbol{x}) = 1$ for any inner vertex $v \in V^{\text{inner}}$.*

It is not so difficult to prove the above theorem. For example, Hiroshima, Miyamoto and Sugihara gave a short and elementary proof in their paper [5].

If we restrict the Delaunay triangulations to non-degenerate ones, the conditions C3 and C4 are respectively changed in the following way.

C3' For each outer vertex, the sum of all the associated angle variables is *less than* 180.

C4' For each inner edge, the sum of the associated pair of the angle values facing the edge is *less than* 180.

A non-degenerate version of Theorem 1 is as follows.

Theorem 2 ([5]) *A 2-connected combinatorial triangulation $G = (V, E, C)$ is a non-degenerate Delaunay graph if and only if the set of conditions C1, C2, C3', C4', C5 and C6 is satisfiable.*

Thus, we get a necessary and sufficient condition for a combinatorial triangulation to be a (non-degenerate) Delaunay graph. However, the conditions stated in Theorems 1 and 2 are not useful for the recognition of a Delaunay graph, because we do not know any finite-step algorithm for judging the satisfiability of these conditions.

3 Main Result

Now we describe a theorem which yields an efficient method for recognizing Delaunay graphs. The following theorem says that when we only need to judge whether a given combinatorial triangulation is a Delaunay graph (or not), we can drop condition C6, surprisingly.

Theorem 3 ([5, 9, 10]) *A 2-connected combinatorial triangulation $G = (V, E, C)$ is a Delaunay graph if and only if the set of conditions C1–C5 is satisfiable.*

We can judge the satisfiability of the set of conditions C1–C5 in finite steps because the conditions C1 through C5 are linear in the variables and the method for checking their satisfiability has been established using linear programming (see [5] for detail). Especially, the obtained linear programming problem satisfies that all the non-zero coefficients are $+1$ or -1, and thus it is solvable in strongly polynomial time [4].

The following theorem deals with the non-degenerate case.

Theorem 4 ([5, 9, 10]) *A 2-connected combinatorial triangulation $G = (V, E, C)$ is a non-degenerate Delaunay graph if and only if the set of conditions C1, C2, C3', C4' and C5 is satisfiable.*

We employ the fixed point theorem and give simple proofs of Theorem 3 and Theorem 4.

Theorem 5 (Fixed Point Theorem [2])
Every continuous map $f : B^m \to B^m$ defined on an m-dimensional closed ball B^m has a fixed point (a point $x \in B^m$ with $f(x) = x$).

It is well known that we can extend the above theorem to a continuous map defined on a convex compact set.

Before describing our proof, we give a sketch of an important procedure, which transforms a feasible solution of the linear inequality system defined by C1–C5. Let us recall a vector of angle variables shown in Figure 3, that satisfies conditions C1–C5, but not C6. Now we construct a (new) vector by increasing angle variables c-facing around the inner vertex v_1 by α degree, and decreasing angle variables cc-facing around v_1 by α degree. After this procedure, conditions C1–C4 are preserved. When we set $\alpha = 2$, the obtained vector of angle variables, shown in Figure 2, satisfies conditions C1–C6.

Now we describe the above procedure precisely. Given a non-negative vector $\boldsymbol{x} \geq \boldsymbol{0}$ of angle variables satisfying C1–C4, an inner vertex v and a real number α, we introduce a vector $\boldsymbol{x}(\alpha)$ defined by

$$x(\alpha)_j = \begin{cases} x_j + \alpha, & j \in X_v^{\mathrm{CC}}, \\ x_j - \alpha, & j \in X_v^{\mathrm{C}}, \\ x_j, & \text{otherwise}. \end{cases} \tag{2}$$

The following lemma shows some properties of $\boldsymbol{x}(\alpha)$.

Lemma 6 *Let $\boldsymbol{x} \geq \boldsymbol{0}$ be a non-negative vector of angle variables satisfying C1–C4 and $\boldsymbol{x}(\alpha)$ be a vector defined by (2) w.r.t. an inner vertex $v \in V^{inner}$. For any $\alpha \in \mathbb{R}$, vector $\boldsymbol{x}(\alpha)$ satisfies conditions C1–C4. We define*

$$\begin{aligned} \alpha_{\max} &= \max\{\alpha \in \mathbb{R} \mid \boldsymbol{x}(\alpha) \geq \boldsymbol{0}\}, \\ \alpha_{\min} &= \min\{\alpha \in \mathbb{R} \mid \boldsymbol{x}(\alpha) \geq \boldsymbol{0}\}. \end{aligned}$$

If $\alpha_{\min} < \alpha_{\max}$, then $F_v(\boldsymbol{x}(\alpha)) : (\alpha_{\min}, \alpha_{\max}) \to \mathbb{R}$ is a continuous monotone increasing function w.r.t. α.

Proof. It is easy to show that $\boldsymbol{x}(\alpha)$ satisfies conditions C1–C4. The continuity of $F_v(\boldsymbol{x}(\alpha))$ with respect to α is obvious. Let $C(v) \subseteq C$ be a set of cycles including v. For each cycle $c' \in C(v)$, angle variable $x_{c'}^{\mathrm{CC}}$ ($x_{c'}^{\mathrm{C}}$) denotes associated cc-facing (c-facing) angle variable around v. Condition C1, non-negativity of $\boldsymbol{x}$, and inequality $\alpha_{\min} < \alpha_{\max}$ imply that $0 < x_{c'}^{\mathrm{CC}} + x_{c'}^{\mathrm{C}} \leq 180$ ($\forall c' \in C(v)$). We transform the following differentiation and obtain that

$$\begin{aligned} &\frac{\mathrm{d}\log F_v(\boldsymbol{x}(\alpha))}{\mathrm{d}\alpha} \\ &= \sum_{j \in X_v^{\mathrm{CC}}} \frac{\mathrm{d}\log\sin(x_j + \alpha)}{\mathrm{d}\alpha} - \sum_{j \in X_v^{\mathrm{C}}} \frac{\mathrm{d}\log\sin(x_j - \alpha)}{\mathrm{d}\alpha} \\ &= \sum_{j \in X_v^{\mathrm{CC}}} \frac{\cos(x_j + \alpha)}{\sin(x_j + \alpha)} + \sum_{j \in X_v^{\mathrm{C}}} \frac{\cos(x_j - \alpha)}{\sin(x_j - \alpha)} \\ &= \sum_{c' \in C(v)} \left(\frac{\cos(x_{c'}^{\mathrm{CC}} + \alpha)}{\sin(x_{c'}^{\mathrm{CC}} + \alpha)} + \frac{\cos(x_{c'}^{\mathrm{C}} - \alpha)}{\sin(x_{c'}^{\mathrm{C}} - \alpha)} \right) \\ &= \sum_{c' \in C(v)} \frac{\sin(x_{c'}^{\mathrm{CC}} + x_{c'}^{\mathrm{C}})}{\sin(x_{c'}^{\mathrm{CC}} + \alpha)\sin(x_{c'}^{\mathrm{C}} - \alpha)} > 0, \end{aligned}$$

where the last inequality is derived from the facts that (1) $\forall \alpha \in (\alpha_{\min}, \alpha_{\max})$, $\forall c' \in C(v)$, $\sin(x_{c'}^{\mathrm{CC}} + \alpha)\sin(x_{c'}^{\mathrm{C}} - \alpha) > 0$ (2) $\forall c' \in C(v)$, $\sin(x_{c'}^{\mathrm{CC}} + x_{c'}^{\mathrm{C}}) \geq 0$, and (3) $\exists c' \in C(v)$, $\sin(x_{c'}^{\mathrm{CC}} + x_{c'}^{\mathrm{C}}) > 0$ (obtained from C2). Thus, both $\log F_v(\boldsymbol{x}(\alpha))$ and $F_v(\boldsymbol{x}(\alpha))$ are monotonically increasing. ☐

In the following, we show that if there exists a vector of angle variables satisfying C1–C5, then there also exists a vector of angle variables satisfying C1–C6 which is obtained by adopting the above procedure around all inner vertices simultaneously.

Proof. (Proof of Theorem 3.) From Theorem 1, we only have to show that we once obtain angle variables satisfying C1–C5 there is a vector of angle variables satisfying C1–C6.

Let $\boldsymbol{b} \in \mathbb{R}^J$ be a vector of angle variables satisfying conditions C1–C5, where $J = \{1, 2, \ldots, 3|C|\}$ is a set of indices of angle variables. We define a matrix M whose rows are indexed by J, columns are indexed by the vertex set V, and each entry m_{iv} is defined as follows:

$$m_{iv} = \begin{cases} 1, & \text{angle variable } x_i \text{ is cc-facing around } v, \\ -1, & \text{angle variable } x_i \text{ is c-facing around } v, \\ 0, & \text{otherwise}. \end{cases}$$

Figure 5 shows a matrix M corresponding to Figure 1.

Let $\widetilde{M}$ be a column submatrix of M corresponding to inner vertices V^{inner}. It is easy to see that the vector of angle variables $\widetilde{M}\boldsymbol{y} + \boldsymbol{b}$ is obtained from $\boldsymbol{b}$ by increasing angle variables c-facing around the inner vertex v by y_v, and decrease angle variables cc-facing around v by

$$\begin{array}{c|cccc|}
 & v_1 & v_2 & v_3 & v_4 \\ \hline
x_1 & 0 & -1 & 1 & 0 \\
x_2 & 1 & 0 & -1 & 0 \\
x_3 & -1 & 1 & 0 & 0 \\
x_4 & 0 & 0 & -1 & 1 \\
x_5 & 1 & 0 & 0 & -1 \\
x_6 & -1 & 0 & 1 & 0 \\
x_7 & 0 & 1 & 0 & -1 \\
x_8 & 1 & -1 & 0 & 0 \\
x_9 & -1 & 0 & 0 & 1 \\ \hline
\end{array}$$

Figure 5: A matrix M corresponding to Figure 1.

y_v, for each inner vertex $v \in V^{\text{inner}}$. Lemma 6 directly implies that for any vector $\boldsymbol{y} \in \mathbb{R}^{V^{\text{inner}}}$, a vector of angle variables $\widetilde{M}\boldsymbol{y} + \boldsymbol{b}$ also satisfies conditions C1–C4.

We introduce a subset $\Omega \subseteq \mathbb{R}^{V^{\text{inner}}}$ defined by

$$\Omega := \left\{ \boldsymbol{y} \in \mathbb{R}^{V^{\text{inner}}} \;\middle|\; \widetilde{M}\boldsymbol{y} + \boldsymbol{b} \geq \boldsymbol{0} \right\}.$$

Here, we briefly prove the boundedness of Ω by showing that every vector $\boldsymbol{y} \in \Omega$ satisfies $-180|V|\boldsymbol{1} \leq \boldsymbol{y} \leq 180|V|\boldsymbol{1}$. Let $\{u, v\}$ be an inner edge of G. Since $\{u, v\}$ is an inner edge, there exists an angle b_j (in the vector $\boldsymbol{b}$) which is both c-facing around u and cc-facing around v. There also exists an angle $b_{j'}$ (in vector $\boldsymbol{b}$) which is both cc-facing around u and c-facing around v. When both u and v are inner vertices, every vector $\boldsymbol{y} \in \Omega$ satisfies

$$-180 \leq -b_{j'} \leq y_u - y_v \leq b_j \leq 180. \tag{3}$$

If $(u, v) \in V^{\text{inner}} \times V^{\text{outer}}$, then we have

$$-180 \leq -b_{j'} \leq y_u \leq b_j \leq 180. \tag{4}$$

For any inner vertex u, there exists a minimal path Γ_u on G connecting u and an outer vertex. From the minimality, Γ_u consists of inner edges. The telescoping sum of inequalities (3) and (4) w.r.t. inner edges in Γ_u gives

$$-180|V| \leq y_u \leq 180|V|.$$

From the above, Ω becomes a compact convex set.

For any pair $(\boldsymbol{y}, v) \in \Omega \times V^{\text{inner}}$, we define following two values:

$$\alpha_{\max}(\boldsymbol{y}, v) := \max\{\alpha \in \mathbb{R} \mid \boldsymbol{y} + \alpha \boldsymbol{e}_v \in \Omega\},$$
$$\alpha_{\min}(\boldsymbol{y}, v) := \min\{\alpha \in \mathbb{R} \mid \boldsymbol{y} + \alpha \boldsymbol{e}_v \in \Omega\},$$

where $\boldsymbol{e}_v \in \{0,1\}^{V^{\text{inner}}}$ is a unit vector whose entry is equal to 1 if and only if the corresponding index is equal to v. (Here we note that both the maximum and the minimum always exist, because Ω is a bounded closed set and is nonempty; clearly $\boldsymbol{b} \in \Omega$.) Since $\boldsymbol{y} \in \Omega$, inequalities $\alpha_{\min}(\boldsymbol{y}, v) \leq 0 \leq \alpha_{\max}(\boldsymbol{y}, v)$ hold. When $\alpha_{\min}(\boldsymbol{y}, v) < \alpha_{\max}(\boldsymbol{y}, v)$, we have that

$$\lim_{\alpha \to \alpha_{\max}(\boldsymbol{y}, v)} F_v(\boldsymbol{y} + \alpha \boldsymbol{e}_v) = +\infty,$$
$$\lim_{\alpha \to \alpha_{\min}(\boldsymbol{y}, v)} F_v(\boldsymbol{y} + \alpha \boldsymbol{e}_v) = +0,$$

and thus Lemma 6 and the intermediate value theorem imply that there exists a unique value α^* in the open interval $(\alpha_{\min}(\boldsymbol{y}, v), \alpha_{\max}(\boldsymbol{y}, v))$ satisfying equality $F_v(\boldsymbol{y} + \alpha^* \boldsymbol{e}_v) = 1$. Now we introduce a map $f_v : \Omega \to \Omega$ for each $v \in V^{\text{inner}}$ defined by

$$f_v(\boldsymbol{y}) = \begin{cases} \boldsymbol{y}, & \text{if } \alpha_{\min}(\boldsymbol{y}, v) = \alpha_{\max}(\boldsymbol{y}, v) = 0, \\ \boldsymbol{y} + \alpha^* \boldsymbol{e}_v, & \text{if } \alpha_{\min}(\boldsymbol{y}, v) < \alpha_{\max}(\boldsymbol{y}, v), \end{cases}$$

where α^* is a unique value satisfying $F_v(\boldsymbol{y} + \alpha^* \boldsymbol{e}_v) = 1$. It is obvious that for each inner vertex v, the corresponding map f_v is continuous.

Lastly, we define a map $f : \Omega \to \Omega$ as:

$$f(\boldsymbol{y}) := \frac{1}{|V^{\text{inner}}|} \sum_{v \in V^{\text{inner}}} f_v(\boldsymbol{y}),$$

where $f(\boldsymbol{y})$ is the gravity center of vectors $\{f_v(\boldsymbol{y}) \mid v \in V^{\text{inner}}\}$. Since f_v is continuous for each inner vertex v, f is also continuous.

Now we apply the fixed point theorem to the continuous map f and obtain a result that there exists a fixed point $\boldsymbol{y}^* \in \Omega$, i.e., $\boldsymbol{y}^*$ satisfies $f(\boldsymbol{y}^*) = \boldsymbol{y}^*$.

Every fixed point $\boldsymbol{y}^*$ satisfies that

$$\begin{aligned} &\forall v \in V^{\text{inner}}, \\ &\alpha_{\min}(\boldsymbol{y}^*, v) = \alpha_{\max}(\boldsymbol{y}^*, v) = 0 \text{ or } F_v(\boldsymbol{y}^*) = 1. \end{aligned} \tag{5}$$

Otherwise, there exists at least one inner vertex v' satisfying $\alpha_{\min}(\boldsymbol{y}^*, v') < \alpha_{\max}(\boldsymbol{y}^*, v')$ and $F_{v'}(\boldsymbol{y}^*) \neq 1$. Then v' also satisfies $f_{v'}(\boldsymbol{y}^*) \neq \boldsymbol{y}^*$, which implies $f(\boldsymbol{y}^*) \neq \boldsymbol{y}^*$. It is a contradiction.

We have shown that there exists a non-negative vector of angle variables satisfying C1–C4. Next we discuss condition C5, which also yields condition C6. In the following, we show that $\widetilde{M}\boldsymbol{y}^* + \boldsymbol{b} > \boldsymbol{0}$ for *any* fixed point $\boldsymbol{y}^*$.

When a vertex v satisfies $F_v(\boldsymbol{y}^*) = 1$, it is obvious that $\alpha_{\min}(\boldsymbol{y}^*, v) < 0 < \alpha_{\max}(\boldsymbol{y}^*, v)$. Since $\boldsymbol{y}^*$ is a fixed point, property (5) implies that for any $v \in V^{\text{inner}}$,

$$\alpha_{\min}(\boldsymbol{y}^*, v) = 0 \text{ if and only if } \alpha_{\max}(\boldsymbol{y}^*, v) = 0.$$

Put $\boldsymbol{x}^* = \widetilde{M}\boldsymbol{y}^* + \boldsymbol{b}$. If an inner vertex v has a cc-facing angle variable x_j satisfying $x^*_j = 0$, then $\alpha_{\min}(\boldsymbol{y}^*, v) = 0$ and thus $\alpha_{\max}(\boldsymbol{y}^*, v) = 0$, which implies that v also has a c-facing angle variable $x_{j'}$ satisfying $x^*_{j'} = 0$. Similarly, when an inner vertex v has a c-facing angle variable x_j satisfying $x^*_j = 0$, then $\alpha_{\max}(\boldsymbol{y}^*, v) = 0$ and thus $\alpha_{\min}(\boldsymbol{y}^*, v) = 0$, which implies that v also has a cc-facing angle variable $x_{j'}$ satisfying $x^*_{j'} = 0$.

Let us consider a directed graph H whose incident matrix is $M^\top$: i.e., a directed graph obtained from G by substituting a pair of parallel arcs with opposite direction for each edge in E (see Figure 6). Digraph H has vertex set V and edge set J, which is an index set of angle variables. Each angle variable x_j corresponds to an arc in H from u to v where x_j is a c-facing variable around u and cc-facing variable around v.

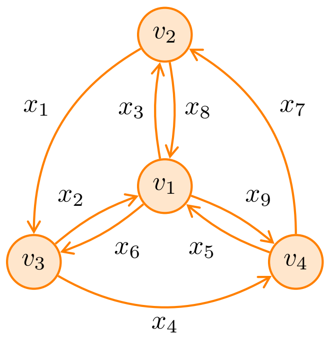


Figure 6: A directed graph (V, J) corresponding to Figure 1.

If an arc a of H satisfies that the corresponding angle variable, denoted by x_a, satisfies $x_a^* = 0$, we say that a is a *critical* arc. In the directed graph, if an inner vertex v has an incoming critical arc, then v also has at least one outgoing critical arc.

Now we show $\boldsymbol{x}^* > \boldsymbol{0}$. Assume on the contrary that there exists an angle variable x_j satisfying $x_j^* = 0$. Then, there exists a critical arc in H. Let A_0 be a set of critical arcs. From the above discussion, a digraph defined by (V, A_0) has either (Case 1) "a directed elementary path Γ_1 connecting a pair of outer vertices and passing only inner vertices" or (Case 2) "a directed elementary cycle Γ_2 consisting of inner vertices."
Case 1. Let $\chi_1 \in \{0,1\}^J$ be a characteristic vector of the set of arcs in Γ_1. Since Γ_1 consists of critical edges, $\chi_1^\top \boldsymbol{x}^* = 0$ hold. Every inner vertex v has an incoming arc in Γ_1 if and only if v has an outgoing arc in Γ_1. Accordingly, the equality $\chi_1^\top \widetilde{M} = \boldsymbol{0}$ hold. Thus we have that

$$0 = \chi_1^\top \boldsymbol{x}^* = \chi_1^\top(\widetilde{M}\boldsymbol{y}^* + \boldsymbol{b}) = \chi_1^\top \widetilde{M}\boldsymbol{y}^* + \chi_1^\top \boldsymbol{b} = \chi_1^\top \boldsymbol{b} > 0.$$

Contradiction.
Case 2. Let $\chi_2 \in \{0,1\}^J$ be a characteristic vector of the set of arcs in Γ_2. Since Γ_2 consists of inner vertices and critical edges, both $\chi_2^\top \widetilde{M} = \boldsymbol{0}$ and $\chi_2^\top \boldsymbol{x}^* = 0$ hold. Thus we have that

$$0 = \chi_2^\top \boldsymbol{x}^* = \chi_2^\top(\widetilde{M}\boldsymbol{y}^* + \boldsymbol{b}) = \chi_2^\top \widetilde{M}\boldsymbol{y}^* + \chi_2^\top \boldsymbol{b} = \chi_2^\top \boldsymbol{b} > 0.$$

Contradiction.
Now we have shown that every fixed point $\boldsymbol{y}^*$ satisfies condition C5 and thus every inner vertex v satisfies $\alpha_{\min}(\boldsymbol{y}^*, v) < 0 < \alpha_{\max}(\boldsymbol{y}^*, v)$. From property (5), every inner vertex v satisfies $F_v(\boldsymbol{y}^*) = 1$. As a consequence, condition C6 is satisfied. □

A proof of Theorem 4 is almost the same. Actually, we only have to replace C3 and C4 with C3' and C4' respectively in our proofs of Lemma 6 and Theorem 3.

Acknowledgements

We wish to express our thanks to Naoyuki Kamiyama and Mizuyo Takamatsu for fruitful discussions. We also thank anonymous reviewers.

References

[1] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[2] L. E. Brouwer. Über abbilidungen von mannigfaltigkeiten. *Math. Annalen*, 71:97–115, 1912.

[3] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Heidelberg, 1987.

[4] E. Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, 34(2):250–256, 1986.

[5] T. Hiroshima, Y. Miyamoto, and K. Sugihara. Another proof of polynomial-time recognizability of Delaunay graphs. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E83-A:627–638, 2000.

[6] C. D. Hodgson, I. Rivin, and W. D. Smith. A characterization of convex hyperbolic polyhedra and of convex polyhedra inscribed in the sphere. *Bulletin of the American Mathematical Society*, 27:246–251, 1991.

[7] Y. Oishi and K. Sugihara. Topology-oriented divide-and-conquer algorithm for Voronoi diagrams. *CVGIP: Graphical Model and Image Processing*, 57(4):303–314, 1995.

[8] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams.* Wiley and Sons, England, 2nd edition, 2000.

[9] I. Rivin. Euclidean structures on simplicial surfaces and hyperbolic volume. *Annals of Mathematics*, 139:553–580, 1994.

[10] I. Rivin. A characterization of ideal polyhedra in hyperbolic 3-space. *Annals of Mathematics*, 143:51–70, 1996.

[11] K. Sugihara and M. Iri. Construction of the Voronoi diagram for "one million" generators in single-precision arithmetic. *Proc. IEEE*, 80:1471–1484, 1992.

[12] K. Sugihara and M. Iri. A robust topology-oriented incremental algorithm for Voronoi diagrams. *Int. J. Comput. Geometry Appl.*, 4(2):179–228, 1994.

Lower Bounds for the Number of Small Convex k-Holes*

Oswin Aichholzer† Ruy Fabila-Monroy‡ Thomas Hackl† Clemens Huemer§ Alexander Pilz†
Birgit Vogtenhuber†

Abstract

Let S be a set of n points in the plane in general position, that is, no three points of S are on a line. We consider an Erdős-type question on the least number $h_k(n)$ of convex k-holes in S, and give improved lower bounds on $h_k(n)$, for $3 \leq k \leq 5$. Specifically, we show that $h_3(n) \geq n^2 - \frac{32n}{7} + \frac{22}{7}$, $h_4(n) \geq \frac{n^2}{2} - \frac{9n}{4} - o(n)$, and $h_5(n) \geq \frac{3n}{4} - o(n)$.

1 Introduction

Let S be a set of n points in the plane in general position, that is, no three points of S lie on a common straight line. A *k-hole* of S is a simple polygon, P, spanned by k points from S, such that no other point of S is contained in the interior of P. A classical existence question raised by Erdős [8] is: "What is the smallest integer $h(k)$ such that any set of $h(k)$ points in the plane contains at least one convex k-hole?". Esther Klein observed that every set of 5 points contains a convex 4-hole, and Harborth [12] showed that every set of 10 points determines a convex 5-hole. Both bounds are tight w.r.t. the cardinality of S. Only in 2007/08 Nicolás [14] and independently Gerken [11] proved that every sufficiently large point set contains a convex 6-hole. On the other hand, Horton [13] showed that there exist arbitrarily large sets which do not contain any convex 7-hole; see [1] for a brief survey.

A generalization of Erdős' question is: "What is the least number $h_k(n)$ of convex k-holes determined by any set of n points in the plane?". In this paper we concentrate on this question for $3 \leq k \leq 5$, that is, the number of empty triangles (3-holes), convex 4-holes, and convex 5-holes. We denote by $h_k(S)$ the number of convex k-holes determined by S, and by $h_k(n) = \min_{|S|=n} h_k(S)$ the number of convex k-holes any set of n points in general position must have. Throughout this paper let $\operatorname{ld} x = \frac{\log x}{\log 2}$ be the binary logarithm. Furthermore, we denote with $\operatorname{CH}(S)$ the convex hull of S and with $\partial \operatorname{CH}(S)$ the boundary of $\operatorname{CH}(S)$.

We start in Section 2 by providing improved bounds on the number of convex 5-holes. In particular, increasing the so far best bound $h_5(n) \geq \frac{n}{2} - O(1)$ [16] to $h_5(n) \geq \frac{3n}{4} - n^{0.87447} + 1.875$. In Section 3 we combine these results with a technique recently introduced by García [9, 10], and improve the currently best bounds on the number of empty triangles and convex 4-holes, $h_3(n) \geq n^2 - \frac{37n}{8} + \frac{23}{8}$ and $h_4(n) \geq \frac{n^2}{2} - \frac{11n}{4} - \frac{9}{4}$ (both in [10]), to $h_3(n) \geq n^2 - \frac{32n}{7} + \frac{22}{7}$ and $h_4(n) \geq \frac{n^2}{2} - \frac{9n}{4} - 1.2641\, n^{0.926} + \frac{199}{24}$, respectively.

2 Convex 5-holes

The currently best upper bound on the number of convex 5-holes, $h_5(n) \leq 1.0207n^2 + o(n^2)$ is by Bárány and Valtr [5], and it is widely conjectured that $h_5(n)$ grows quadratically. Still, to this date not even a super-linear lower bound is known.

As early as in 1987 Dehnhardt presented a lower bound of $h_5(n) \geq 3\lfloor \frac{n}{12} \rfloor$ in his thesis [6]. Unfortunately, this result, published in German only, remained unknown to the scientific community until recently. Thus, the best known lower bound was $h_5(n) \geq \lfloor \frac{n-4}{6} \rfloor$, obtained by Bárány and Károlyi [4]. In the presentation of [9] this bound was improved to $h_5(n) \geq \frac{2}{9}n - \frac{25}{9}$. A slightly better bound $h_5(n) \geq 3\lfloor \frac{n-4}{8} \rfloor$ was presented in [2], which was then sharpened to $h_5(n) \geq \lceil \frac{3}{7}(n-11) \rceil$ in [3]. The latest and so far best bound of $h_5(n) \geq \frac{n}{2} - O(1)$ is due to Valtr [16]. In this section we further improve this bound to $h_5(n) \geq \frac{3}{4}n - o(n)$.

We start by fine-tuning the proof from [3], showing $h_5(n) \geq \lceil \frac{3}{7}(n-11) \rceil$, by utilizing the results $h_5(10) = 1$ [12], $h_5(11) = 2$ [6], and $h_5(12) \geq 3$ [6]. Although this does not lead to an improved lower bound of $h_5(n)$ for large n, it provides better lower bounds for small values of n; see Table 1.

*OA and BV supported by the ESF EUROCORES programme EuroGIGA – CRP 'ComPoSe', Austrian Science Fund (FWF): I648-N18. RFM partially supported by Conacyt of Mexico, grant 153984. TH supported by the Austrian Science Fund (FWF): P23629-N18 'Combinatorial Problems on Geometric Graphs'. CH is partially supported by projects MTM2009-07242, Gen. Cat. DGR 2009SGR1040, and ESF EUROCORES programme EuroGIGA, CRP ComPoSe: MICINN Project EUI-EURC-2011-4306, for Spain. AP is a recipient of a DOC-fellowship of the Austrian Academy of Sciences.

†Institute for Software Technology, University of Technology, Graz, Austria, [oaich|thackl|apilz|bvogt]@ist.tugraz.at

‡Departamento de Matemáticas, Cinvestav, Mexico City, Mexico, ruyfabila@math.cinsvestav.edu.mx

§Departament de Matemàtica Aplicada IV, Universitat Politècnica de Catalunya, Barcelona, Spain, clemens.huemer@upc.edu

n	10	11	12	13	14	15	16	17	18	19	20..23	24	25	26	27..30	31
$h_5(n)$	1	2	3	3..4	3..6	3..9	≥ 3	≥ 4	≥ 5	≥ 6	≥ 6	≥ 7	≥ 8	≥ 9	≥ 9	≥ 10
n	32	33	34..37	38	39	40	41..44	45	46	47	48..50	51	52	53	54	55..56
$h_5(n)$	≥ 11	≥ 12	≥ 12	≥ 13	≥ 14	≥ 15	≥ 15	≥ 16	≥ 17	≥ 18	≥ 18	≥ 19	≥ 19	≥ 20	≥ 21	≥ 21

Table 1: The updated bounds on $h_5(n)$ for small values of n.

Lemma 1 *Every set S of n points in the plane in general position with $n = 7 \cdot m + 9 + t$ (for any natural number $m \geq 0$ and $t \in \{1, 2, 3\}$) contains at least $h_5(n) \geq 3m + t = \frac{3n-27+4t}{7}$ convex 5-holes.*

Proof. Because of $h_5(10) = 1$, $h_5(11) = 2$, and $h_5(12) \geq 3$ this is true for $m = 0$. Obviously $h_5(n) \geq h_5(n-1)$. Hence, $h_5(n) \geq 3$ for any $n \geq 12$.

If there exists a point $p \in ((\partial\,\mathrm{CH}\,(S)) \cap S)$ that is a point of a convex 5-hole, then $h_5(S) \geq 1 + h_5(S \setminus \{p\}) \geq 1 + h_5(n-1)$. In this case, the lemma is true by induction, as for $t = 1$ and $m > 0$, $h_5(n-1) = h_5(7 \cdot m + 9) \geq h_5(7 \cdot (m-1) + 9 + 3)$. (The case $t \in \{2, 3\}$ is trivial.)

Otherwise, each point $p \in ((\partial\,\mathrm{CH}\,(S)) \cap S)$ is not a point of a convex 5-hole. For $m > 0$ choose one such point p (e.g. the bottom-most one) and partition $S \setminus \{p\}$ (in clockwise order around p) into the following successive disjoint subsets: S_0 containing the first 7 points; S'_0 containing the next 4 points; $(m-1)$ pairs of subsets: S_i containing 3 points and S'_i containing 4 points $(1 \leq i \leq (m-1))$; and the subset S_{rem} containing the remaining $(t+4)$ points. See Figure 1 for a sketch.

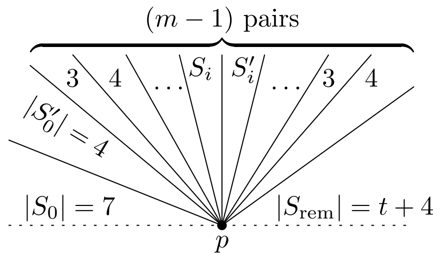


Figure 1: Partition of $S \setminus \{p\}$ clockwise around an extreme point p: starting with the pair S_0, S'_0; continuing with $(m-1)$ pairs of sets S_i, S'_i, for $1 \leq i \leq (m-1)$, with $|S_i| = 3$ and $|S'_i| = 4$; and ending with the remainder set S_{rem}.

The subset $S_0 \cup S'_0 \cup \{p\}$ has cardinality 12 and thus contains at least 3 convex 5-holes. The same is true for each subset $S'_{i-1} \cup S_i \cup S'_i \cup \{p\}$ $(1 \leq i \leq (m-1))$. Finally, the subset $S'_{m-1} \cup S_{\text{rem}} \cup \{p\}$ has cardinality $(9 + t)$ and therefore contains at least t convex 5-holes. Note that we count every convex 5-hole at most once, as the considered subsets of 10, 11, and 12 points, respectively, overlap in at most 4 points. In total this gives at least $3 + (m-1) \cdot 3 + t = 3 \cdot \frac{n-9-t}{7} + t = \frac{3n-27+4t}{7}$ convex 5-holes. $\square$

Corollary 2 *Every set S of 17 points in the plane in general position contains at least $h_5(17) \geq 4$ convex 5-holes.*

Table 1 shows the bounds on $h_5(n)$ obtained by Lemma 1, for some small values of n. By Harborth [12] $h_5(10) = 1$, and by Dehnhardt [6] $h_5(11) = 2$ and $h_5(12) \geq 3$. The bounds for $n = 51$ and for $57 \leq n < 62250$ (not shown in the table) are due to $h_5(n) \geq \lceil \frac{n}{2} \rceil - 7$ from Valtr [16]. The bounds $h_5(12) \leq 3$, $h_5(13) \leq 4$, $h_5(14) \leq 6$, and $h_5(15) \leq 9$ are from [3, 17].

In the following theorem we present an improved lower bound on $h_5(n)$ for larger n.

Theorem 3 *Every set S of $n \geq 12$ points in the plane in general position contains at least $h_5(n) \geq \frac{3n}{4} - n^{\mathrm{ld}\,\frac{11}{6}} + \frac{15}{8} = \frac{3n}{4} - o(n)$ convex 5-holes.*

Proof. For $12 \leq n < 17$ we count three convex 5-holes for S. For $17 \leq n < 24$ we can count four convex 5-holes for S by Corollary 2.

If $n \geq 24$ consider an (almost) halving line ℓ of S which splits S into S_L ($|S_L| = \lceil \frac{n}{2} \rceil$) and S_R ($|S_R| = \lfloor \frac{n}{2} \rfloor$) and does not contain any point of S. See Figure 2.

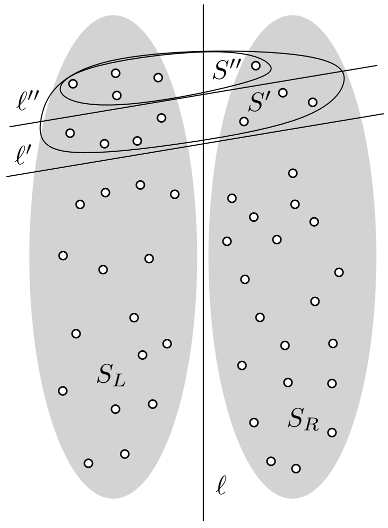


Figure 2: A point set S split by a halving line ℓ into two point sets $S_L, S_R \subset S$. The line ℓ' cuts off a set $S' \subseteq S$, consisting of 8 points of S_L and 4 points of S_R. The line ℓ'' is parallel to ℓ' and halves $S_L \cap S'$.

Furthermore, consider a line ℓ' that intersects ℓ and cuts off a set $S' \subseteq S$, consisting of eight points from S_L

and four points from S_R. That this is in fact possible is folklore, see e.g. Exercise 4.5 (b) in [7]. Let a line ℓ'' be parallel to ℓ' and split $S' \cap S_L$ into two groups of four points, and let $S'' \subset S'$ be the set which is cut off by ℓ''. Note that neither ℓ' nor ℓ'' contain any points of S.

As $|S'| = 12$ we have that S' contains at least three convex 5-holes. We distinguish two cases.

Case 1: S' contains at least three convex 5-holes which are not intersected by ℓ. Then each of these 5-holes contains only points from S_L and thus at least one point above ℓ''. We count the three convex 5-holes for the set S_L and continue on $S \backslash S''$.

Case 2: S' contains at most two convex 5-holes which are not intersected by ℓ. Then at least one convex 5-hole in S' is intersected by ℓ. We count one convex 5-hole for the halving line ℓ and continue on $S \backslash S'$.

Note that in both cases we cut off at least four points from S_L, but at most four points from S_R. Thus, we can repeat this process until we have processed all $\lceil \frac{n}{2} \rceil$ points of S_L. Let c_L be the number of convex 5-holes counted for ℓ when processing S_L. Hence, Case 2 appeared c_L times, and Case 1 appeared at least $\lfloor \frac{1}{4} \cdot (\lceil \frac{n}{2} \rceil - 8c_L) \rfloor - 1$ times. Therefore, the number of convex 5-holes we counted in S_L (i.e., not intersecting ℓ) is $h_5(S_L) \geq 3 \left(\lfloor \frac{1}{4} \left(\lceil \frac{n}{2} \rceil - 8c_L \right) \rfloor - 1 \right)$.

Repeating the same procedure for S_R (exchanging the roles of S_L and S_R), we obtain $h_5(S_R) \geq 3 \left(\lfloor \frac{1}{4} \left(\lfloor \frac{n}{2} \rfloor - 8c_R \right) \rfloor - 1 \right)$, where c_R is the number of convex 5-holes which we counted for ℓ when processing S_R. Note that any convex 5-hole intersected by ℓ, which we counted while processing S_L, might have occurred again when processing S_R. Thus, the total number c of convex 5-holes intersected by ℓ is at least $\max\{c_L, c_R\} \geq \frac{c_L + c_R}{2}$. As $h_5(S) = h_5(S_L) + h_5(S_R) + c$, we obtain

$$\begin{aligned} h_5(S) \geq\ & 3 \cdot \left(\left\lfloor \frac{1}{4} \cdot \left(\left\lceil \frac{n}{2} \right\rceil - 8c_L \right) \right\rfloor - 1 \right) \\ & + 3 \cdot \left(\left\lfloor \frac{1}{4} \cdot \left(\left\lfloor \frac{n}{2} \right\rfloor - 8c_R \right) \right\rfloor - 1 \right) + \frac{c_L + c_R}{2} \ . \end{aligned}$$

Considering that

$$\left\lfloor \frac{\lceil \frac{n}{2} \rceil}{4} \right\rfloor + \left\lfloor \frac{\lfloor \frac{n}{2} \rfloor}{4} \right\rfloor = \begin{cases} 2 \cdot \left\lfloor \frac{\frac{n}{2}}{4} \right\rfloor & \dots\ n \text{ is even} \\ \left\lfloor \frac{\frac{n+1}{2}}{4} \right\rfloor + \left\lfloor \frac{\frac{n-1}{2}}{4} \right\rfloor & \dots\ n \text{ is odd} \end{cases}$$

is $\geq \frac{n}{4} - \frac{6}{4}$ in both cases, careful transformation gives

$$h_5(S) \geq \frac{3n}{4} - 11 \cdot \frac{c_L + c_R}{2} - \frac{21}{2} \qquad (1)$$

as a first lower bound for the number of convex 5-holes in S. Using $h_5(S) = c + h_5(S_L) + h_5(S_R)$, and the fact that the (almost) halving line ℓ splits S such that $|S_L| = \lceil \frac{n}{2} \rceil$ and $|S_R| = \lfloor \frac{n}{2} \rfloor$, we get $h_5(S) \geq \frac{c_L + c_R}{2} + h_5(\lceil \frac{n}{2} \rceil) + h_5(\lfloor \frac{n}{2} \rfloor) \geq \frac{c_L + c_R}{2} + h_5(\lceil \frac{n-1}{2} \rceil) + h_5(\lceil \frac{n-1}{2} \rceil)$, and hence, a second lower bound for $h_5(S)$:

$$h_5(S) \geq \frac{c_L + c_R}{2} + 2 \cdot h_5(\left\lceil \frac{n-1}{2} \right\rceil) \ . \qquad (2)$$

Combining this with the bound (1), we obtain

$$\begin{aligned} h_5(S) \geq \max \Bigg\{ & \left(\frac{3n}{4} - 11 \cdot \frac{c_L + c_R}{2} - \frac{21}{2} \right), \\ & \left(\frac{c_L + c_R}{2} + 2 \cdot h_5(\left\lceil \frac{n-1}{2} \right\rceil) \right) \Bigg\} \ . \end{aligned} \qquad (3)$$

Note that the first term in inequality (3) is strictly monotonically decreasing in $\frac{c_L + c_R}{2}$, while the second term is strictly monotonically increasing in $\frac{c_L + c_R}{2}$. Thus, the minimum of the lower bound in (3) is reached if both bounds are equal.

$$\begin{aligned} \frac{3n}{4} - 11 \cdot \frac{c_L + c_R}{2} - \frac{21}{2} &= \frac{c_L + c_R}{2} + 2 \cdot h_5(\left\lceil \frac{n-1}{2} \right\rceil) \\ \frac{3n}{4} - \frac{21}{2} - 2 \cdot h_5(\left\lceil \frac{n-1}{2} \right\rceil) &= 12 \cdot \frac{c_L + c_R}{2} \\ \frac{c_L + c_R}{2} &= \frac{n}{16} - \frac{7}{8} - \frac{1}{6} \cdot h_5(\left\lceil \frac{n-1}{2} \right\rceil) \end{aligned}$$

Plugging this result for $\frac{c_L + c_R}{2}$ into the lower bound (2) for $h_5(S)$, we obtain a lower bound for $h_5(S)$ for any S with n points. Therefore, this also leads to a lower bound for $h_5(n)$.

$$\begin{aligned} h_5(n) &\geq \frac{n}{16} - \frac{7}{8} - \frac{1}{6} \cdot h_5(\left\lceil \frac{n-1}{2} \right\rceil) + 2 \cdot h_5(\left\lceil \frac{n-1}{2} \right\rceil) \\ &= \frac{n}{16} - \frac{7}{8} + \frac{11}{6} \cdot h_5(\left\lceil \frac{n-1}{2} \right\rceil) \ . \end{aligned} \qquad (4)$$

We show by induction that this recursion resolves to $h_5(n) \geq \frac{3n}{4} - n^{\mathrm{ld} \frac{11}{6}} + \frac{15}{8}$, for $n \geq 12$. We know that $h_5(12), \dots, h_5(16) \geq 3$ and $h_5(17), \dots, h_5(23) \geq 4$ (see first paragraph of this proof). As $\frac{3n}{4} - n^{\mathrm{ld} \frac{11}{6}} + \frac{15}{8}$ is monotonically increasing for $12 \leq n \leq 23$, it is sufficient to check the induction base for $n = 16$ and $n = 23$: $h_5(16) \geq 3 \geq 2.578 \geq \frac{3 \cdot 16}{4} - 16^{\mathrm{ld} \frac{11}{6}} + \frac{15}{8}$ and $h_5(23) \geq 4 \geq 3.609 \geq \frac{3 \cdot 23}{4} - 23^{\mathrm{ld} \frac{11}{6}} + \frac{15}{8}$. For $n \geq 24$ we insert the claim into the recursive formula:

$$\begin{aligned} h_5(n) &\geq \frac{n}{16} - \frac{7}{8} + \frac{11}{6} \cdot h_5(\left\lceil \frac{n-1}{2} \right\rceil) \\ &\geq \frac{n}{16} - \frac{7}{8} + \frac{11}{6} \cdot \left(\frac{3 \frac{n-1}{2}}{4} - \left(\frac{n-1}{2} \right)^{\mathrm{ld} \frac{11}{6}} + \frac{15}{8} \right) \\ &= \frac{3n}{4} + \frac{15}{8} - \frac{11}{6} \cdot \frac{1}{2^{\mathrm{ld} \frac{11}{6}}} \cdot (n-1)^{\mathrm{ld} \frac{11}{6}} \\ &\geq \frac{3n}{4} - n^{\mathrm{ld} \frac{11}{6}} + \frac{15}{8} \ . \end{aligned}$$

The last inequality is true because $(n-1)^{\mathrm{ld} \frac{11}{6}} < n^{\mathrm{ld} \frac{11}{6}}$.

This proves the claim and the theorem as we have: $h_5(n) \geq \frac{3n}{4} - n^{0.87447} + 1.875 = \frac{3n}{4} - o(n)$. $\square$

3 Empty triangles and convex 4-holes

For this section we are going to use some definitions and notation used in [15, 9, 10]. Let S be a set of n points in the plane in general position. We need to define a total order on the points of S. In addition, this order has to define a line ℓ_q through every point $q \in S$, such that each point $r \in S$ is either in the closed halfplane "below" ℓ_q, i.e., $q \geq r$, or in the open halfplane "above" ℓ_q, i.e., $q < r$. In [10] the points of S are sorted in increasing order of the ordinate y (with the additional restriction that no two points have equal ordinate). Observe though, that of course any direction is a valid order for the points of S. Furthermore, observe that also a cyclic order around some point $p \in ((\partial \operatorname{CH}(S)) \cap S)$ is a valid order for the points of $S \backslash \{p\}$, as there exists a line ℓ through p, such that all points of $S \backslash \{p\}$ are in an open halfplane bounded by ℓ. This will be crucial for the proof of Lemma 6 where we will order the points of a set $S \backslash \{p\}$ around such a point p. Note that, because of the general position assumption for S, no two points in $S \backslash \{p\}$ are equivalent in this order. Anyhow, for simplicity, and apart from the aforementioned exception, we will use the order along the ordinate of S, as in [10].

Let P be a convex 5-hole spanned by points of S and let v be the *top vertex* of P, i.e., the vertex of P with highest order. We name an empty triangle *generated by* P if it is spanned by v and the two vertices of P that are not adjacent (on the boundary of P) to v. Let $h_{3|5}(S)$ be the number of such triangles determined by S, and let $h_{3|5}(n) = \min_{|S|=n} h_{3|5}(S)$ be the number of empty triangles generated by convex 5-holes that every set of n points spans at least. Likewise, we name a convex 4-hole *generated by* P if it is spanned by all vertices of P except for one of the two vertices of P that are adjacent (on the boundary of P) to v. Observe that each convex 5-hole generates two convex 4-holes by this definition. Let $h_{4|5}(S)$ be the number of such 4-holes determined by S, and let $h_{4|5}(n) = \min_{|S|=n} h_{4|5}(S)$ be the number of convex 4-holes generated by convex 5-holes that every set of n points spans at least.

García [10] recently proved that $h_3(S) = n^2 - 5n + H + 4 + h_{3|5}(S) \geq n^2 - 5n + H + 4 + h_{3|5}(n)$ and $h_4(S) = \frac{n^2}{2} - \frac{7n}{2} + H + 3 + h_{4|5}(S) \geq \frac{n^2}{2} - \frac{7n}{2} + H + 3 + h_{4|5}(n)$, where H is the number of points of $((\partial \operatorname{CH}(S)) \cap S)$. Consequently, this gives $h_3(n) \geq n^2 - 5n + 7 + h_{3|5}(n)$ and $h_4(n) \geq \frac{n^2}{2} - \frac{7n}{2} + 6 + h_{4|5}(n)$, as $H \geq 3$. Observe that this implies that $h_{3|5}(S)$ and $h_{4|5}(S)$ (and of course $h_{3|5}(n)$ and $h_{4|5}(n)$) do not depend on the chosen order of the points. As changing the order does not change the point set, $h_3(S)$ and $h_4(S)$ are of course independent of the order. Furthermore, García proved that the number of empty triangles (or convex 4-holes) not generated by convex 5-holes is an invariant of the point set. Hence, although the empty triangles and convex 4-holes generated by convex 5-holes may change with different orders, their numbers stay the same.

Proving $h_{3|5}(n) \geq 3 \cdot \left\lfloor \frac{n-4}{8} \right\rfloor$ and $h_{4|5}(n) \geq 6 \cdot \left\lfloor \frac{n-4}{8} \right\rfloor$, García presented the improved bounds $h_3(n) \geq n^2 - \frac{37n}{8} + \frac{23}{8}$ and $h_4(n) \geq \frac{n^2}{2} - \frac{11n}{4} - \frac{9}{4}$. We will improve these bounds on $h_{3|5}(n)$ and $h_{4|5}(n)$. Showing that for each convex 5-hole counted in Lemma 1 we may count one empty triangle generated by convex 5-holes and two convex 4-holes generated by convex 5-holes will already give an improved bound for both, $h_{3|5}(n)$ and $h_{4|5}(n)$. But using a slightly adapted version of the proof from Theorem 3 will improve the bound on $h_{4|5}(n)$ even further. To this end we have to first prove the base case, i.e., sets of 10, 11, and 12 points.

Having a close look at the example shown in Figure 3, one can see that as soon as the triangle $\triangle$ (or the convex 4-hole $\diamond$) is generated by more than one convex 5-hole, there must exist at least one convex 6-hole. We state this fact in more detail and prove it in the following lemma. Note that a similar approach and figure has been used in [10].

Lemma 4 *Let S be a set of $n \geq 6$ points in the plane in general position. Let $\triangle$ ($\diamond$) be an empty triangle (a convex 4-hole) of S. If $\triangle$ ($\diamond$) is generated by at least two convex 5-holes, $\pentagon_1$ and $\pentagon_2$, of S, then there exists at least one convex 6-hole, $\hexagon_1$, of S, containing $\pentagon_1$, and one convex 6-hole, $\hexagon_2$, of S, containing $\pentagon_2$, where $\hexagon_1 = \hexagon_2$ is possible.*

Proof. See Figure 3 (top). Assume that there exists at least one empty triangle, $\triangle = \langle p_i, p_j, p_k \rangle$, with p_k being the top vertex, that is generated by two different convex 5-holes. Let one of them, $\pentagon_1$, be spanned by the points p_i, p_j, p_L, p_k, p_R (the points shown as full dots in the figure). As $\triangle$ is generated by another convex 5-hole, $\pentagon_2$, there must be at least one additional point in one of the regions L_h, L_l, R_h, and R_l. Otherwise, the new pentagon would not be empty, not be convex, or $\triangle$ would not be generated by it (recall that p_k must be the highest point). W.l.o.g. assume that there exists at least one point p_{new} in R_l. It is easy to see that in this case there exists a convex 4-hole spanned by the points p_i, p_k, p_R, p'_R ($p'_R = p_{new}$ is possible, but not necessary). Together with p_j and p_L this forms a convex 6-hole which contains $\pentagon_1$. Starting the argument with $\triangle$ being generated by $\pentagon_2$, proves that also $\pentagon_2$ is contained in a convex 6-hole.

The argumentation is analogous for a convex 4-hole, $\diamond$, that is generated by two different convex 5-holes. See Figure 3 (bottom). The only difference to the previous case (with $\triangle$) is that the additional point p_{new} can not exist in either L_l or R_l, depending on which convex 4-hole (either $\diamond = \langle p_i, p_j, p_L, p_k \rangle$ or $\diamond = \langle p_i, p_j, p_k, p_R \rangle$) is considered. The former situation is depicted in Figure 3 (bottom). □

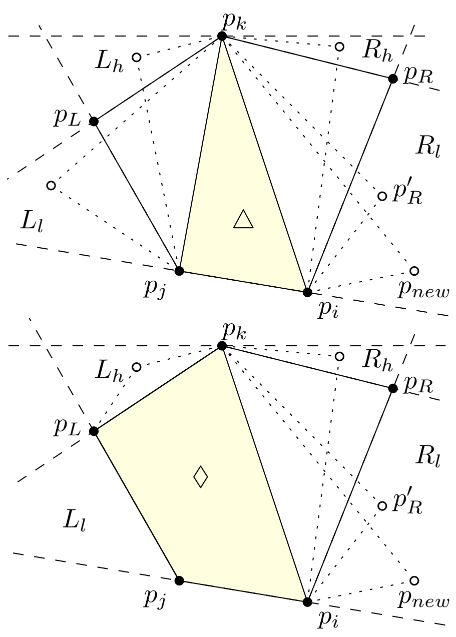


Figure 3: Auxiliary figure for the proof of Lemma 4.

Using Lemma 4 we are able to provide the base cases $10 \leq n \leq 12$ for $h_{3|5}(n)$ and $h_{4|5}(n)$. The proof is omitted in this extended abstract.

Lemma 5 *Every set of 10, 11, or 12 points in the plane in general position contains (i) at least 1, 2, and 3, respectively, different empty triangles generated by convex 5-holes (i.e., $h_{3|5}(10) = 1$, $h_{3|5}(11) = 2$, and $h_{3|5}(12) = 3$) and (ii) at least 2, 4, and 6, respectively, different convex 4-holes generated by convex 5-holes (i.e., $h_{4|5}(10) = 2$, $h_{4|5}(11) = 4$, and $h_{4|5}(12) = 6$).*

These base cases allow a lemma similar to Lemma 1. The proof follows the lines of the proof of Lemma 1 and is omitted in this extended abstract.

Lemma 6 *Every set S of n points in the plane in general position with $n = 7 \cdot m + 9 + t$ (for any natural number $m \geq 0$ and $t \in \{1, 2, 3\}$) contains at least $h_{3|5}(n) \geq \frac{3n-27+4t}{7}$ empty triangles generated by convex 5-holes and at least $h_{4|5}(n) \geq 2 \cdot \frac{(3n-27+4t)}{7}$ convex 4-holes generated by convex 5-holes.*

As mentioned above, this lemma already improves the bounds for $h_{3|5}(n)$ and $h_{4|5}(n)$. We will further improve the bound for $h_{4|5}(n)$ in Theorem 8. In the following theorem we state only the bound for $h_{3|5}(n)$.

Theorem 7 *Every set S of $n \geq 12$ points in the plane in general position contains at least $h_{3|5}(n) \geq 3 \cdot \left\lfloor \frac{n-12}{7} \right\rfloor + 3 + f(|S_{rem}|) \geq \left\lceil \frac{3n-27}{7} \right\rceil$ empty triangles generated by convex 5-holes. The point set $S_{rem} \subset S$ is the remainder set with $0 \leq |S_{rem}| \equiv (n-12) \mod 7 \leq 6$, and $f(0 \ldots 4) = 0$, $f(5) = 1$, and $f(6) = 2$.*

Proof. The first inequality in the bound, $h_{3|5}(n) \geq 3 \cdot \left\lfloor \frac{n-12}{7} \right\rfloor + 3 + f(|S_{\text{rem}}|)$, is simply a reformulation of the bound in Lemma 6. The second inequality results from taking the minimum of the first inequality over all possible values for $|S_{\text{rem}}|$. (This minimum is obtained by $|S_{\text{rem}}| = 4$.) □

The basic principles of the proof of the following theorem are the same as in the proof of Theorem 3. The main difference is that, for excluding over-counting, a slightly different counting is needed. The proof is omitted in this extended abstract and we only state the result.

Theorem 8 *Every set S of $n \geq 12$ points in the plane in general position contains at least $h_{4|5}(n) \geq \frac{5n}{4} - \frac{383}{303} \cdot n^{\operatorname{ld}\frac{19}{10}} + \frac{55}{24} = \frac{5n}{4} - o(n)$ convex 4-holes generated by convex 5-holes.*

Remark: To use the principles of the proof of Theorem 3 also for empty triangles generated by convex 5-holes, a very disadvantageous splitting is necessary to avoid over-counting. This would lead to a bound inferior to the one from Theorem 7.

Recall that García [10] recently proved $h_3(S) \geq n^2 - 5n + H + 4 + h_{3|5}(n)$ and $h_4(S) \geq \frac{n^2}{2} - \frac{7n}{2} + H + 3 + h_{4|5}(n)$. Combining these results with Theorem 7 and Theorem 8 we can state the following corollary.

Corollary 9 *Every set S of $n \geq 12$ points in the plane in general position and with H points on the boundary of its convex hull contains at least $h_3(S) \geq n^2 - 5n + H + 4 + \left\lceil \frac{3n-27}{7} \right\rceil$ empty triangles and at least $h_4(S) \geq \frac{n^2}{2} - \frac{9n}{4} - \frac{383}{303} \cdot n^{\operatorname{ld}\frac{19}{10}} + H + \frac{127}{24}$ convex 4-holes. Consequently, $h_3(n) \geq n^2 - \frac{32n}{7} + \frac{22}{7}$ and $h_4(n) \geq \frac{n^2}{2} - \frac{9n}{4} - 1.2641\, n^{0.926} + \frac{199}{24}$.*

4 Conclusion

In this paper we improved the lower bounds on the least number $h_k(n)$ of convex k-holes any set of n points contains, for $3 \leq k \leq 5$. The question whether there exists a super-linear lower bound for the number of convex 5-holes remains unsettled, though.

Still, we are able to answer two questions that Dehnhardt [6] asked in 1987. Already in [3] a set of 12 points containing only three convex 5-holes has been presented, implying $h_5(12) = 3$. This disproved Dehnhardt's conjecture of $h_5(12) = 4$. Recall that we know from García [10], that $h_3(S) = n^2 - 5n + H + 4 + h_{3|5}(S)$ and $h_4(S) = \frac{n^2}{2} - \frac{7n}{2} + H + 3 + h_{4|5}(S)$, where $h_{3|5}(S)$ ($h_{4|5}(S)$) is the number of empty triangles (convex 4-holes) generated by convex 5-holes in S.

Consider the set S_{12} with $n = 12$ points and $H = 3$, depicted in Figure 4. It can be easily checked that

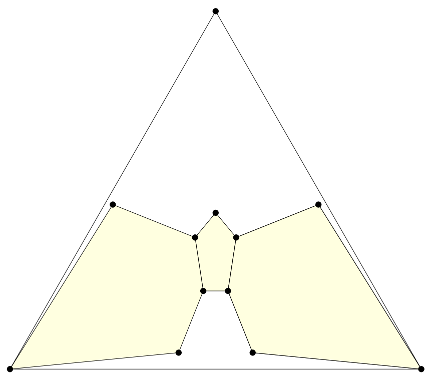


Figure 4: Set of 12 points with triangular convex hull, generating the minimal number of 3-holes (94), convex 4-holes (42), and convex 5-holes (3). The coordinates (x, y) of the 12 points are: (0, 0); (100, 0); (50, 87); (50, 38); (55, 32); (53, 19); (47, 19); (45, 32); (41, 4); (59, 4); (25, 40); (75, 40).

this point set contains only the 3 shown convex 5-holes. Hence, $h_{3|5}(S_{12}) = 3$ and $h_{4|5}(S_{12}) = 6$, as by Lemma 5 $h_{3|5}(12) = 3$ and $h_{4|5}(12) = 6$. Inserting into the above equations, we get $h_3(S_{12}) = h_3(12) = 144-60+3+4+3 = 94$ and $h_4(S_{12}) = h_4(12) = 72-42+3+3+6 = 42$, as $h_3(12) \geq 94$ and $h_4(12) \geq 42$ (by [6]). Of course, $h_3(S_{12})$ and $h_4(S_{12})$ can also be derived by counting all empty triangles and convex 4-holes in S_{12}. This disproves two conjectures of Dehnhardt in [6], namely $h_3(12) = 95$ and $h_4(12) = 44$.

Furthermore, his question for a set of n points that minimizes at least one of $h_3(n)$, $h_4(n)$, and $h_5(n)$, but not all of them is answered by the set of 12 points presented in [3], which has only 3 convex 5-holes but contains (non-minimal) 95 empty triangles and (non-minimal) 43 convex 4-holes.

Acknowledgments

We thank Alfredo García for valuable discussions and an anonymous referee for helpful comments.

References

[1] O. Aichholzer. [Empty] [colored] k-gons - Recent results on some Erdős-Szekeres type problems. In *Proc. XIII Encuentros de Geometría Computacional ECG2009*, pages 43–52, Zaragoza, Spain, 2009.

[2] O. Aichholzer, T. Hackl, and B. Vogtenhuber. On 5-holes and 5-gons. In *Proc. XIV Encuentros de Geometría Computacional ECG2011*, pages 7–10, Alcalá de Henares, Spain, 2011.

[3] O. Aichholzer, T. Hackl, and B. Vogtenhuber. On 5-holes and 5-gons. In A. Marquez, P. Ramos, and J. Urrutia, editors, *Special issue: XIV Encuentros de Geometría Computacional ECG2011*, Lecture Notes in Computer Science (LNCS), page to appear. Springer, 2012.

[4] I. Bárány and G. Károlyi. Problems and results around the Erdős–Szekeres convex polygon theorem. In J. Akiyama, M. Kano, and M. Urabe, editors, *Discrete and Computational Geometry*, volume 2098 of *Lecture Notes in Computer Science (LNCS)*, pages 91–105. Springer, 2001.

[5] I. Bárány and P. Valtr. Planar point sets with a small number of empty convex polygons. *Studia Scientiarum Mathematicarum Hungarica*, 41(2):243–266, 2004.

[6] K. Dehnhardt. *Leere konvexe Vielecke in ebenen Punktmengen.* PhD thesis, TU Braunschweig, Germany, 1987. In German.

[7] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer, 1987.

[8] P. Erdős. Some more problems on elementary geometry. *Australian Mathematical Society Gazette*, 5:52–54, 1978.

[9] A. García. A note on the number of empty triangles. In *Proc. XIV Encuentros de Geometría Computacional ECG2011*, pages 101–104, Alcalá de Henares, Spain, 2011.

[10] A. García. A note on the number of empty triangles. In A. Marquez, P. Ramos, and J. Urrutia, editors, *Special issue: XIV Encuentros de Geometría Computacional ECG2011*, Lecture Notes in Computer Science (LNCS), page to appear. Springer, 2012.

[11] T. Gerken. Empty convex hexagons in planar point sets. *Discrete and Computational Geometry*, 39(1–3):239–272, 2008.

[12] H. Harborth. Konvexe Fünfecke in ebenen Punktmengen. *Elemente der Mathematik*, 33:116–118, 1978. In German.

[13] J. Horton. Sets with no empty convex 7-gons. *Canadian Mathematical Bulletin*, 26(4):482–484, 1983.

[14] C. Nicolás. The empty hexagon theorem. *Discrete and Computational Geometry*, 38(2):389–397, 2007.

[15] R. Pinchasi, R. Radoičić, and M. Sharir. On empty convex polygons in a planar point set. *J. Comb. Theory Ser. A*, 113:385–419, April 2006.

[16] P. Valtr. On empty pentagons and hexagons in planar point sets. In *Proc. 18th Computing: Australasian Theory Symposium CATS2012*, pages 47–48, Melbourne, Australia, 2012.

[17] B. Vogtenhuber. *Combinatorial Aspects of [Colored] Point Sets in the Plane.* PhD thesis, Institute for Software Technology, Graz University of Technology, Graz, Austria, 2011.

What makes a Tree a Straight Skeleton?[*]

Oswin Aichholzer[†] Howard Cheng[‡] Satyan L. Devadoss[§] Thomas Hackl[†] Stefan Huber[¶]
Brian Li[§] Andrej Risteski[‖]

Abstract

Let G be a cycle-free connected straight-line graph with predefined edge lengths and fixed order of incident edges around each vertex. We address the problem of deciding whether there exists a simple polygon P such that G is the straight skeleton of P. We show that for given G such a polygon P might not exist, and if it exists it might not be unique. For the later case we give an example with exponentially many suitable polygons. For small star graphs and caterpillars we show necessary and sufficient conditions for constructing P.

Considering only the topology of the tree, that is, ignoring the length of the edges, we show that any tree whose inner vertices have degree at least 3 is isomorphic to the straight skeleton of a suitable convex polygon.

1 Introduction

The straight skeleton $\mathcal{S}(P)$ of a simple polygon P is a skeleton structure like the Voronoi diagram, but consists of straight-line segments only. Its definition is based on a so-called *wavefront propagation* process that corresponds to mitered offset curves. Each edge e of P emits a wavefront that moves with unit speed to the interior of P. Initially, the wavefront of P consists of parallel copies of all edges of P. However, during the wavefront propagation, topological changes occur: An *edge event* happens if a wavefront edge shrinks to zero length. A *split event* happens if a reflex wavefront vertex meets a

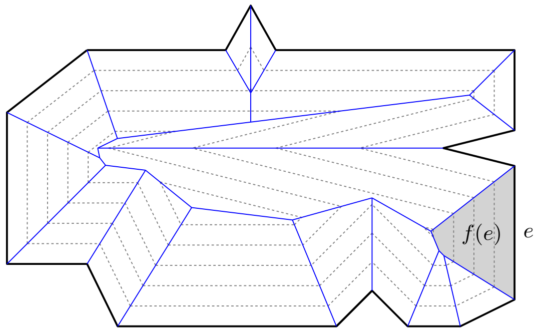


Figure 1: The straight skeleton (thin) of a simple polygon (bold) is defined by the propagating wavefront (dotted).

wavefront edge and splits the wavefront into pieces, see Figure 1. The *straight skeleton* $\mathcal{S}(P)$ is defined as the set of loci that are traced out by the wavefront vertices and it partitions P into polygonal faces. Each face $f(e)$ belongs to a unique edge e of P. Each straight-skeleton edge belongs to two faces, say $f(e_1)$ and $f(e_2)$, and lies on the bisector of e_1 and e_2.

Straight skeletons have many applications, like automated roof construction, computation of mitered offset curves, topology-preserving collapsing of areas in geographic maps, or solving fold-and-cut problems. See [4] and Chapter 5.2 in [3] for further information and detailed definitions.

Although straight skeletons were introduced to computational geometry in 1995 by Aichholzer et al. [1], their roots actually go back to the 19th century. In textbooks about the construction of roofs (see e.g. [6], pages 86–122) using the angle bisectors (of the polygon defined by the ground walls) was suggested to design roofs where rainwater can run off in a controlled way. This construction is called *Dachausmittlung* and became rather popular. See [5] for related and partially more involved methods to obtain roofs from the ground plan of a house. In this book detailed explanations of the constructions and drawings of the resulting roofs can be found.

Maybe not surprisingly, none of this early work mentions the ambiguity of the non-algorithmic definition of the construction. It can be shown that using solely bisector graphs does not necessarily lead to a unique roof

[*]A preliminary version of this paper appeared at EuroCG 2012. Research of O. Aichholzer partially supported by the ESF EUROCORES programme EuroGIGA – ComPoSe, Austrian Science Fund (FWF): I 648-N18. T. Hackl was funded by the Austrian Science Fund (FWF): P23629-N18. S. Huber was funded by the Austrian Science Fund (FWF): L367-N15. H. Cheng, S.L. Devadoss, B. Li, and A. Risteski were funded by NSF grant DMS-0850577.

[†]Institute for Software Technology, Graz University of Technology, `[oaich|thackl]@ist.tugraz.at`

[‡]University of Arizona, Tucson, AZ 85721, `howardc@email.arizona.edu`

[§]Williams College, Williamstown, MA 01267, `[satyan.devadoss|brian.t.li]@williams.edu`

[¶]Department of Mathematics, Universität Salzburg, Austria, `shuber@cosy.sbg.ac.at`

[‖]Princeton University, Princeton, NJ 08544, `ristesk@princeton.edu`

construction, and actually does not even guarantee a plane partition of the interior of the defining boundary. See [1] for a detailed explanation and examples.

An interesting inverse problem was motivated to us by Lior Pachter and investigations started in [2]: Which graphs are the straight skeleton of some polygon? In other words, how can straight skeletons be characterized among all graphs?

2 Finding straight skeletons of given topology

First of all, the straight skeleton $\mathcal{S}(P)$ of a simple polygon is known to be connected and cycle-free. Hence, we can rephrase our question as follows: which trees T are realized as straight skeletons of simple polygons? At first we concentrate on the topological structure of trees only and ignore their geometry.

Theorem 1 *For any tree T, whose inner vertices have at least degree 3, there exists a feasible (convex) polygon P such that $\mathcal{S}(P)$ possesses the same topology as T.*

Note that within convex polygons the straight skeleton and the Voronoi diagram are identical. Hence, by the above theorem, any tree is also isomorphic to the Voronoi diagram of a suitable convex polygon.

Proof. We first choose any inner vertex v of T to be the root of T. Then we construct a regular polygon P_1 with $d(v)$ sides, where $d(v)$ denotes the degree of v. The straight skeleton $\mathcal{S}(P_1)$ is a star graph comprising one inner node and $d(v)$ incident edges, which correspond to v and its incident edges in T. It remains to attach the corresponding subtrees from T to each leaf vertex of $\mathcal{S}(P_1)$, if there are any.

In the remainder of the proof we describe an inductive step by which we locally transform a polygon P_i to P_{i+1} in order to attach to a leaf vertex u of $\mathcal{S}(P_i)$ a missing number $k = d(u) - 1$ of incident edges, where $d(u)$ denotes the degree of the vertex of T that corresponds to u. Applying this technique recursively — e.g., in a breadth-first search fashion starting from v — gives us finally a polygon P_m, where m denotes the number of inner nodes of T, whose straight skeleton $\mathcal{S}(P_m)$ is topologically equivalent to T by construction. Furthermore, in our induction step we guarantee that all polygons $P_1, \ldots, P_m$ remain convex.

For the induction step, we consider a leaf vertex u in $\mathcal{S}(P_i)$ and we denote by e the incident straight-skeleton edge of u. As u is a leaf vertex of $\mathcal{S}(P_i)$ it is also a polygon vertex of P_i. Hence, e lies on the straight-skeleton faces $f(s_1)$ and $f(s_2)$ of the two incident polygon edges s_1 and s_2 of u. Since P_i is convex by induction, all faces of $\mathcal{S}(P_i)$ are convex, too. Hence, the projection lines of the mid point u' of e onto s_1 and s_2 are completely contained in $f(s_1)$ and $f(s_2)$, respectively. Let us consider the circular arc C that is centered at u' and tangential to s_1 and s_2 such that C forms a round convex cap of the corner u of P_i, see Figure 2. In the induction step, we locally bevel the corner u of P_i by any convex polygonal chain with $k \geq 2$ vertices that is tangential to C. By that the edge e is truncated to u' and we obtain k additional straight-skeleton edges $e_1, \ldots, e_k$ that are incident to u', as desired. The resulting polygon P_{i+1} is again convex and a locally beveled version of P_i. Note that the remaining straight skeleton of P_i remains unchanged for P_{i+1}. ☐

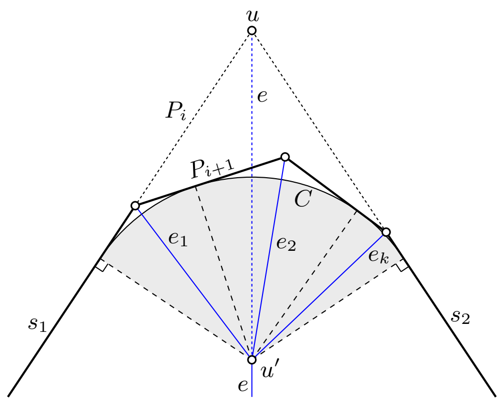


Figure 2: The induction step in order to add k edges to a terminal vertex of $\mathcal{S}(P_i)$, with $k = 3$, by beveling the corner u of P_i.

3 Abstract geometric trees

The original problem motivated by Lior Pachter and for which investigations started in [2] does not only ask for a specific topology of $\mathcal{S}(G)$, but also asks for certain geometric requirements that are to be fulfilled by $\mathcal{S}(P)$. In particular, we want to find a polygon P for which (i) $\mathcal{S}(P)$ has a specific topology, (ii) the edges of $\mathcal{S}(P)$ have a specific length and (iii) the cyclic order of incident edges at vertices of $\mathcal{S}(P)$ is given.

To give a more formal problem definition we denote with *abstract geometric graphs* the set of combinatorial graphs, where the length of each edge and the cyclic order of incident edges around every vertex is predefined (and cannot be altered). Let $\mathcal{G}$ be the set of cycle-free connected abstract geometric graphs. Denote with $E(G)$ an embedding of $G \in \mathcal{G}$ in the plane, that is, the vertices of G are points in $\mathbb{R}^2$ and the edges of G are straight-line segments of the predefined length, connecting the corresponding points and respecting the predefined cyclic order of incident edges around each vertex. Further, denote with $P_{E(G)}$ the polygon resulting from connecting the leaves of G (with straight-line segments) in cyclic order for the embedding $E(G)$. We call a simple polygon $P_{E(G)}$ *suitable* if its straight skeleton $\mathcal{S}(P_{E(G)}) = E(G)$, for the embedding $E(G)$. If there

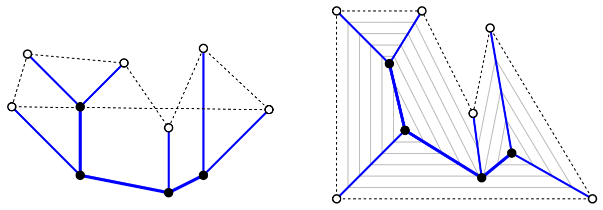


Figure 3: Example of a feasible cycle-free connected abstract geometric graph G (leaves of G are shown as white dots). Left: Arbitrary embedding $E(G)$ and (non-simple) polygon $P_{E(G)}$ (dotted). Right: Suitable polygon $P_{E'(G)}$ for a different embedding $E'(G)$, which is equal to $\mathcal{S}(P_{E'(G)})$. A set of wavefronts of $P_{E'(G)}$ at different points in time are depicted in gray.

exists a suitable polygon for a graph $G \in \mathcal{G}$, we call G *feasible*, see Figure 3.

The obvious questions which arise from these definitions are: Which graphs of $\mathcal{G}$ are feasible? Are the suitable polygons for feasible graphs unique modulo rigid motions? How can one construct a suitable polygon for a feasible graph?

3.1 Star graphs

All polygon edges whose straight-skeleton faces contain a common vertex u (of the straight skeleton) have equal orthogonal distance t to u, because their wavefront edges reach u at the same time t. That is, the supporting lines of those polygon edges are tangential to the circle with center u and radius t. Thus, in this section we consider a subset of $\mathcal{G}$, the so called star graphs. A *star graph* $S_n \in \mathcal{G}$, for $n \geq 3$ has $(n+1)$ vertices, one vertex u with degree n and n leaves $v_1, \ldots, v_n$ ordered counter-clockwise around u. The length of each edge uv_i, with $1 \leq i \leq n$, is denoted by l_i. W.l.o.g. let $l_1 = \max_i l_i$. Observe that the polygon $P_{E(S_n)}$ is star shaped and $v_i v_{i+1}$ (with $v_{n+k} := v_{1+(k-1) \bmod n}$) are its edges.

Observation 1 *If $S_n \in \mathcal{G}$ is a feasible star graph and $P_{E(S_n)}$ is a suitable polygon of S_n, then (1) all straight-skeleton faces are triangles, (2) two consecutive vertices v_i, v_{i+1} can not both be reflex, (3) $l_i < l_{i\pm1}$ for each reflex vertex v_i of $P_{E(S_n)}$, and (4) all edges of $P_{E(S_n)}$ have equal orthogonal distance t to u, with $t \in (0, \min_i l_i]$.*

As a given $S_n \in \mathcal{G}$ is possibly not feasible and a suitable polygon may not be known or might not exist, we define a polyline $L_{S_n}(t, A)$: The vertices $v_1, \ldots, v_{n+1}$ of $L_{S_n}(t, A)$ are the leaves, $v_1, \ldots, v_n$, of S_n, in the same order as for S_n, and one additional vertex v_{n+1} succeeding v_n. The vertices $v_1, \ldots, v_n, v_{n+1}$ have the corresponding distances (predefined in S_n) $l_1, \ldots, l_n, l_1$ to u. A is an assignment for each vertex whether it should

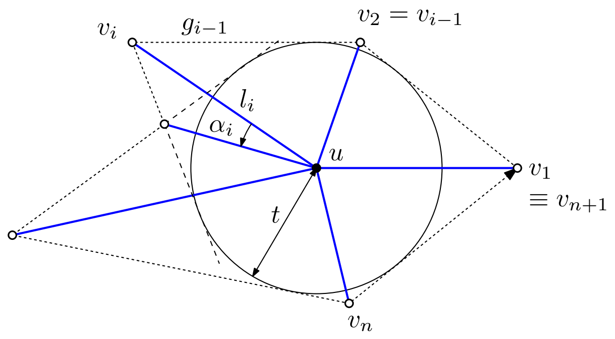


Figure 4: Construction of $L_{S_n}(t, A)$ (and $E(S_n)$) for a given S_n and a fixed distance t and assignment A.

be convex or reflex, as seen from u. As $l_1 = \max_i l_i$, v_1 and v_{n+1} are always convex (fact (3) in Observation 1). For the remaining vertices any convex/reflex assignment, which respects the facts (2) and (3) in Observation 1, can be considered. The edges of $L_{S_n}(t, A)$ have equal orthogonal distance t to u. Of course, not all possible combinations of t and an arbitrary embedding $E(S_n)$ allow such a polyline, but it is possible to construct $L_{S_n}(t, A)$ and $E(S_n)$ simultaneously for a fixed $t \in (0, \min_i l_i]$.

For a fixed assignment A and a fixed $t \in (0, \min_i l_i]$ we construct $L_{S_n}(t, A)$ (and $E(S_n)$) in the following way. Consider the circle C with center u and radius t. Start with v_1 at polar coordinate $(l_1, 0)$, with u as origin. For each v_i, $i = 2 \ldots (n+1)$, consider a tangent g_{i-1} to C (such that the vertices will be placed counter-clockwise around the circle) through v_{i-1}. If v_{i-1} is convex, then there exist two points with distance l_i (l_1 for v_{n+1}) on g_{i-1}. If v_i is assigned to be reflex, then v_i is placed on the point closer to v_{i-1}, and if v_i is assigned to be convex, then v_i is placed on the other point. If v_{i-1} is reflex, then there exists only one applicable point for placing v_i on g_{i-1}. See Figure 4.

The $L_{S_n}(t, A)$ constructed this way is unique (for fixed t and A), and may be not simple (e.g. when circling C many times), simple but not closed ($v_{n+1} \not\equiv v_1$), or simple and closed ($v_{n+1} \equiv v_1$). In the latter case, the construction reveals a witness pair (t, A) for the existence of some $E(S_n)$, a suitable polygon $P_{E(S_n)}$, and thus the feasibility of S_n.

It is easy to see that for each suitable polygon $P_{E(S_n)}$, there exists a polyline $L_{S_n}(t, A)$ (just duplicate the vertex v_1). Hence, deciding feasibility of S_n is equivalent to finding an assignment A and a $t \in (0, \min_i l_i]$ such that $L_{S_n}(t, A)$ is closed and simple. For a polyline $L_{S_n}(t, A)$ and a corresponding embedding $E(S_n)$, we denote with α_i, $i = 1 \ldots n$, the counter-clockwise angle at u, spanned by uv_i and uv_{i+1}. Note that for a suitable polygon $P_{E(S_n)}$ α_i can be defined the same way, with $v_{n+1} \equiv v_1$. It is easy to see that the sum of all α_i is 2π if and only if $L_{S_n}(t, A)$ is closed and simple.

Lemma 2 *Let $S_n \in \mathcal{G}$, distance $t \in (0, \min_i l_i]$ and assignment A be fixed, and let $L_{S_n}(t, A)$ be the resulting polyline. Then $\alpha_A(t) := \sum_{i=1}^n \alpha_i$ can be expressed as*

$$\alpha_A(t) = 2 \sum_{\substack{i=1 \\ v_i \text{ convex}}}^{n} \arccos\frac{t}{l_i} - 2 \sum_{\substack{i=1 \\ v_i \text{ reflex}}}^{n} \arccos\frac{t}{l_i}\ . \quad (1)$$

Proof. Recall that v_1 and v_{n+1} are convex by the assumption $l_1 = l_{n+1} = \max_i l_i$. It is easy to see that $\alpha_i = \arccos\frac{t}{l_i} + \arccos\frac{t}{l_{i+1}}$ if v_i is convex and v_{i+1} is convex, $\alpha_i = \arccos\frac{t}{l_i} - \arccos\frac{t}{l_{i+1}}$ if v_i is convex and v_{i+1} is reflex, and $\alpha_i = -\arccos\frac{t}{l_i} + \arccos\frac{t}{l_{i+1}}$ if v_i is reflex and v_{i+1} is convex. If v_i is reflex then $\alpha_{i-1} + \alpha_i$ sums up to $\left(\arccos\frac{t}{l_{i-1}} + \arccos\frac{t}{l_i}\right) + \left(\arccos\frac{t}{l_i} + \arccos\frac{t}{l_{i+1}}\right) - 4\arccos\frac{t}{l_i}$, because $v_{i\pm1}$ are both convex (fact (2) in Observation 1). Thus, summing over all α_i results in the claimed formula. □

We define a suitable polygon to be *unique* if it is the only suitable polygon modulo rigid motions. For the following result we use the first derivative of α_A:

$$\alpha'_A(t) = 2 \sum_{\substack{i=1 \\ v_i \text{ reflex}}}^{n} \frac{1}{\sqrt{l_i^2 - t^2}} - 2 \sum_{\substack{i=1 \\ v_i \text{ convex}}}^{n} \frac{1}{\sqrt{l_i^2 - t^2}}. \quad (2)$$

Lemma 3 *A suitable convex polygon for a star graph S_n exists if and only if $\sum_i \arccos\frac{\min_i l_i}{l_i} \leq \pi$. If a suitable convex polygon exists then it is unique.*

Proof. As all vertices are assumed to be convex, we obtain $\alpha_A(0) = n\pi > 2\pi$. Furthermore, we observe that $\alpha_A(t)$ is monotonically decreasing since $\alpha'_A(t) < 0$ for all $t \in (0, \min_i l_i]$. Hence, there is a $t \in (0, \min_i l_i]$ with $\alpha(t) = 2\pi$ if and only if $\alpha_A(\min_i l_i) \leq 2\pi$ which is $\sum_i \arccos\frac{\min_i l_i}{l_i} \leq \pi$. If this is the case the solution is unique as $\alpha(t)$ is monotonic. □

For $n = 3$, $\alpha_A(0) = 3\pi$ and $\alpha_A(\min_i l_i) < 2\pi$, and thus we immediately get the following corollary.

Corollary 4 *For every S_3 there exists a unique suitable convex polygon.*

Considering star graphs with $n = 5$, we show in the following lemma that they are not always feasible, and that suitable polygons (if they exist) are not always unique.

Lemma 5 *There exist infeasible star graphs, $S_n \in \mathcal{G}$. Further, there exist feasible star graphs for which multiple suitable polygons exist.*

Proof. To prove the first claim consider a star graph with $n = 5$, $l_1 = l_2 = l_3 = l_4 = 1$, and $l_5 = 0.25$. There exist only two possible assignments: either all vertices convex or all but v_5 convex. It is easy to check that for both assignments $\sum_i \alpha_i > 2\pi$, for every $t \in (0, \min_i l_i]$. To prove the second claim consider a star graph with $n = 5$, $l_1 = l_3 = 1$, $l_2 = 0.6$, $l_4 = 0.79$, and $l_5 = 0.75$. Assign all vertices convex, except for v_2. Then $\sum_i \alpha_i$ evaluates to 2π for $t \approx 0.537$ and $t \approx 0.598$. Hence, there exist (at least) two different suitable polygons for this star graph. □

Note that for the latter example two suitable polygons exist that even share the same reflexivity assignment. In the following we discuss sufficient and necessary conditions for the feasibility of a star graph S_4. By Lemma 3 we know in which cases suitable convex polygons exist. The remaining cases are solved by the following lemma.

Lemma 6 *Consider an S_4 for which no suitable convex polygon exists. A suitable non-convex polygon exists if and only if $\frac{1}{\min_i l_i} < \sum_{j=1, l_j \neq \min_i l_i} \frac{1}{l_j}$.*

Proof. First of all, if a polyline has two or more reflex vertices assigned then $\alpha_A(t) < 2\pi$, as each positive summand in Equation (1) is bound by $\pi/2$. Hence, we only need to consider polylines with exactly one reflex vertex, which implies $\alpha_A(0) = 2\pi$.

For simplicity, we may reorder v_i and l_i such that $l_4 = \min_i l_i$. We show that for suitable non-convex polygons v_4 needs to be reflex. Assume to the contrary that some v_k, with $1 \leq k \leq 3$, is reflex. In this case we obtain that $\alpha'_A(t) < 0$ as $1/\sqrt{l_4^2 - t^2}$ dominates $1/\sqrt{l_k^2 - t^2}$ for all $t \in [0, l_4)$. But since $\alpha_A(0) = 2\pi$ we see that $\alpha_A(t) < 2\pi$ for all $t \in (0, \min_i l_i]$.

Observe that the assumption in the lemma, that no suitable convex polygon exists, is equivalent to $\alpha_A(l_4) > 2\pi$. Recall that $\alpha_A(0) = 2\pi$. Hence, if $\alpha'_A(0) < 0$ then there exists a $t \in (0, l_4)$ such that $\alpha_A(t) = 2\pi$, as α_A is continuously differentiable.

Finally, we show that if $\alpha'_A(0) \geq 0$ then $\alpha'_A(t) > 0$ for all $t \in (0, l_4)$. Hence, there is no $t \in (0, l_4]$ such that $\alpha_A(t) = 2\pi$. From Equation (2) we get that $\alpha'_A(t) > 0$ is equivalent to

$$\frac{1}{\sqrt{l_4^2 - t^2}} > \sum_{i=1}^{3} \frac{1}{\sqrt{l_i^2 - t^2}} \quad \Leftrightarrow \quad 1 > \sum_{i=1}^{3} \sqrt{1 - \frac{l_i^2 - l_4^2}{l_i^2 - t^2}}$$

The right side of this equivalence is true since

$$1 \geq \sum_{i=1}^{3} \sqrt{1 - \frac{l_i^2 - l_4^2}{l_i^2}} > \sum_{i=1}^{3} \sqrt{1 - \frac{l_i^2 - l_4^2}{l_i^2 - t^2}}, \quad (3)$$

where the first inequality is given by $\alpha'_A(0) \geq 0$ and the second inequality holds for all $t \in (0, l_4)$.

To conclude, we have shown that if no suitable convex polygon exists for some S_4, then a suitable non-convex polygon exists for this S_4 if and only if $\alpha'(0) < 0$, which is equivalent to $\frac{1}{\min_i l_i} < \sum_{j=1, l_j \neq \min_i l_i} \frac{1}{l_j}$, as claimed in the lemma. □

3.2 Caterpillar graphs

The techniques developed in the previous section can be generalized to so-called *caterpillar graphs*. A caterpillar graph $G \in \mathcal{G}$ is a graph that becomes a path if all its leaves (and their incident edges) are removed. We call this path the *backbone* of G. Figure 3 shows a caterpillar graph whose backbone comprises three backbone edges.

In general, a caterpillar graph has m backbone vertices, consecutively denoted by $v_0^1, \ldots, v_0^m$. We denote the adjacent vertices of a backbone vertex v_0^i, with k_i incident edges, by $v_1^i, \ldots, v_{k_i}^i$, such that $v_{k_i}^i = v_0^{i+1}$ for $1 \leq i < m$. Furthermore, we denote by l_j^i the length of the edge $v_0^i v_j^i$, see Figure 5. Let us consider a polygon P whose straight skeleton $\mathcal{S}(P)$ forms a caterpillar graph.

Observation 2 *All edges of P whose straight-skeleton faces contain the same backbone vertex v_0^i have identical orthogonal distance to v_0^i.*

We denote this orthogonal distance by r_i. Hence, the supporting lines of the corresponding polygon edges are tangents to the circle of radius r_i centered at v_0^i, see Figure 5.

Lemma 7 *The radii $r_2, \ldots, r_m$ of a suitable polygon $P_{E(G)}$ for some given caterpillar graph G are determined by r_1 and the predefined edge lengths of G according to the following recursions, for $1 \leq i < m$:*

$$r_{i+1} = r_i + l_{k_i}^i \sin \beta_i$$

$$\beta_i = \beta_{i-1} + (1 - k_i/2)\pi + \sum_{\substack{j=1 \\ v_j^i \neq v_0^{i-1}}}^{k_i - 1} \begin{cases} \arcsin \frac{r_i}{l_j^i} & v_j^i \textit{ is convex} \\ \pi - \arcsin \frac{r_i}{l_j^i} & v_j^i \textit{ is reflex} \end{cases}$$

For $i = 1$ we define that $\beta_0 = 0$ and $v_j^1 \neq v_0^0$ being true for all $1 \leq j < k_1$.

Proof. Denote with e one of the two edges of $P_{E(G)}$ whose faces of $\mathcal{S}(P_{E(G)})$ contain the edge $v_0^i v_0^{i+1}$. The supporting line of e is tangential to the circles at v_0^i and v_0^{i+1}. Considering the shaded right-angled triangle in Figure 5, we obtain $r_{i+1} - r_i = l_{k_i}^i \cdot \sin \beta_i$.

Consider the polygon P_i' (bold in Figure 5) which comprises the edges of $P_{E(G)}$ whose faces of $\mathcal{S}(P_{E(G)})$ contain v_0^i, trimmed by two additional edges orthogonal to $v_0^{i-1} v_0^i$ and $v_0^i v_0^{i+1}$, respectively. P_i' comprises k_i+2 vertices (k_1+1 for P_1') and hence, the sum of inner angles equals $k_i \pi$ ((k_1−1)π for P_1'). On the other hand, we can express this sum as follows (also for P_1'), which implies the second recursion:

$$k_i \pi = 2\pi + 2\beta_{i-1} - 2\beta_i + 2 \sum_{\substack{j=1 \\ v_j^i \neq v_0^{i-1}}}^{k_i - 1} \begin{cases} \arcsin \frac{r_i}{l_j^i} & v_j^i \text{ is convex} \\ \pi - \arcsin \frac{r_i}{l_j^i} & v_j^i \text{ is reflex} \end{cases}$$ □

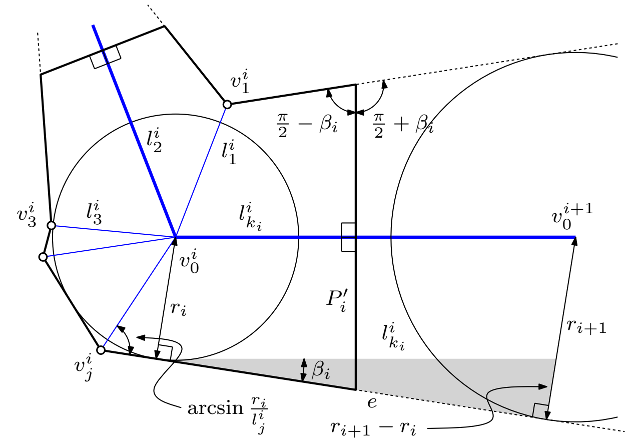


Figure 5: A section of a polygon P for which $\mathcal{S}(P)$ is a caterpillar graph.

Corollary 8 *The sum of the inner angles of $P_{E(G)}$ with convexity assignment A is a function*

$$\alpha_A(r_1) = 2 \sum_{j=1}^{n} \begin{cases} \arcsin \frac{r_{v_j}}{l_j} & v_j \textit{ is convex} \\ \pi - \arcsin \frac{r_{v_j}}{l_j} & v_j \textit{ is reflex} \end{cases}, \tag{4}$$

where r_{v_j} denotes the radius of the circle at the backbone vertex that is adjacent to v_j and l_j denotes the length of the incident edge of G.

The previous corollary provides us with a tool in order to find suitable polygons $P_{E(G)}$ for caterpillar graphs G. We know that for any suitable polygon $P_{E(G)}$ the identity $\alpha_A(r_1) = (n-2)\pi$ must hold. Hence, we can determine all suitable polygons $P_{E(G)}$ as follows: for all 2^n possible assignments A we determine all r_1 such that $\alpha_A(r_1) = (n-2)\pi$.

For any such pair (A, r_1) we construct a polyline $v_1, \ldots, v_n, v_{n+1}$ by a similar method as outlined for star graphs: shooting rays tangential to circles centered at the backbone vertices v_0^i. In order to switch over from v_0^i to v_0^{i+1}, we consider the previously constructed ray, which needs to be tangential to the two circles centered at both, v_0^i and v_0^{i+1}, respectively. As the length of the edge $v_0^i v_0^{i+1}$ is given, the center v_0^{i+1} of the next circle is uniquely determined, cf. Figure 5. If there is any non-backbone edge with length $l_j^i < r_i$ then there is no suitable polygon for that particular pair (A, r_1). For each candidate polyline we check whether it is closed, simple and forms a suitable polygon. Note that all suitable polygons can be constructed by the above method.

Lemma 9 *There is at most a finite number of suitable polygons $P_{E(G)}$ for a caterpillar graph G.*

Proof. As α_A is analytic, there are no accumulation points in the set $\{r_1 : \alpha_A(r_1) = (n-2)\pi\}$. Otherwise, α_A would be identical to $(n-2)\pi$. In other words, there is only a finite number of possible pairs (A, r_1) that correspond to a suitable polygon. $\square$

Lemma 10 *There exists a caterpillar graph with* $3m$ *vertices having* 2^{m-2} *suitable polygons.*

Proof. We consider a caterpillar graph with m backbone vertices, for which the two outer backbone vertices are of degree three and the $m-2$ inner backbone vertices are of degree four. We set the length of the non-backbone edges to $\sqrt{2}/4$ and the length of the backbone edge $e_k = v_0^k v_0^{k+1}$ to $3/4 \cdot 2^{k-1}$. We embed the backbone as a rectilinear path and at each v_0^k, with $1 < k < m$, we either make a left or right turn from e_{k-1} to e_k. This gives us 2^{m-2} possible embeddings, see Figure 6. We now consider the polygon P shown in Figure 6, which forms a rectilinear hose (a mitered offset curve) around the embedding of G with thickness $\frac{1}{2}$. It remains to show that P is not self-overlapping.

For each edge e_k we consider the axis-parallel square $A_k \supseteq e_k$ with side length 2^k that has v_0^{k+1} as a midpoint of one of its sides. By our choice of edge lengths we observe that $A_{k-1} \subseteq A_k$ and A_{k-1} and A_k share a common vertex. We say that a polygon edge belongs to e_i if e_i is contained in its straight-skeleton face. Let s denote a polygon edge that belongs to e_{k+1}. By construction, s cannot overlap with polygon edges belonging to e_k. Using an induction-type argument, all polygon edges belonging to e_i, with $i < k$, are contained in the one axis-parallel half of A_k that does not contain v_0^{k+1}. Hence, s does not intersect polygon edges that belong to $e_1, \ldots, e_k$. It follows by induction that P is not self-overlapping. $\square$

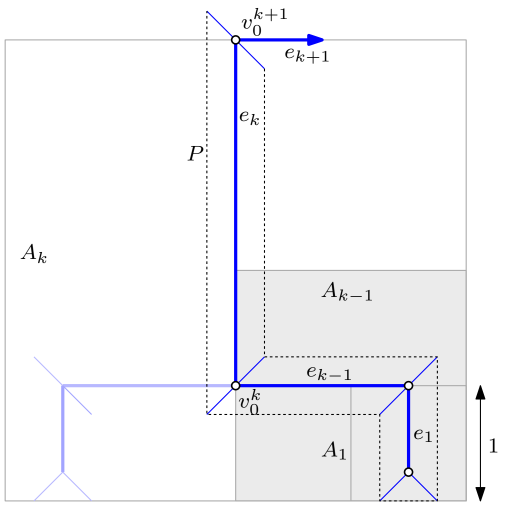


Figure 6: For the above caterpillar graph an exponential number of polygons exist by making left or right turns at backbone vertices.

4 Conclusion

In this work, we considered the inverse problem of computing the straight skeletons of simple polygons. First, we proved that each tree, whose inner vertices have degree at least 3, can be realized as the straight skeleton of a convex polygon. The constructive proof also provides the outline for an algorithm to construct a suitable polygon.

Next, we considered a more restrictive version of the original question by predefining the lengths of the straight-skeleton edges and their circular orders at straight-skeleton vertices. For star graphs we showed that there exists a unique suitable convex polygon for every S_3. Further, we derived that for general star graphs feasibility is neither guaranteed nor unique, and we gave sufficient and necessary conditions for the existence of a suitable polygon for all S_4. Furthermore, we gave a simple necessary and sufficient condition that tells us when a suitable convex polygon exists for a star graph.

Concerning the more general caterpillar graphs we provided a basic method for constructing all suitable polygons and we proved that the number of suitable polygons is finite. Furthermore, we showed that an n-vertex caterpillar graph may possess $2^{n/3-2}$ suitable polygons. Finding a tight upper bound on the number of suitable polygons is an open question. Finally, the major open questions concern arbitrary trees: How to decide feasibility? Are there at most finitely many feasible polygons or is there even an entire continuum of feasible polygons? After all, a partial result is given in [2]: there are at most $2n-5$ suitable convex polygons for an arbitrary abstract geometric tree with n leaves.

References

[1] O. Aichholzer, D. Alberts, F. Aurenhammer, and B. Gärtner. *A novel type of skeleton for polygons.* Journal of Universal Computer Science, 1(12):752–761, 1995.

[2] H. Cheng, S.L. Devadoss, B. Li, A. Risteski. *Skeletal rigidity of phylogenetic trees.* 2012. `arXiv:1203.5782v1 [cs.CG]`

[3] S.L. Devadoss, J. O'Rourke. *Discrete and Computational Geometry.* Princeton University Press, Princeton and Oxford, 2011.

[4] S. Huber. *Computing straight skeletons and motorcycle graphs: theory and practice.* Shaker Verlag, Aachen, 2012. ISBN 978-3-8440-0938-5.

[5] E. Müller. *Lehrbuch der darstellenden Geometrie für technische Hochschulen.* Band 2, Verlag B.G.Teubner, Leipzig & Berlin, 1916.

[6] G.A.V. Peschka. *Kotirte Ebenen und deren Anwendung.* Verlag Buschak & Irrgang, Brünn, 1877.

3D Skeletonization as an Optimization Problem

Denis Khromov[*] Leonid Mestetskiy[†]

Abstract

We present a novel approach to 1D curve-skeletonization of 3D objects. The skeletonization process is reduced to a numerical optimization problem. The method provides a strict way to evaluate and compare various 1D skeletons of the same 3D object. We describe a particular implementation of our approach and discuss experiment results.

1 Introduction

A skeleton of a shape is a graph that captures major topological and metrical properties of the shape. It may be considered as a 1D thinning of the original shape. Skeletons are useful in computer vision since it is much easier to extract a shape's features from a graph rather than from its boundary description. The skeleton of a two-dimensional shape is usually defined as a shape's medial axis. The medial axis is the set of all points having more than one closest point on the shape's boundary. The medial axis of a 2D shape is always a 1D set. Such a set can be computed efficiently. However, a 3D medial axis contains 2D sheets and therefore is not a graph.

A curve-skeleton is a 1D skeleton of a 3D shape. It is known to be an ill-defined object [3]. There is no common strict definition recognized by a significant number of papers on curve-skeletons. A curve-skeleton is usually defined as an object produced by some particular algorithm. Different algorithms produce different skeleton graphs; those graphs have different properties. Examples of such algorithms can be found in [3, 11]. Therefore it is almost impossible to compare different algorithms with each other. It is only possible to evaluate the computational performance and the visual quality of different skeletons.

On the other hand, there is an intuitive idea of what a correct curve-skeleton should be. It is clear that different curve-skeletons of the same 3D object are visually similar, even if they are defined differently. Our goal is a mathematical formalization of this intuitive idea.

In this article we present a method to evaluate the quality of a curve-skeleton. We define a curve-skeleton as a continuous thinning of the original object. We also describe the procedure of reconstruction which allows to produce a 3D shape from a 1D skeleton. The shape produced from a curve-skeleton is called a silhouette. The measure of similarity between the original shape and the skeleton's silhouette is an evaluation of the skeleton's quality. Thus the problem of skeletonization can be formulated as a numerical optimization problem. To construct a skeleton of the object means to find the best approximation of the object by the skeleton's silhouette. We describe our implementation of this idea. The implementation includes the measure of similarity, the first approximation algorithm, and numerical methods for the minimization process.

The key feature of our method is the numerical evaluation of the curve-skeleton's quality. However, our work shares some common ideas with the recent papers on the subject. We use Reeb graphs to compute the first approximation of our skeletons. Reeb graphs are widely used in topological shape analysis [1, 4, 5, 10, 14]. The primitives we use to approximate the original object are very similar to the sphere swept volumes (SSV). There are papers presenting the usage of SSV for 3D shape approximation [2, 7]. In [9] a deformable model is used to describe the object's shape; the curve-skeleton itself is obtained from this deformable model. There are very few methods involving the optimization of a skeleton's quality. One example is the paper [12], where an iterative least squares optimization is performed to shrink the model and obtain its accurate thinned representation.

2 Definitions

As mentioned above, there are many definitions of a 3D curve-skeleton. The common idea of these definitions is that the curve-skeleton is a thinned 1D representation of the 3D object [3]. It can be formalized in terms of homotopy.

Let Ω be a connected open set embedded in $\mathbb{R}^3$ with boundary $\partial\Omega$. Let $\overline{\Omega}$ be its closure:

$$\overline{\Omega} = \Omega \cup \partial\Omega. \quad (1)$$

Let $\Gamma \subset \overline{\Omega}$ be a 3D representation of some graph G such that every edge of G is mapped onto a smooth curve $\gamma \in \mathbb{R}^3$. We denote the Euclidean distance between points $x, y \in \mathbb{R}^3$ by $\rho(x, y)$.

Definition 1 Γ *is a curve-skeleton of* Ω *if there is a*

[*]Moscow State University, denis.v.khromov@gmail.com
[†]Moscow State University, l.mest@ru.net

continuous function

$$H : [0;1] \times \partial\Omega \to \overline{\Omega} \qquad (2)$$

such that

$$H(0,x) = x, H(1,\partial\Omega) = \Gamma. \qquad (3)$$

The function

$$\sigma(x) \equiv H(1,x) \qquad (4)$$

is called a skeleton mapping.

A skeleton mapping describes the correspondence between the surface of the object and its skeleton.

One of the advantages of a 2D skeleton is the ability to recover the original shape from a skeleton. That is possible due to a distance transform function (DTF). A DTF defines "width" of the shape for every skeleton point. Then the shape is a union of discs: the centers of these discs are situated on the skeleton branches, and the radii are defined by the DTF. We need some analogue of a 2D distance transform in order to preserve the reconstruction possibility for 3D curve-skeletons.

Definition 2 *A radial function r is a non-negative real-valued function defined on a curve-skeleton:*

$$r : \Gamma \to \mathbb{R}, r(x) \geq 0 \ \forall x \in \Gamma. \qquad (5)$$

Definition 3 *A silhouette of a curve-skeleton Γ with a radial function r is a set*

$$S(\Gamma, r) = \bigcup_{x\in\Gamma} B_{r(x)}(x), \qquad (6)$$

where $B_{r(x)}(x)$ is a ball with the center in x and the radius $r(x)$:

$$B_{r(x)}(x) = \{y \in \mathbb{R}^3 : \rho(x,y) \leq r(x)\}. \qquad (7)$$

A single curve with its silhouette is called a fat curve [8]. Fat curves can be used for an approximation of tubular objects [6]. An example of a silhouette of a skeleton is shown in Figure 1.

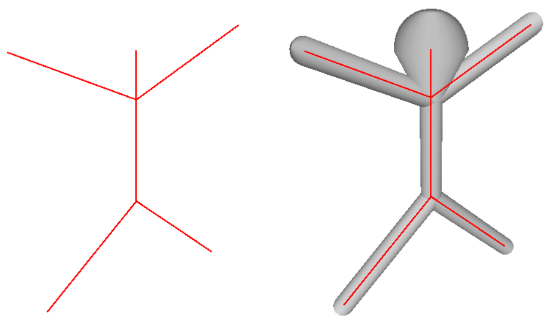


Figure 1: A 3D graph (left) and its silhouette produced by some radial function (right).

A silhouette can be considered as a reconstruction of the original 3D object. Unlike 2D skeletons, a 3D silhouette is merely an approximation of Ω. So we need a numerical measure of similarity between the original shape Ω and the silhouette $S(\Gamma, r)$. It is possible to define it as a distance between two sets in $\mathbb{R}^3$ (for example, the Hausdorff distance). But this approach may produce skeletons that do not correspond to the intuitive idea of what a correct curve-skeleton is. An example is shown in Figure 2. The graph Γ consists of one parabola, and the radial function r is such that the silhouette $S(\Gamma, r)$ looks like a straight tubular object because of its self-intersection. It can be considered as an ordinary subset of $\mathbb{R}^3$. In that case its curve-skeleton is expected to look like a single straight line segment, not a parabola. But the silhouette of the parabola is a perfect approximation for this object if we use something like the Hausdorff distance.

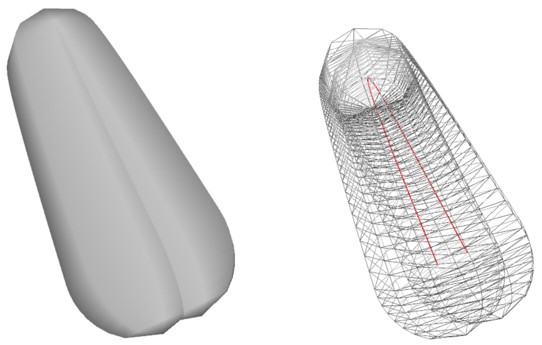


Figure 2: A silhouette of the parabola (red): solid (left) and wireframe (right).

In this paper we propose the following function.

Definition 4 *Let Γ be a curve-skeleton of Ω with a radial function r. An approximation error of (Γ, r) is a value*

$$\mathcal{E}(\Omega,\Gamma,r) = \int_{x\in S=\partial\Omega} \Big(\rho^2\big(x,\sigma(x)\big) - r^2\big(\sigma(x)\big)\Big)^2 dS. \qquad (8)$$

This function takes into account the correspondence σ between $\partial\Omega$ and Γ. It takes small values when the distance between $x \in \partial\Omega$ and $\sigma(x) \in \Gamma$ is close to $r\big(\sigma(x)\big)$. It is satisfied when the skeleton's curve gives the right representation of the approximated shape. Thus Equation 8 gives a mathematical meaning to the intuitive concept of a 3D curve-skeleton.

3 First Approximation

The main idea of our method is a numerical optimization of a curve-skeleton in order to minimize the approximation error given in (8). This approach suggests that there is a first approximation of a curve-skeleton. It can be constructed by almost any existing skeletonization algorithm.

We construct the initial curve-skeleton using Reeb graphs.

Definition 5 *Let f be a continuous function defined on a compact manifold M. A Reeb graph is a quotient space*

$$R_f = M/\sim_f, \tag{9}$$

where $\sim_f$ is an equivalence relation such that

$$x \sim_f y \Leftrightarrow \exists l \subset \partial\Omega : x, y \in l, f(z) \equiv \text{Const}\ \forall z \in l, \tag{10}$$

and l is a connected curve in $\partial\Omega$.

Algorithms based on Reeb graphs are rather effective. They always produce topologically correct skeletons. This is important since it is difficult to change the topology of a skeleton during the numerical optimization process. Another important advantage of Reeb graphs is that the skeleton mapping σ is generated explicitly.

The first approximation is computed in a few steps. A continuous scalar function

$$f : \partial\Omega \to \mathbb{R} \tag{11}$$

is called a mapping function. The corresponding Reeb graph R_f is a graph-like space. It describes the topology of Ω. We need to embed it into $\mathbb{R}^3$ in order to generate a 3D representation of the graph.

The embedding is defined as follows:

$$\Gamma = \bigcup_{l \in L} \text{MEB}(l), \tag{12}$$

$$\sigma(x) = \text{MEB}(l \in L : x \in l). \tag{13}$$

$\text{MEB}(l)$ is the center of a minimum enclosing ball covering the set l. Here minimum enclosing balls play the same role as maximum inscribed balls do in the definition of a medial axis.

The appearance of a skeleton is determined by the mapping function f. In our implementation it is defined as follows. Let $p \in \partial\Omega$ be a fixed point called a pole. The mapping function $f(x)$ equals the geodesic distance over $\partial\Omega$ between x and the pole p. This function does not depend on the object's orientation in space. This is a serious advantage over functions like a height map. Another benefit of our mapping function is that it has a low computational complexity.

4 Implementation

In this section we discuss the implementation of our method for 3D objects represented by polygonal meshes. Let M be a triangulated closed surface with a set of vertices V and a set of edges E. We assume that V is uniform and detailed enough.

To avoid confusion with the mesh and its vertices and edges, in this section we will use the term "joint" for a curve-skeleton vertex and the term "bone" for an edge. By J denote the set of joints and by B denote the set of bones. In our implementation, all bones are straight line segments and all radial functions are linear. Each joint

$$j = (x_j, r_j),\ x_j \in \mathbb{R}^3, r_j \geq 0 \tag{14}$$

is defined by 4 real values: the position x_j and the value of the radial function (or radius) r_j. Each bone

$$b = (j_1, j_2),\ j_1, j_2 \in J \tag{15}$$

connecting joints j_1 and j_2 describes a curve and a radial function given by equations

$$\gamma_b(t) = tx_{j_1} + (1-t)x_{j_2}, \tag{16}$$

$$r_b(t) = tr_{j_1} + (1-t)r_{j_2}, \tag{17}$$

where $t \in [0; 1]$.

The skeleton mapping value for a vertex $v \in V$ is defined by the bone $b^v = (j_1^v, j_2^v)$ and the parameter t_v:

$$\big(j_1^v, j_2^v, t_v\big), (j_1^v, j_2^v) \in B, t_v \in [0; 1], \tag{18}$$

$$\sigma(v) = \gamma_{(j_1^v, j_2^v)}(t_v) = t_v x_{j_1^v} + (1-t_v)x_{j_2^v} \tag{19}$$

The computation of this mapping will be explained below.

The integral in (8) is approximated with a sum $\tilde{\mathcal{E}}$ over all vertices:

$$\begin{aligned} \tilde{\mathcal{E}} &= \sum_{v \in V} \Big(\rho^2\big(v, \sigma(v)\big) - r^2\big(\sigma(v)\big)\Big)^2 = \\ &= \sum_{v \in V} \Big(\rho^2\big(v, t_v x_{j_1^v} + (1-t_v)x_{j_2^v}\big) - \\ &\qquad - \big(t_v r_{j_1^v} + (1-t_v) r_{j_2^v}\big)^2\Big)^2. \end{aligned} \tag{20}$$

4.1 First Approximation

In order to find the first approximation we need to

1. select the pole vertex;
2. compute the mapping function f and its contour lines;
3. construct the skeleton itself.

The geodesic distance between two vertices $v_1, v_2 \in V$ is approximated by the length of the shortest chain of edges from E connecting v_1 and v_2. The advantage of this method is that it is rather fast. In some cases it may produce a rough approximation. However, that is not an issue since we don't need the exact values of geodesic distance. The accuracy of this method is enough to produce contour lines which define an acceptable Reeb graph.

The pole vertex is computed in three steps:

1. let v_1 be a random vertex from V;
2. let v_2 be the vertex farthest from v_1;
3. the pole vertex p is the vertex that is the farthest from v_2.

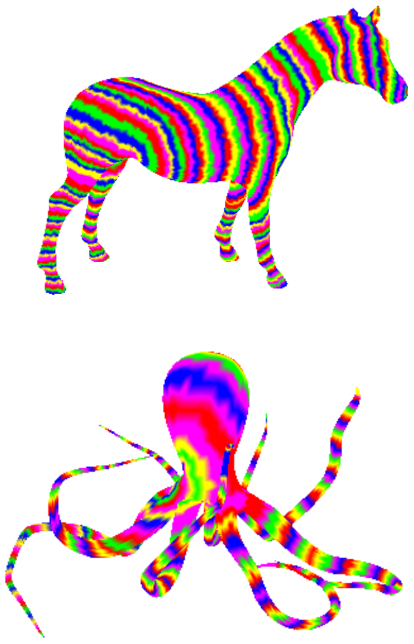


Figure 3: Contour lines of the mapping function.

The set V is divided into contour lines of f. It is impossible to obtain exact contour line of f because we work with the discrete set of points V. So the following algorithm is performed. Let $l_k \subset V$ be some contour line. The contour line l_{k+1} produced on the next step of the algorithm is a maximum connected subset of V such that

$$\exists u \in l_{k+1} : \tag{21}$$

$$\exists v_1 \in l_k \quad (v_1, u) \in E, \tag{22}$$

$$\forall v_2 \in l_{k+1} \quad f(v_2) \leq f(u). \tag{23}$$

More then one connected subset satisfying these conditions means that we have reached a joint where a group of skeleton branches converge: each subset corresponds to a new branch. Figure 3 demonstrates an example: contour lines are painted in different colors.

Each contour line corresponds to a skeleton joint $j \in J$. The position of this joint x_j is computed as a center of the minimum enclosing ball. We use Welzl's algorithm to find the MEB [13]. The radius of this ball is taken for the first approximation of the radial function value r_j.

4.2 Numerical Optimization

This step of the algorithm involves the minimization of the function (20). This function is a polynomial of degree 4 with $4|J|$ variables (since we have $|J|$ joints and each is defined by 4 scalar variables). It can be represented in the form of the sum

$$\begin{aligned}\mathcal{E}(z) = \sum_{v\in V}\left((a_v z + b_v)^2 - c_v\right)^2 = \\ = z^4\left(\sum_{v\in V} a_v^4\right) + \\ +z^3\left(\sum_{v\in V} 4a_v^3\right) + \\ +z^2\left(\sum_{v\in V}(6a_v^2 b_v^2 - 2b_v^2 c_v)\right) + \\ +z\left(\sum_{v\in V}(4a_v b_v^3 - 4a_v b_v c_v)\right) + \\ +\left(\sum_{v\in V}(b_v^4 - 2b_v^2 c_v + c_v^2)\right).\end{aligned} \tag{24}$$

with respect to each single variable z. For example, for $z = r_j$

$$a_v = t_v, b_v = (1-t_v) r_{j_2^v}, c_v = \rho^2\left(v, \sigma(v)\right) \tag{25}$$

if $j_1^v = j$,

$$a_v = (1-t_v), b_v = t_v r_{j_1^v}, c_v = \rho^2\left(v, \sigma(v)\right) \tag{26}$$

if $j_2^v = j$ and

$$a_v = b_v = c_v = 0 \tag{27}$$

otherwise.

We use a gradient descent to minimize $\tilde{\mathcal{E}}$. Let J_k be a $4|J|$-dimensional vector describing the set of joints computed in step k. Then J_{k+1} is given by

$$J_{k+1} = J_k - \lambda \nabla \tilde{\mathcal{E}}(J_k), \tag{28}$$

where λ is a solution of the minimization problem

$$\arg\min_{\lambda\in\mathbb{R}} \tilde{\mathcal{E}}(J_k - \lambda\nabla\tilde{\mathcal{E}}(J_k)). \tag{29}$$

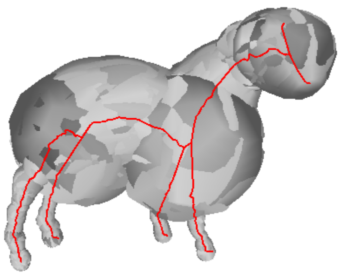
It is a polynomial of degree 4 with respect to only one variable. Therefore it is very easy to find its minimum analytically. $\tilde{\mathcal{E}}$ and $\nabla\tilde{\mathcal{E}}$ are computed using (24). The procedure (28) is performed until the condition

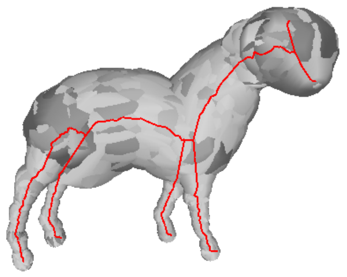
$$|\tilde{\mathcal{E}}(J_k) - \tilde{\mathcal{E}}(J_{k+1})| < \varepsilon \tag{30}$$

is satisfied, where $\varepsilon > 0$ is a fixed parameter of the algorithm. In our experiments

$$\varepsilon = 10^{-3}\tilde{\mathcal{E}}(J_0), \tag{31}$$

where J_0 is the first approximation. The optimization process is demonstrated in Figure 4. It shows the first approximation and two serial iterations of the gradient descent for the horse model from Figure 3.


(a) First approximation, $\tilde{\mathcal{E}}(J_0) = 7.96$


(b) After one iteration, $\tilde{\mathcal{E}}(J_1) = 1.64$

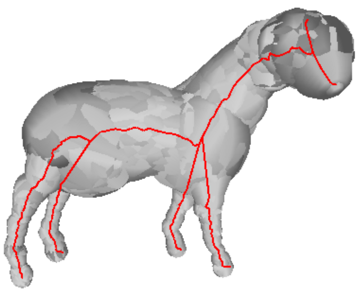

(c) After two iterations, $\tilde{\mathcal{E}}(J_2) = 0.84$

Figure 4: The curve-skeleton of the horse and its silhouette.

Note that this process may lead to negative radial functions. Negative values in radii have no physical meaning. We change the sign of each negative r_j since they are included in $\tilde{\mathcal{E}}$ with the second degree.

5 Experiments and Discussion

Some examples of our algorithm's work are shown in Figure 5. The resulting curve-skeletons are demonstrated with their corresponding silhouettes. These 3D models are often used in articles on curve-skeletons, so they are suitable for visual comparison of our skeletonization method with other ones.

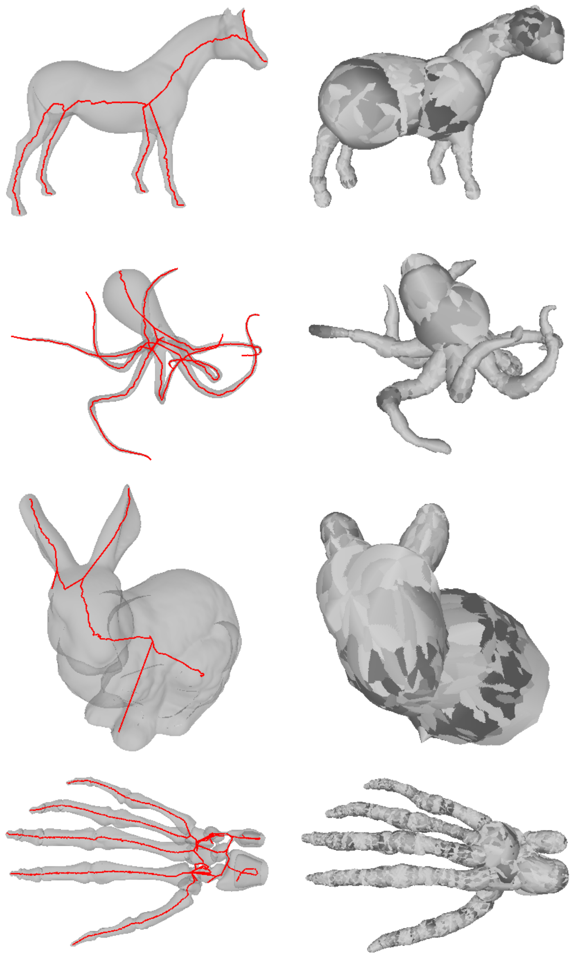


Figure 5: The results produced by our algorithm: the skeletons (left) and their silhouettes (right) for various 3D objects.

An object with a full-dimensional medial axis is shown in Figure 6. It does not have any significant tubular fragments. Curve-skeletons are not very useful for objects like this. The resulting silhouette is not a valuable approximation.

Some desirable properties of a curve-skeleton are listed in [3]. Our approach guarantees some of them:

- the topological properties (such as homotopic invariance) are provided by the usage of Reeb graphs;
- the invariance under isometric transformations is not always guaranteed by methods based on Reeb

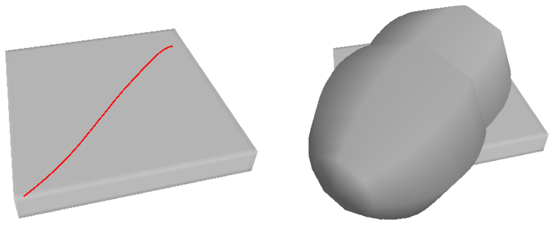


Figure 6: An object with a full-dimensional medial axis.

graphs (for example, the mapping function can depend significantly on one of the coordinates), but in our case this property is satisfied;

- the possibility of reconstruction is a key feature of our method;
- the skeletons are thin by definition;
- the centeredness is the value that is strictly measured and optimized with our algorithm.

Some properties are not satisfied:

- robustness is not guaranteed since some images may produce skeleton branches for visually insignificant protrusions of the shape. These branches should be considered as noise which can't be compensated by the minimization of (20);
- smoothness is another property that is provided neither by Reeb graphs nor by the optimization process. However, the usage of curves of higher degrees instead of piecewise linear curves could fix that problem.

6 Conclusions and Future Work

We have demonstrated an approach to 3D curve-skeletonization which allows to evaluate skeletons and to choose the best one among others. This method is implemented. The implementation and experiments prove the practical utility of the method. Below we list several open issues for further research.

1. Tuning the skeleton mapping σ during the minimization process. Currently it is fixed, which imposes strict requirements on the quality of the first approximation and its skeleton mapping.
2. Taking mesh edges and faces into account. Current algorithm uses vertices only. This means that skeletons of the detailed meshes are more accurate than for the low polygon meshes.

References

[1] M. Attene, S. Biasotti, M. Spagnuolo. Shape understanding by contour driven retiling. *The Visual Computer*, 19:2–3, 2003.

[2] M. Bae, J. Kim, Y.J. Kim. User-guided volumetric approximation using swept sphere volumes for physically based animation. *Computer Animation and Virtual Worlds*, 2012.

[3] N.D. Cornea, D. Silver. Curve-skeleton properties, applications, and algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 13:530–548, 2007.

[4] M. Hilaga, Y. Shinagawa, T. Kohmura, T.L. Kunii. Topology matching for fully automatic similarity estimation of 3D shapes. *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, 203–212, 2001.

[5] R.E. Khoury, J. Vandeborre, M. Daoudi. 3D mesh Reeb graph computation using commute-time and diffusion distances. *Proc. SPIE* 8290, 82900H, 2012.

[6] D. Khromov. Curve-Skeletons based on the fat graph approximation. *Lecture Notes in Computer Science*, Volume 6915/2011, 239–248, 2011.

[7] E. Larsen , S. Gottschalk , M. C. Lin , D. Manocha. Fast proximity queries with swept sphere volumes. *IEEE International Conference on Robotics and Automation*, 3719-3726, 2000.

[8] L.M. Mestetskii. Fatcurves and representation of planar figures. *Computers & Graphics*, Volume 24, Issue 1, February 2000.

[9] A. Sharf, T. Lewiner, A. Shamir, L. Kobbelt. On-the-fly curve-skeleton computation for 3D shapes. *In Proceedings of Comput. Graph. Forum*,323-328, 2007.

[10] Y. Shi, R. Lai, S. Krishna, N. Sicotte, I. Dinov, A.W. Toga. Anisotropic Laplace-Beltrami eigenmaps: bridging Reeb graphs and skeletons. *Computer Vision and Pattern Recognition Workshop*, pp. 1-7, 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2008.

[11] K. Siddiqi, S. M. Pizer . Medial Representations: mathematics, algorithms and applications. *Springer*, 2008.

[12] Y. Wang, T. Lee. Curve-Skeleton extraction using iterative least squares optimization. *IEEE Transactions on Visualization and Computer Graphics*, 14:926–936, 2008.

[13] E. Welzl. Smallest enclosing disks (balls and ellipsoids). *Results and New Trends in Computer Science*, 1991.

[14] Y. Xiao, J.P. Siebert, N. Werghi. A discrete Reeb graph approach for the segmentation of human body scans. *Fourth International Conference on 3-D Digital Imaging and Modeling*, 2003.

Dynamic Computational Topology for Piecewise Linear Curves [*]

Hugh P. Cassidy [†] Thomas J. Peters[‡] Kirk E. Jordan[§]

Abstract

A piecewise linear (PL) approximation often serves as the graphics representation for a parametric curve. Algorithms for preserving correct topology for a single static image are available, but significant challenges remain to ensure correct topology when the PL curve is changing shape during synchronized visualization with an ongoing simulation, such as a molecule writhing over time. A tubular neighborhood of the curve is defined to preserve topology under perturbation, but as the perturbed geometry approaches the boundary of that tubular neighborhood, any required update of the neighborhood should maintain the synchronization. The algorithimic performance of these updates is directly dependent upon the number of approximating edges and the techniques presented here decrease that data volume versus previous methods, as shown by a comprehensive comparative analysis and a representative example.

1 Introduction & Related Work

Molecules undergoing computer simulations are often represented by parametric curves. The simulation code describes how points on the curve move under changes in critical variables such as temperature, pressure and acidity. For graphics display, PL approximations are invoked [11]. The literature on ensuring that a *static* PL approximation retains crucial topological characteristics of the model is relatively well developed for both curves [14] and surfaces [1, 2, 10]. The more subtle challenge of maintaining topological fidelity during perturbations of PL approximations [3] is addressed here, with innovations for efficient update of topological constraints during visualization that is synchronized to an ongoing molecular simulation. These perturbations are constrained within a tubular neighborhood of c, as shown in Figure 1. This tubular neighborhood has a bounding pipe surface [14] of radius r, which depends upon two properties of c,

- the mimimal Euclidean distance between points of c that are distant in arc-length, and
- maximal curvature.

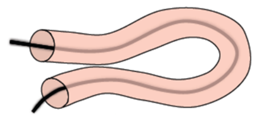


Figure 1: A tubular neighborhood about c.

For polynomial curves, the maximal curvature can be easily computed and will not be discussed further. Our attention is devoted to the first value, known as the *minimum separation distance* for c, abbreviated as MSD_c. A *double normal* is defined to be a line segment which is normal to c at both end points of the line segment. We define MSD_c to be the minimum length of all double normal line segments of c. Even for polynomial curves, the computation of MSD_c entails some subtlety. We show that an approximant of MSD_c, denoted as λ_c can be computed, within user-specified error bounds, from a PL approximation of c. More importantly, we provide a comprehensive comparative analysis and a representative example to justify our efficient updates of λ_c, as the geometry continues to perturb beyond the original constraints imposed by λ_c.

For graphics, once a PL curve has been shown to preserve the topological embedding of the parametric curve, then perturbations of the PL approximation are used to generate subsequent graphical images. By obvious extensions of previous methods [13], the originally calculated λ_c serves as a bound to continue to preserve the desired topological embedding. However, as the limiting value of λ_c is approached, with a perturbed PL curve ℓ, then computation of the updated limit can proceed purely on ℓ.

A concept closely related to these pipe surfaces is the thickness of a knot [5, 6]. Applications to molecular modeling [19] and detailed numeric algorithmic development [4, 7] have appeared, but these algorithms do not address the crucial update efficiencies considered here, even while their definition of *doubly-critical self-distances* is the same as MSD_c.

[*]The first author was partially supported by NSF Grant CNS 0923158. The second author was partially supported by NSF grants CCF 0429477, CNS 0923158 and CMMI 1053077 as well as by the Joint Study Agreement W1056109 with IBM, an IBM Faculty Award and two IBM Doctoral Fellowships. All statements here are the responsibility of the author, not of the National Science Foundation nor of IBM.

[†]Department of Computer Science and Engineering, University of Connecticut, Storrs CT, hugh.cassidy@engr.uconn.edu

[‡]Department of Computer Science and Engineering, University of Connecticut, Storrs CT, tpeters@engr.uconn.edu

[§]T.J. Watson Research Center, IBM, Cambridge, MA, kjordan@us.ibm.com

2 Preliminaries

We present the curve and topological definitions that are central to this work.

2.1 Class of Parametric Curves

As often occurs, this investigation is first restricted to the class of Bézier curves, as their polynomial representation avoids many cumbersome details, while still supporting theoretical inisights that can easily be generalized to the wider class of non-uniform rational B-spline (NURBS) curves. A degree n Bézier curve with control points, $P = \{p_0, \cdots, p_n\}$ is given by

$$c(t) = \sum_{i=0}^{n} \binom{n}{i} (i-t)^{n-i} p_i, \qquad t \in [0,1],$$

where the PL curve formed by consecutively connecting $p_0, \cdots, p_n$ is called the control polygon of c [18]. A subdivision algorithm operates on P to generate two PL curves, each having $n+1$ vertices, denoted, respectively as P_L and P_R, as shown in Figure 2. The union $P_L \cup P_R$ is also a control polygon for c but lies closer to c than the original control polygon. This process can be repeated to obtain a PL graphical approximation that is within a prescribed distance of the curve c. This is only an initial *static* approximation and the focus here is for methods to ensure that this approximant retains crucial topological characteristics as it changes over time.

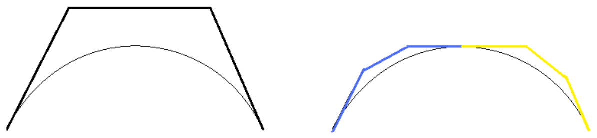


Figure 2: Initial & subdivided control polygons of c.

For ease of exposition[1], we assume that the subdivision parameter is 1/2, so that the fundamental subdivision operation is to find midpoints of line segments. We remark that the statement, proof and use of Lemma 1 directly depend upon a subdivision parameter of 1/2.

2.2 Crucial Topological Characteristics

The traditional measure of topological equivalence is *homeomorphism*. A homeomorphism is a mapping, $f : X \longrightarrow Y$, between two subsets X and Y of $\mathbb{R}^n$ such that:

1. f is bijective,
2. f and f^{-1} are continuous.

[1]The reader can modify our results for other parameters.

Homeomorphic equivalence does not capture the embedding of a curve within $\mathbb{R}^3$. In Figure 3 the right image is an unknot and the left is a trefoil. These structures are homeomorphic even though they are embedded differently in $\mathbb{R}^3$.

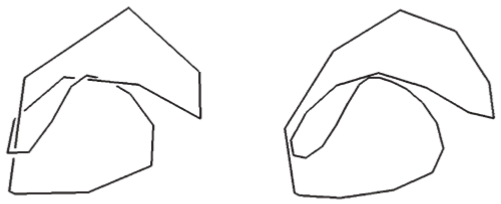


Figure 3: PL knots in $\mathbb{R}^3$

We use the stronger equivalence of *ambient isotopy* to also preserve embedding of c in $\mathbb{R}^3$. The knots in figure 3 are not ambient isotopic. Two subspaces, X and Y, of $\mathbb{R}^n$ are said to be ambient isotopic if there exists a continuous function $H : \mathbb{R}^n \times [0,1] \longrightarrow \mathbb{R}^n$ such that

1. $H(\cdot, 0)$ is the identity on $\mathbb{R}^n$,
2. $H(X, 1) = Y$, and
3. $\forall t \in [0,1], H(\cdot, t)$ is a homeomorphism.

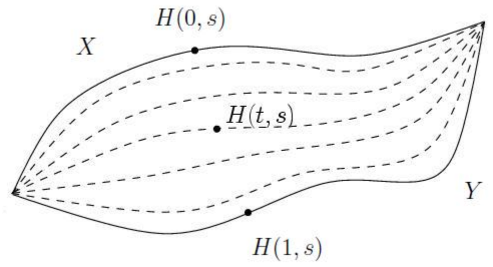


Figure 4: An isotopic deformation of X into Y.

The parameter t can be considered as variable representing time for application to animation and dynamic visualization as illustrated in Figure 4.

3 Background and Notation

In this section we state some previously established results and define notation to be used throughout this paper. Let $c(t)$ be a Bézier curve with control points $P = \{p_0, \cdots, p_n\}$. The PL approximations presented will converge to c in both distance and derivative.

3.1 Approximation of c in Distance

Given the polygon generated by P the *second centered difference* of p_i is given by

$$\Delta_2 p_i = p_{i-1} - 2p_i + p_{i+1}.$$

We define $\Delta_2 p_0 = \Delta_2 p_n = 0$. The *maximal second centered difference* of the polygon generated by P is given by

$$\|\Delta_2 P\|_\infty = \max_{0 \le i \le n} \|\Delta_2 p_i\|_\infty.$$

For a degree n Bézier curve $c(t)$ with control points $P = \{p_0, \cdots, p_n\}$, after m uniform subdivisions the maximal Hausdorff distance between the control polygon and the curve is given by [17]

$$\left(\frac{1}{2}\right)^{2m} \|\Delta_2 P\|_\infty N_\infty(n).$$

Here $N_\infty(n) = \dfrac{\lceil n/2 \rceil \lfloor n/2 \rfloor}{2n}$. Note that this distance is actually attained. So subdividing m_1 times guarantees that the PL structure is within a specified tolerance ϵ where

$$m_1 = \left\lceil -\frac{1}{2} \log_2 \left(\frac{\epsilon}{\|\Delta_2 P\|_\infty N_\infty(n)} \right) \right\rceil \qquad (1)$$

3.2 PL Approximation of c in Derivative

Define the derivative operator Δ on the control points P as follows [12],

$$\begin{aligned} \Delta P &= \{\Delta p_0, \Delta p_1, \cdots, \Delta p_{n-1}\} \\ &= n\{p_1 - p_0, p_2 - p_1, \cdots, p_n - p_{n-1}\}. \end{aligned}$$

The curve generated by the control points ΔP is called the hodograph or derivative curve of c.

Define $\mathcal{L}(P, [0,1])$ to be the uniform parameterization of the control polygon $P = \{p_0, p_1, \cdots, p_n\}$. So

$$\mathcal{L}(P, [0,1]) \left(\frac{j}{n}\right) = p_j$$

and $\mathcal{L}(P, [0,1])$ is linear on the intervals $\left[\frac{j}{n}, \frac{j+1}{n}\right]$. For a Bézier curve $c(t)$ defined on $[0,1]$ with control points $P = \{p_0, \cdots, p_n\}$ the discrete derivative is defined as [16]

$$D[c(t)] = \mathcal{L}(\Delta P, [0,1]).$$

In other words, the discrete derivative is the uniform parameterization of the control polygon of the hodograph.

After applying m subdivisions to $c(t)$ 2^m curves are generated with control points $\left\{p_0^{m,1}, \cdots, p_n^{m,1}, p_0^{m,2}, \cdots, p_n^{m,2}, \; \cdots \; , p_0^{m,2^m} \cdots, p_n^{m,2^m}\right\}$. Subdividing m times divides $[0,1]$ into the intervals $\left[\frac{k}{2^m}, \frac{k+1}{2^m}\right]$ for $k = 0, 1, \cdots, 2^m - 1$. Each interval is associated with a unique subcurve. Now define

$$P' = n\left\{\Delta p_0^{k,1}, \Delta p_1^{k,1}, \cdots, \Delta p_{n-1}^{k,1}, \Delta p_0^{k,2}, \cdots, \Delta p_{n-1}^{m,2^m}\right\}.$$

Now write

$$P' = \{p'_0, \cdots, \; p'_{n2^i}\}$$

The discrete derivative of $c(t)$ after m subdivisions is

$$D_m[c(t)] = \mathcal{L}(P', [0,1]).$$

For a degree n curve subdivided m times we have [8]

$$\|D_m[c(t)] - \frac{d}{dt}c(t)\|_\infty \le \left(\frac{1}{2}\right)^{2m+1} N_\infty(n-1)n\|\Delta_2(\Delta P)\|_\infty.$$

Subdividing m_2 times ensures that we can approximate the derivative within a specified ϵ_d where

$$m_2 = \left\lceil -\frac{1}{2}\left(1 + \log_2 \frac{\epsilon_d}{N_\infty(n-1)n\|\Delta_2(\Delta P)\|_\infty}\right)\right\rceil \quad (2)$$

3.3 Establishing Double Normals

For a Bézier curve c and distinct $s, t \in [0,1]$ define the quadratic form

$$\langle c(s), c(t) \rangle_{\mathcal{D}} = \frac{[c(s) - c(t)] \cdot c'(s)}{\|c(s) - c(t)\|}.$$

Notice that $c(s)$ and $c(t)$ establish a double normal if and only if

$$\langle c(s), c(t) \rangle_{\mathcal{D}} = \langle c(t), c(s) \rangle_{\mathcal{D}} = 0.$$

The subscript $\mathcal{D}$ here denotes the fact that we are using the continuous derivative.

4 Using PL Structure to Calculate $\mathrm{MSD_c}$

In the previous section we presented a PL approximation to a Bézier curve and its derivatives. Also we defined a quadratic form that we can use to test if given points on the curve form a double normal. The transition to use of the discrete derivative is established through the modified quadratic form

$$\langle c(s), c(t) \rangle_d = \frac{[c(s) - c(t)] \cdot D_m[c(s)]}{\|c(s) - c(t)\|}.$$

The subscript d here indicates that we are using the discrete derivative.

4.1 Testing for Candidate Double Normals

Assume that the user has provided some $\epsilon > 0$ and that we have refined the PL structure with m subdivisions, so it is within $\frac{\epsilon}{2}$ of the curve and the derivative is approximated by the discrete derivative within $\frac{\epsilon}{2}$.

Define $\vec{\gamma} \in \mathbb{R}^3$ so that $D_m[c(s)] = c'(s) + \vec{\gamma}$. Note that $\|\vec{\gamma}\| \le \frac{\epsilon}{2}$. Also, notice that if $c(s)$ and $c(t)$ establish a double normal, i.e. $\langle c(s), c(t) \rangle_{\mathcal{D}} = 0$, then

$$\begin{aligned} \langle c(s), c(t) \rangle_d &= \frac{[c(s) - c(t)] \cdot D_m[c(s)]}{\|c(s) - c(t)\|} \\ &= \frac{[c(s) - c(t)] \cdot [c'(s) + \vec{\gamma}]}{\|c(s) - c(t)\|} \\ &= \frac{[c(s) - c(t)] \cdot \vec{\gamma}}{\|c(s) - c(t)\|} \end{aligned}$$

The Cauchy-Schwarz inequality states that for all vectors x and y of an inner product space it is true that

$$|\langle x, y\rangle| \leq \|x\|\|y\|.$$

So applying the Cauchy-Schwarz inequality yields

$$|\langle c(s), c(t)\rangle_d| \leq \frac{\epsilon}{2}$$

Consider two line segments from the PL representation of c and calculate the minimum distance between the segments. Suppose this minimum distance is realized by the line segment with end points $\mathcal{L}(P,[0,1])(t_0)$ and $\mathcal{L}(P,[0,1])(s_0)$. Then, if

$$|\langle c(s_0), c(t_0)\rangle_d| \leq \frac{\epsilon}{2} \quad \text{and} \quad |\langle c(t_0), c(s_0)\rangle_d| \leq \frac{\epsilon}{2}$$

we consider the line to be a good approximation of a double normal.

4.2 Estimating $\mathbf{MSD_c}$

Denote the exact MSD_c by σ. There exist distinct points $c(s)$ and $c(t)$, such that $d(c(s), c(t)) = \sigma$. We compute λ_c, our estimate for σ as the minimum of the distance between all pairs of disjoint edges of the approximating control polygon. There will exist distinct points p and q from those edges such that $d(p, q) = \lambda_c$.

Lemma 1 *Let ℓ_0 be the length of the longest edge of a given control polygon before any subdivision has occured and ℓ_m be the length of the longest edge after m subdivisions. Then*

$$\ell_m \leq \frac{\ell_0}{2^m}.$$

Proof. This follows from the definition of subdivision with subdivision parameter of 1/2. □

Applying the triangle inequality we see that

$$\sigma = d(c(s), c(t)) \leq d(c(s), p) + d(p, q) + d(q, c(t))$$

Let L_1 be the line segment in the PL approximation that contains the point p, and L_2 be the line segment passing through $c(s)$ with the same length as L_1 so that L_1 and L_2 form opposite sides of a rectangle (see figure 5). Let d_1 denote the diagonal of the rectangle.

We note that $d_1 < d_1^2$, *unless* $d_1 < 1$. In the ensuing analyses, we wish to consider specific numeric bounds on *epsilon*, where we will typically assume that $\epsilon << \ell_m$. To do so, we will make the further simplifying assumption that $\ell_m \geq 1$. Theoretically, the length of each of the finitely many edges could be divided by $\ell_m > 0$, normalizing the measuring scale. Pragmatically, this ensures that the user can conveniently choose values of ϵ and ϵ_d that are small relative to 1.

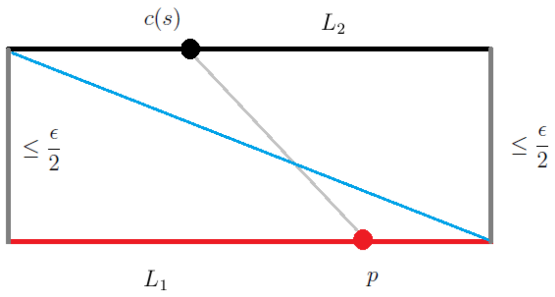


Figure 5: Estimating distance with right triangle

Note that

$$|L_1| = |L_2| \leq \frac{\ell_0}{2^m}$$

and

$$d(c(s), p) \leq d_1.$$

Applying the Pythagorean theorem we have

$$d_1 \leq d_1^2 \leq \left(\frac{\epsilon}{2}\right)^2 + \left(\frac{\ell_0}{2^m}\right)^2 = \frac{\epsilon^2}{4} + \frac{\ell_0^2}{2^{2m}}.$$

So

$$d(c(s), p) \leq \frac{\epsilon^2}{4} + \frac{\ell_0^2}{2^{2m}}.$$

Similarly

$$d(c(t), q) \leq \frac{\epsilon^2}{4} + \frac{\ell_0^2}{2^{2m}}.$$

So we have

$$\begin{aligned} \sigma &\leq 2\left(\frac{\epsilon^2}{4} + \frac{\ell_0^2}{2^{2m}}\right) + \lambda_c \\ &= \frac{\epsilon^2}{2} + \frac{\ell_0^2}{2^{2m-1}} + \lambda_c. \end{aligned}$$

Also,

$$\begin{aligned} \lambda_c &= d(p, q) \\ &\leq d(p, c(s)) + d(c(s), c(t)) + d(c(t), q) \\ &\leq \frac{\epsilon^2}{2} + \frac{\ell_0^2}{2^{2m-1}} + \sigma \end{aligned}$$

So,

$$\sigma \in [\lambda_c - E, \lambda_c + E], \quad \text{where } E = \frac{\epsilon^2}{2} + \frac{\ell_0^2}{2^{2m-1}}.$$

In order to choose a number of subdivisions to minimize E, recall that

$$\left(\frac{1}{2}\right)^{2m} \|\Delta_2 P\|_\infty N_\infty(n) \leq \epsilon,$$

and the fact that ϵ depends on m. Let δ denote our maximum error tolerance, and establish $E \leq \delta$. Let $K = \|\Delta_2 P\|_\infty N_\infty(n)$. Then

$$\left(\frac{1}{2}\right)^{2m} K < \epsilon \quad \text{and} \quad E = 2\left(\frac{\epsilon^2}{4} + \frac{\ell_0^2}{2^{2m}}\right). \quad \text{So}$$

$$E \quad < \quad 2\left(\frac{\epsilon^2}{4}+\frac{\epsilon\ell_0^2}{K}\right)=\frac{\epsilon^2}{2}+\frac{2\epsilon\ell_0^2}{N_\infty(n)\|\Delta_2 P\|_\infty}$$

So if we choose ϵ so that

$$\frac{\epsilon^2}{2}+\frac{2\epsilon\ell_0^2}{N_\infty(n)\|\Delta_2 P\|_\infty}\le\delta,$$

then clearly $E \le \delta$. This involves solving the following quadratic in ϵ,

$$(1/2)\epsilon^2+\frac{2\ell_0^2\epsilon}{N_\infty(n)\|\Delta_2 P\|_\infty}-\delta\le 0. \tag{3}$$

Choose half the smallest positive root of this quadratic to substitute for ϵ in equation (1).

```
MSD_c Estimation Algorithm

Input: $\delta, c, \epsilon_d$
0. Calculate $m_1, m_2$ and let $m = \max\{m_1, m_2\}$.
1. Subdivide $m$ times to get PL approximation of $c$.
2. Compute $d(l_i, l_j)$, the distances between line segments in the PL structure, where $\mathcal{L}(P,[0,1])(t_0)$ and $\mathcal{L}(P,[0,1])(s_0)$ realize $d(l_i, l_j)$.
3. If $|\langle c(s_0), c(t_0)\rangle_d| \le \epsilon_d$ and $|\langle c(t_0), c(s_0)\rangle_d| \le \epsilon_d$ then keep as double normal.
4. Take minimum from Step 3. as $\lambda_c$.

Output: $\lambda_c$
```

Figure 6: Algorithm for estimating $\mathrm{MSD_c}$

4.3 Efficient Updates by Data Reduction

The previous approach [15] constructed a PL approximation of c by uniformly partitioning $[0,1]$ as

$$0 = s_0 < s_1 < \cdots < s_{v-1} < s_v = 1,$$

where

$$|s_{i+1}-s_i| < \min\left\{\frac{\epsilon}{\sqrt{3}K_0}, \frac{\sin\left(\frac{\epsilon}{2}\right)\mu_0}{K_1}\right\}.$$

Here K_0 is the maximum value of $\|c'(t)\|_\infty$, K_1 is the maximum value of $\|c''(t)\|_\infty$ and μ_0 is the minimum value of $\|c'(t)\|_\infty$. The points in the partition are the end points of the curves resulting from subdivision. The number of subdivisions required, $\tilde{m}$, is given by $\tilde{m} = \max\{\tilde{m}_1, \tilde{m}_2\}$ where

$$\tilde{m}_1 = \left\lceil -\log_2\left(\frac{\epsilon}{\sqrt{3}K_0}\right)\right\rceil \text{ and}$$

$$\tilde{m}_2 = \left\lceil -\log_2\left(\frac{\sin\left(\frac{\epsilon}{2}\right)\mu_0}{K_1}\right)\right\rceil$$

Our algorithm will use fewer subdivisions when $m < \tilde{m}$, given by our comprehensive analysis of 4 cases:
Case 1: $m_1 < \tilde{m}_1$. This is true if

$$-\frac{1}{2}\log_2\left(\frac{\epsilon}{\|\Delta_2 P\|_\infty N_\infty(n)}\right) < -\log_2\left(\frac{\epsilon}{\sqrt{3}K_0}\right)+1.$$

In other words if

$$\frac{3K_0^2}{\|\Delta_2 P\|_\infty N_\infty(n)\epsilon} > \frac{1}{4} \tag{4}$$

then $m_1 < \tilde{m}_1$. If there is no restriction on ϵ then clearly we can choose ϵ small enough so (4) holds. Otherwise (4) will hold unless K_0 is small, the degree of the curve is very large or $\|\Delta_2 P\|_\infty$ is large (i.e. the control polygon has a narrow spike). So for example if $\epsilon = 0.01$ and the curve is cubic

$$\frac{3K_0^2}{\|\Delta_2 P\|_\infty N_\infty(n)\epsilon} = 300\left(\frac{M_0^2}{\|\Delta_2 P\|_\infty}\right).$$

So the ratio on the right would need to be less than $\frac{1}{1200}$ for the above inequality to be reversed.
Case 2: $m_2 < \tilde{m}_2$ when

$$\frac{\sin\left(\frac{\epsilon}{4}\right)\mu_0\sqrt{n\|\Delta_2(\Delta P)\|_\infty N_\infty(n-1)}}{\sqrt{\epsilon}K_1} < \sqrt{8}$$

This holds unless K_1 is small, μ_0 is large, $\|\Delta_2(\Delta P)\|_\infty$ is large, or the degree is large. So for a cubic with $\epsilon = 0.01$ for this inequality to be false $\dfrac{\mu_0}{K_1} > 130.6$.
Case 3: $m_1 < \tilde{m}_2$ if

$$\frac{\sin\left(\frac{\epsilon}{2}\right)\mu_0\sqrt{\|\Delta_2 P\|_\infty N_\infty(n)}}{K_1\sqrt{\epsilon}} < 2.$$

This will not hold for large μ_0, small K_1 or curves of large degree. Note that $\|\Delta_2 P\|_\infty$ is an approximation of K_1.
Case 4: $m_2 < \tilde{m}_1$ if

$$\frac{N_\infty(n-1)\|\Delta_2(\Delta P)\|_\infty n\epsilon}{K_0^2} < \frac{8}{3},$$

which holds unless K_0 is small, $\|\Delta_2(\Delta P)\|_\infty$ is large or the degree is large. So outside of the circumstances described above $m < \tilde{m}$. The analysis suggests avoiding 'high degree' curves, as can generally be done in graphics [11]. The other characteristics in the shape of the curve will be assessed in problem specific contexts. Recall, also that the number of approximating segments is exponential in m and $\tilde{m}$, so that even modest differences here can have a significant effect on performance of the topological updating, which depends directly on the number of approximating segments. This is now shown with a representative example.

5 Representative Example

Consider a composite cubic Bézier curve, consisting of five sub-curves, that forms a trefoil knot (figure 7). The control points and calculations are given in the extended version of this paper [9]. Choosing $\delta = 0.25$ and $\epsilon_d = 0.01$, gives $\epsilon = 0.01$, so $m = 6$. The maximum error is given by $E = 0.0034415$. This gives a maximum relative error of approximately 0.0078. Note that 6 subdivisions yields 960 line segments. Comparatively the previous approach involved 10,240 line segments [15].

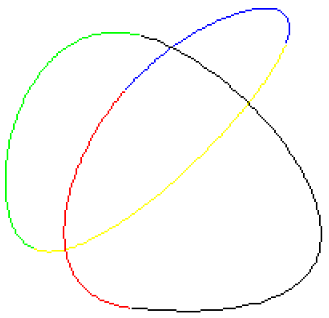


Figure 7: Composite Bézier Curve

6 Conclusion and Future work

We present theory, accompanied by an illustrative example, for efficient updates to constraints for preserving the topological embedding of curves during dynamic visualization of molecular simulations. In high performance computing applications, an accompanying visualization could have millions of frames, so it also becomes important to assess accumulated numerical errors over the total time interval upon realistically challenging data sets. In principle, the dynamic topological results presented here for curves should extend to surfaces, but practical testing on surface data remains a future consideration.

References

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

[2] N. Amenta, T. J. Peters, and A. C. Russell. Computational topology: ambient isotopic approximation of 2-manifolds. *Theoretical Computer Science*, 305(1-3):3–15, 2003.

[3] L.-E. Andersson, T. J. Peters, N. F. Stewart, and S. M. Doney. Polyhedral perturbations that preserve topological form. *Computer Aided Geometric Design*, 12(8):785–799, 1995.

[4] T. Ashton, J. Cantarella, M. Piatek, and E. J. Rawdon. Knot tightening by constrained gradient descent. *Experimental Mathematics*, 20(1):57–90, 2011.

[5] J. Cantarella, G. Kuperberg, R. B. Kusner, and J. M. Sullivan. The second hull of a knotted curve. *American Journal of Mathematics*, 125(6):1335 – 1348, 2003.

[6] J. Cantarella, R. B. Kusner, and J. M. Sullivan. On the minimum ropelength of knots and links. *Inventiones Mathematica*, 150(2):257 – 286, 2002.

[7] J. Cantarella, M. Piatek, and E. Rawdon. Visualizing the tightening of knots. In *VIS '05: Proceedings of the conference on Visualization '05*, pages 575–582, Washington, DC, USA, 2005. IEEE Computer Society.

[8] H. Cassidy and T.J.Peters. Spline operators for subdivision and differentiation, November 2011. pre-print.

[9] H. Cassidy, T.J.Peters, and K. Jordan. Dynamic computational topology for piecewise linear curves. www.engr.uconn.edu/ tpeters/some-pubs.html, June 2012. pre-print.

[10] K. L. Clarkson. Building triangulations using epsilon-nets. In J. M. Kleinberg, editor, *STOC*, pages 326–335. ACM, 2006.

[11] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice, second edition.* Addison-Wesley Professional, 1990.

[12] J. Gravesen. De Casteljau's algorithm revisited. In *Mathematical methods for curves and surfaces, II (Lillehammer, 1997)*, Innov. Appl. Math., pages 221–228. Vanderbilt Univ. Press, Nashville, TN, 1998.

[13] K. E. Jordan, L. E. Miller, E. L. F. Moore, T. J. Peters, and A. C. Russell. Modeling time and topology for animation and visualization with examples on parametric geometry. *Theoretical Computer Science*, 405:41–49, 2008.

[14] T. Maekawa, N. M. Patrikalakis, T. Sakkalis, and G. Yu. Analysis and applications of pipe surfaces. *Computer Aided Geometric Design*, 15:437–458, 1998.

[15] L. Miller, E. Moore, T. Peters, and A. Russell. Topological neighborhoods for spline curves : Practice and theory. In P. Hertling et al., editors, *Reliable Implementation of Real Number Algorithms: Theory and Practice*, volume 5045 of *LNCS*, pages 149–161. Springer, 2008.

[16] G. Morin and R.Goldman. On the smooth convergence of subdivision and degree elevation for Bézier curves. *Computer Aided Geometric Design*, 18:657–666, 2001.

[17] D. Nairn, J. Peters, and D. Lutterkort. Sharp, quantitative bounds on the distance between a polynomial piece and its bézier control polygon. *Computer Aided Geometric Design*, 16:613–631, 1999.

[18] L. Piegl and W. Tiller. *The NURBS Book*. Springer, 2nd edition, 1997.

[19] P. Plunkett, M. Piatek, et al. Total curvature and total torsion of knotted polymers. *Macromolecules*, 40(10):38603867, 2007.

Finding a Lost Treasure in Convex Hull of Points From Known Distances

Bahman Kalantari[*]

Abstract

Given a set of points $S = \{v_1, \dots, v_n\} \subset \mathbb{R}^m$, and a set of positive numbers r_i, $i = 1, \dots, n$, we wish to determine if there exists $p \in \mathrm{Conv}(S)$ such that $d(p, v_i) = r_i$ for all $i = 1, \dots, n$, where $d(\cdot, \cdot)$ denotes the Euclidean distance. We refer to this as the *ambiguous convex hull problem*. Given $\epsilon > 0$, we describe an algorithm that in $O(mn\epsilon^{-2} \ln \epsilon^{-1})$ arithmetic operations computes $p_\epsilon \in \mathrm{Conv}(S)$ such that one of the three conditions hold; (1): $|d(p_\epsilon, v_i) - r_i| < \epsilon$, for all $i = 1, \dots, n$; (2): $d(p_\epsilon, v_i) < r_i$, for all $i = 1, \dots, n$; (3): $d(p_\epsilon, v_i) > r_i$, for all $i = 1, \dots, n$. In case of (2), no point p with prescribed distances belongs to $\mathrm{Conv}(S)$. In case of (3), no point p with prescribed distances exists. In case of (1), we give an estimate on $d(p_\epsilon, p)$. The algorithm is a variation of the Triangle Algorithm in [8] for the convex hull decision problem where p is given explicitly.

1 Introduction

The *convex hull decision problem* is to test if a given point $p \in \mathbb{R}^m$ lies in the convex hull of a given set of points $S = \{v_1, \dots, v_n\} \subset \mathbb{R}^m$, denoted by $\mathrm{Conv}(S)$. This problem is a very special case of the *convex hull problem*, a problem that represents various descriptions of a polytope that is either specified as the convex hull of a finite point set or the intersection of a finite number of halfspaces, see Goodman and O'Rourke [5]. For more general convex hull problems and corresponding algorithms see, [5], Clarkson [4], Chan [1], Chazelle [2].

The convex hull decision problem is a fundamental problem in computational geometry and in linear programming (LP). A general LP problem can be formulated as a single LP feasibility problem, see e.g. Chvátal [3]. Then, given a bound on the norm of the vertices, a number that can be estimated for integer inputs, the latter problem in turn can be converted into the convex hull decision problem. Any LP algorithm can be applied to solve the convex hull decision problem. It can be argued that several polynomial-time algorithms for LP are in fact specifically designed to solve the convex hull decision problem with $p = 0$. These include, the ellipsoid algorithm of Khachiyan [11], the projective algorithm of Karmarkar[9], and the positive semidefinite diagonal matrix scaling algorithm of Khachiyan and Kalantari [12]. See [6] and also [7].

In a recent article, [8], we offered characterization theorems and a simple fully polynomial-time approximation algorithm, called the *Triangle Algorithm* for the convex hull decision problem having the following properties: Given $\epsilon \in (0, 1)$, in $O(mn\epsilon^{-2})$ arithmetic operations the algorithm computes a point $p_\epsilon \in \mathrm{Conv}(S)$, where either $d(p_\epsilon, p) \leq \epsilon d(p, v_j)$ for some j; or for all $i = 1, \dots, n$, $d(p_\epsilon, v_i) < d(p, v_i)$. The following characterization theorem in [8] plays an important role in the development and correctness of the Triangle Algorithm.

Theorem 1 *Let $S = \{v_1, \dots, v_n\} \subset \mathbb{R}^m$ be a given set of points and let $p \in \mathbb{R}^m$ be given. Then exclusively, either $p \in \mathrm{Conv}(S)$ and for any $p' \in \mathrm{Conv}(S)$ there exists $v_j \in S$ such that $d(p', v_j) \geq d(p, v_j)$, or $p \notin \mathrm{Conv}(S)$ and there exists $p' \in \mathrm{Conv}(S)$ such that $d(p', v_i) < d(p, v_i)$, for all $i = 1, \dots, n$.*

Each $p' \in \mathrm{Conv}(S)$ that satisfies $d(p', v_i) < d(p, v_i)$, for all $i = 1, \dots, n$, acts as a *witness* to the infeasibility of p. The set W_p of all such witnesses is the intersection of $\mathrm{Conv}(S)$ and open balls B_i of radius $d(p, v_i)$ centered at v_i, $i = 1, \dots, n$ and forms a convex open set in the relative interior of $\mathrm{Conv}(S)$. A corollary of Theorem 1 reveals a property of a set of intersecting open balls:

If a set of n open balls in $\mathbb{R}^m$ has a common boundary point p, their intersection is empty, if and only if p lies in the convex hull of their centers.

This property suggests we can define a geometric "dual" problem to the convex hull decision problem, the *intersecting open balls problem*:

Given a set of n open balls in $\mathbb{R}^m$ that are known to have a common boundary point p, determine if they have a nonempty intersection.

The Triangle Algorithm in particular is capable of solving the intersecting open balls problem. It can be used to solve some other versions of the convex hull problem, e.g. the *irredundancy problem* where all the vertices are to be determined. The Triangle Algorithm can also be used to solve a linear programming problem (see [8]) and as such offers an alternative to polynomial-time algorithms for linear programming, as well as the simplex method whose randomized version is shown to run in polynomial-time, see [10]. All known polynomial-time LP algorithms for integer inputs have operation complexity that is polynomial in m, n, and the size of

[*]Department of Computer Science, Rutgers University, kalantari@cs.rutgers.edu

encoding of the input data, often denoted by L, see [11]. As approximation schemes, polynomial-time algorithms for LP have complexity polynomial in the dimension of the data and $\ln(\epsilon^{-1})$. As is well-known, for integral inputs, by rounding, any approximate solution having sufficient precision can be turned into an exact solution. It is likely the Triangle Algorithm performs better than its worst-case analysis. Its average case or randomized versions could prove to give much better complexity. For fixed ϵ it produces approximate solutions in $O(mn)$ operations.

In this article we consider the *ambiguous convex hull problem.* It differs with the convex hull decision problem in that we do not know p explicitly, but we are given a set of distances r_i, $i = 1, \dots, n$, presumed to equal $d(p, v_i)$. In particular, the distances may not even be valid distances of any point p in $\mathrm{Conv}(S)$, or in its complement. Thus there is ambiguity to the problem. Two scenarios where the ambiguous convex hull problem applies well are as follows. The first, where we are given claimed distances of a hidden treasure from a set of known sites. The second, where we wish to determine the feasibility of building a site at prescribed distances in the convex hull of a given set of sites. Analogous to the Triangle Algorithm itself, the variation to be described here, called *Blindfold Triangle Algorithm*, is geometric in nature, simple in description, and very easy to implement, having worst-case complexity of $O(mn\epsilon^{-2} \ln \epsilon^{-1})$ arithmetic operations.

If a subset of $m + 1$ points in S that are in general position are identifiable, we can determine the coordinates of p by solving an $m \times m$ linear system, see Section 5. Doing so will reduce the problem to the convex hull decision problem. However, our goal is to avoid solving such linear systems. Furthermore, as will be seen in the final section of the article, we wish to argue that, at least in some cases, solving a linear system corresponds to an ambiguous convex hull problem. Thus such linear systems can be solved approximately in $O(m^2\epsilon^{-2} \ln(\epsilon^{-1}))$ (see Section 6).

While the complexity analysis of the Blindfold Triangle Algorithm is independent of Triangle Algorithm itself, it makes use of Theorem 1. For the sake of completeness, we will first give a proof of this theorem, somewhat different than the proof presented in [8].

2 A Voronoi Diagram-Based Proof of Theorem 1

In this section we present a simple proof of Theorem 1. Suppose $p \in \mathrm{Conv}(S)$. Let p' be any point in $\mathrm{Conv}(S) - \{p\}$. Consider $V(p) = \{x \in \mathbb{R}^m : \quad d(p, x) < d(p', x)\}$, i.e. the Voronoi cell of p with respect to the two point set $\{p, p'\}$. We wish to show $\overline{V}(p) = \{x \in \mathbb{R}^m : \quad d(p, x) \leq d(p', x)\}$ intersects S. If not, then S is a subset of $V(p') = \{x \in \mathbb{R}^m : \quad d(p', x) < d(p, x)\}$. Since $V(p')$ is convex, it follows that it must contain $\mathrm{Conv}(S)$. This contradicts that $p \in \mathrm{Conv}(S)$.

Conversely, suppose $p \notin \mathrm{Conv}(S)$. Let p' be the point in $\mathrm{Conv}(S)$ that is closest to p. We claim p' is a witness, i.e. $d(p', v_i) < d(p, v_i)$ for all i. For any $v_i \neq p'$, the angle $\angle pp'v_i$ cannot be acute since otherwise this would contradict p' being the closest point of $\mathrm{Conv}(S)$ to p. This implies that $d(p', v_i) < d(p, v_i)$. If $p' = v_j$ for some j, then clearly the inequality is also satisfied for v_j. Hence the proof of the Theorem 1.

Remark This simple proof also implies the separating hyperplane theorem for the convex hull of a finite point set. The converse is however not true, that is, the separating hyperplane theorem does not imply Theorem 1. In this sense Theorem 1 is a stronger separation theorem. Theorem 1 can also be stated for a polytope that is described by the intersection of halfspaces. The proof however would have to rely on a further result: any point in the polytope can be written as the convex combination of its vertices.

3 Getting Closer to The Treasure

Given a point $p' \in \mathrm{Conv}(S)$, as the current estimate to the location of p, we wish to compute a new point $p'' \in \mathrm{Conv}(S)$ that reduces the current distance $d(p', p)$, referred here as the *gap*. However, since the location of p is unknown we must select p'' in such a way that guarantees improvement. The following theorem describes how this can be achieved.

Theorem 2 *Let $\epsilon > 0$ be given. Consider three given points $p, p', v \in \mathbb{R}^m$ satisfying the following conditions:*
(i) $r' = d(p', v) \geq r = d(p, v)$.
(ii) $\delta = d(p', p) \geq \epsilon$.
Let p'' be the point on the line segment $p'v$ such that

$$d(p', p'') = \frac{\epsilon^2}{2r'}. \tag{1}$$

Then if $d(p'', p) = \delta'$, we have

$$\frac{\delta'}{\delta} \leq \sqrt{1 - \frac{\epsilon^2}{2(r'^2 + r^2)}}. \tag{2}$$

In particular, if $r' \leq 2r$, then

$$\frac{\delta'}{\delta} \leq \sqrt{1 - \frac{\epsilon^2}{10r^2}}. \tag{3}$$

Proof. Without loss of generality we may assume that p, p', v lie in a Euclidean plane, where $v = (0, 0)$, $p' = (r', 0)$ and $p = (x, y)$, see Figure 1. Consider the concentric circles of radii r and r' centered at v. Let q be a point lying on the circle of radius r', satisfying $d(p', q) = \epsilon$. See Figure 1 where one of the two such

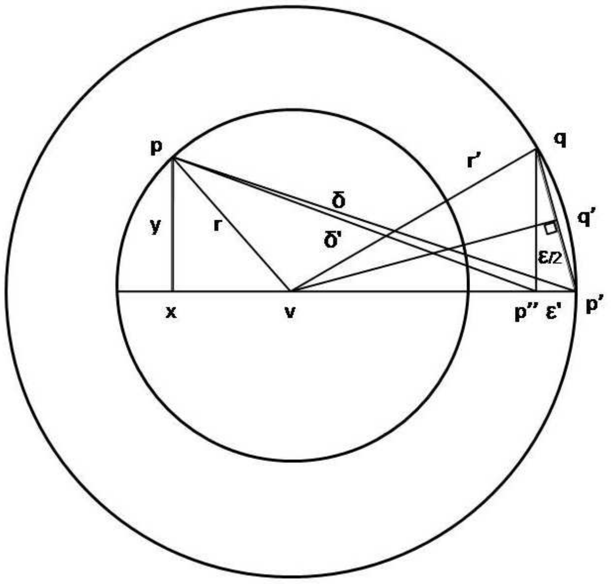


Figure 1: Reduction of $\delta = d(p', p)$ to $\delta' = d(p'', p)$.

points are shown. Draw the line from q perpendicular to the line $p'v$ and let p'' be the base of this line. We claim that this point coincides with p'' as defined in (1). We do so by computing $d(p', p'') = \epsilon'$. Let q' be the midpoint of the chord qp'. By a property of a circle, the line vq' is perpendicular to the line qp'. Consider the right triangles $\triangle qp'p''$ and $\triangle q'p'v$. They are similar since they have a common angle, $\angle qp'p''$. Therefore, we have

$$\frac{d(p', p'')}{d(p', q)} = \frac{d(p', q')}{d(v, p')}. \tag{4}$$

Substituting, equivalently we have,

$$\frac{\epsilon'}{\epsilon} = \frac{\epsilon/2}{r'}. \tag{5}$$

This gives (1). From this, that $p = (x, y)$, and since by assumption (ii) $\delta \geq \epsilon$, it follows that

$$x \in [-r, \rho], \quad \rho = \min\{r, r' - \epsilon'\}. \tag{6}$$

Now consider x as a variable ranging in the interval $[-r, \rho]$. Since $y^2 = r^2 - x^2$, the corresponding ratio, δ'/δ can be written explicitly as a function of x:

$$\frac{\delta'}{\delta} = \sqrt{\frac{(r' - \epsilon' - x)^2 + (r^2 - x^2)}{(r' - x)^2 + (r^2 - x^2)}}. \tag{7}$$

We will compute a bound on the maximum of the above ratio for $x \in [-r, \rho]$. It is more convenient to consider

$$f(x) = \frac{(r' - \epsilon' - x)^2 + (r^2 - x^2)}{(r' - x)^2 + (r^2 - x^2)}. \tag{8}$$

We will prove that for $x \in [-r, \rho]$ we have

$$f(x) \leq \left(1 - \frac{\epsilon' r'}{r'^2 + r^2}\right). \tag{9}$$

We will consider two cases.

Case I. $r \leq r' - \epsilon'$. We first claim that in this case for any $x \in (0, \rho]$, we have

$$f(-x) \geq f(x). \tag{10}$$

It is easy to verify that the above is true if and only if

$$\epsilon' x \Big(r'(r' - \epsilon') - r^2\Big) \geq 0. \tag{11}$$

This holds true under the given assumption of the case. Thus it suffices to consider the maximum of $f(x)$ in the interval $[-r, 0]$. Differentiating $f(x)$ and simplifying its numerator, we get

$$2\epsilon' \Big(\epsilon' r' - (r'^2 - r^2)\Big). \tag{12}$$

This quantity never vanishes unless,

$$\epsilon' = \frac{(r'^2 - r^2)}{r'}. \tag{13}$$

It can be shown for this value of ϵ' both the numerator and the denominator of $f(x)$ have the same root,

$$x = \frac{r^2 + r'^2}{2r'} \geq r, \tag{14}$$

a value outside of the interval $[-r, 0]$. Thus, we only need to compare $f(0)$ and $f(-r)$. In fact since $\epsilon' \leq r' - r$, the quantity in (12) is negative, thus $f(x)$ is decreasing. Hence the maximum value of $f(x)$ on $[-r, \rho]$ is

$$f(-r) = \frac{(r' + r - \epsilon')^2}{(r' + r)^2} = \left(1 - \frac{\epsilon'}{r' + r}\right)^2. \tag{15}$$

We claim

$$f(-r) \leq \left(1 - \frac{\epsilon' r'}{r'^2 + r^2}\right). \tag{16}$$

Equivalently, we claim

$$\left(1 - \frac{\epsilon' r'}{r'^2 + r^2}\right) \geq \left(1 - \frac{\epsilon'}{r' + r}\right)^2 =$$

$$1 + \frac{\epsilon'^2}{(r' + r)^2} - \frac{2\epsilon'}{r' + r}. \tag{17}$$

Simplifying, this amounts to showing

$$\frac{2}{r' + r} \geq \frac{r'}{r'^2 + r^2} + \frac{\epsilon'}{(r' + r)^2}. \tag{18}$$

It is easy to show that

$$\frac{2r'}{(r'+r)^2} \geq \frac{r'}{r'^2+r^2}. \tag{19}$$

Finally, we can verify the following

$$\frac{2}{r'+r} \geq \frac{2r'}{(r'+r)^2} + \frac{\epsilon'}{(r'+r)^2}. \tag{20}$$

Thus we have proved (16).

Case II. $r > r' - \epsilon'$. By our previous statement on a root of $f'(x)$ and (14), and since in this case $\rho = r' - \epsilon'$, $f'(x)$ does not vanish on $[-r, \rho]$, so we only need to bound $f(r' - \epsilon')$. We claim

$$f(r'-\epsilon') = 1 - \frac{\epsilon'^2}{r^2 - r'^2 + 2r'\epsilon'} < (1 - \frac{\epsilon' r'}{r'^2+r^2}). \tag{21}$$

This is equivalent to proving

$$\frac{r'}{r'^2+r^2} < \frac{\epsilon'}{r^2 - r'^2 + 2r'\epsilon'}. \tag{22}$$

After simplification this is equivalent to

$$r'(r^2 - r'^2) < \epsilon'(r^2 - r'^2). \tag{23}$$

This is valid. Finally, substituting $\epsilon' = \epsilon^2/2r'$, we have proved (2), and using $r' \leq 2r$ in (2) we get (3). □

Corollary 3 *Let p, p', p'', v be as in Theorem 2 and set $R = \max\{d(p, v_i), i = 1, \ldots, n\}$. If $r' \leq 2r$, we have*

$$\delta' \leq \delta \sqrt{1 - \frac{\epsilon^2}{10R^2}} \leq \delta \exp\left(\frac{-\epsilon^2}{20R^2}\right). \tag{24}$$

Proof. The first inequality follows from Theorem 2 and the definition of R. To prove the next inequality, we use that $1 + x \leq \exp(x)$, and set $x = -\epsilon^2/10R^2$. □

4 The Blindfold Triangle Algorithm

In this section we describe a variation of the Triangle Algorithm described in [8] for solving the convex hull decision problem. As the Triangle Algorithm itself, given $p' \in \text{Conv}(S) - \{p\}$, the algorithm searches for a triangle $\triangle pp'v_j$ where $v_j \in S$ such that $d(p', v_j) \geq r_j$. Given such a triangle, the algorithm uses v_j as a pivot point to compute a new iterate $p'' \in \text{Conv}(S)$ such that $d(p'', p) < d(p', p)$. It replaces p'' with p' and repeats. Since the coordinates of p are not known, nor do we know if such prescribed distances correspond to an actual point, we will need to adjust the Triangle Algorithm to take conservative, but improving steps while making use of Theorem 2. For this reason we refer to this version as the *Blindfold Triangle Algorithm.*

The input to the algorithm is a prescribed tolerance $\epsilon > 0$, and a set of distances r_i, $i = 1, \ldots, n$, assumed to correspond to $d(p, v_i)$ for some point p. We assume to have selected an initial point $p' \in \text{Conv}(S)$. The Blindfold Triangle Algorithm is described as follows.

- **Step 1.** Pick any $p' \in \text{Conv}(S)$ (e.g. $p' = \frac{1}{n}\sum_{i=1}^{n} v_i$), check if

  $$|d(p', v_i) - r_i| \leq \epsilon, \quad \forall i = 1, \ldots, n.$$

  If the above holds, stop. We shall refer ro p' as an *ϵ-approximate solution.* Otherwise, go to Step 2.

- **Step 2.** Check if $d(p', v_i) > r_i$, $\forall i = 1, \ldots n$. If the above holds, stop. Otherwise, go to Step 3.

- **Step 3.** Check if there exists v_j such that $d(p', v_j) \geq r_j$. We shall refer to such v_j as the *pivot point.* If no such v_j exists, then $d(p', v_i) < d(p, v_i)$, for all $i = 1, \ldots, n$. Stop. Otherwise, go to Step 4.

- **Step 4.** If $r'_j = d(p', v_j) \geq 2r_j$, then set $p'' = v_j$. Replace p' with p'', go to Step 1. Otherwise, set p'' to be the point that takes a step of size $\epsilon' = \epsilon^2/2r'_j$ from p' in the direction of v_j. Replace p' with p'', go to Step 1. We refer to p'' as the *iterate.*

When p'' is not equal to v_j it is given by

$$p'' = \alpha p' + (1-\alpha)v_j, \quad \alpha = 1 - \frac{\epsilon'}{r'_j}. \tag{25}$$

Since p'' is a convex combination of p' and v_j, it will remain in $\text{Conv}(S)$. First, we state a result that characterizes the stopping conditions in the algorithm.

Theorem 4 *The algorithm termination is categorized as follows:*

(1): If the algorithm terminates in Step 1, it has determined an ϵ-approximate solution.

(2): If the algorithm terminates in Step 2, no point p with prescribed distances exists.

(3): If the algorithm terminates in Step 3, no point p with prescribed distances belongs to $\text{Conv}(S)$.

Proof. Part (1) is clear from definition. The proof of (2) follows from the fact that if such point p existed, by Theorem 1 it could not belong to $\text{Conv}(S)$. Consider the Voronoi cell $V(p')$ with respect to the two point set $\{p, p'\}$. Analogous to the proof of Theorem 1 in the present article we can argue that $\overline{V}(p') = \{x : d(x, p') \leq d(x, p)\}$ must necessarily contain a v_j, hence $d(p', v_j) \leq d(p, v_j)$. But this contradicts the termination criterion of Step 2. Part (3), follows from Theorem 1. □

Next we state some basic properties of the algorithm to be used in its complexity analysis. The proof is omitted as it is straightforward and analogous to that in [8].

Proposition 5 *The algorithm satisfies:*

(i) In each iteration the step-size α lies in $(0, 1]$.

(ii) Given an explicit representation of p' as a convex combination of v_i's, p'' can also be explicitly written as a convex combination of v_i's.

(iii) Each iteration of the algorithm uses at most $O(mn)$ arithmetic operations and comparisons.

(iv) Each v_i can be selected as an iterate p'' at most once (see definition of iterate in Step 4).

(v) If the point p exists, and if v_j is selected as a pivot point more than once, then except possibly for its first selection as an iterate, in any subsequent selection of v_j as a pivot point the iterate p'' will satisfy $d(p, p'') \leq r_j$.

By part (iv) of Proposition 5, the number of iterations where an element $v_j \in S$ is selected as an iterate is at most n. Therefore, except for n iterations, in any other iteration of the algorithm the inequality $r' \leq 2r$ holds and thus inequality (3) in Theorem 2 holds so that (24) in Corollary 3 applies. The analysis of complexity of the Blindfold Triangle Algorithm will make repeated use of (24). For the sake of simplicity in the forgoing complexity analysis we will exclude the occurrence of exceptional iterations where $r' > 2r$ so that we assume (24) applies in every iteration.

Theorem 6 *Assume p is in $\mathrm{Conv}(S)$. Pick $p_0 \in \mathrm{Conv}(S) - \{p\}$, let $\delta_0 = d(p_0, p) \geq \epsilon$. Set $R = \max\{d(p, v_i), i = 1, \dots, n\}$. Then the algorithm computes an ϵ-approximate solution in a finite number of iterations k_ϵ satisfying,*

$$k_\epsilon = \left\lceil \frac{20R^2}{\epsilon^2} \ln\left(\frac{\delta_0}{\epsilon}\right) \right\rceil = O\left(\epsilon^{-2} \ln(\frac{1}{\epsilon})\right).$$

Proof. From Corollary 3, given $p' \in \mathrm{Conv}(S)$ in an iteration, if the algorithm gets to compute p'', then if p with prescribed distances exists, we must have $d(p, p') > \epsilon$. Otherwise, from the triangle inequality we would have

$$|d(p', v_i) - d(p, v_i)| \leq \epsilon, \quad \forall i = 1, \dots, n.$$

Thus, Corollary 3 applies and from its repeated application we have

$$\delta_k \leq \delta_{k-1} \exp\left(- \frac{\epsilon^2}{20R^2}\right) \leq \cdots \leq \delta_0 \exp\left(- k\frac{\epsilon^2}{20R^2}\right).$$

In order to satisfy $\delta_k \leq \epsilon$, it suffices to solve for the smallest $k = k_\epsilon$ satisfying

$$\delta_0 \exp\left(- k\frac{\epsilon^2}{20R^2}\right) \leq \epsilon.$$

This gives the claimed k_ϵ. □

Remark. In each iteration we can continue to use the same pivot point v_j so long as $d(p', v_j) \geq r_j$, making the search for a pivot as efficient as possible.

5 Estimation of the Gap

Suppose we have computed a point p' in the convex hull of a subset of $m + 1$ points in S, forming a full-dimensional simplex in $\mathbb{R}^m$. Without loss of generality assume these points are $v_0, \dots, v_m$. Thus p' can be written as a convex combination of these points in a unique fashion. Also, the set of vectors $v_i - v_0$, $i = 1, \dots, m$ forms a linearly independent set in $\mathbb{R}^m$. We wish to represent p and p' in terms of $v_0, \dots, v_m$ and use it to estimate the gap $d(p, p')$, given the following

$$d(v_i, p) = r_i, \quad d(v_i, p') = r'_i, \quad i = 0, \dots, m, \tag{26}$$

$$|r_i - r'_i| \leq \epsilon, \quad i = 0, \dots, m. \tag{27}$$

Squaring the equations in (26), subtracting the first from the remaining m equations gives,

$$d^2(v_i, p) - d^2(v_0, p) = r_i^2 - r_0^2, \quad i = 1, \dots, m, \tag{28}$$

$$d^2(v_i, p') - d^2(v_0, p') = r_i'^2 - r_0'^2, \quad i = 1, \dots, m. \tag{29}$$

Equivalently, for $i = 1, \dots, m$ this gives

$$(v_0 - v_i)^T p = \gamma_i, \quad (v_0 - v_i)^T p' = \gamma'_i, \tag{30}$$

where for $i = 1, \dots, m$ we have

$$\gamma_i = \frac{1}{2}\left(r_i^2 - r_0^2 - d^2(v_i, 0) + d^2(v_0, 0)\right),$$

$$\gamma'_i = \frac{1}{2}\left(r_i'^2 - r_0'^2 - d^2(v_i, 0) + d^2(v_0, 0)\right). \tag{31}$$

Let W be the $m \times m$ matrix whose i-th row is

$$v_0^T - v_i^T, \quad i = 1, \dots, m.$$

Let $\gamma = (\gamma_1, \dots, \gamma_m)^T$, and $\gamma' = (\gamma'_1, \dots, \gamma'_m)^T$. Then we have

$$Wp = \gamma, \quad Wp' = \gamma'.$$

This implies

$$p - p' = W^{-1}(\gamma - \gamma').$$

From (27), and assuming Δ is the diameter of $\mathrm{Conv}(S)$, we have

$$|r_i^2 - r_i'^2| \leq \epsilon(r_i + r'_i) \leq 2\epsilon\Delta, \quad i = 0, \dots, m.$$

Thus for $i = 1, \dots, m$ we have

$$|\gamma_i - \gamma'_i| \leq \frac{1}{2}\left(|r_i^2 - r_i'^2| + |r_0'^2 - r_0'^2|\right) \leq 2\epsilon\Delta.$$

Hence we conclude

$$d(\gamma, \gamma') \leq 2\sqrt{m}\Delta\epsilon,$$

implying the following bound on the gap

$$d(p, p') \leq \|W^{-1}\| 2\sqrt{m}\Delta\epsilon, \tag{32}$$

where $\| \cdot \|$ denotes the 2-norm of a matrix.

Remark. When the data is integral, there exists an ϵ_* such that if $d(p, p') \leq \epsilon_*$, p also lies in the convex hull of $v_0, \dots, v_m$. This follows from the usual LP sensitivity. Thus, the algorithm can correctly solve the ambiguous convex hull problem when viewed as a decision problem.

6 Solving a Linear System as an Ambiguous Convex Hull Problem

Theorem 7 *Consider a linear system $Ax = b$, where A is an $m \times m$ invertible matrix. Let p be the solution to the equation. Assume we are given $v_0 \in \mathbb{R}^m$ and r_0 such that $d(p, v_0) = r_0$. Let $e = (1, \dots, 1)^T \in \mathbb{R}^m$, let v_i be the i-th row of the matrix $V = ev_0^T - A$, and let $r_i = d(v_i, p)$, $i = 1, \dots, m$. Then p is the unique solution to the set of equations*

$$d(v_i, p) = r_i, \quad i = 0, \dots, m.$$

Moreover, for $i = 1, \dots, m$ we have,

$$r_i^2 = 2b_i + r_0^2 + d^2(v_i, 0) - d^2(v_0, 0).$$

In particular, if $p \in \mathrm{Conv}\{v_0, \dots, v_m\}$, the Blindfold Triangle Algorithm can give an ϵ-approximate solution to $Ax = b$ in $O(m^2\epsilon^{-2}\ln(\epsilon^{-1}))$ arithmetic operations.

Proof. From the relationship between A, V, v_0, b, p, the definition of W and γ in (31) we have

$$Ap = b = Wp = (ev_0^T - V)p = \frac{1}{2}\gamma.$$

This completes the proof. $\square$

Theorem 7 suggests interesting computational possibilities in using the Blindfold Triangle Algorithm to solve a linear system, given that the distance of the solution to a single point is known, e.g. given the Euclidean norm of the solution as is the case when A is an orthogonal matrix. Such an approach would compute an approximate solution faster than the traditional method of computing LU factorization.

7 Remarks

In this article we have presented a variation of the Triangle Algorithm for the convex hull decision problem, called the *Blindfold Triangle Algorithm*. It tries to determine if there exists a point p in the convex hull of a given set of points S, knowing only a set of distances, presumably corresponding to the points in S. In contrast with the Triangle Algorithm it takes smaller steps because it does not know the coordinates of p, nor does it know if such a point exists. Despite the conservative steps it is only slower than the Triangle Algorithm by a factor of $\ln(\epsilon^{-1})$. This could also mean that the Triangle Algorithm itself should have a better complexity (see [8]). An interesting application of the Blindfold Triangle Algorithm is in solving certain linear systems (e.g. orthogonal coefficient matrix), offering a new approach for solving this problem (see Theorem 7). We are optimistic that both the Triangle Algorithm as well as the Blindfold Triangle Algorithm will offer alternative algorithms for the convex hull decision problem, its variations, linear programming, and more. These algorithms suggest new lines of research in several different areas. We hope to carry out some computational testing as well as pursue theoretical and practical applications of the results.

Acknowledgements. I am grateful to three anonymous reviewers for a very careful reading of this article and for their helpful comments and suggestions that resulted in improvements. I also thank Iraj Kalantari for a discussion on the use of a bisecting hyperplane to prove Theorem 1, resulting in a simpler geometric proof than the one given in [8]. As suggested by a reviewer, it is also possible to give a direct proof Theorem 1, removing the embedded contradiction.

References

[1] T. M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete Comput. Geom.*, 16(4):369–387, 1996.

[2] B. Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete Comput. Geom.*, 10:377–409, 1993.

[3] V. Chvátal. *Linear Programming.* W.H. Freeman and Company, New York, 1983.

[4] K.L. Clarkson. More output-sensitive geometric algorithm. In Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science, 695–702, 1994.

[5] J. E. Goodman, J. O'Rourke (Editors). *Handbook of Discrete and Computational Geometry*, 2nd Edition (Discrete Mathematics and Its Applications), Chapman & Hall Boca Raton, 2004.

[6] Y. Jin and B. Kalantari. A procedure of Chvátal for testing feasibility in linear programming and matrix scaling. *Linear Algebra and its Applications*, 416:795–798, 2006.

[7] B. Kalantari. Canonical problems for quadratic programming and projective methods for their solution. *Contemporary Mathematics*, 114:243–263, 1990.

[8] B. Kalantari. A characterization theorem and an algorithm for a convex hull problem. arXiv:1204.1873v1, 2012.

[9] N. Karmarkar. A new polynomial time algorithm for linear programming, *Combinatorica*, 4:373-395, 1984.

[10] J. A. Kelner and D. A. Spielman. A randomized polynomial-time simplex algorithm for linear programming. Proceedings of the 38th Annual ACM Symposium on Theory of Computing, 2006.

[11] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademia Nauk SSSR*, 1093–1096, 1979.

[12] L. Khachiyan and B. Kalantari. Diagonal matrix scaling and linear programming. *SIAM J. Optim.*, 4:668–672, 1992.

Optimal Average Case Strategy for Looking around a Corner

Reza Dorrigiv [*] Alejandro López-Ortiz [†] Selim Tawfik [†]

Abstract

A robot is free to move in a non-convex polygonal region, starting against an edge on the boundary. Ahead of the robot at one unit of distance is a corner, i.e. a reflex vertex (see Figure 1). The angle θ is unknown to the robot. The robot's task is to look at the region around the corner, which it initially cannot see.

We let $\varphi = \pi - \theta$. If $\varphi \geq \pi/2$, the robot is best off moving straight to the vertex. However, if φ is close to 0, much shorter paths exist, making this solution suboptimal. Therefore we ask: What is the best path for the robot to follow?

In this paper, we look into the problem of finding an optimal average-case strategy under a homogeneous probability distribution for φ. The average-case performance of a strategy is measured by its average cost, defined as the expected value of the strategy's competitive function. Given a value for φ, the competitive function of a strategy gives the ratio of the distance the robot travels to look around the corner (as prescribed by the strategy) to the shortest distance it must travel to do so.

We give strong evidence that an optimal average-case strategy exists and achieves an average cost of ~ 1.189.

1 Introduction

The corner exploration problem was first examined by Icking, Klein and Ma in [7]. The authors measure the performance of a strategy by its competitive factor, defined as the maximum value attained by its competitive function (lower is better). Under this measure, they show there exists an optimal strategy characterized as the solution of a certain differential equation. This strategy's competitive function is in fact identically equal to a constant $c \approx 1.21218$. Thus both its competitive factor and its average cost are equal to c. Although this strategy's competitive factor is optimal, strategies with better average costs exist.

The problem of finding an optimal average-case strategy reduces to that of finding a curve which minimizes an integral giving the average cost. The appropriate tool for handling such problems is the calculus of variations, on which we state a couple of useful theorems in the appendix. This said, formal proofs of some of our results would require complex techniques from that field. For the problem at hand, we obtain near optimal results using discretizations of the instance.

After giving discretizations of the problem, we apply some techniques from the calculus of variations to produce a strong candidate for an optimal average-case strategy. The average cost of this strategy is ~ 1.189. This is better than the average cost of the strategy described in [7], which is ~ 1.21218 as we noted earlier. However, we should state that the competitive factor of our strategy is worse: ~ 1.3136 for ours vs. ~ 1.21218 for [7].

Unfortunately, we do not have a closed-form for our strategy, so we propose a closed-form approximation whose average cost exceeds ours by less than 0.202%.

2 Competitive Strategies

2.1 Preliminaries

We need some of the early definitions and results in [7], which we reproduce here.

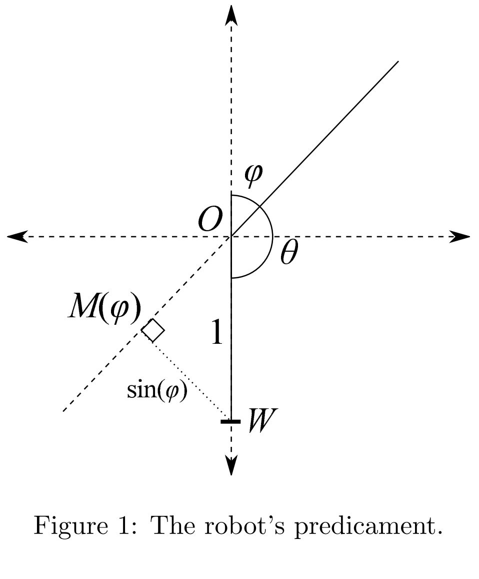


Figure 1: The robot's predicament.

We start by introducing a coordinate system with the origin located at the corner, and with the robot's starting position, W, one unit away from the origin. We let φ be the angle between the invisible wall and the pro-

*Faculty of Computer Science, Dalhousie University, Halifax, NS, B3H 4R2, Canada, email: rdorrigiv@cs.dal.ca

†David Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, N2L 3G1, Canada, email: {alopez-o,stawfik}@uwaterloo.ca

longation of the visible wall, $M(\pi)$. The distance from W to $M(\varphi)$ is denoted by $a(\varphi)$. We observe that

$$a(\varphi) = \begin{cases} \sin(\varphi) & : \quad \varphi < \frac{\pi}{2} \\ 1 & : \quad \varphi \geq \frac{\pi}{2} \end{cases} \qquad (1)$$

A *strategy* is a curve which starts at the point W and finishes on the prolongation of the visible wall, $M(\pi)$. For any possible value of φ, there is a point on such a curve from which the other wall is visible, the intersection of the curve with $M(\varphi)$.

We let $A_S(\varphi)$ be the length of the path described by strategy S between W and the first point of intersection with $M(\varphi)$. The *competitive function*, $f_S(\varphi)$, of S is the ratio of $A_S(\varphi)$ to $a(\varphi)$ and its *competitive factor*, c_S, is the supremum of the values taken by $f_S(\varphi)$ in $(0, \pi]$.

$$f_S(\varphi) = \frac{A_S(\varphi)}{a(\varphi)}, \qquad c_S = \sup_{\varphi \in (0,\pi]} f_S(\varphi)$$

We say that S is *competitive* if $c_S < \infty$. If a strategy reaches $M(\varphi')$ for the first time, turns back and meets $M(\varphi')$ again, this part of the path may be cut off and replaced by a radial line segment, giving a better strategy. Strategies with radial line segments can then be approximated arbitrarily closely by strategies that can be described in polar coordinates.

Definition 1 ([7]) A curve $S = (\varphi, s(\varphi))$, where s is defined on $[0, \pi]$, is called a *strategy* for the corner problem if the following holds.

(i) s is a continuous function on an interval $[0, \sigma]$, where $\sigma \leq \pi$.

(ii) On the open interval $(0, \sigma)$, s is piecewise continuously differentiable and $s'(0)$ exists (possibly $\pm\infty$).

(iii) s is *rectifiable*, i.e. s has finite arc length.[1]

(iv) $s(0) = 1$.

(v) If $s(\sigma) \neq 0$ then $\sigma = \pi$.

The last property says that the strategy must end somewhere on $M(\pi)$, possibly the corner. By agreeing that $s(\varphi) = 0$ for $\sigma < \varphi \leq \pi$, we can regard $s(\varphi)$ as defined on all of $[0, \pi]$.

Lemma 1 ([7]) *Let $S = (\varphi, s(\varphi))$ be a strategy. Then S is competitive iff $|s'(0)| < \infty$. The estimation*

$$c_S \geq \sqrt{s'^2(0) + 1}$$

holds for the competitive factor.

[1]This criterion is not listed in [7], but it is assumed. Since $s'(\sigma)$ is not known to exist, it is necessary to require this.

Thus competitive strategies are piecewise continuously differentiable in $[0, \pi)$. Using the formula for arc length in polar coordinates, we have

$$A_S(\varphi) = \int_0^{\varphi} \sqrt{s^2(t) + s'^2(t)}\, dt \qquad (2)$$

By the fundamental theorem of calculus, $A'_S(\varphi) = \sqrt{s^2(\varphi) + s'^2(\varphi)}$ on $[0, \pi)$, and so A_S is continuous therein. Since s is rectifiable, $A_S(\pi) < \infty$ so that A_S is bounded on $[0, \pi]$. By L'Hôpital's rule,

$$\begin{aligned} \lim_{\varphi \to 0} f_S(\varphi) &= \lim_{\varphi \to 0} \frac{\sqrt{s^2(\varphi) + s'^2(\varphi)}}{\cos(\varphi)} \\ &= \sqrt{s'^2(0) + 1} \end{aligned}$$

Defining $f_S(0) := \sqrt{s'^2(0) + 1}$, f_S is continuous on $[0, \pi)$ and bounded on $[0, \pi]$. In particular, f_S is integrable on $[0, \pi]$ (this is given by Lebesgue's criterion for Riemann-integrability, see [1] pp. 171).

2.2 The Objective

We are interested in finding a competitive strategy S which minimizes the average value taken by $f_S(\varphi)$ in the interval $[0, \pi]$. More precisely, we wish to minimize the expected value of the ratio of the distance traveled before the corner is seen over the shortest path to the line of sight:

$$\begin{aligned} E[f_S] &= \frac{1}{\pi} \int_0^{\pi} f_S(\varphi)\, d\varphi \\ &= \frac{1}{\pi} \int_0^{\pi} \int_0^{\varphi} \frac{\sqrt{s^2(t) + s'^2(t)}}{a(\varphi)}\, dt\, d\varphi \end{aligned} \qquad (3)$$

which we define as the *average cost* of f_S. Since f_S is integrable on $[0, \pi]$, the above integral exists and is finite.

Observation 1 *If $S = (\varphi, s(\varphi))$ is an optimal strategy, then s is non-increasing on $[0, \pi]$. Indeed, the robot is always better off staying at the same radial distance from the corner than getting farther from it.*

3 A Discretization

As a first attempt to gain insight into the problem, we look at a discretization of it. To this end, we start by partitioning the interval $[0, \pi]$ into n equal subintervals (for simplicity, we choose n to be even), so that we get partition points $x_0, \ldots, x_n$ with $x_0 = 0$, $x_n = \pi$ and $x_{i+1} - x_i = \frac{\pi}{n}$ for $i = 0, \ldots, n-1$. Putting $\theta = \frac{\pi}{n}$, we will assume that the angle φ in the original problem can only take values in $\{k\,\theta \;:\; 1 \leq k \leq n\}$ with equal probability $\frac{1}{n}$.

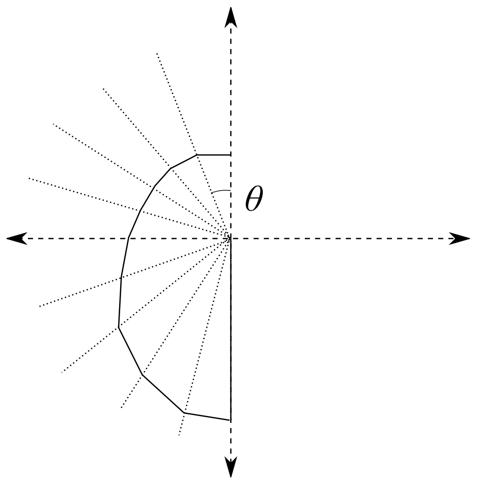


Figure 2: Robot's path in the discretization.

For non-negative values $y_0, \ldots, y_n$, let $P(y_0, \ldots, y_n)$ denote the polygonal path which joins the polar points $(x_0, y_0), \ldots, (x_n, y_n)$ in that order. The problem is then to find values for $y_0, \ldots, y_n$ which minimize the average cost incurred by the strategy $P(y_0, \ldots, y_n)$. The requirement that the robot starts one unit away from the origin is worked in by demanding that $y_0 = 1$. Let $L_i(y_0, \ldots, y_n)$ denote the i^{th} segment of the path $P(y_0, \ldots, y_n)$. The law of cosines gives us

$$\begin{aligned} |L_i(y_0, \ldots, y_n)|^2 &= y_{i-1}^2 + y_i^2 - 2\cos(\theta) y_{i-1} y_i \\ &= (y_{i-1} - \cos(\theta) y_i)^2 + \sin^2(\theta) y_i^2 \end{aligned}$$

And so $|L_i(y_0, \ldots, y_n)|$ may be expressed as the norm of a vector

$$\|(y_{i-1} - \cos(\theta) y_i, \sin(\theta) y_i)\| \qquad (4)$$

Notice that the expression above is *convex* in the arguments $y_0, \ldots, y_n$. For $1 \le k \le n$, the robot travels a distance of $\sum_{i=1}^{k} |L_i(y_0, \ldots, y_n)|$ before it reaches the ray $k\theta$. Remembering (1), we see that the competitive function, f_S takes on values according to

$$f_S(k\theta) = \sum_{i=1}^{k} \frac{|L_i(y_0, \ldots, y_n)|}{\sin(k\theta)}$$

if $1 \le k \le \frac{n}{2}$ and

$$f_S(k\theta) = \sum_{i=1}^{k} |L_i(y_0, \ldots, y_n)|$$

if $\frac{n}{2} + 1 \le k \le n$. The average cost of the strategy $P(y_0, \ldots, y_n)$ is thus given by

$$\frac{1}{n} \sum_{k=1}^{\frac{n}{2}} \sum_{i=1}^{k} \frac{|L_i(y_0, \ldots, y_n)|}{\sin(k\theta)} + \frac{1}{n} \sum_{k=\frac{n}{2}+1}^{n} \sum_{i=1}^{k} |L_i(y_0, \ldots, y_n)|$$

Now, the convexity of (4) gives us the convexity of the above expression and so the problem at hand is actually a convex program. Using a numerical solver [2], we are able to find solutions for $n = 100$, plotted below:

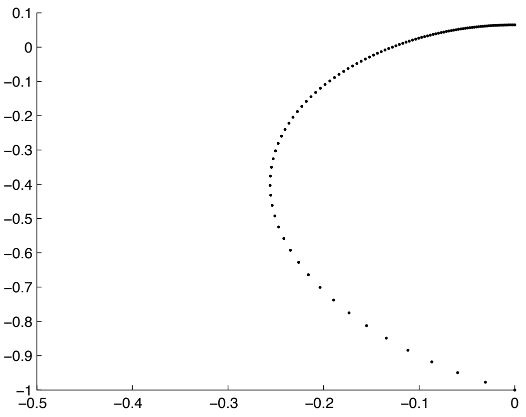


Figure 3: Optimal discrete strategy for $n = 100$.

This plot suggests that an optimal continuous strategy exists. The solutions for $n = 1000$ and $n = 5000$ further support this hypothesis. Note that the solution does not reach the corner as is the case with the strategy found in [7]. Instead, it continually approaches the corner until it reaches the ray π, at a distance of ~ 0.067 from the corner.

Observation 2 *This discretization method suggests that if $H = (\varphi, h(\varphi))$ is an optimal continuous strategy, then h is differentiable at π and $h'(\pi) = 0$. To see this, consider y_n in relation to y_{n-1}: once the robot has reached the ray $(n-1)\,\theta$, it is best off heading to the ray $n\,\theta = \pi$ in the shortest possible path, which is the line segment perpendicular to $n\,\theta$. Thus $y_n = \cos(\theta)\, y_{n-1}$ and we expect the difference quotient*

$$\frac{y_n - y_{n-1}}{\theta} = \frac{(\cos(\theta) - 1)}{\theta}\, y_{n-1}$$

to converge to $h'(\pi)$ as θ approaches 0, if it converges at all. Since

$$\lim_{\theta \to 0} \frac{(\cos(\theta) - 1)}{\theta} = 0$$

[2]We used CVX, a package for specifying and solving convex programs [6].

and $0 \le y_{n-1} \le 1$, we would have $h'(\pi) = 0$. It is actually possible to prove this rigorously with the techniques of the calculus of variations. An argument along similar lines, by considering y_n and y_{n-1} in relation to y_{n-2}, gives $h''(\pi) = \infty$.

4 A Continuous Solution

Our discretization has given us good evidence that an optimal continuous strategy exists. We now explore this possibility with more appropriate tools coming from the calculus of variations. First however We start by considering yet another discretization which will reinforce our evidence. We also derive an expression for the average cost E in the form of a single integral.

Next, we apply the techniques of the calculus of variations to the problem of minimizing E. Namely, we solve the Euler-Lagrange equation associated with E. In general, proving that a given solution to the Euler-Lagrange equation is a minimizer for a variational problem is difficult and attempting to do so would lead us too far deep into the theory of the calculus of variations. Instead, we will be content with the near perfect match we observe between our candidate solution and the discrete optimal average-case solutions.

4.1 The Average Cost as a Single Integral

In view of (3), the expression for the average cost is

$$\begin{aligned} E[f_S] &= \frac{1}{\pi}\int_0^\pi f_S(\varphi)\,d\varphi \\ &= \frac{1}{\pi}\int_0^\pi \int_0^\varphi \frac{\sqrt{s(t)^2 + s'^2(t)}}{a(\varphi)}\,dt\,d\varphi \end{aligned} \tag{5}$$

Before continuing, we define a function b:

$$b(\varphi) = \begin{cases} \ln\left(\frac{1-\cos(\varphi)}{\sin(\varphi)}\right) & : \quad \varphi < \frac{\pi}{2} \\ \varphi - \frac{\pi}{2} & : \quad \varphi \ge \frac{\pi}{2} \end{cases} \tag{6}$$

Note that $b(\varphi)$ is continuous and that $b'(\varphi) = \frac{1}{a(\varphi)}$ on $(0, \pi]$, i.e. b is an antiderivative for the reciprocal of a. Using Fubini's theorem (see [3] pp. 64) we may change the order of integration in (5):

$$\frac{1}{\pi}\int_0^\pi \int_t^\pi \frac{\sqrt{s(t)^2 + s'^2(t)}}{a(\varphi)}\,d\varphi\,dt$$

Finally, invoking the fundamental theorem of calculus, this gives

$$E[f_S] = \int_0^\pi \frac{\sqrt{s(t)^2 + s'^2(t)}\left(\frac{\pi}{2} - b(t)\right)}{\pi}\,dt \tag{7}$$

4.1.1 Another Discretization

The above derivation suggests we try to approximate the average cost E by a Riemann sum. Suppose $S = (\varphi, s(\varphi))$ is an optimal strategy. Choosing an $n > 0$ sufficiently large, we let $\theta = \frac{\pi}{n}$ and $\varphi_i = i\ \theta$. We approximate $s'(\varphi_i)$ by the Newton quotient

$$\frac{s(\varphi_{i+1}) - s(\varphi_i)}{\theta}$$

Putting $y_i = s(\varphi_i)$, we make an approximation for $E[f_S]$:

$$\frac{1}{\pi}\sum_{i=0}^{n-1} \sqrt{y_i^2 + \left(\frac{y_{i+1} - y_i}{\theta}\right)^2}\left(\frac{\pi}{2} - b(\varphi_i)\right)\theta \tag{8}$$

Naturally, we look for values $y_0, \ldots, y_n$ which will minimize (8). Fortunately, this problem is once again convex and in fact computationally easier than the one before. Using a numerical solver, we are able to increase n to 100000, the result is shown below together with our previous discretization:

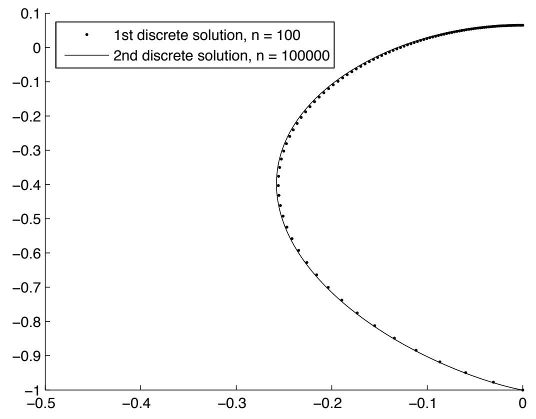


Figure 4: Optimal solutions for two types of discretizations

Notice the two solutions almost agree everywhere they are both defined. The value of (8) in this case is ~ 1.1892.

4.2 An Optimal Strategy

If we regard E as a functional acting on functions defined in $[0, \pi]$ and piecewise continuously differentiable in $[0, \pi)$, our goal is to find such a function h, subject to $h(0) = 1$ and $h(x) \ge 0$ for $x \in [a, b]$, that minimizes E. In general, proving the existence of a continuous minimizer for variational problems such as this one is not an easy task. Instead, we will take the corroborating results from the first and second discretization as proof one exists, and that its average cost should be close to 1.1892. The discretizations suggest we look for such a minimizer with $h(\pi) > 0$. Before proceeding, we verify that we may apply the techniques of the calculus of variations:

Lemma 2 *Suppose h minimizes (7) and $h(\pi) > 0$. Then h is C^2 on $(0, \pi)$.*

Proof. As we have observed, h is non-increasing on $[0, \pi]$, thus $h(t) > 0$ for all $t \in [0, \pi]$. If we define

$$F(x, y, z) = \frac{1}{\pi}\sqrt{y^2 + z^2}\left(\frac{\pi}{2} - b(x)\right)$$

then

$$E(h) = \int_0^\pi F(t, h(t), h'(t))\, dt$$

Now we have

$$\frac{\partial^2 F}{\partial z^2}(t, h(t), h'(t)) = \frac{1}{2\pi}\frac{(\pi - 2b(t))^2}{(h(t)^2 + h'(t)^2)^{\frac{3}{2}}}$$

which remains non-zero in $(0, \pi)$. By the criterion for the regularity of minimizers, h is C^2 in $(0, \pi)$. □

As shown in the appendix, the above lemma ensures h satisfies the Euler-Lagrange equation associated with (7) in $(0, \pi)$. Although too long to produce here, this equation may be expressed in the explicit form

$$h'' = G(t, h, h')$$

so long as $h(t) > 0$ and $0 < t < \pi$. Thus if we know the values for, say, $h(\frac{\pi}{2})$ and $h'(\frac{\pi}{2})$, then we can determine h by the Existence-Uniqueness Theorem for ordinary differential equations. With this mind, we use the discretization results for $n = 100000$ to estimate values for $h(\frac{\pi}{2})$ and $h'(\frac{\pi}{2})$. The resulting solution to the Euler-Lagrange equation associated with (7) is plotted below alongside the optimal discrete solution for $n = 100$.

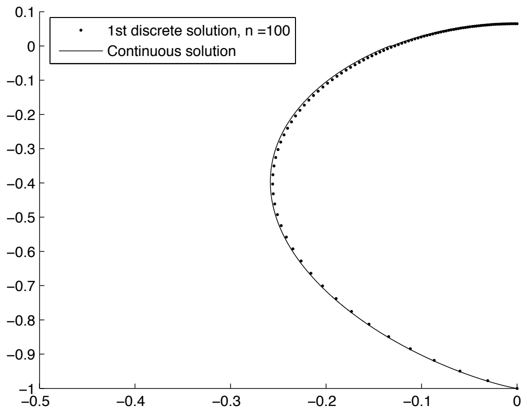


Figure 5: Optimal discrete solution together with optimal continuous solution

The very close match between the continuous solution and the discrete one gives us strong evidence that we have indeed found an optimal strategy. The average cost of this strategy is ~ 1.189. Its competitive factor is ~ 1.3136.

5 Conclusion

Our results give us strong evidence that we have found an optimal average case strategy. It achieves an average cost of ~ 1.189. Although we do not have a closed-form expression for this strategy, a good approximation is given by

$$\frac{1 + 7\,\theta}{35 + 21\,\theta + 22\,\theta^2}$$

which has an average cost of ~ 1.1914, which exceeds the presumed optimal average cost by less than 0.202%.

References

[1] T. M. Apostol. *Mathematical Analysis*. Addison Wesley, 1974.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[3] J. C. Burkill. *The Lesbesgue Integral*. Cambridge University Press, 1951.

[4] J. C. Butcher. *Numerical Methods for Ordinary Differential Equations*. Wiley, 2008.

[5] M. Grant and S. Boyd. Graph implementations for nonsmooth convex programs. In V. Blondel, S. Boyd, and H. Kimura, editors, *Recent Advances in Learning and Control*, Lecture Notes in Control and Information Sciences, pages 95–110. Springer-Verlag Limited, 2008. `http://stanford.edu/~boyd/graph_dcp.html`.

[6] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 1.21. `http://cvxr.com/cvx`, Apr. 2011.

[7] C. Icking, R. Klein, and L. Ma. How to look around a corner. In *Proceedings of the 5th Canadian Conference on Computational Geometry*, pages 443–448, Waterloo, Ontario, 1993.

[8] J. Jost and X. Li-Jost. *Calculus of Variations*. Cambridge University Press, 1998.

[9] M. Mesterton-Gibbons. *A Primer on the Calculus of Variations and Optimal Control Theory*. American Mathematical Society, 2009.

6 Appendix: The Calculus of Variations

Our analysis requires us to minimize expressions of the form

$$J[r] = \int_a^b F\left(x, r(x), r'(x)\right)\, dx \tag{9}$$

where $F : (a, b) \times \mathbb{R}^2 \to [0, \infty)$ is a given function of class C^2 in $(a, b) \times \Omega$ where Ω is an open region in $\mathbb{R}^2$. J is the *objective function*, otherwise known as a *functional.* The goal is to find a minimizing function $r : [a, b] \to \mathbb{R}$ over the class of *admissible* functions. For us, these are the functions that are piecewise continuously differentiable in (a, b) with $(r(x), r'(x)) \in \Omega$ for all x where $r'(x)$ is defined, and moreover satisfy some boundary condition $r(a) = \alpha$ for some $\alpha \in \mathbb{R}$.

Suppose for now that r is such a minimizer. Assume furthermore that r is in fact C^2 in (a, b). It can be shown that

$$\frac{\partial F}{\partial r}(x, r(x), r'(x)) - \frac{d}{dx}\left(\frac{\partial F}{\partial r'}(x, r(x), r'(x))\right) = 0 \quad (10)$$

holds everywhere in (a, b) (see [8] for example). Equation (10) is known as the *Euler-Lagrange equation.* It is a second order differential equation in r and must be satisfied by any r which minimizes J subject to the conditions we have imposed. It is important to note that simply satisfying (10) is not enough to guarantee that r is a minimizer: the condition is necessary, but not sufficient.

In the above, we have assumed minimizers for (9) are of class C^2 in (a, b). The following result gives us conditions on F under which this is justified.

Proposition 3 *Regularity of Minimizers Let r be a piecewise continuously differentiable minimizer for the above problem, and let F be as in* (9). *If $\frac{\partial^2 F}{\partial r'^2}(x, r(x), r'(x))$ does not vanish anywhere in (a, b), then r is C^2 in (a, b).*

A proof of this statement goes along the same lines as Theorem 1.2.3 and 1.2.3 in [8]. Note that because r is piecewise continuously differentiable, the above implies it is C^1 in $[a, b]$.

Computational Geometry in Air Traffic Management

Joseph S. B. Mitchell[*]

The next generation of air transportation system will have to use technology to be able to cope with the ever increasing demand for flights. Several challenging optimization problems arise in trying to maximize efficiency while maintaining safe operation in air traffic management (ATM). Constraints and issues unique to air transportation arise in the ATM domain, including weather hazards, turbulence, no-fly zones, and three-dimensional routing. The challenge is substantially compounded when the constraints vary in time and are not known with certainty, as is the case with weather hazards. Human oversight is provided by air traffic controllers, who are responsible for safe operation within a portion of airspace known as a sector.

In this talk we discuss algorithmic methods that can be used in modeling and solving air traffic management problems, including routing of traffic flows, airspace configuration into load-balanced sectors, and capacity estimation in the face of dynamic and uncertain constraints and demands. We highlight several open problems of interest to computational geometers.

This research has been supported by grants from the National Science Foundation (CCF-0729019, CCF-1018388), NASA Ames, and Metron Aviation. The talk is based on collaborative work with many, including Anthony D. Andre, Dominick Andrisani, Estie Arkin, Amitabh Basu, Jit-Tat Chen, Nathan Downs, Moein Ganji, Robert Hoffman, Rafal Kicinger, Joondong Kim, Victor Klimenko, Irina Kostitsyna, Shubh Krishna, Jimmy Krozel, Changkil Lee, Tenny Lindholm, Tim Myers, Anne Pääkkö, Steve Penny, Valentin Polishchuk, Joseph Prete, Girishkumar Sabhnani, Robert Sharman, Philip J. Smith, Amy L. Spencer, Ali Tafazzoli, Gregory Wong, Shang Yang, Arash Yousefi, Jingyu Zou.

[*]Stony Brook University, `joseph.mitchell@stonybrook.edu`

Competitive Routing on a Bounded-Degree Plane Spanner

Prosenjit Bose* Rolf Fagerberg§ André van Renssen* Sander Verdonschot*

Abstract

We show that it is possible to route locally and competitively on two bounded-degree plane 6-spanners, one with maximum degree 12 and the other with maximum degree 9. Both spanners are subgraphs of the empty equilateral triangle Delaunay triangulation. First, in a weak routing model where the only information stored at each vertex is its neighbourhood, we show how to find a path between any two vertices of a 6-spanner of maximum degree 12, such that the path has length at most $95/\sqrt{3}$ times the straight-line distance between the vertices. In a slightly stronger model, where in addition to the neighbourhood of each vertex, we store $O(1)$ additional information, we show how to find a path that has length at most $15/\sqrt{3}$ times the Euclidean distance both in a 6-spanner of maximum degree 12 and a 6-spanner of maximum degree 9.

1 Introduction

A t-spanner of a weighted graph G is a connected subgraph H with the property that for all pairs of vertices, the weight of the shortest path between the vertices in H is at most t times the weight of the shortest path in G, for some fixed constant $t \geq 1$. The constant t is referred to as the *spanning ratio*. The graph G is referred to as the *underlying graph*. In our setting, the underlying graph is the complete graph on a set of n points in the plane and the weight of an edge is the Euclidean distance between its endpoints (see [9] for a comprehensive overview of spanners).

In communication networks, in addition to being a constant spanner, a desirable property is the ability to route messages on the network such that the total distance travelled by the message is at most a constant times the spanning ratio. Inability to route effectively defeats the purpose of building a spanner in the first place. Network routing strategies such as Dijkstra's algorithm [7] require knowledge of the whole network topology to compute a short route. In many settings, this assumption is impractical. As such, we focus on *local* routing strategies (see [8] for a discussion of various models wrt. local routing). In a local routing strategy, the decision to forward a message depends on information stored at the node where the message currently resides, location of the source, location of the destination and the contents of a small memory. Typically, the information stored at a node is the set of direct neighbours.

Formally, an algorithm A is a k-memory routing algorithm, if the vertex to which a message is forwarded from the current vertex s is a function of s, t, $N(s)$, and M, where t is the destination vertex, $N(s)$ is the set of direct neighbours of s and M is a memory of size k, stored with the message. For our purposes, we consider a unit of memory to consist of a $\log_2 n$ bit integer or a point in $\mathbb{R}^2$. Our model also assumes that the only information stored at each vertex of the graph is $N(s)$. The algorithm A is *d-competitive* provided that the total distance travelled by the message is not more than d times the Euclidean distance between source and destination. We refer to the constant d as the *routing ratio*.

We present the first competitive k-memory routing algorithm to route on a bounded-degree plane spanner. Our algorithm routes on a 6-spanner with maximum degree 12, which is a subgraph of the empty equilateral triangle Delaunay triangulation [4]. We then present another competitive k-memory routing algorithm that routes on a subgraph with maximum degree 9. However, for this algorithm, we need to slightly enhance our model by storing a constant number of bits and points at each vertex (in addition to the neighbourhood of the vertex) to help guide the routing process.

2 Graphs

The empty equilateral triangle Delaunay triangulation was one of the first plane graphs that was shown to be a spanner [6]. It is internally triangulated and has a spanning ratio of 2. Recently, Bonichon *et al.* showed that it is equivalent to the half-θ_6-graph [2] and that it contains a bounded-degree spanner as a subgraph [3]. In this section, we describe the construction of the half-θ_6-graph and two of its bounded-degree subgraphs.

Given a set P of points in the plane, we consider each point $v \in P$ and partition the plane into 6 cones with apex v, each defined by two rays at consecutive multiples of $\pi/3$ radians from the positive x-axis. We

*School of Computer Science, Carleton University. Research supported in part by NSERC. Email: jit@scs.carleton.ca, andre@cg.scs.carleton.ca, sander@cg.scs.carleton.ca.

§Department of Mathematics and Computer Science, University of Southern Denmark. Email: rolf@imada.sdu.dk. Partially supported by the Danish Council for Independent Research, Natural Sciences.

label the cones $\overline{C}_1, C_0, \overline{C}_2, C_1, \overline{C}_0$ and C_2, in counter-clockwise order around v, starting from the positive x-axis (see Figure 1a). The cones C_0, C_1 and C_2 are called *positive*, while the others are called *negative*.

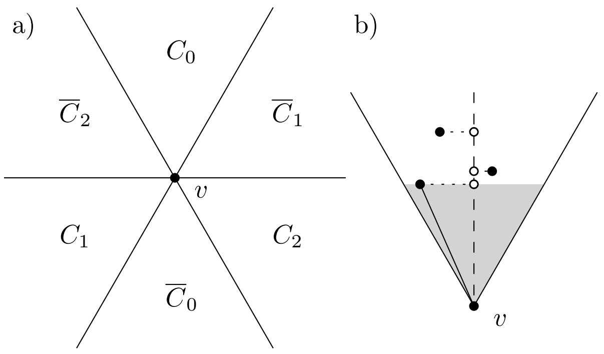


Figure 1: a) The cones around a vertex v. b) The construction of the half-θ_6-graph. In each positive cone, v connects to the vertex with the closest projection on the bisector of that cone.

To build the half-θ_6-graph, we consider each vertex v and add an edge between v and the 'closest' vertex in each of its positive cones. However, instead of using the Euclidean distance, we measure distance by projecting each vertex onto the bisector of the cone. We call the vertex in this cone whose projection is closest to v the *closest vertex* and connect it to v with an edge (see Figure 1b). For simplicity, we assume that no two points lie on a line parallel to a cone boundary.

Given two points a and b such that b lies in a positive cone of a, we define their *canonical triangle* T_{ab} to be the triangle bounded by the cone of a that contains b and the line through b perpendicular to the bisector of that cone. For example, the shaded region in Figure 1b is the canonical triangle of v and its closest vertex. The construction of the half-θ_6-graph can alternatively be described as adding an edge between two vertices if and only if their canonical triangle is empty. This property will play an important role in our proofs.

Each vertex in the half-θ_6-graph has at most one incident edge in each positive cone, but it can have an unbounded number of incident edges in its negative cones. We describe two transformations that allow us to bound the total degree of each vertex. The transformations are adapted from Bonichon *et al.* [3].

The first transformation discards all edges in each negative cone, except for three: the first and last edges in clockwise order around the vertex and the edge to the closest vertex (see Figure 2a). This results in a subgraph with maximum degree 12, which we call G_{12}.

To reduce the degree even further, we note that since the half-θ_6-graph is internally triangulated, consecutive neighbours of v within a negative cone are connected by edges. We call the path formed by these edges the

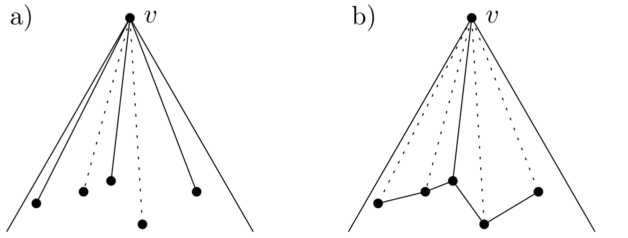


Figure 2: The construction for G_{12} (a) and G_9 (b). Solid edges are kept, while dotted edges are discarded if no other vertex wants to keep them.

canonical path. So instead of keeping three edges per negative cone, we keep only the edge to the closest vertex, but force the edges of the canonical path to be kept as well (see Figure 2b). We call the resulting graph G_9. Since the half-θ_6-graph is planar, both subgraphs are planar as well. In a previous paper [5], we showed that G_9 is a 6-spanner with maximum degree 9. We give an adapted version of the proof for the spanning ratio below.

Theorem 1 *G_9 is a 3-spanner of the half-θ_6-graph.*

Proof. We show that for every edge (s, v) in the half-θ_6-graph, there is a path of length at most $3 \cdot |sv|$ in G_9. This path consists of the edge to the closest vertex, followed by the edges on the canonical path between the closest vertex and v. We will refer to it as the *approximation path*.

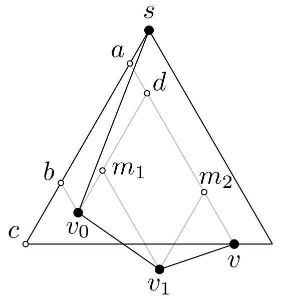


Figure 3: The approximation path.

Let v_0 be the closest vertex and let $v_1, \ldots, v_k = v$ be the other vertices on the approximation path. We assume without loss of generality that s lies in C_0 of v and that v lies to the right of v_0. We shoot rays parallel to the boundaries of C_0 from each vertex. Let m_i be the intersection of the right ray of v_{i-1} and the left ray of v_i (see Figure 3). Let a and b be the intersections of the left boundary of $\overline{C}_0$ of s with the left rays of v and v_0, respectively, and let c be the intersection of this left boundary with the horizontal line through v. Finally, let d be the intersection of the right ray of v_0 and the left ray of v. We can bound the length of the approximation path as follows:

$$\begin{aligned} & |sv_0| + \sum_{i=1}^{k} |v_{i-1}v_i| \\ \leq\; & |sb| + |bv_0| + \sum_{i=1}^{k} |v_{i-1}m_i| + \sum_{i=1}^{k} |m_i v_i| \\ =\; & |sb| + |bv_0| + |ab| + |dv| \quad \{\text{by projection}\} \\ =\; & |sb| + |ab| + |av| \\ \leq\; & |sc| + 2 \cdot |cv| \end{aligned}$$

The last inequality follows from the fact that v_0 is the closest vertex to s. Let α be $\angle csv$. Some basic trigonometry gives us that $|sc| = \frac{2}{\sqrt{3}} \cdot \sin\left(\alpha + \frac{\pi}{3}\right) \cdot |sv|$ and $|cv| = \frac{2}{\sqrt{3}} \cdot \sin(\alpha) \cdot |sv|$. Thus the approximation path is at most $\frac{2}{\sqrt{3}} \cdot \left(\sin\left(\alpha + \frac{\pi}{3}\right) + 2\sin(\alpha)\right)$ times as long as (s, v). Since this function is increasing in $[0, \frac{\pi}{3}]$, the maximum is achieved for $\alpha = \pi/3$, where it is 3. Therefore every edge of the half-θ_6-graph can be approximated by a path that is at most 3 times as long and the theorem follows. □

Note that the part of the approximation path that lies on the canonical path has length at most $2 \cdot |cv| = \frac{4}{\sqrt{3}} \cdot \sin(\alpha) \cdot |sv|$. This function is also increasing in $[0, \frac{\pi}{3}]$ and its maximal value is 2, so the total length of this part is at most $2 \cdot |sv|$.

Since the half-θ_6-graph is a 2-spanner, this shows that G_9 is a 6-spanner. Bonichon *et al.* [3] also showed that all edges on the canonical path are either first or last in their respective negative cones, making G_9 a subgraph of G_{12}. Hence G_{12} is a 6-spanner as well.

3 Routing on G_{12}

We now turn our attention to finding a competitive path from a current vertex s to a given destination t in G_{12}. In a previous paper, we showed that it is possible to route competitively on the half-θ_6-graph [4].

Theorem 2 ([4], Corollary 4.1) *Let u and w be two vertices with w in a positive cone of u. There exists a 0-memory routing algorithm on the half-θ_6-graph with routing ratio*

i) 2 *when routing from u to w,*

ii) $5/\sqrt{3} = 2.886\ldots$ *when routing from w to u.*

and this is best possible for deterministic local routing schemes.

Next we present a slightly modified version of the routing algorithm for the half-θ_6-graph. The difference lies in the fact that the original algorithm does not keep state. However, the same proofs can be used to show

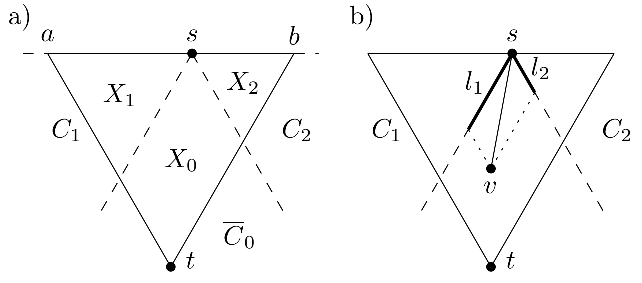


Figure 4: a) The points and regions involved in negative routing. b) The projected length of an edge.

that the stateful version in this paper finds a path with the same routing ratio.

Before we can describe the actual algorithm, we need a few definitions. We assume without loss of generality that t lies in C_0 or $\overline{C}_0$ of s. If t lies in $\overline{C}_0$, the cones around s split T_{ts} into three regions, which we call X_0, X_1 and X_2, as shown in Figure 4a. Formally, let $X_0 = \overline{C}_0 \cap T_{ts}$, $X_1 = C_1 \cap T_{ts}$ and $X_2 = C_2 \cap T_{ts}$. Further, we let a be the corner of T_{ts} that is on the boundary of C_1 and b the corner on the boundary of C_2. For brevity, we use "an edge in X_0" to denote an edge incident to s with the other endpoint in X_0.

We also need the concept of the *projected length* of an edge onto a neighbouring cone. For an edge (s, v) in $\overline{C}_0$, the neighbouring cones of s are C_1 and C_2. Let $\vec{e_1}$ and $\vec{e_2}$ be unit vectors parallel to the boundary of $\overline{C}_0$ with C_1 and C_2, respectively. Since $\vec{e_1}$ and $\vec{e_2}$ are linearly independent, the vector $\vec{sv}$ can be uniquely written as $l_1 \cdot \vec{e_1} + l_2 \cdot \vec{e_2}$. We define the projected length of (s, v) on C_1 as l_1 and on C_2 as l_2 (see Figure 4b).

Our algorithm distinguishes three cases and keeps track of a *preferred side*, which is one of the positive cones, or undefined. The preferred side is stored as state in the message. If t lies in a positive cone of s, we are in case A. If t lies in a negative cone of s and no preferred side has been set yet, we are in case B. If t lies in a negative cone of s and a preferred side has been set, we are in case C. The algorithm works as follows on the half-θ_6-graph.

- In case A, follow the unique edge in the positive cone containing t.
- In case B, if there are edges in X_0, follow an arbitrary one. Otherwise, if there is an edge in the smaller of X_1 and X_2, follow that edge. Otherwise, follow the edge in the larger of X_1 and X_2 and set the other as the preferred side. At least one of these edges must exist [4].
- In case C, if there are edges in X_0, follow the one with the largest projected distance on the preferred side. Otherwise, follow the edge in the positive cone that is not on the preferred side. Again, at least one of these edges must exist [4].

This algorithm constructs a path between two vertices in the half-θ_6-graph. To approximate this path in G_{12} and G_9, we simulate each step of the algorithm. Note that we can decide which case we are in based solely on the coordinates of s and t and whether the preferred side has been set. The following five headlines refer to original steps of the algorithm on the half-θ_6-graph, and the text after a headline describes how to simulate that step in G_{12}. We discuss modifications for G_9 in Section 4.

Follow an edge (s, v) in a positive cone. If the edge is still there, we simply follow it. If it is not, the edge was removed because s is on the canonical path of v and it is not the closest, first or last vertex on the path. Since G_{12} is a supergraph of G_9, we know that all of the edges of the canonical path are kept and every vertex on the path originally had an edge to v in the same positive cone. Therefore it suffices to search the canonical path for any vertex with an edge in this positive cone and follow this edge. Since the edges connecting v to the first and last vertices on the path are always kept, the edge we find in this way must lead to v.

This method is guaranteed to reach v, but we want to find a *competitive* path to v. Therefore we will use exponential search along the canonical path: we start by following the shorter of the two edges of the canonical path incident to s. If the endpoint of this edge does not have an edge in our positive cone, we return to s and travel twice the length of the first edge in the other direction. We keep returning to s and doubling the maximum travel distance until we find a vertex x that does have an edge in our positive cone. If x is not the closest to v, by the triangle inequality, following its edge to v is shorter than continuing our search until we reach the closest and following its edge. So for the purpose of bounding the distance travelled, we can assume that x is closest to v. Let d be the distance between s and x along the canonical path. By using exponential search to find x, we travel at most 9 times this distance [1] and afterwards we follow (s, x). From the spanning proof, we know that $d \leq 2 \cdot |sv|$ and $d + |xv| \leq 3 \cdot |sv|$. Thus the total length of our path is at most $9 \cdot d + |xv| = 8 \cdot d + (d + |xv|) \leq 16 \cdot |sv| + 3 \cdot |sv| = 19 \cdot |sv|$.

Determine if there are edges in X_0. In the regular half-θ_6-graph we can look at all our neighbours and see if any of them lie in X_0. However, in G_{12}, these edges may have been removed. Fortunately, we can still determine if they existed in the original half-θ_6-graph. To do this, we look at the first and last vertex along the canonical path in this cone. If these vertices do not exist, s did not have any incoming edges in this cone, so there can be no edges in X_0. If first and last are the same vertex, this was the only incoming edge to s from this cone, so we simply check if its endpoint lies in X_0. The interesting case is when first and last exist and are distinct. If either of them lies in X_0, we have our answer, so assume that both lie outside of X_0. Since they cannot have t in their positive cone, they must lie in one of two regions, which we call S_1 and S_2 (see Figure 5a).

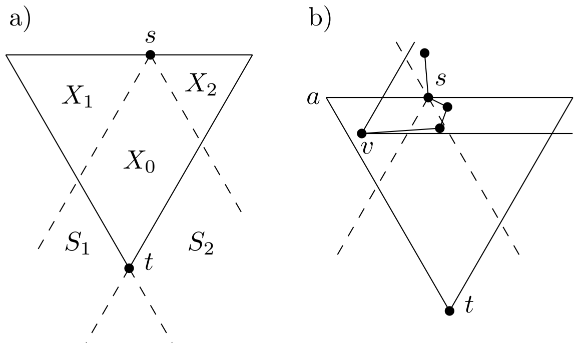


Figure 5: a) Possible regions for the first and last vertex. b) A vertex v in X_1.

If both first and last lie in the same region (say S_1), there can be no edge in X_0, since any vertex on the canonical path between them in X_0 would lie in C_0 of the last vertex. This would prevent the last vertex from having an edge to s, which is a contradiction.

On the other hand, if first lies in S_1 and last in S_2, both X_1 and X_2 have to be empty, since s was the closest vertex to both. Thus if there are no vertices in X_0 (different from t and s), t must have an edge to s, which gives us an edge in X_0. On the other hand, if there are vertices in X_0, the same holds for the topmost vertex in X_0, so in either case there must be an edge in X_0. This shows that we can check whether there was an edge in X_0 in the half-θ_6-graph using only the coordinates of the first and last vertex.

Follow an arbitrary edge in X_0. If the half-θ_6-graph has edges in X_0, we simulate following an arbitrary one of these by first following the edge to the closest vertex in the negative cone. If this vertex is in X_0, we are done. Otherwise, we follow the canonical path in the direction of X_0 and stop once we are inside. This traverses exactly the approximation path of the edge, and hence travels a distance at most 3 times the length of the edge.

Determine if there is an edge in X_1 or X_2. Since these regions are symmetric, we will consider only the case for X_1. It is contained in a positive cone of s, so it contains at most one edge incident to s. If the edge is still there, we can simply test whether it is in X_1 or not. However, if s does not have a neighbour in this cone, we need to find out whether it used to have one in the original half-θ_6-graph and if so, whether it was in X_1. Since this step is only needed in case B after we determine that there are no edges in X_0, we can use this information to guide our search. Specifically, we know that if we find an edge, we should follow it.

Therefore we simply attempt to follow the edge in this cone. If there is a vertex v in X_1 and the edge (s, v) was removed (see Figure 5b), we must encounter the closest vertex after travelling at most $2 \cdot |sv|$ along the canonical path. Since all edges in X_1 have length at most $|as|$, we use exponential search, travelling at most $2 \cdot |as|$ from s. Once we explored both sides of the path to a distance of $2 \cdot |as|$ without encountering a vertex with an edge in the correct positive cone, we return to s and conclude that there was no edge in X_1. If we do find such a vertex, we test whether its edge leads into X_1 and follow it if it does. If the edge does not lead into X_1, either the edge of s in C_1 had its endpoint outside of X_1, or s did not have an edge in C_1. Either way, we return to s and conclude that there was no edge in X_1.

If there was an edge in X_1, we travelled the same distance as if we were simply following the edge: at most $19 \cdot |sv|$. If we return to s unsuccessfully, we travelled at most $20 \cdot |as|$: 9 times $2 \cdot |as|$ during the exponential search and $2 \cdot |as|$ to return to s.

Follow the edge in X_0 with the largest projected distance on the preferred side. In the half-θ_6-graph, we have sufficient information about our neighbours to simply compute their projected distances. However, a lot of these edges might have been removed in the construction of G_{12}. To help find the correct edge, we first prove the following property.

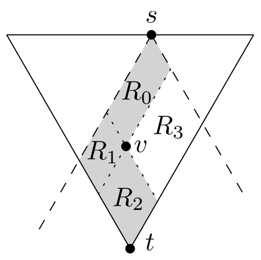


Figure 6: Situation around the first vertex in X_0.

Lemma 3 *In the half-θ_6-graph, the first or last edge in X_0 in counter-clockwise order around s has the largest projected distance on the preferred side.*

Proof. We consider only the case where the preferred side is C_1. The case for C_2 is analogous. Let v be the endpoint of the first edge in X_0 in counter-clockwise order around s. The lines through v parallel to the boundaries of $\overline{C}_0$ partition X_0 into four regions. In counter-clockwise order, starting at the top, we call these R_0, R_1, R_2 and R_3 (see Figure 6). Now let us consider the possible locations of other vertices on the canonical path in this negative cone of s.

Since v has an edge to s, R_0 must be empty. There can also be no neighbours of s in R_1, as these would have come before v in the counter-clockwise ordering around s. Finally, for vertices in R_2, v will always be closer than s, so there can be no neighbours of s in R_2 either. Thus all other vertices of the canonical path must either be outside X_0 or in R_3. Since the projected distance of (s, v) is at least as large as the projected distance to any vertex in R_3, (s, v) has the largest projected distance among all edges in X_0. $\square$

To follow this edge, we first follow the edge to the closest vertex. If this lands us in X_0, we then follow the canonical path towards the preferred side and stop at the last vertex on the canonical path that is in X_0. If the closest is not in X_0, we follow the canonical path towards X_0 and stop at the first or last vertex in X_0, depending on which side of X_0 we started on. This follows the approximation path of the edge, so the distance travelled is at most 3 times the length of the edge.

Routing ratio. This shows that we can simulate the routing algorithm on G_{12}. Note that in contrast to the routing algorithm on the half-θ_6-graph, we maintain state in the message. We need to store not only the preferred side, but also information for the exponential search, including distance travelled. The exact routing ratios are as follows.

Theorem 4 *Let u and w be two vertices with w in a positive cone of u. There exists an $O(1)$-memory routing algorithm on G_{12} with routing ratio*

i) $19 \cdot 2 = 38$ *when routing from u to w,*

ii) $19 \cdot 5/\sqrt{3} = 54.848\ldots$ *when routing from w to u.*

Proof. As shown above, we can simulate every edge followed by the algorithm by travelling at most 19 times the length of the edge. The only additional cost is incurred in case B, when we try to follow an edge in the smaller of X_1 and X_2, but this edge does not exist. In this case, we travel an additional $20 \cdot |as|$, where a is the corner closest to s. Fortunately, this can happen at most once during the execution of the algorithm, as it prompts the transition to case C, after which the algorithm never returns to case B. Looking at the original proof for the routing ratio [4], we observe that in the transition from case B to C, there is $2 \cdot |as|$ of unused potential. Since we are trying to show a routing ratio of 19 times the original, we can charge the additional $20 \cdot |as|$ to the $38 \cdot |as|$ of unused potential. $\square$

4 Routing on G_9

In this section, we explain how to modify the described simulation strategies so that they work for G_9, where the first and last edges are not guaranteed to be present. We discuss only those steps that rely on the presence of these edges.

Follow an edge (s, v) in a positive cone. Because the first and last edges are not always kept, we cannot guarantee that the first vertex we reach with an edge in this positive cone is still part of the same canonical path. Therefore our original exponential search solution does not work. Instead, we need to store one bit of information at s, namely in which direction we have to follow the canonical path to reach the closest vertex to v. Knowing this, we just follow the canonical path in this direction until we reach a vertex with an edge in this positive cone. This vertex must be the closest, so it gives us precisely the approximation path and therefore we travel at most $3 \cdot |sv|$.

Determine if there are edges in X_0. In G_{12}, this test was based on the coordinates of the endpoints of the first and last edge. Since these might be missing in G_9, we store the coordinates of these vertices at s. This allows us to perform the check without increasing the distance travelled.

Determine if there is an edge in X_1 or X_2. As in the positive routing simulation, we now know where to go to find the closest. Therefore we simply follow the canonical path in this direction from s and stop when we reach a vertex with an edge in the correct positive cone, or when we have travelled $2 \cdot |as|$. If there is an edge, we follow exactly the approximation path, giving us 3 times the length of the edge. If there is no edge, we travel $2 \cdot |as|$ back and forth, for a total of $4 \cdot |as|$.

Routing Ratio. Since the other simulation strategies do not rely on the presence of the first or last edges, we can now analyze the routing ratio obtained on G_9.

Theorem 5 *Let u and w be two vertices with w in a positive cone of u. By storing $O(1)$ additional information at each vertex, there exists an $O(1)$-memory routing algorithm on G_{12} and G_9 with routing ratio*

i) $3 \cdot 2 = 6$ *when routing from u to w,*

ii) $3 \cdot 5/\sqrt{3} = 8.660\ldots$ *when routing from w to u.*

Proof. The simulation strategy for G_{12} followed the approximation path for each edge, except when following an edge in a positive cone. Since our new strategy follows the approximation path there as well, our new routing ratio is only 3 times the one for the half-θ_6-graph. Note that this is still sufficient to charge the additional $4 \cdot |sa|$ travelled to the transition from case B to C. Since G_9 is a subgraph of G_{12}, this strategy works on G_{12} as well. □

5 Conclusion

We presented two competitive $O(1)$-memory routing algorithms for bounded-degree subgraphs of the half-θ_6-graph. To the best of our knowledge, these are the first competitive routing algorithms on bounded-degree plane spanners. The first strategy works on G_{12} and achieves a routing ratio of 19 times the routing ratio on the half-θ_6-graph. The second algorithm works on G_9 as well as G_{12} and reduces the routing ratio to 3 times the original. However, it achieves this only by storing information at vertices. Note that this routing ratio is optimal for the positive case, as the spanning ratio of 6 is tight for both graphs.

An interesting open problem is whether the routing ratio for the negative case can be improved, or if, as for the half-θ_6-graph, we can find a matching lower bound. And is it possible to improve the routing ratio on G_{12} without storing information at the vertices? Or does there exist a 0-memory routing algorithm for these graphs? It would also be interesting to see if there are other bounded-degree plane spanners that are easier to route on or allow better routing ratios. For example, a slight modification transforms G_9 into a plane 6-spanner with maximum degree 6 [3]. Is it possible to route competitively on that graph as well?

References

[1] R. A. Baeza-Yates, J. C. Culberson, and G. J. E. Rawlins. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993.

[2] N. Bonichon, C. Gavoille, N. Hanusse, and D. Ilcinkas. Connections between theta-graphs, Delaunay triangulations, and orthogonal surfaces. In *WG*, pages 266–278, 2010.

[3] N. Bonichon, C. Gavoille, N. Hanusse, and L. Perkovic. Plane spanners of maximum degree six. In *ICALP (1)*, pages 19–30, 2010.

[4] P. Bose, R. Fagerberg, A. van Renssen, and S. Verdonschot. Competitive routing in the half-θ_6-graph. In *SODA*, pages 1319–1328, 2012.

[5] P. Bose, R. Fagerberg, A. van Renssen, and S. Verdonschot. On plane constrained bounded-degree spanners. In *LATIN*, pages 85–96, 2012.

[6] P. Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences*, 39(2):205–219, 1989.

[7] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[8] X. Li. *Wireless Ad Hoc and Sensor Networks.* Cambridge University Press, 2008.

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

Optimal Bounds on Theta-Graphs: More is not Always Better

Prosenjit Bose* Jean-Lou De Carufel* Pat Morin* André van Renssen* Sander Verdonschot*

Abstract

We present tight upper and lower bounds on the spanning ratio of a large family of θ-graphs. We show that θ-graphs with $4k+2$ cones ($k \geq 1$ and integer) have a spanning ratio of $1 + 2\sin(\theta/2)$, where θ is $2\pi/(4k+2)$. We also show that θ-graphs with $4k+4$ cones have spanning ratio at least $1 + 2\tan(\theta/2) + 2\tan^2(\theta/2)$, where θ is $2\pi/(4k+4)$. This is somewhat surprising since, for equal values of k, the spanning ratio of θ-graphs with $4k+4$ cones is greater than that of θ-graphs with $4k+2$ cones, showing that increasing the number of cones can make the spanning ratio worse.

1 Introduction

In a weighted graph G, let the distance $\delta_G(u,v)$ between two vertices u and v be the length of the shortest path between u and v in G. A subgraph H of G is a *t-spanner* of G if for all pairs of vertices u and v, $\delta_H(u,v) \leq t \cdot \delta_G(u,v), t \geq 1$. The *spanning ratio* of H is the smallest t for which H is a t-spanner. The graph G is referred to as the *underlying graph*.

We consider the situation where the underlying graph G is a straightline embedding of the complete graph on a set of n points in the plane denoted by K_n, with the weight of an edge (u,v) being the Euclidean distance $|uv|$ between u and v. A spanner of such a graph is called a *geometric spanner*. We look at a specific type of geometric spanner: θ-graphs.

Introduced independently by Clarkson [4] and Keil [5], θ-graphs are constructed as follows (a more precise definition follows in the next section): for each vertex u, we partition the plane into m disjoint cones with apex u, each having aperture $\theta = 2\pi/m$. When m cones are used, we denote the resulting θ-graph as θ_m. The θ-graph is constructed by, for each cone with apex u, connecting u to the vertex v whose projection along the bisector of the cone is closest. Ruppert and Seidel [6] showed that the spanning ratio of these graphs is at most $1/(1 - 2\sin(\theta/2))$, when $\theta < \pi/3$, i.e. there are at least seven cones.

*School of Computer Science, Carleton University. Research supported in part by FQRNT and NSERC. Email: jit@scs.carleton.ca, jdecaruf@cg.scs.carleton.ca, morin@scs.carleton.ca, andre@cg.scs.carleton.ca, sander@cg.scs.carleton.ca.

Recently, Bonichon *et al.* [1] showed that the θ_6-graph has spanning ratio 2. This was done by dividing the cones into two sets, positive and negative cones, such that each positive cone is adjacent to two negative cones and vice versa. It was shown that when edges are added only in the positive cones, in which case the graph is called the half-θ_6-graph, the resulting graph is equivalent to the TD-Delaunay triangulation (the Delaunay triangulation where the empty region is an equilateral triangle) whose spanning ratio is 2 as shown by Chew [3]. An alternative, inductive proof of the spanning ratio of the θ_6-graph was presented by Bose *et al.* [2].

Tight bounds on spanning ratios are notoriously hard to obtain. The standard Delaunay triangulation (where the empty region is a circle) is a good example. It has been studied for over 20 years and the upper and lower bounds still do not match. Also, even though it was introduced about 25 years ago, the spanning ratio of the θ_6-graph has only recently been shown to be finite and tight, making it the first and, until now, only θ-graph for which tight bounds are known.

In this paper, we generalize the results from Bose *et al.* [2]. We look at two families of θ-graphs: the $\theta_{(4k+2)}$-graph and the $\theta_{(4k+4)}$-graph, where k is an integer and at least 1. We show that the $\theta_{(4k+2)}$-graph has a tight spanning ratio of $1 + 2\sin(\theta/2)$ and that the $\theta_{(4k+4)}$-graph has a strictly larger spanning ratio of at least $1 + 2\tan(\theta/2) + 2\tan^2(\theta/2)$, for their respective values of θ.

2 Preliminaries

Let a *cone* C be the region in the plane between two rays originating from the same point (referred to as the apex of the cone). When constructing a θ_m-graph, for each vertex u of K_n consider the rays originating from u with the angle between consecutive rays being $\theta = 2\pi/m$. Each pair of consecutive rays defines a cone. The cones are oriented such that the bisector of some cone coincides with the vertical line through u.

The θ_m-graph is constructed as follows: for each cone C of each vertex u, add an edge from u to the closest vertex in that cone, where distance is measured along the bisector of the cone. More formally, we add an edge between two vertices u and v if $v \in C$ and for all vertices $w \in C$ ($v \neq w$), $|uv'| \leq |uw'|$, where v' and w' denote the orthogonal projection of v and w on the bisector of C.

For ease of exposition, we only consider point sets in general position: no two vertices lie on a line parallel to one of the rays that define the cones and no two vertices lie on a line perpendicular to the bisector of one of the cones. This implies that each vertex adds at most one edge per cone to the graph.

Given a vertex w in cone C of vertex u, we define the *canonical triangle* T_{uw} to be the triangle defined by the borders of C and the line through w perpendicular to the bisector of C. We use m to denote the midpoint of the side of T_{uw} opposite u and α to denote the unsigned angle between uw and um. See Figure 1. Note that for any pair of vertices u and w, there exist two canonical triangles: T_{uw} and T_{wu}.

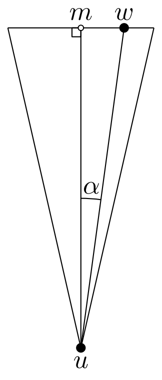


Figure 1: The canonical triangle T_{uw}

3 Spanning Ratio of the $\theta_{(4k+2)}$-Graph

In this section, we give matching upper and lower bounds on the spanning ratio of the $\theta_{(4k+2)}$-graph, for any integer $k \geq 1$. The proof is a generalization of the proof given by Bose *et al.* [2]. We first show that the $\theta_{(4k+2)}$-graph has a very nice geometric property:

Lemma 1 *Any line perpendicular to the bisector of a cone is parallel to the boundary of some cone.*

Proof. The angle between the bisector of a cone and the boundary of that cone is $\theta/2$ and the angle between the bisector and the line perpendicular to this bisector is $\pi/2 = ((2k+1)/2) \cdot \theta$. Thus the angle between the line perpendicular to the bisector and the boundary of the cone is $2\pi - \theta/2 - ((2k+1)/2) \cdot \theta = k \cdot \theta$. Since a cone boundary is placed at every multiple of θ, the line perpendicular to the bisector is parallel to the boundary of some cone. $\square$

This property implicitly helps when bounding the spanning ratio of the $\theta_{(4k+2)}$-graph. However, before deriving this bound, we first prove a useful geometric lemma.

Lemma 2 *Given a convex quadrilateral $abcd$ such that no three of its vertices lie on a line, $\angle abc = \angle adc$, $\angle bad \leq \angle bcd$, and $\angle bad \leq 2 \cdot \angle bac$. It holds that $|ad| + |dc| \leq |ab| + |bc|$.*

Proof. Since $\angle bad \leq 2 \cdot \angle bac$, the bisector of $\angle bad$ intersects bc. Let x be this intersection. Let y be the intersection of ad and the line through x, parallel to cd. Since $\angle bad \leq \angle bcd$, the line through d parallel to bc intersects xy. Let z be this intersection. See Figure 2. These definitions imply that $|zd| = |xc|$ and $|zx| = |dc|$.

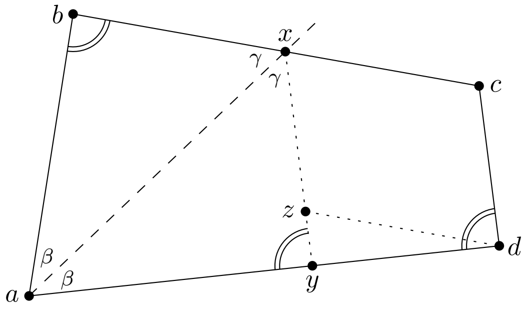


Figure 2: Quadrilateral $abcd$

Since $\angle bax = \angle yax$ and $\angle abx = \angle adc = \angle ayx$, we have that $\angle bxa = \angle yxa$. Since ax is part of both triangle abx and triangle ayx, the law of sines implies that $|ab| = |ay|$ and $|bx| = |yx|$. We now rewrite $|ad| + |dc|$ and $|ab| + |bc|$:

$$\begin{aligned} |ad| + |dc| &= |ay| + |yd| + |dc| \\ &= |ay| + |yd| + |zx| \end{aligned}$$

$$\begin{aligned} |ab| + |bc| &= |ab| + |bx| + |xc| \\ &= |ay| + |yx| + |xc| \\ &= |ay| + |yz| + |zx| + |zd| \end{aligned}$$

Therefore $|ad| + |dc| \leq |ab| + |bc|$ if and only if $|yd| \leq |yz| + |zd|$, which follows from the triangle inequality. $\square$

Theorem 3 *Let u and w be two vertices in the plane. Let m be the midpoint of the side of T_{uw} opposite u and let α be the unsigned angle between uw and um. There exists a path in the $\theta_{(4k+2)}$-graph of length at most*

$$\left(\left(\frac{1 + \sin\left(\frac{\theta}{2}\right)}{\cos\left(\frac{\theta}{2}\right)} \right) \cdot \cos\alpha + \sin\alpha \right) \cdot |uw|.$$

Proof. We prove the theorem by induction on the area of T_{uw} (formally, induction on the rank, when ordered by area, of the canonical triangles for all pairs of vertices). Let a and b be the upper left and right corners of T_{uw}, let p and q be the intersections of T_{uw} and the lower boundaries of the uppermost cones of w that intersect T_{uw}, and let x and y be the left and right intersections of T_{uw} and the boundaries of the cone of w that contains u. See Figure 3.

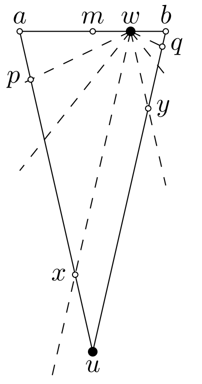


Figure 3: The canonical triangle T_{uw} with a, b, p, q, x, and y being the various intersections of its sides

Our inductive hypothesis is the following, where $\delta(u, w)$ denotes the length of the shortest path from u to w in the $\theta_{(4k+2)}$-graph:

- If axw is empty, then $\delta(u, w) \leq |ub| + |bw|$.
- If byw is empty, then $\delta(u, w) \leq |ua| + |aw|$.
- If neither axw nor byw is empty, then $\delta(u, w) \leq \max\{|ua| + |aw|, |ub| + |bw|\}$.

We first show that this induction hypothesis implies the theorem. Basic trigonometry gives us the following equalities: $|um| = |uw| \cdot \cos\alpha$, $|mw| = |uw| \cdot \sin\alpha$, $|am| = |bm| = |uw| \cdot \cos\alpha \cdot \tan(\theta/2)$, and $|ua| = |ub| = |uw| \cdot \cos\alpha / \cos(\theta/2)$. Thus the induction hypothesis gives that $\delta(u, w)$ is at most $|ua| + |am| + |mw| = |uw| \cdot (((1 + \sin(\theta/2))/\cos(\theta/2)) \cdot \cos\alpha + \sin\alpha)$.

Base case: T_{uw} has rank 1. Since the triangle is a smallest triangle, w is the closest vertex to u in that cone. Hence the edge (u, w) is part of the $\theta_{(4k+2)}$-graph, and $\delta(u, w) = |uw|$. From the triangle inequality, we have $|uw| \leq \min\{|ua| + |aw|, |ub| + |bw|\}$, so the induction hypothesis holds.

Induction step: We assume that the induction hypothesis holds for all pairs of vertices with canonical triangles of rank up to i. Let T_{uw} be a canonical triangle of rank $i + 1$.

If (u, w) is an edge in the $\theta_{(4k+2)}$-graph, the induction hypothesis follows by the same argument as in the base case. If there is no edge between u and w, let v be the vertex closest to u in the cone of u that contains w, and let a' and b' be the upper left and right corners of T_{uv}. See Figure 4. By definition, $\delta(u, w) \leq |uv| + \delta(v, w)$, and by the triangle inequality, $|uv| \leq \min\{|ua'| + |a'v|, |ub'| + |b'v|\}$.

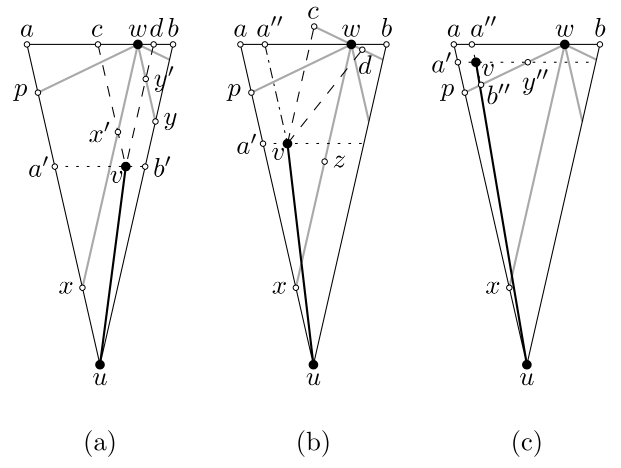


Figure 4: The three cases: (a) v lies in $uxwy$, (b) v lies in xpw, (c) v lies in paw

We perform a case analysis based on the location of v: (a) v lies in $uxwy$, (b) v lies in xpw, (c) v lies in paw, (d) v lies in yqw, and (e) v lies in qbw. Case (d) is analogous to Case (b) and Case (e) is analogous to Case (c), so we only discuss the first three cases.

Case (a): Vertex v lies in $uxwy$. Let c and d be the upper left and right corners of T_{vw}, and let x' and y' be the left and right intersections of T_{vw} and the boundaries of the cone of w that contains v. See Figure 4a. Since T_{vw} has smaller area than T_{uw}, we apply the inductive hypothesis on T_{vw}. Our task is to prove all three statements of the inductive hypothesis for T_{uw}.

1. If axw is empty, then $cx'w$ is also empty, so by induction $\delta(v, w) \leq |vd| + |dw|$. Since v, d, b, and b' form a parallelogram, we have:

$$\begin{aligned} \delta(u, w) &\leq |uv| + \delta(v, w) \\ &\leq |ub'| + |b'v| + |vd| + |dw| \\ &= |ub| + |bw|, \end{aligned}$$

   which proves the first statement of the induction hypothesis.

2. If byw is empty, an analogous argument proves the second statement of the induction hypothesis.

3. If neither axw nor byw is empty, by induction we have $\delta(v, w) \leq \max\{|vc| + |cw|, |vd| + |dw|\}$. Assume, without loss of generality, that the maximum of the right hand side is attained by its second argument $|vd| + |dw|$ (the other case is analogous).

Since vertices v, d, b, and b' form a parallelogram, we have that:

$$\begin{aligned}\delta(u,w) &\leq |uv| + \delta(v,w)\\ &\leq |ub'| + |b'v| + |vd| + |dw|\\ &\leq |ub| + |bw|\\ &\leq \max\{|ua| + |aw|, |ub| + |bw|\},\end{aligned}$$

which proves the third statement of the induction hypothesis.

Case (b): Vertex v lies in xpw. Since v lies in axw, the first statement in the induction hypothesis for T_{uw} is vacuously true. It remains to prove the second and third statement in the induction hypothesis. Let c and d be the upper and lower right corners of T_{vw}, and let a'' be the intersection of aw and the line through v, parallel to ua. See Figure 4b. Since T_{vw} is smaller than T_{uw}, by induction we have $\delta(v,w) \leq \max\{|vc| + |cw|, |vd| + |dw|\}$. We perform a case analysis based on this: (i) $\delta(v,w) \leq |vd| + |dw|$, (ii) $\delta(v,w) \leq |vc| + |cw|$.

Case (i): Since $\angle va''w$ and $\angle vdw$ are both the angle between the boundary of a cone and the line perpendicular to the bisector of that cone, we have $\angle va''w = \angle vdw = k\cdot\theta$. Also, we have that $\angle a''vd \leq \angle a''wd$, since $\angle a''vd \leq k\cdot\theta$ and

$$\begin{aligned}\angle a''wd &= 2\pi - \angle va''w - \angle vdw - \angle a''vd\\ &\geq (4k+2)\cdot\theta - 3k\cdot\theta\\ &= (k+2)\cdot\theta\end{aligned}$$

Furthermore, since $\angle a''vw > \angle a''vc \geq \theta$ and $\angle a''vd = \angle a''vc + \theta \leq 2\cdot\angle a''vc$, we have that $\angle a''vd < 2\cdot\angle a''vw$.

Hence we can apply Lemma 2 to quadrilateral $va''wd$, which gives us that $|vd| + |dw| \leq |va''| + |a''w|$. Since $|uv| \leq |ua'| + |a'v|$ and v, a'', a, and a' form a parallelogram, we have that $\delta(u,w) \leq |ua| + |aw|$, proving the second and third statement in the induction hypothesis for T_{uw}.

Case (ii): Let z be the lower corner of T_{wv}. Since $vcwz$ form a parallelogram, we know that $|vc| + |cw| = |wz| + |zv|$. We now look at quadrilateral $wzva''$. Analogous to Case (i), we have that $\angle wzv = \angle wa''v = k\cdot\theta$, $\angle a''wz \leq \angle a''vz$, and $\angle a''wz < 2\cdot\angle a''wv$. Hence we can apply Lemma 2 to quadrilateral $wzva''$, which gives us that $|wz| + |zv| \leq |va''| + |a''w|$, proving the second and third statement in the induction hypothesis for T_{uw}.

Case (c): Vertex v lies in paw. Since v lies in axw, the first statement in the induction hypothesis for T_{uw} is vacuously true. It remains to prove the second and third statement in the induction hypothesis. Let a'' and b'' be the upper and lower left corners of T_{wv}, and let y'' be the intersection of T_{wv} and the lower boundary of the cone of v that contains w. See Figure 4c. Note that y'' is also the right intersection of T_{uv} and T_{wv}. Since v is the closest vertex to u, T_{uv} is empty. Hence, $b''y''v$ is empty. Since T_{wv} is smaller than T_{uw}, we can apply induction on it. As $b''y''v$ is empty, the first statement of the induction hypothesis for T_{wv} gives $\delta(v,w) \leq |va''| + |a''w|$. Since $|uv| \leq |ua'| + |a'v|$ and v, a'', a, and a' form a parallelogram, we have that $\delta(u,w) \leq |ua| + |aw|$, proving the second and third statement in the induction hypothesis for T_{uw}. □

Since $((1+\sin(\theta/2))/\cos(\theta/2))\cdot\cos\alpha + \sin\alpha$ is increasing for $\alpha \in [0, \theta/2]$, for $\theta \leq \pi/3$, it is maximized when $\alpha = \theta/2$, and we obtain the following corollary:

Corollary 4 *The $\theta_{(4k+2)}$-graph is a $\left(1 + 2\cdot\sin\left(\frac{\theta}{2}\right)\right)$-spanner of K_n.*

The upper bounds given in Theorem 3 and Corollary 4 are tight, as shown in Figure 5: we place a vertex v arbitrarily close to the upper corner of T_{uw} that is furthest from w. Likewise, we place a vertex v' arbitrarily close to the lower corner of T_{wu} that is furthest from u. Both shortest paths between u and w visit either v or v', so the path length is arbitrarily close to $(((1+\sin(\theta/2))/\cos(\theta/2))\cdot\cos\alpha + \sin\alpha)\cdot|uw|$, showing that the upper bounds are tight.

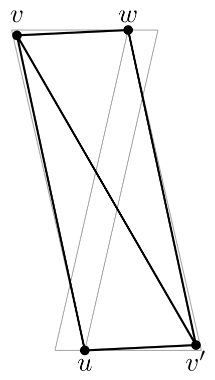


Figure 5: The lower bound for the $\theta_{(4k+2)}$-graph

4 Spanning Ratio of the $\theta_{(4k+4)}$-Graph

The $\theta_{(4k+2)}$-graph has the nice property that any line perpendicular to the bisector of a cone is parallel to the boundary of a cone (Lemma 1). As a result of this, if u, v, and w are vertices with v in one of the upper corners of T_{uw}, then T_{wv} is completely contained in T_{uw}. The $\theta_{(4k+4)}$-graph does not have this property. In this section, we show how to exploit this to construct a lower bound for the $\theta_{(4k+4)}$-graph whose spanning ratio exceeds the worst case spanning ratio of the $\theta_{(4k+2)}$-graph.

Theorem 5 *The worst case spanning ratio of the $\theta_{(4k+4)}$-graph is at least $1 + 2\tan\left(\frac{\theta}{2}\right) + 2\tan^2\left(\frac{\theta}{2}\right)$.*

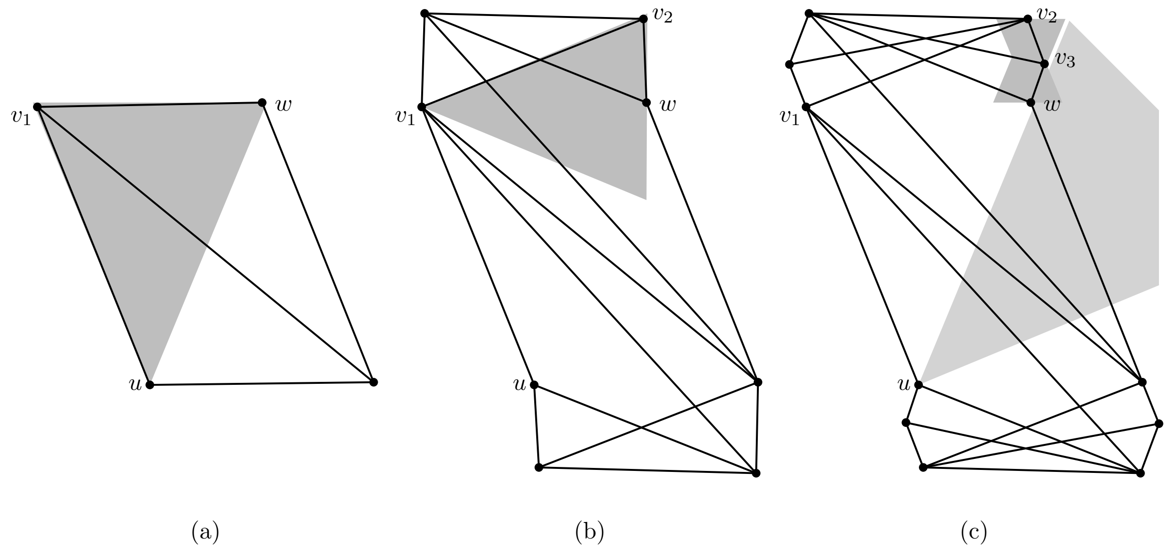


Figure 6: The construction of the lower bound for the $\theta_{(4k+4)}$-graph

Proof. We construct the lower bound example by extending the shortest path between two vertices u and w in three steps. We describe only how to extend one of the shortest paths between these vertices. To extend all shortest paths, the same modification is performed in each of the analogous cases, as shown in Figure 6.

First, we ensure that there is no edge between u and w by placing a vertex v_1 in the upper corner of T_{uw} that is furthest from w. See Figure 6a. Next, we place a vertex v_2 in the corner of T_{v_1w} that lies in the same cone of u as w and v_1. See Figure 6b. Finally, we place a vertex v_3 in the intersection of T_{v_2w} and T_{wv_2} to ensure that there is no edge between v_2 and w. See Figure 6c. Note that we cannot place v_3 in the lower right corner of T_{v_2w} since this would cause an edge between u and v_3 to be added, creating a shortcut to w.

One of the shortest paths in the resulting graph visits u, v_1, v_2, v_3, and w. Thus, to obtain a lower bound for the $\theta_{(4k+4)}$-graph, we compute the length of this path.

Let m be the midpoint of the side of T_{uw} opposite u. By construction, we have that $\angle v_1um = \angle wum = \angle v_2v_1w = \angle v_3v_2w = \angle v_3wv_2 = \theta/2$. See Figure 7. We can express the various line segments as follows:

$$\begin{aligned}
|uv_1| &= |uw| \\
|v_1w| &= 2\sin\left(\frac{\theta}{2}\right)\cdot|uw| \\
|v_1v_2| &= 2\tan\left(\frac{\theta}{2}\right)\cdot|uw| \\
|v_2w| &= 2\sin\left(\frac{\theta}{2}\right)\tan\left(\frac{\theta}{2}\right)\cdot|uw| \\
|v_2v_3| &= |v_3w| \;=\; \tan^2\left(\frac{\theta}{2}\right)\cdot|uw|
\end{aligned}$$

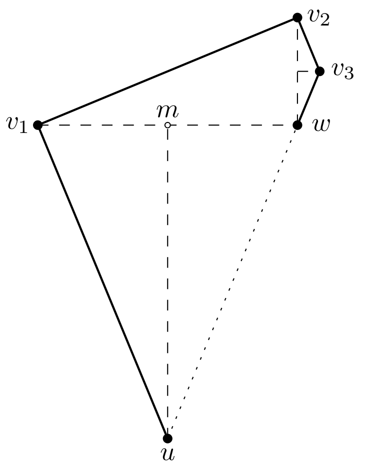


Figure 7: The lower bound for the $\theta_{(4k+4)}$-graph

Hence, the total length of the shortest path is $|uv_1| + |v_1v_2| + |v_2v_3| + |v_3w| = (1 + 2\tan(\theta/2) + 2\tan^2(\theta/2)) \cdot |uw|$. □

Finally, we show that increasing the number of cones of a θ-graph by 2 from $4k + 2$ to $4k + 4$ increases the worst case spanning ratio.

Theorem 6 *The worst case spanning ratio of the $\theta_{(4k+4)}$-graph is greater than that of the $\theta_{(4k+2)}$-graph, for any integer $k \geq 1$.*

Proof. Recall that the worst case spanning ratio of the $\theta_{(4k+2)}$-graph is $1 + 2\sin(\pi/(4k + 2))$ and that of the $\theta_{(4k+4)}$-graph is at least $1 + 2\tan(\pi/(4k + 4)) + 2\tan^2(\pi/(4k + 4))$. To prove the theorem, it suffices to show that $\tan(\pi/(4k+4)) + \tan^2(\pi/(4k+4))$ is greater than $\sin(\pi/(4k+2))$, for any integer $k \geq 1$.

For $x \in (0, \pi/6]$, it holds that $\sin x < x$, $\tan x > x$, and $\tan^2 x > x^2$. Since $k \geq 1$, both $\pi/(4k+2)$ and $\pi/(4k+4)$ are in the range $(0, \pi/6]$. Therefore, we have that:

$$\begin{aligned}\sin\left(\frac{\pi}{4k+2}\right) &< \frac{\pi}{4k+2} \\ &< \frac{\pi}{4k+4} + \left(\frac{\pi}{4k+4}\right)^2 \\ &< \tan\left(\frac{\pi}{4k+4}\right) + \tan^2\left(\frac{\pi}{4k+4}\right),\end{aligned}$$

as required. □

5 Conclusion

We showed that the $\theta_{(4k+2)}$-graph has a tight spanning ratio of $1 + 2\sin(\theta/2)$. This is the first time tight spanning ratios have been found for a large family of θ-graphs. Previously, the only θ-graph for which tight bounds were known was the θ_6-graph.

Furthermore, we showed that the $\theta_{(4k+4)}$-graph has a spanning ratio of at least $1 + 2\tan(\theta/2) + 2\tan^2(\theta/2)$. This result is somewhat surprising since, for equal values of k, the worst case spanning ratio of the $\theta_{(4k+4)}$-graph is greater than that of the $\theta_{(4k+2)}$-graph, showing that increasing the number of cones can make the spanning ratio worse.

There remain a number of open problems, such as finding lower bounds for the $\theta_{(4k+3)}$-graph and the $\theta_{(4k+5)}$-graph, and finding tight spanning ratios of the $\theta_{(4k+3)}$, $\theta_{(4k+4)}$, and $\theta_{(4k+5)}$-graphs. The best known upper bound for these graphs is $1/(1 - 2\sin(\theta/2))$. Furthermore, for the θ_4 and θ_5-graphs, neither upper nor lower bounds are known.

References

[1] N. Bonichon, C. Gavoille, N. Hanusse, and D. Ilcinkas. Connections between theta-graphs, Delaunay triangulations, and orthogonal surfaces. In *Proceedings of the 36th International Conference on Graph Theoretic Concepts in Computer Science (WG 2010)*, pages 266–278, 2010.

[2] P. Bose, R. Fagerberg, A. van Renssen, and S. Verdonschot. Competitive routing in the half-θ_6-graph. In *Proceedings of the 23rd ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 1319–1328, 2012.

[3] P. Chew. There are planar graphs almost as good as the complete graph. *Journal of Computer and System Sciences*, 39(2):205–219, 1989.

[4] K. Clarkson. Approximation algorithms for shortest path motion planning. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC 1987)*, pages 56–65, 1987.

[5] J. Keil. Approximating the complete Euclidean graph. In *Proceedings of the 1st Scandinavian Workshop on Algorithm Theory (SWAT 1988)*, pages 208–213, 1988.

[6] J. Ruppert and R. Seidel. Approximating the *d*-dimensional complete Euclidean graph. In *Proceedings of the 3rd Canadian Conference on Computational Geometry (CCCG 1991)*, pages 207–210, 1991.

Near-Linear-Time Deterministic Plane Steiner Spanners and TSP Approximation for Well-Spaced Point Sets

Glencora Borradaile* David Eppstein†

Abstract

We describe an algorithm that takes as input n points in the plane and a parameter ϵ, and produces as output an embedded planar graph having the given points as a subset of its vertices in which the graph distances are a $(1+\epsilon)$-approximation to the geometric distances between the given points. For point sets in which the Delaunay triangulation has bounded sharpest angle, our algorithm's output has $O(n)$ vertices, its weight is $O(1)$ times the minimum spanning tree weight, and the algorithm's running time is bounded by $O(n\sqrt{\log\log n})$. We use this result in a similarly fast deterministic approximation scheme for the traveling salesperson problem.

1 Introduction

A *spanner* of a set of points in a geometric space [13] is a sparse graph having those points as its vertices, and with its edge lengths equal to the geometric distance between the endpoints, such that the graph distance between any two points accurately approximates their geometric distance. More precisely, the *dilation* of a spanner is the smallest number δ for which the graph distance of every pair of points is at most δ times their geometric distance. It has long been known that very good spanners exist: for every constant $\epsilon > 0$ and constant dimension d, it is possible to find a spanner for every set of n points in $O(n \log n)$ time such that the dilation of the spanner is at most $1+\epsilon$, its weight is at most a constant times the weight of the minimum spanning tree, and its degree is constant [4].

A spanner is *plane* if no two of its edges (represented as planar line segments) intersect except at their shared endpoints [10]. Plane spanners with bounded dilation are known; for instance, the Delaunay triangulation is a spanner in this sense [9]. However, it is not possible for these spanners to have dilation arbitrarily close to one. For instance, for four points at the corners of a square, any plane graph must avoid one of the diagonals and have dilation at least $\sqrt{2}$. However, the addition of *Steiner points* allows smaller dilation for pairs of original points. For instance, the plane graph formed by overlaying all possible line segments between pairs of input points has dilation exactly one, although its $\Theta(n^4)$ combinatorial complexity is high. Less trivially, in the *pinwheel tiling*, a certain aperiodic tiling of the plane, any two vertices of the tiling at geometric distance D from each other have graph distance $D + o(D)$ [18]. We define a *plane Steiner δ-spanner* for a set of points to be a graph that contains the points as a subset of its vertices, is embedded with straight line edges and no crossings in the plane, and achieves dilation δ for pairs of points in the original point set. We do not require pairs of points that are not both original to be connected by short paths.

Arikati et al. [2] show how to construct a plane Steiner spanner in $O(n \log n)$ time, but do not bound the total weight of the graph. Of course, spanners may also be constructed by forming an arrangement of line segments [12] representing the edges of a nonplanar spanner graph; this planarization does not change the spanner's weight, but may add a large number of edges and vertices. A paper of Klein [15] on graph spanners provides an alternative basis for plane Steiner spanner construction. Generalizing a previous result of Althöfer et al. [1], Klein shows that any n-vertex planar graph with a specified subset of vertices may be thinned to provide a planar Steiner $(1+\epsilon)$-spanner for the graph distances on the specified subset, with weight $O(1/\epsilon^4)$ times the weight of the minimum Steiner tree of the subset, in time $O((n \log n)/\epsilon)$. Klein combined this result with methods from another paper [16] to provide a polynomial time approximation scheme for the traveling salesperson problem in weighted planar graphs. Using Klein's method to reduce the weight of the geometric spanner formed by the arrangement of all line segments connecting pairs of a given point set would lead to a low weight plane $(1+\epsilon)$ Steiner spanner for the point set, but again with a large number of vertices and edges. Ideally, we would prefer plane Steiner spanners that not only have low weight, but also have a linear number of edges and vertices.

Small and low-weight plane Steiner spanners in turn could be used with Klein's planar graph algorithms to derive a deterministic polynomial time approximation

*School of Electrical Engineering and Computer Science, Oregon State University. Supported in part by the National Science Foundation under grant CCF-0963921.

†Department of Computer Science, University of California, Irvine. Supported in part by the National Science Foundation under grant 0830403, and by the Office of Naval Research under MURI grant N00014-08-1-1015.

scheme for the Euclidean TSP. The previous randomly shifted quadtree approximation scheme of Arora [3] and guillotine subdivision approximation scheme of Mitchell [17] have runtimes that are polynomial for fixed ϵ but with an exponent depending on ϵ; in contrast, Klein's method takes time linear in the spanner size for any fixed ϵ. However, combining Klein's method with the nonlinear-size Steiner spanners described above would not improve on a different deterministic TSP approximation scheme announced by Rao and Smith [19]. Their method is based on *banyans*, a generalized type of spanner that must accurately approximate all Steiner trees, and it takes $O(n \log n)$ time for any fixed ϵ and any fixed dimension, although its details do not appear to have been published yet.

These past results raise several questions. Are banyans necessary for fast TSP approximation, or is it possible to make do with more vanilla forms of spanners? How quickly may low-weight plane Steiner spanners be constructed, and how quickly may the TSP be approximated? And how few vertices are necessary in a plane Steiner spanner?

In this work we provide some partial answers, for planar point sets that are well-spaced in the sense that their Delaunay triangulation avoids sharp angles. We show that, when both ϵ and the sharpest angle in the Delaunay triangulation are bounded by fixed constants, then there exist plane Steiner $(1+\epsilon)$-spanners with $O(n)$ vertices whose weight is $O(1)$ times the minimum spanning tree weight (with a near-linear dependence on ϵ, improving the quartic dependence in Klein's construction). The dependence on ϵ and the sharpest angle is given in Theorem 8. Our spanners may be constructed in linear time given the Delaunay triangulation, or (by combining a fast Delaunay triangulation algorithm of Buchin and Mulzer [11] with fast integer sorting algorithms [14]) in time $O(n\sqrt{\log \log n})$ for points with integer coordinates. Combining these spanners with the methods from Klein [16] leads to near-linear-time TSP approximation for the same class of point sets.

2 Delaunay triangulations without sharp angles

The *Delaunay triangulation* DT of a set S of points (called *sites*) is a triangulation in which the circumcircle of each triangle does not contain any sites in its interior. For points in general position (no four cocircular) the Delaunay triangulation is uniquely defined and its sharpest angle α is at least as large as the sharpest angle in any other triangulation. As we show in this section, Delaunay triangulations that do not have any triangles with sharp angles have two key properties: their total weight $w(\text{DT})$ is small relative to the weight $w(\text{MST})$ of the minimum spanning tree, and every point in the plane is covered by only a few circumcircles.

Lemma 1

$$w(DT) = f_w(\alpha) w(MST) \text{ where } f_w(\alpha) = \frac{1+\cos\alpha}{1-\cos\alpha}.$$

Proof. The proof follows closely that of Lemma 3.1 of Klein [15]. Let T be the MST. (Recall $T \subseteq \text{DT}$.) Let H_0 be the non-self-crossing Euler tour of T. We consider the edges of $\text{DT} \setminus T$ in a leaf-to-root order with respect to the dual tree T^*: $e_1, e_2, \ldots, e_k$. e_1 makes a triangle with edges a_1 and b_1 of H_0. Recursively define H_i as the tour resulting from removing a_i and b_i from H_{i-1} and adding e_i: $w(H_i) = w(H_{i-1}) + w(e_i) - w(a_i) - w(b_i)$.

Since the α is the smallest angle of triangle $e_i a_i b_i$ and e_i is longest when $w(a_i) = w(b_i)$, we get $w(e_i) \le (w(a_i) + w(b_i)) \cos\alpha$.

Combining, we get:
$w(H_i) \le w(H_{i-1}) + (1 - 1/\cos\alpha)\, w(e_i)$.

Summing:
$w(H_k) \le w(H_0) + (1 - 1/\cos\alpha) \sum_i w(e_i)$.

By rearranging and using the facts $w(H_k) \ge 0$ and $w(H_0) = 2\, w(\text{MST})$, the lemma follows. □

Lemma 2 *The number of Delaunay circumdisks whose interiors contain a given point in the plane is at most*

$$f_e(\alpha) = 2\pi/\alpha. \tag{1}$$

Proof. The lemma trivially holds for points that are sites. Let x be a non-site point in the plane. Then the Delaunay triangles whose circumcircles contain x are exactly the ones that get removed from the Delaunay triangulation if we add x to S and re-triangulate. Therefore, the number of Delaunay circumcircles that contain x is the same as the degree of x in the Delaunay triangulation, DT_x of $S \cup \{x\}$.

Let d be the degree of x in DT_x. Then, one of the triangles, xqr, in DT_x incident to x has an angle at x of at most $2\pi/d$. Edge qr must be a side of a triangle qrs in DT replacing triangle xqr, because after the removal of x, line segment qr is still a chord of the empty circle that circumscribed xqr; however, triangle qrs has a circumcircle that extends at least as far from qr on the side of the triangle as the circumcircle of xqr, and therefore angle qsr is at least as sharp as angle qxr. So it must be that $2\pi/d > \alpha$ or $d < 2\pi/\alpha$, proving the lemma. □

3 Portals for chords

As we now show, it is possible to space a set of *portals* along an edge of a Delaunay triangulation in such a way that any chord of a Delaunay circumcircle must pass close to one of the portals, relative to the chord length.

Lemma 3 *Let AD be a chord of a circle O, let B and C be points interior to segment AD, and let EF be another*

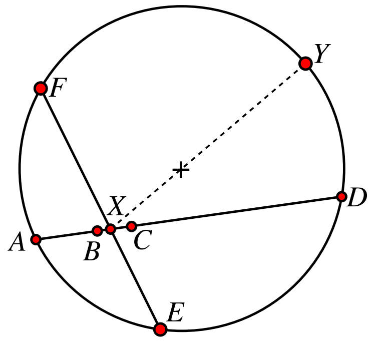


Figure 1: Figure for Lemma 3

chord of O, crossing AD between B and C. Then the distance from chord EF to the nearer of the two points B and C is at most $|EF| \cdot |BC|/2\min(|AB|, |CD|)$.

Proof. The points of the lemma are illustrated in Figure 1. We assume without loss of generality that F is on the side of AD that contains the center of O, as drawn in the figure; let Y be the point of O farthest from X, lying on the line through X and the center of O. Note that the distance from line EF to the closer of B and C is at most $\min(|BX|, |CX|) \le |BC|/2$, so it remains to prove that $|EF| \ge \min(|AB|, |CD|)$. But if F lies on the arc between A and Y, then $|EF| \ge |FX| \ge |AB|$, and if F lies on the arc between Y and D then $|EF| \ge |FX| \ge |CD|$. In either case the result follows. $\square$

Lemma 4 *Let s be a line segment in the plane, and let $\epsilon > 0$. Then there exists a set $P_{s,\epsilon}$ of $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ points on s with the property that, for every circle O for which s is a chord, and for every chord t of O that crosses s, t passes within distance $\epsilon|t|$ of a point in $P_{s,\epsilon}$.*

Proof. Our set $P_{s,\epsilon}$ includes both endpoints of s and its midpoint. In the subset of s from one endpoint p_0 to the midpoint, we add a sequence of points p_i, where p_1 is at distance $O(\epsilon^2 s)$ from p_0 with a constant of proportionality to be determined later and where for each $i > 1$, p_i is at distance $\epsilon\, d(p_0, p_{i-1})$ from p_i. Because the distance from p_0 increases by a $(1+\epsilon)$ factor at each step, the set formed in this way contains $O(\frac{1}{\epsilon}\log\frac{1}{\epsilon})$ points.

If chord t crosses s between some two points p_i and p_{i+1} for $i \ge 1$, or between the last of these points and the midpoint of s, then Lemma 3 ensures that the nearer of these two points is within distance $\epsilon|t|$ of the chord.

Otherwise, t crosses s between p_0 and p_1. Let r be the radius of O, necessarily at least $|s|/2$, and suppose that t passes within distance δr of p_0. Because of the choice of p_1, $\delta = O(\epsilon^2)$. By the Pythagorean theorem, $|t| \ge r\sqrt{2\delta - \delta^2} = \Omega(|r|\sqrt{\delta})$, and combining this information with the definition of δ shows that t is within distance $O(\sqrt{\delta}|t|) = O(\epsilon|t|)$ of p_0. By choosing the constant of proportionality in the placement of p_1 appropriately we can ensure that this distance is at most $\epsilon|t|$. $\square$

We call the points in $P_{s,\epsilon}$ *portals.*

4 Spanning the portals within each triangle

Within each triangle of the Delaunay triangulation, we will use a plane Steiner spanner that connects the portals that lie on the triangle edges. For this special case, we use a construction that generalizes to an arbitrary set P of points on the boundary of an arbitrary planar convex set K. Given a range of angles $\theta \pm \delta$, we say that a path is $(\theta \pm \delta)$-*angle-bounded* if it is piecewise linear and each linear segment remains within this range of angles, and we say that a point p on the boundary of K is $(\theta \pm \delta)$-*extreme* if there does not exist a $(\theta \pm \delta)$-angle-bounded path from p to a point interior to K.

Lemma 5 *Every $(\theta \pm \delta)$-angle-bounded path has length $1 + O(\delta^2)$ times the distance between its endpoints.*

Proof. The most extreme case is a path that follows two sides of an isosceles triangle having the endpoints of the path as base, for which the length is the length of the base multiplied by $1/\cos\delta = 1 + O(\delta^2)$. $\square$

Lemma 6 *Let P be a set of n points on the boundary of a convex set K with perimeter ℓ, let θ be an angle, and let $\delta > 0$ be a positive number. Then in time $O(n \log n)$ we can construct a set S of $O(n)$ line segments within K, with total length $O((\ell \log n)/\delta)$, with the property that for every point p in P there exists a $(\theta \pm \delta)$-angle-bounded path in S from p to a $(\theta \pm \delta)$-extreme point of K.*

Proof. We consider the points of P in an order we will later define; for each such point p that is not itself $(\theta \pm \delta)$-extreme, we extend two line segments with angles $\theta - \delta$ and $\theta + \delta$ until reaching either an extreme point of K or one of the previously constructed line segments. Thus, a $(\theta \pm \delta)$-angle-bounded path from p may be found by following either of these two line segments, and continuing to follow each line segment hit in turn by the previous line segment on the path, until reaching an extreme point.

The non-extreme points of P, because K is convex, form a contiguous sequence along the boundary of K. We extend segments from the two endpoints of this sequence, then from its median, and then finally we continue recursively in the two subsequences to the left and right of the median, as shown in Figure 2.

The segments from the first two points of P contribute a total length of ℓ to S. For each subsequent point p, the length of each added segment is at most proportional to $1/\delta$ times the length of the part of the boundary of K that extends from p to the most recently previously considered point in the same direction. Because of the ordering of the points, each point along the boundary is

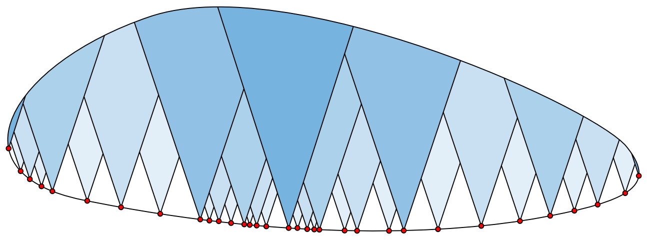


Figure 2: Figure for Lemma 6

charged in this way for $O(\log n)$ segments, so adding this quantity over all points, the total length of the segments is $O((\ell \log n)/\delta)$ as claimed. We may construct S in $O(n \log n)$ time by using binary search to determine the endpoint on K of each segment. $\square$

Lemma 7 *Let P be a set of n points on the boundary of a convex set K with perimeter ℓ, and let $\epsilon > 0$ be a positive number. Then in time $O(n^2/\epsilon)$ we can construct a plane Steiner $(1+\epsilon)$-spanner for P, with all spanner edges in K, with $O(n^2/\epsilon)$ edges and vertices, and with total length $O((\ell \log n)/\epsilon)$.*

Proof. We choose $\delta = O(\sqrt{\epsilon})$ (with a constant of proportionality determined later), partition the circle into $O(1/\delta)$ arcs of angle 2δ, let θ_i be the angle at the center of the ith arc, and apply Lemma 6 to each of the arcs $\theta_i \pm \delta$. We overlay the resulting system of $O(n/\delta)$ line segments; when two line segments from different arcs both have the same angle and starting point, we choose the longer of the two to use in the overlay. The resulting arrangement of line segments has $O(n^2/\epsilon)$ edges and vertices and total length $O((\ell \log n)/\epsilon)$ as required, and can be constructed in time $O(n^2/\epsilon)$ using standard line segment arrangement construction algorithms [12].

To see that this is a spanner, we must show that every pair (p, q) of points in P may be connected by a short path. Let θ be the angle formed by the segment from p to q, choose i such that $\theta + \delta \leq \theta_i \leq \theta + 3\delta$, and use Lemma 6 to find a $(\theta_i \pm \delta)$-angle-bounded path pp' in the spanner from p to a $(\theta_i \pm \delta)$-extreme point p'. Because of the angle bound, p' must be clockwise of q. Similarly, we may choose θ_j within $O(\delta)$ of $\pi + \theta$, and find a $(\theta_j \pm \delta)$-angle-bounded path qq' to a $(\theta_j \pm \delta)$-extreme point q' that is counterclockwise of p. These two paths (depicted in Figure 3) must cross at at least one point x, and the combination of the path from p to x and from x to q lies within the spanner and is $(\theta \pm O(\delta))$-angle-bounded. By Lemma 5, this path has length at most $1+O(\delta^2)$ times the distance between its endpoints, and by choosing the constant of proportionality in the definition of δ appropriately we can cause this factor to be at most $1+\epsilon$. $\square$

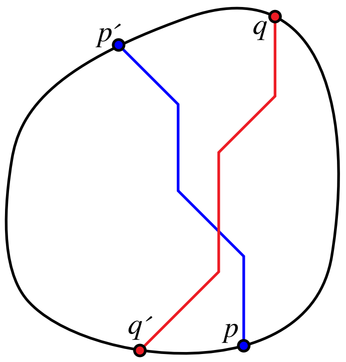


Figure 3: Figure for Lemma 7

5 Spanner construction

We now have all the pieces for our overall spanner construction.

Theorem 8 *Let P be a planar point set whose Delaunay triangulation is given and has sharpest angle α, and let $\epsilon > 0$ be given. Then in time $O(n \log^2(1/(\alpha\epsilon))/(\alpha^2\epsilon^3))$ we can construct a plane Steiner $(1+\epsilon)$-spanner for P with $O(n \log^2(1/(\alpha\epsilon))/(\alpha^2\epsilon^3))$ vertices and edges, and with total length $O(w(MST)\log(1/(\alpha\epsilon))/(\alpha^2\epsilon))$.*

Proof. We apply Lemma 3 to place portals along the edges of the triangulation, such that each chord s of a Delaunay circle passes within distance $O(\alpha\epsilon|s|)$ of a portal on each Delaunay edge that it crosses. We then apply Lemma 7 within each Delaunay triangle to construct a $1+O(\epsilon)$-spanner for the portals on the boundary of that triangle.

The construction time is bounded by the time to construct the spanners within each triangle. Since there are $O(\log(1/(\alpha\epsilon))/(\alpha\epsilon))$ portals on each triangle, the time to construct the spanner for a single triangle is $O(\log^2(1/(\alpha\epsilon))/(\alpha^2\epsilon^3))$ and the total time over the whole graph is $O(n \log^2(1/(\alpha\epsilon))/(\alpha^2\epsilon^3))$. This bound also applies to the number of vertices and edges in the constructed spanner. By Lemma 1, the total perimeter of the Delaunay triangles is $O(w(\text{MST})/\alpha^2)$, and combining this bound with the length bound of Lemma 7 gives total length $O(w(\text{MST})\log(1/(\alpha\epsilon))/(\alpha^2\epsilon))$ for the spanner edges.

To show that this is a spanner, we must find a short path between any two of the input points p and q. By Lemma 3, the line segment pq passes within distance $O(\alpha\epsilon|s|)$ of a portal on every Delaunay edge that it crosses, where s is the chord of one of the Delaunay circles for the crossed edge. By Lemma 2, the total length of all of these chords is $O(|pq|/\alpha)$, so we may replace pq by a polygonal path that contains a portal on each crossed Delaunay edge, expanding the total length by a factor of at most $1 + O((\alpha\epsilon)/\alpha) = 1 + O(\epsilon)$. Then, by Lemma 7 we may replace each portal-to-portal segment in this path by a path within the spanner for the portals in a single Delaunay triangle, again expanding the total length by a factor of at most $1 + O(\epsilon)$. By choosing constants of proportionality appropriately, we may make the total length expansion be at most $1 + \epsilon$. □

6 Approximating the TSP

An algorithm of Klein [16] provides a linear time approximation scheme for the traveling salesperson problem in a planar graph. Its first step is to find a low-weight spanner of the graph. A subsequent paper, also by Klein [15] describes an algorithm that, given a planar graph G and a subset S of the nodes, finds a subgraph of G whose weight is $O(\epsilon^{-4})$ times that of the minimum-weight tree spanning S and that is a $(1 + \epsilon)$-spanner for the shortest-path metric on S [15]. This subset spanner construction can be substituted for the first step of Klein's approximation scheme, resulting in an algorithm for approximating the TSP on the subset S. However, in this more general result, the spanner construction takes time $O(n \log n)$, so the total time for the approximation scheme is $O(n \log n)$ for any fixed $\epsilon > 0$.

For points in the plane, we may substitute our own faster low-weight spanner construction for the first step of the approximation scheme. The remaining steps of the approximation use only the facts that the points we are seeking to connect into a tour are vertices in a planar graph, and that the whole graph has total weight proportional to the minimum spanning tree of the given points. Thus, we obtain the following result:

Theorem 9 *For any fixed α and ϵ, the optimal traveling salesman tour of sets of n points in the plane with sharpest Delaunay triangulation angle at most α may be approximated to within a $1 + \epsilon$ factor in time $O(n)$ plus the time needed to construct the Delaunay triangulation.*

It would also be possible to design a TSP approximation scheme more directly using a framework used by Borradaile, Klein and Mathieu [8] to solve the Steiner tree problem; details on how this framework applies to TSP were given by Borradaile, Demaine and Tazari [7] in generalizing the planar framework to bounded-genus graphs. Their algorithm, as interpreted for point sets in the Euclidean plane, would partition the triangles of the Delaunay triangulation into layers according to their depth from the infinite face in the dual graph so that the boundary between layers is an ϵ fraction of the optimal solution. This can be achieved with depth $f_w(\alpha)/\epsilon = O\left(\frac{1}{\alpha^2\epsilon}\right)$; each layer has tree-width polynomial in this depth. The problem is then solved using dynamic programming, where the dynamic programs are additionally indexed by the portals. The base cases are made to correspond to the triangles in which the intersection with any tour can be enumerated. The size of the dynamic program is therefore bounded singly-exponentially in $1/\epsilon$ and $1/\alpha$. This task is slightly easier in the geometric setting than in the planar graph setting as computing shortest paths is trivial.

7 Looking ahead

The most obvious question posed by this work is: how do we remove the dependence on α? The dependence on α appears in two places: in the number of circumcircles that enclose a point and in the weight of the Delaunay triangulation. We believe that it should be possible to remove these dependencies on α by treating groups of skinny triangles as a single region. In fact, using this idea, we are able to remove each dependency separately, but not together.

An alternative approach to removing this dependence would be to augment the input to remove all sharp angles from its Delaunay triangulation, but this may sometimes need a number of added points that cannot be bounded by a function of n [5]. A construction based on quadtrees shows that every point set may be augmented with $O(n)$ points so that the Delaunay triangulation has no obtuse angles [5]; the resulting triangulation may also be modified to have the bounded circumcircle enclosure property, despite having some sharp angles, and may be constructed as efficiently as sorting [6]. Applying our spanner construction method to the augmented input would allow us to completely eliminate the dependence on α in the time and output complexity of our spanners, but at the expense of losing control over their total weight. Once a spanner is constructed in this way, Klein's subset spanner [15] can be used to reduce its weight, allowing it to be used in an algorithm to approximate the TSP for arbitrary planar point sets in time $O(n \log n)$ for any fixed $\epsilon > 0$, but this does not improve on the time bound of Rao and Smith [19].

Unlike in the methods of Arora [3], Mitchell [17], Rao and Smith [19] and Borradaile, Klein and Mathieu [8], the approximation error in our method is charged locally as opposed to globally. In the quad-tree based approximation schemes, the error incurred is charged to the dissection lines that form the quad tree. In the planar approximation-scheme framework for Steiner tree, the error incurred is charged to an $O(\text{MST})$-weight subgraph called the mortar graph which acts much like the quad-tree decomposition. Our charging scheme is much more similar to that used by Klein for the subset tour problem in planar graphs [15]. However, in applying the planar approximation-scheme frameworks of either Klein or Borradaile, Klein and Mathieu, some error is incurred in partitioning the graph into pieces of bounded treewidth. This error is proportional to the graph that is partitioned, which in our case is either the spanner (for Klein's scheme) or the triangulations (for Borradaile et. al.'s scheme). This error is indirectly related to OPT by way of the $O(\text{MST})$ weight of the spanner and triangulation. By current techniques, this source of error does not seem avoidable.

Finally, our spanner construction more closely ties Euclidean and planar distance metrics together. By unifying the approximation schemes in these two related metrics, it may be possible to generalize these methods to other two dimensional metrics.

References

[1] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete Comput. Geom.*, 9(1):81–100, 1993.

[2] S. Arikati, D. Z. Chen, L. Chew, G. Das, M. Smid, and C. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Proc. 4th Eur. Symp. Alg.*, volume 1136 of *LNCS*, pages 514–528, 1996.

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

[4] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: short, thin, and lanky. In *Proc. 27th ACM Symp. Theory of Computing*, pages 489–498, 1995.

[5] M. Bern, D. Eppstein, and J. Gilbert. Provably good mesh generation. *J. Comput. Sys. Sci.*, 48(3):384–409, 1994.

[6] M. Bern, D. Eppstein, and S.-H. Teng. Parallel construction of quadtrees and quality triangulations. *Int. J. Computational Geometry & Applications*, 9(6):517–532, 1999.

[7] G. Borradaile, E. Demaine, and S. Tazari. Polynomial-time approximation schemes for subset-connectivity problems in bounded-genus graphs. *Algorithmica*, to appear.

[8] G. Borradaile, P. Klein, and C. Mathieu. An $O(n \log n)$ approximation scheme for Steiner tree in planar graphs. *ACM Trans. Algorithms*, 5(3):1–31, 2009.

[9] P. Bose, L. Devroye, M. Löffler, J. Snoeyink, and V. Verma. The spanning ratio of the Delaunay triangulation is greater than $\pi/2$. In *Proc. 21st Canad. Conf. Comput. Geom.*, 2009.

[10] P. Bose and M. Smid. On plane geometric spanners: a survey and open problems. Manuscript, 2009.

[11] K. Buchin and W. Mulzer. Delaunay triangulations in $O(\text{sort}(n))$ time and more. *J. ACM*, 58(2):A6, 2011.

[12] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *J. ACM*, 39(1):1–54, 1992.

[13] D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 9, pages 425–461. Elsevier, 2000.

[14] Y. Han and M. Thorup. Integer sorting in $O(n\sqrt{\log \log n})$ expected time and linear space. In *Proc. 43rd Annual Symp. Foundations of Computer Science*, pages 135–144, 2002.

[15] P. Klein. A subset spanner for planar graphs, with application to subset TSP. In *Proc. 38th ACM Symp. Theory of Computing*, pages 749–756, 2006.

[16] P. Klein. A linear-time approximation scheme for TSP in undirected planar graphs with edge-weights. *SIAM J. Comput.*, 37(6):1926–1952, 2008.

[17] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: a simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM J. Comput.*, 28(4):1298–1309, 1999.

[18] C. Radin and L. Sadun. The isoperimetric problem for pinwheel tilings. *Comm. Math. Phys.*, 177(1):255–263, 1996.

[19] S. B. Rao and W. D. Smith. Approximating geometrical graphs via "spanners" and "banyans". In *Proc. 30th ACM Symp. Theory of Computing*, pages 540–550, 1998.

On the Strengthening of Topological Signals in Persistent Homology through Vector Bundle Based Maps

Eric Hanson Francis Motta Chris Peterson Lori Ziegelmeier*

1 Abstract

Persistent homology is a relatively new tool from topological data analysis that has transformed, for many, the way data sets (and the information contained in those sets) are viewed. It is derived directly from techniques in computational homology but has the added feature that it is able to capture structure at multiple scales. One way that this multi-scale information can be presented is through a barcode. A barcode consists of a collection of line segments each representing the range of parameter values over which a generator of a homology group persists. A segment's length relative to the lenght of other segments is an indication of the strength of a corresponding topological signal. In this paper, we consider how vector bundles may be used to re-embed data as a means to improve the topological signal. As an illustrative example, we construct maps of tori to a sequence of Grassmannians of increasing dimension. We equip the Grassmannian with the geodesic metric and observe an improvement in barcode signal strength as the dimension of the Grassmannians increase.

2 Introduction

The need to efficiently extract critical information from large data sets has been growing for decades and is central to a variety of scientific, engineering and mathematical challenges. In many settings, underlying constraints on the data allow it to be considered as a sampling of a topological space. It is a fundamental problem in topological data analysis to develop theory and tools for recovering a topological space from a noisy, discrete sampling. The tools that one might choose to use on a given problem depend on the density, quality, and quantity of the data, on the ambient space from where the sampling is drawn, and on the complexity of the topological space as a sub-object of an ambient space. In this paper, we will focus on data consisting of points sampled from an algebraic variety (the zero locus of a system of polynomials). The data points are obtained using the tools of numerical algebraic geometry. Derived from techniques in homotopy continuation, numerical algebraic geometry allows one to use numerical methods to cheaply sample a large collection of low-noise points from an algebraic set. Persistent homology (PH) allows one to use such a sample to gain insight into the topological structure of the algebraic variety. Implementations of persistent homology are readily available and have been used in a variety of applications, ranging from the analysis of experimental data to analyzing the topology of an algebraic variety. However, as with any algorithm, there are computational limitations. Generally, the time and space required for the persistence computation grows rapidly with the size of the input sample, so the maximum size of a sample is limited. Often, applications of PH start with noisy, real-world data, which may also be limited in size [16]. However, our consideration begins with effectively unlimited, arbitrarily accurate data. Experience shows that as one increases the sample size of a fixed space, the quality of the topological signals produced by PH improves. Since the computational complexity of persistent homology limits the size of a sample, methods of preprocessing data that improve the topological signal, without increasing the sample size, are desirable.

In this paper, we consider how topological re-embeddings affect the topological signal obtained from persistent homology. First, a construction of PH and the inherent challenges of interpreting its output is briefly introduced. Then, we will provide details about the setting in which we have applied this embedding technique, using computational topology to analyze projective algebraic varieties. Lastly, results for a specific example are displayed and interpreted.

3 Background

3.1 Persistent Homology

Beginning with a finite set of data points, which are viewed as a noisy sampling of a topological space, assume one has a way of building the matrix of pairwise distances between points in the data set. From this distance matrix, one constructs a nested sequence of simplicial complexes indexed by a parameter t. Fixing a field $\mathbb{K}$, for each simplicial complex, one builds an associated chain complex of vector spaces over $\mathbb{K}$. The i^{th} homology of the chain complex is a vector space and its dimension corresponds to the i^{th} Betti number, $\beta_i(\mathbb{K})$,

*Colorado State University, Department of Mathematics {hanson, motta, peterson, ziegelme}@math.colostate.edu

of the corresponding topological space. For each pair $t_1 < t_2$, there is a pair of simplicial complexes, S_{t_1} and S_{t_2}, and an inclusion map $j : S_{t_1} \hookrightarrow S_{t_2}$. This inclusion map induces a chain map between the associated chain complexes which further induces a linear map between the corresponding i^{th} homology vector spaces. For each i, the totality of the collection of i^{th} homology vector spaces and induced linear maps can be encoded as a graded $\mathbb{K}[t]$-module known as the persistence module. The i^{th} bar code is a way of presenting the invariant factors of the persistence module. As the invariant factors of the persistence module directly relate to the Betti numbers, from the bar code one can visualize the Betti numbers as a function of the scale, t, and can visualize the number of independent homology classes that persist across a given time interval $[t_i, t_j]$. For foundational material and overviews of computational homology in the setting of persistence, see [8, 21, 12, 6, 9, 20, 15].

One commonly used method for building a nested sequence of simplicial complexes from a distance matrix is through a *Vietoris-Rips* complex [12]. This is done by first building the 1-skeleton of the simplicial complex then determining the higher dimensional faces as the clique complex of the 1-skeleton. More precisely, fix $t > 0$, a collection of points X, and a metric, $d(x_i, x_j)$ for $x_i, x_j \in X$. The 1-skeleton of the Vietoris-Rips complex, $C_t(X)$, is defined by including the edge $x_i x_j \in C_t(X)$ if $d(x_i, x_j) \leq t$. A higher dimensional face is included in $C_t(X)$ if all of its lower dimensional sub-faces are in $C_t(X)$. In other words, the abstract k-simplices of $C_t(X)$ are given by unordered $(k+1)$-tuples of sample points whose pairwise distances do not exceed the parameter t.

Given a collection of data points, the resulting Vietoris-Rips complex, and its homology, is highly dependent on the choice of parameter t. To reconcile this ambiguity, persistence exploits that if $t_1 < t_2$ then C_{t_1} is a sub simplicial complex of C_{t_2}. In other words, as t grows so do the Vietoris-Rips complexes, giving an inclusion from earlier complexes to those which appear later. The idea then is to not only consider the homology for a single specified choice of parameter, but rather track topological features through a range of parameters [12]. Those which persist over a large range of values are considered signals of underlying topology, while the short lived features are taken to be noise inherent in approximating a topological space with a finite sample [10].

For clarity, consider 4 points in the plane with distance matrix

$$\begin{bmatrix} 0 & t_2 & t_5 & t_3 \\ t_2 & 0 & t_1 & t_6 \\ t_5 & t_1 & 0 & t_4 \\ t_3 & t_6 & t_4 & 0 \end{bmatrix}.$$

We label the points a, b, c and d and build the sequence of Vietoris-Rips simplicial complexes up to $\mathbf{C}_{t_5}$. Table 1 shows the Betti information (where β_i is the dimension of the i^{th} homology vector space) for the example illustrated in Figure 1 over the range of parameter values $t \geq 0$.[1]

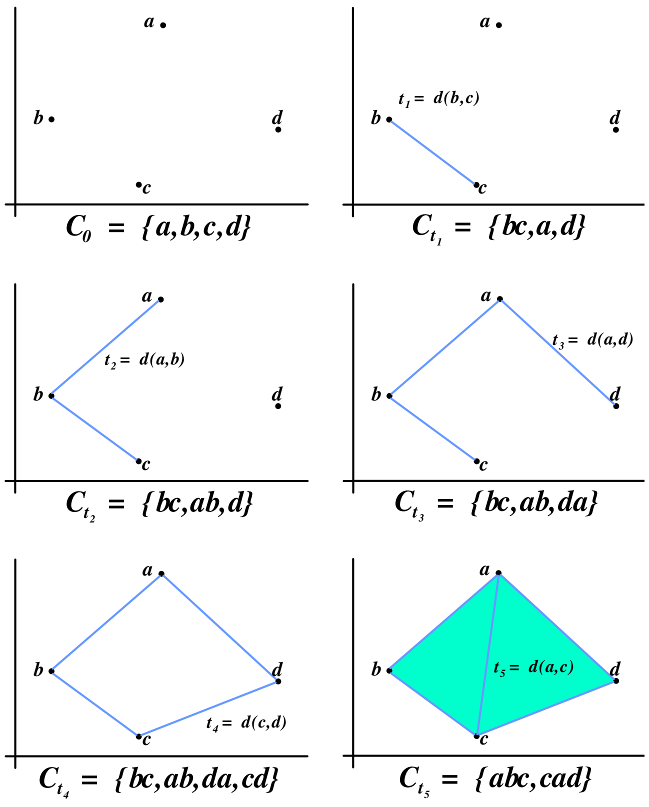


Figure 1: A sequence of Vietoris-Rips simplicial complexes shown geometrically and abstractly along with their maximal faces.

filtration times (t)	β_0	β_1
$0 \leq t < t_1$	4	0
$t_1 \leq t < t_2$	3	0
$t_2 \leq t < t_3$	2	0
$t_3 \leq t < t_4$	1	0
$t_4 \leq t < t_5$	1	1
$t_5 \leq t$	1	0

Table 1: Persistent homology data

Even in this simple example, the amount of information created by the persistent homology computation is non-trivial. Furthermore, an effective rendering of the complexes in Figure 1 is only possible because there are very few points in the example. In the 4-point example, at time t_6 the simplicial complex $\mathbf{C}_{t_6}$ becomes three-

[1]For finite data there will only be finitely many parameter values where the simplicial complex changes.

dimensional. As the vertex set or the dimension of the ambient space grows, visualizing the sequence of complexes is not practical.

The *barcode* is a visual method for presenting some of the homological information in a sequence of chain maps. In particular, it displays the structure of the invariant factors of the i^{th} persistence module. Figure 2 is the barcode corresponding to the example of the four points in the plane described in Figure 1.

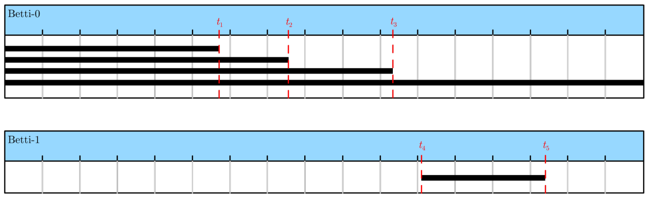


Figure 2: Barcodes corresponding to Figure 1

The computational requirements of the persistence computation is related to the sample size. It is often the case that computing the persistent homology using the Rips filtration is impractical. There is an alternative construction, introduced by Carlsson and de Silva, called the *witness complex* [7, 13]. Starting with a large sample set X, one picks a distinguished subset $L \subset X$ of *landmark points.* The witness complex is a family of simplicial complexes built on L using information from the entire set X.

To build the witness complex, first use the landmark set to assign to each point $x \in X$ the numbers $m_k(x)$ corresponding to the distance from x to its $(k+1)$-th nearest landmark point. For each integer k $(0 < k < |X|)$ and vertices $\{l_{j_i} | 0 \leq i \leq k\} \subset L$, include the k-simplex $[l_{j_0} l_{j_1} ... l_{j_k}]$ in the complex (at time t) if there exists a point $x \in X$ such that $\max\{d(l_{j_i}, x) | 0 \leq i \leq k\} \leq t + m_k(x)$, and if all of its faces are in the complex [1].

The output of the witness filtration is sensitive to the choice of landmark set. One technique for choosing a landmark set, called sequential maxmin, is implemented in the freely distributed persistent homology software package JPlex [17]. The procedure for using sequential maxmin is to first pick a point $l_0 \in X$ then inductively choose the i-th landmark point from X by choosing the point furthest from the set of $(i-1)$ points already chosen. In practice, this seems to produce a stronger topological signal than choosing L randomly, so it is the method we will utilize.

3.2 Algebraic Varieties and Numerical Algebraic Geometry

A motivating problem for this paper is the computation of the Betti numbers of a complex projective algebraic variety from numerically obtained sample points. The method we use to obtain sample points derive from several algorithms in numerical algebraic geometry.

The term *numerical algebraic geometry* is often used to describe a wide ranging set of numerical methods to extract algebraic and geometric information from polynomial systems. The field includes a diverse collection of algorithms (both numeric and numeric-symbolic). The class of numerical algorithms that we use are rooted in homotopy continuation. The idea of homotopy continuation is to link a pair of polynomial systems through a deformation and to relate features of the two systems through this deformation. For example, one can track known, isolated, complex solutions of one polynomial system to unknown, complex solutions of a second polynomial system through a deformation of system parameters.

Let Z be the complex algebraic variety associated to an ideal in $\mathbb{C}[z_1, \ldots, z_N]$. With numerical homotopy continuation methods combined with monodromy breakup, it is practical to produce sets of numerical data points which numerically lie on each of the irreducible components of Z [19, 18].

There are several important features of the methods of numerical algebraic geometry that are worth highlighting. The first feature is the ability to refine sample points to arbitrarily high precision via Newton's method. A second feature is the ability to produce an arbitrary number of sample points on any given component. A third feature is the parallelizability of these numerical methods. For instance, 10,000 processors could be used in parallel to track 10,000 paths and could be used in parallel to refine the accuracy of each sample point to arbitrarily high precision. The basic algorithms of numerical algebraic geometry (including monodromy breakup) are implemented in the freely available software package, Bertini [4].

It is important to note that sampling is computationally inexpensive, so obtaining large sample sets does not pose a significant challenge. However, it is not clear that this sampling technique will provide points that are well distributed for the purpose of persistent homology computations.

4 Main Idea

4.1 Theory

By its very nature, persistent homology characterizes intrinsic topological features which should be relatively insensitive to the metric used to build a pairwise distance matrix. However, experiments show that the signal *strength* is impacted by the choice of metric. In our experience, even if the topological features remain the same, the ability to correctly interpret information from a barcode depends on the strength of the signal. We will consider the barcode signal strength of mappings of an algebraic variety into various Grassmannians.

The Grassmannian $Gr(n,k)$ is a manifold parametrizing all k dimensional subspaces of a fixed n dimensional vector space. The Grassmann manifold $Gr(n+1,1)$ is the projective space $\mathbb{P}^n$, and from this vantage point, Grassmannians can be viewed as generalizations of projective spaces. These manifolds can be given a topological structure, a differential structure and even the structure of a projective variety (e.g. via the Plucker embedding).

Points in an n-dimensional projective space correspond to 1-dimensional subspaces of a fixed $(n+1)$-dimensional vector space. A natural notion of distance is given by the smallest angle between the subspaces. We would like to define the distance between points on other Grassmannians by extending this definition. As a starting point, it can be shown that any unitarily invariant metric on a Grassmannian can be written in terms of the principal angles between the corresponding subspaces. The principal angles between a pair of subspaces A, B in $\mathbb{C}^n$ can be determined as follows. First, determine matrices M and N whose columns form orthonormal bases for A and B. Next, determine the singular value decomposition $M^*N = U\Sigma V^*$. The singular values of M^*N are the diagonal entries of Σ. These singular values are the cosines of the principal angles between A and B (see [5]). If A and B are k-dimensional, then there will be principal angles $\Theta(A,B) = (\theta_1, \theta_2, \dots, \theta_k)$ with $0 \le \theta_1 \le \theta_2 \le \cdots \le \theta_k \le \pi/2$. There are many common metrics computed as functions of the principal angles [2]. For instance, the Fubini-Study metric induced by the Plucker embedding is $d_{FS}(A,B) = \cos^{-1}\left(\prod_{i=1}^{k}\cos\theta_i\right)$. We have found that the Fubini-Study metric does not, in general, yield a strong signal, and instead, we restrict our attention to the geodesic distance

$$d(A,B) = \sqrt{\theta_1^2 + \ldots + \theta_k^2}.$$

Since we wish to compare the effect of considering a sample in various Grassmannian embeddings, it remains to define what we mean by relative topological signal strength. Imagine we know that our sample was taken from a topological space whose i^{th} Betti number is b_i. Assuming that the b_i longest segments in the barcode represent these topological features, we will measure signal strength as the ratio of the sum of the lengths of the b_i longest segments to the sum of the total length of all the segments in the i^{th} Betti barcode, including noise. Note that noise consisting of many segments of total length m and noise consisting of a single segment of length m cannot be distinguished by this statistic. To cope with this limitation we also consider the ratio of the length of the b_i^{th} longest segment to the $(b_i+1)^{th}$ longest segment in the barcode.

4.2 Embeddings into the Grassmannian

Consider a complex projective curve, $C \subset \mathbb{P}^2$, defined by the zero locus of a homogenous polynomial $F(x,y,z)$. When we think of the zero set as a projective variety, then each point, $[x:y:z]$ on C, corresponds to a 1-dimensional subspace of $\mathbb{C}^3$ (note that the homogeneity of the equation leads to the conclusion that if (x,y,z) is a solution then so is (cx,cy,cz) for any $c \in \mathbb{C}$). Thus, points on a projective variety correspond to one-dimensional subspaces of $\mathbb{C}^3$ constrained to lie on the vanishing locus of a homogeneous polynomial. From this point of view, C is a sub-object of $\mathbb{P}^2 = Gr(3,1)$. We can sample random points on C with several different methods. If we wish to build a distance matrix from these points, then we should consider the distance between a pair of points as the principal angle between the one dimensional spaces to which they correspond.

Consider the matrix

$$E(x,y,z) := \begin{bmatrix} 0 & z & -y \\ -z & 0 & x \\ y & -x & 0 \end{bmatrix},$$

and observe that for any point $(x,y,z) \neq (0,0,0)$, the rank of $E(x,y,z)$ is 2. This can be seen by observing that the determinant of $E(x,y,z)$ is identically zero and that the locus of conditions such that all 2×2 minors of $E(x,y,z)$ are zero force $x=y=z=0$. For each value of (x,y,z), we consider the row space of $E(x,y,z)$. Note also that

$$\begin{bmatrix} 0 & z & -y \\ -z & 0 & x \\ y & -x & 0 \end{bmatrix}\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}.$$

As a consequence, the row space of $E(x,y,z)$ is the same as the row space of $E(cx,cy,cz)$, and $E(x,y,z)$ can be viewed as a rule for attaching a smoothly varying 2 dimensional subspace to each point of $\mathbb{P}^2$. In other words, $E(x,y,z)$ determines a rank two vector bundle on $\mathbb{P}^2$. For each one dimensional subspace of $\mathbb{C}^3$, we can determine a 2-dimensional subspace of $\mathbb{C}^3$ by mapping it to the row space of $E([x:y:z])$. If $\Phi_0 : \mathbb{P}^2 \to Gr(3,2)$ denotes the image of this map, then by restriction this gives a map $\phi_0 : C \to Gr(3,2)$.

For each integer $k > 0$, consider the set of monomials in x,y,z of degree k. We construct new matrices, $E_k(x,y,z)$, by concatenating matrices of the form $m_i \cdot E(x,y,z)$ for each degree k monomial m_i. For example, $E_1(x,y,z)$ is the matrix

$$\begin{bmatrix} 0 & xz & -xy & 0 & yz & -y^2 & 0 & z^2 & -zy \\ -xz & 0 & x^2 & -yz & 0 & yx & -z^2 & 0 & zx \\ xy & -x^2 & 0 & y^2 & -yx & 0 & zy & -zx & 0 \end{bmatrix}.$$

For each k, $E_k(x,y,z)$ has constant rank 2 on $\mathbb{P}^2$ and can be used to define a map $\phi_k : C \to Gr(N_k, 2)$ (where

N_k is the number of columns of $E_k(x,y,z)$). Geometrically, the columns of $E_k(x,y,z)$ corresponds to a "spanning set for the space of sections of the twisted tangent bundle, $\mathbf{T}_{\mathbb{P}^2}(k-1)$". In this way, we can consider images of C, in increasingly large Grassmannians via the maps $\phi_0, \phi_1, \phi_2, \ldots$. It can be shown that for each k, Φ_k embeds $\mathbb{P}^2$ into $Gr(N_k, 2)$ and that ϕ_k embeds C into $Gr(N_k, 2)$.

4.3 Example

Consider the complex projective elliptic curve, $C \subset \mathbb{P}^2$ defined by the equation

$$x^2y + y^2z + z^2x = 0. \tag{1}$$

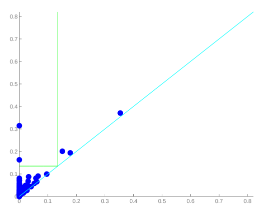
Topologically, C is a torus whose Betti numbers are $\beta_0 = 1, \beta_1 = 2, \beta_2 = 1$.

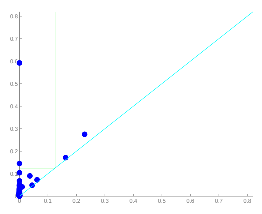
Using Bertini, we sampled 10,000 points satisfying Equation 1. We mapped each point to $Gr(N_k, 2)$ using ϕ_k, for $k = 1, \ldots, 10$. From these 10,000 points we fixed 100 landmark sets, L_i (of size 200) using the sequential maxmin algorithm with random initial points $l_{0,i}$ for $i = 1, \ldots, 100$. For each embedding $\phi_k(C)$ and for each of the fixed landmark sets, we compute the persistent homology barcodes for the zeroth and first Betti numbers using the witness complex construction.

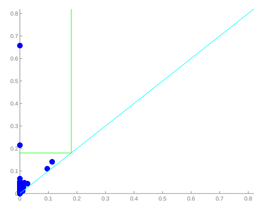
Using the geodesic distance to measure distances between points, Figure 3 shows prototypical Betti-1 barcodes for the images of the 10,000 points in $Gr(N_k, 2)$. In the figure, each segment in the barcode is plotted as a point in the (x,y)-plane with the x-coordinate corresponding to the starting parameter and the y-coordinate corresponding to the ending parameter. Short segments (i.e. topological noise) appear near the $y = x$ line. Notice that as we move the elliptic curve to Grassmannians of higher degree, the two longest segments in the barcode grow in length while the number and lengths of the other segments decrease.

In Figure 4, we plot the relative signal strength of the Betti-1 barcodes, as measured by the ratio of the sum of the length of the two longest segments to the total sum of lengths of all segments, averaged over all landmark sets for each embedding. We observe an increase from approximately 10% to 55% of the total length of the barcodes being accounted for in the longest two segments. We also observe that the improvement of the relative signal strength levels-off after $k = 5$.

Figure 5 compares the second longest segment in the Betti-1 barcode (corresponding to a topological circle) to the third longest segment (representing topological noise). We notice a sharp increase in this measure of signal strength followed by a similarly steep decrease. Together with the content of Figure 4 this indicates that after $k = 5$, the relative length of the longest Betti-1 segment remains somewhat unchanged while the disparity


(a) $k = 1$


(b) $k = 2$


(c) $k = 3$

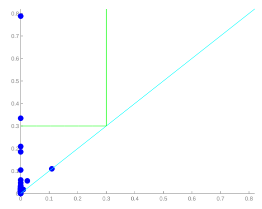

(d) $k = 4$

Figure 3: Betti-1 barcodes for each of the four specified embeddings.

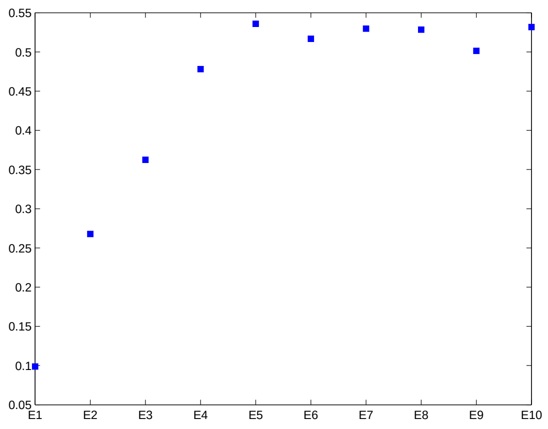


Figure 4: Average ratio of the sum of the longest two barcode lengths to the sum of the lengths of all barcodes for $k = 1, \ldots, 10$.

in the lengths of the second and third longest segments is diminished.

It is worth noting that there is a diminished signal strength from the projective variety to the embeddings in the Grassmannian. Our aim, however, is to compare topological signal strength improvement across successive Grassmannian embeddings.

5 Conclusion

Using the techniques of numerical algebraic geometry, we can sample arbitrarily many points, to an arbitrary degree of accuracy, on any prescribed component of an

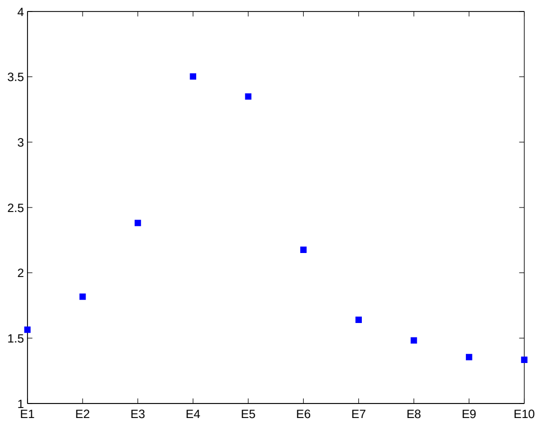


Figure 5: Average ratio of the second longest barcode to the third longest barcode for $k = 1, \ldots, 10$.

algebraic set. Using twists of the tangent bundle to projective space, we can map these points to a sequence of Grassmann manifolds of increasing dimension. With techniques of computational homology, we can build the persistence module and decompose the module into its invariant factors. A visual plot of the starting and ending points of the invariant factors aids in the understanding of the underlying variety as a topological space. Higher embeddings of the data seem to strengthen the topological signal.

For further research, we intend to develop improved sampling techniques for algebraic varieties. We will also conduct experiments to determine if alternate vector bundles or alternate metrics on the Grassmannian can be used to strengthen topological signals.

References

[1] H. Adams, JPlex with Matlab Tutorial. (2011) comptop.stanford.edu/programs/jplex/files/

[2] A. Barg and D.Yu. Nogin, Bounds on packings of spheres in the Grassmann manifold. IEEE Trans. on Info. Theory. 48 (2002), 2450-2454.

[3] D.J. Bates, J. D. Hauenstein, A. J. Sommese, and C. W. Wampler, Adaptive multiprecision path tracking. *SIAM J. Numer. Anal.* 46 (2008), 722-746.

[4] D.J. Bates, J. D. Hauenstein, A. J. Sommese, and C. W. Wampler, Bertini: Software for numerical algebraic geometry. http://www.nd.edu/∼sommese/bertini.

[5] A. Bjorck and G. Golub, Numerical methods for computing angles between linear subspaces. *Mathematics of Computation* 27, (1973), no 123, 579-594.

[6] G. Carlsson, Topology and data. Bulletin of the American Mathematical Society, Vol. 46 (2009), no. 2, 255308.

[7] V. de Silva and G. Carlsson, Topological estimation using witness complexes. *SPBG '04 Symposium on Point-Based Graphics* (2004), 157-166.

[8] H. Edelsbrunner and J. Harer, Persistent homology - a survey. *Contemporary Math* 453 (2008), 257-282.

[9] H. Edelsbrunner and J. Harer, Computational Topology: An Introduction. American Mathematical Society, Providence, RI, 2010. xii+241 pp.

[10] H. Edelsbrunner, D. Letscher, and A. Zomorodian, Topological persistence and simplification. Discrete Computational Geometry, 28:4 (2002), 511-533.

[11] A. Galantai and Cs. J. Hegedus, Jordan's principal angles in complex vector spaces. Numer. Linear Algebra Appl. 13 (2006), 589-598.

[12] R. Ghrist, Barcodes: The persistent topology of data. Bulletin of the American Mathematical Society, Vol. 45 (2008), pp. 61-75.

[13] L. Guibas and S. Oudot, Reconstruction using witness complexes. *Proc. 18th ACM-SIAM Sympos. on Discrete Algorithms* (2007).

[14] A. Hatcher, Algebraic Topology, Cambridge University Press (2002).

[15] T. Kaczynski, K. Mischaikow, and M. Konstantin, Computational Homology. Applied Mathematical Sciences 157. Springer-Verlag, New York, 2004. 480 pp.

[16] D. Mumford, A. Lee, and K. Pederson, The non-linear statistics of high-contrast patches in natural images. Itnl. J. Computer Vision. 54 (2003), 83-103.

[17] H. Sexton and M. Vejdemo-Johansson, JPlex simplicial complex library. http://comptop.standord.edu/programs/jplex/.

[18] A.J. Sommese and C.W. Wampler, The Numerical Solution of Systems of Polynomials. World Scientific Publishing Co. Pte. Ltd., Hackensack, NJ, 2005.

[19] A.J. Sommese, J. Verschelde, and C.W. Wampler, Numerical decomposition of the solution sets of polynomials into irreducible components. *SIAM J. Numer. Anal.* 38 (2001), 2022-2046.

[20] A. Zomorodian, Topology for computing. Cambridge Monographs on Applied and Computational Mathematics, 16. Cambridge University Press, Cambridge, 2005. xiv+243 pp.

[21] A. Zomorodian and G. Carlsson, Computing persistent homology. *Discrete Comput. Geom.* 33 (2005), no. 2, 249274

A Multicover Nerve for Geometric Inference

Donald R. Sheehy*

Abstract

We show that filtering the barycentric decomposition of a Čech complex by the cardinality of the vertices captures precisely the topology of k-covered regions among a collection of balls for all values of k. Moreover, we relate this result to the Vietoris-Rips complex to get an approximation in terms of the persistent homology.

1 Introduction

Computational geometers use topology to certify correctness of geometric constructions and inferences. For example, in surface reconstruction one often wants a homeomorphic reconstruction [8] or in medial axis approximation, one might seek a homotopy equivalence between the approximation and the true medial axis[4]. In some sensor network problems, topological guarantees can certify that the network covers a geometric domain [7]. A growing literature deals explicitly with the inference of topological structure in data sets (see Carlsson [1] for a survey).

Many of these examples depend on the Nerve Theorem or variations thereof to extract topological information from geometry. This classic result in algebraic topology relates the topology of a union of sets to that of a simplicial complex called the nerve (under certain conditions on the intersections of the sets).

In this paper, we extend the Nerve Theorem to consider regions covered by at least k different sets. In the language of sensor networks, this new nerve captures the notion of k-coverage. Whereas the Nerve Theorem can be applied directly for any fixed k, there is little correspondence between the nerves computed for different values of k. We show that a natural filtration of the barycentric decomposition of the nerve can capture this information for all values of k.

Noise and outliers are a major problem in topological data analysis. Even a single outlier can appear as a significant topological feature using standard methods. By considering k-covered regions only, our filtration ignores up to k points locally. This is closely related to a common approach to de-noising data for topological data analysis points are treated as noise if the distance to their kth nearest neighbor is at least some threshold (see [11] and [2] for two notable examples). Our method has the added advantage that it is easy to relate results for different choices of k.

We prove our results in the setting of persistent homology. This allows us to relate the main result also to sets in general metric spaces where it may be difficult to compute k-wise intersections directly.

The specific case we are interested in is the (k,α)-offsets of a point set $P \subset \mathbb{R}^d$, defined as the α-sublevel set of the kth nearest neighbor distance function. Equivalently, this is the subset of $\mathbb{R}^d$ covered by at least k balls of radius α centered at points in P (see Figure 1).

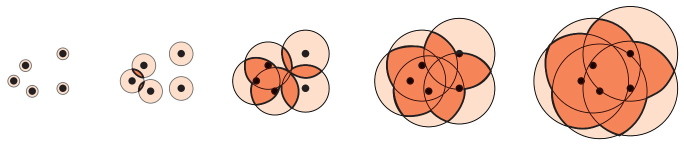


Figure 1: The α-offsets overlaid with the $(2,\alpha)$-offsets for growing values of α.

2 Background

Topology. We will assume a basic knowledge of standard definitions in topology including topological spaces, homotopy equivalence, and homology. The book by Munkres [10] is a good source for all the necessary background. We use $\mathrm{H}_*(X)$ to denote the homology groups of X and $X \simeq Y$ to denote homotopy equivalence.

Simplicial Complexes. A **simplicial complex** S is family of subsets of a vertex set that is closed under taking subsets. That is, if $\sigma' \subset \sigma \in S$ then $\sigma' \in S$. The elements of a simplicial complex are called **simplices** and the elements of the simplices are called **vertices**. The **dimension** of a simplex σ is defined as $|\sigma| - 1$. In this paper, we deal purely with abstract simplicial complexes and do not make any assumptions about how they are embedded.

Given a subset U of the vertices of S, the **induced subcomplex** of S on the vertex set U is the set of simplices of S whose vertices are all in U.

Filtrations. A **filtration** is a nested sequence of topological spaces. In this paper, we deal primarily with

*Geometrica, INRIA Saclay, `don.r.sheehy@gmail.com`

filtrations parameterized by the nonnegative real numbers. So, a filtration $\mathcal{G} = \{G^\alpha\}_{\alpha \geq 0}$ is a family of spaces such that $G^\alpha \subseteq G^\beta$ whenever $0 \leq \alpha \leq \beta$. For brevity, we omit the parameter set and write $\mathcal{G} = \{G^\alpha\}$ when it is obvious that α ranges over $\mathbb{R}_{\geq 0}$. If the spaces in a filtration are all simplicial complexes then we call it a **filtered simplicial complex**. Throughout the paper, superscripts are always used to index into a filtration.

Persistent Homology and Persistence Diagrams The theory of persistent homology describes the way the topology of the spaces in a filtration change as α ranges over $\mathbb{R}_{\geq 0}$. Given a filtered simplicial complex $\mathcal{F}$, there is an efficient algorithm for computing its so-called **persistence diagram** $\operatorname{Dgm}\mathcal{F}$ [13]. This diagram is a multiset of points in the extended plane $(\mathbb{R} \cup \{\infty\})^2$ where every point represents a topological feature. The x- and y-coordinates of a point in the persistence diagram represent the values of α for which that particular feature appeared and disappeared respectively in the filtration. For example, a cycle of edges may form at $\alpha = x$ and then be filled in (killed) by triangles at $\alpha = y$. These are sometimes called the birth and death times of the feature.

By convention the diagonal $x = y$ is included in every persistence diagram. The distance from this diagonal is a measure of how long a feature persisted before being killed.

It is beyond the scope of this paper to give a full treatment of persistent homology; the book by Edelsbrunner and Harer gives a complete background[9].

From Sets to Filtrations. Persistent homology extends homology theory from spaces to filtrations. Below, we present some basic definitions and known results about persistence with an emphasis on the generalization from spaces to filtrations. Often, this means we will overload notation so that the same notation applies to both spaces and filtrations.

First we define the basic set operations on filtrations, defining $\{F^\alpha\} \cup \{G^\alpha\} = \{F^\alpha \cup G^\alpha\}$ and $\{F^\alpha\} \cap \{G^\alpha\} = \{F^\alpha \cap G^\alpha\}$. For any collection T of sets (or filtrations), we use the shorthand notation $\bigcup T = \bigcup_{S \in T} S$ and $\bigcap T = \bigcap_{S \in T} S$.

The first task is to extend a notion of topological equivalence from spaces to filtrations. Since the persistence diagram is a complete invariant of the filtration, two filtrations $\mathcal{F}$ and $\mathcal{G}$ have isomorphic persistent homology if $\operatorname{Dgm}\mathcal{F} = \operatorname{Dgm}\mathcal{G}$. Unfortunately, to prove $\operatorname{Dgm}\mathcal{F} = \operatorname{Dgm}\mathcal{G}$, it does not suffice to have $\mathrm{H}_*(F^\alpha) \cong \mathrm{H}_*(G^\alpha)$ or even $F^\alpha \simeq G^\alpha$. The following lemma gives a sufficient condition. It is a special case of the Persistence Equivalence Theorem [9, page 159]

Lemma 1 *Let $\mathcal{F} = \{F^\alpha\}$ and $\mathcal{G} = \{G^\alpha\}$ be filtrations. If for all $0 \leq \alpha \leq \beta$, there are isomorphisms $\mathrm{H}_*(F^\alpha) \to \mathrm{H}_*(G^\alpha)$ and $\mathrm{H}_*(F^\beta) \to \mathrm{H}_*(G^\beta)$ that commute with the homomorphisms $\mathrm{H}_*(F^\alpha) \to \mathrm{H}_*(F^\beta)$ and $\mathrm{H}_*(G^\alpha) \to \mathrm{H}_*(G^\beta)$ induced by inclusion, then $\operatorname{Dgm}\mathcal{F} = \operatorname{Dgm}\mathcal{G}$.*

Simplicial Maps. Let S and T be simplicial complexes. A map $f : S \to T$ is a **simplicial map** if f maps vertices to vertices and for every $\sigma \in S$, $f(\sigma) \in T$. A simplicial map is defined entirely by how it maps vertices to vertices. A simplicial map that is both injective and surjective is an **isomorphism of simplicial complexes**. We say that $\mathcal{F} = \{F^\alpha\}$ and $\mathcal{G} = \{G^\alpha\}$ are **isomorphic filtered simplicial complexes** if there exists a family of isomorphisms $\phi_\alpha : F^\alpha \to G^\alpha$ such that for all $0 \leq \alpha \leq \beta$, ϕ_α is the restriction of ϕ_β to F^α, denoted $\phi_\alpha = \phi_\beta|_{F^\alpha}$. The following Lemma follows directly from the definition of isomorphic filtrations and Lemma 1.

Lemma 2 *If $\mathcal{F}$ and $\mathcal{G}$ are isomorphic filtered simplicial complexes then $\operatorname{Dgm}\mathcal{F} = \operatorname{Dgm}\mathcal{G}$.*

When $S \subset T$, a map $f : S \to T$ is a **retraction** if $f(\sigma) = \sigma$ for all $\sigma \in S$. A pair of simplicial maps $f, g : S \to T$ are **contiguous** if $f(\sigma) \cup g(\sigma) \in T$ for all $\sigma \in S$. The theory of contiguity is a simplicial analogue of homotopy theory. The following lemma gives a homology analogue of a deformation retraction.

Lemma 3 ([12]) *Let X and Y be simplicial complexes such that $X \subseteq Y$ and let $i : X \hookrightarrow Y$ be the canonical inclusion map. If there exists a simplicial retraction $\pi : Y \to X$ such that $i \circ \pi$ and id_Y are contiguous, then i induces an isomorphism $i_\star : \mathrm{H}_*(X) \to \mathrm{H}_*(Y)$ between the corresponding homology groups.*

Barycentric Decomposition. Let S be a simplicial complex. A **flag** in S is an ordered subset of simplices $\{\sigma_1, \ldots, \sigma_t\} \subseteq S$ such that $\sigma_1 \subset \cdots \subset \sigma_t$. The **barycentric decomposition** of S is the simplicial complex formed by the set of flags of S:

$$\operatorname{Bary} S := \{U \subset S : U \text{ is a flag of } S\}.$$

We also define the barycentric decomposition of a filtered simplicial complex $\{S^\alpha\}$ to be the filtered simplicial complex $\operatorname{Bary}\{S^\alpha\} := \{\operatorname{Bary} S^\alpha\}$.

There is a natural filtration on a barycentric decomposition induced by considering only the flags of some minimum cardinality. We define the complexes in this filtration as

$$k\text{-Bary } S := \{\gamma \in \operatorname{Bary} S : \min_{\sigma \in \gamma} |\sigma| \geq k\}.$$

As before, this definition is extended to filtered complexes $\{S^\alpha\}$ as k-Bary $\{S^\alpha\} := \{k\text{-Bary } S^\alpha\}$.

The operation of taking barycentric decompositions does not change the underlying topology. This fact is expressed in the following lemma, whose proof is trivial and omitted.

Lemma 4 *If $\mathcal{S}$ is a filtered simplicial complex then*

$$\operatorname{Dgm} \mathcal{S} = \operatorname{Dgm} (\operatorname{Bary} \mathcal{S})$$

Note that this lemma is not true if we replace Bary with k-Bary .

Nerves. Let $\mathcal{F} = \{\{F_1^\alpha\}, \ldots, \{F_n^\alpha\}\}$ be a collection of filtrations. Define $\mathcal{F}_\alpha$ to be the collection of sets $\{F_1^\alpha, \ldots, F_n^\alpha\}$.

We say that $\mathcal{F}_\alpha$ is a **good open cover** of $\bigcup \mathcal{F}_\alpha$ if all F_i^α and their intersections are empty or contractible. This condition is easily satisfied if the F_i^α are open convex sets. We say that $\mathcal{F}$ is a **good filtered cover** if $\mathcal{F}^\alpha$ is a good open cover for all $\alpha \geq 0$.

The **nerve** of a collection of sets $\mathcal{F}_\alpha$ is the abstract simplicial complex $\operatorname{Nerve} \mathcal{F}_\alpha := \{U \subseteq \mathcal{F}_\alpha : \bigcap U \neq \emptyset\}$. The nerve of a collection of filtrations $\mathcal{F}$ is the filtered simplicial complex $\operatorname{Nerve} \mathcal{F} := \{\operatorname{Nerve} \mathcal{F}_\alpha\}_{\alpha \geq 0}$.

The following is a classic result in algebraic topology called the Nerve Theorem.

Theorem 5 (The Nerve Theorem) *If $\mathcal{F}_\alpha$ is a good open cover then*

$$\bigcup \mathcal{F}_\alpha \simeq \operatorname{Nerve} \mathcal{F}_\alpha.$$

The extension of the Nerve Theorem to the persistence setting follows from the Persistent Nerve Lemma of Chazal and Oudot [5] and Lemma 1:

Theorem 6 (Persistent Nerves) *If $\mathcal{F}$ is a good filtered cover then*

$$\operatorname{Dgm} \left\{\bigcup \mathcal{F}_\alpha\right\} = \operatorname{Dgm} (\operatorname{Nerve} \mathcal{F}).$$

k-Covers. For any set S, the notation $\binom{S}{k}$ denotes the set of k-element subsets of S. Given a collection $\mathcal{F}$ of sets (or filtrations), the k-Cover of the collection is the set of k-wise intersections:

$$k\text{-Cover}\, \mathcal{F} := \left\{\bigcap U\right\}_{U \in \binom{\mathcal{F}}{k}}.$$

The k-cover of a collection of sets is a new collection of sets. The k-cover of a collection of filtrations is a new collection of filtrations.

3 Barycentric Bifiltration

The Barycentric Čech Filtration. Consider a set of points $P \subset \mathbb{R}^d$. The **Čech complex** at scale α is the nerve of the set of α-balls centered at the points of P. The collection of these complexes at all scales is the **Čech filtration** $\mathcal{C} = \{\mathcal{C}^\alpha\}$. The k-barycentric decomposition of the Čech filtration is $\tilde{\mathcal{C}}_k := k\text{-Bary}\, \mathcal{C}$.

Since $\tilde{\mathcal{C}}_{k+1}^\alpha \subseteq \tilde{\mathcal{C}}_k^\alpha$ for any $\alpha \geq 0$ and $k \in \mathbb{N}$, this gives a filtration in two variables known as a bifiltration, where one dimension is parameterized by (increasing) α and the other is parameterized by (decreasing) k. In fact, the construction of $\tilde{\mathcal{C}}_k$ gives a general recipe for deriving a bifiltration from a filtered simplicial complex.

Our goal is to show that the filtration $\tilde{\mathcal{C}}_k$ has the same persistent homology as the (k, α)-offsets, P_k^α.

Theorem 7 *For any finite set of points $P \subset \mathbb{R}^d$ and any $k \in \mathbb{N}$, the persistence diagrams of the (k, α)-offsets of P and the k-barycentric decomposition of the Čech filtration are identical:*

$$\operatorname{Dgm} \tilde{\mathcal{C}}_k(P) = \operatorname{Dgm} \{P_k^\alpha\}.$$

This theorem follows from a more general result about good filtered covers, Theorem 10 below. It is the special case when the good filtered cover is the collection of balls of radius α centered at the points of P.

The Main Result Before getting to the main result, we set up some definitions and prove two necessary lemmas. Let $\mathcal{F}$ be a good filtered cover and let $k \in N$ be a fixed constant. Define the following filtrations:

$$\begin{aligned} \tilde{\mathcal{J}}_k &:= k\text{-Bary}\, (\operatorname{Nerve} \mathcal{F}) \\ \mathcal{N}_k &:= \operatorname{Nerve}\, (k\text{-Cover}\, \mathcal{F}) \\ \tilde{\mathcal{N}}_k &:= \operatorname{Bary}\, (\mathcal{N}_k) \end{aligned}$$

Formally, the vertices of $\tilde{\mathcal{N}}_k^\alpha$ are the simplices of $\mathcal{N}_k^\alpha$, those collections of k-wise intersections of sets in $\mathcal{F}_\alpha$ that have a nonempty intersection. However, we will instead identify this vertex set with the corresponding collection of k-tuples from $\mathcal{F}_\alpha$. Letting X^α and Y^α be the vertex sets of $\tilde{\mathcal{J}}_k^\alpha$ and $\tilde{\mathcal{N}}_k^\alpha$ respectively, we have

$$X^\alpha = \left\{U \subseteq \mathcal{F}_\alpha : |U| \geq k \text{ and } \bigcap U_\alpha \neq \emptyset\right\}$$
$$Y^\alpha = \left\{V \subseteq \binom{\mathcal{F}_\alpha}{k} : \bigcap_{V' \in V} \bigcap V' \neq \emptyset\right\}$$

The complex $\tilde{\mathcal{N}}_k^\alpha$ contains redundant information. The map $\pi : Y^\alpha \to Y^\alpha$ induces a simplicial map that "projects out" this redundant information. It is defined by

$$\pi(V) = \binom{\bigcup V}{k}.$$

Figure 2 demonstrates the construction of some of the simplicial complexes described above for the special case of the Čech filtration and $k = 2$.

The following Lemma shows that the persistence diagram of $\tilde{\mathcal{N}}_k$ is unchanged by π.

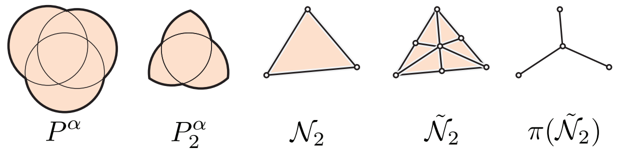


Figure 2: The construction of $\mathcal{N}_2$, its barycentric decomposition, and its image under π.

Lemma 8 $\operatorname{Dgm} \tilde{\mathcal{N}}_k = \operatorname{Dgm} \pi(\tilde{\mathcal{N}}_k)$.

Proof. By Lemma 1, it suffices to show that for all $\alpha \geq 0$, the inclusion $\psi : \pi(\tilde{\mathcal{N}}_k^\alpha) \hookrightarrow \tilde{\mathcal{N}}_k^\alpha$ induces an isomorphism at the homology level. As above, let Y^α be the set of vertices of $\tilde{\mathcal{N}}_k$ and let $s = \max_{V \in Y^\alpha} |V|$. For $i = 0 \dots s$, define

$$Y_i = \{V \in Y^\alpha : |V| \leq i\} \cup \{\pi(V) : V \in Y^\alpha, |V| > i\}.$$

Let A_i be the subcomplex of $\tilde{\mathcal{N}}_k^\alpha$ induced on Y_i. So

$$\pi(\tilde{\mathcal{N}}_k^\alpha) = A_0 \subset \cdots \subset A_s = \tilde{\mathcal{N}}_k^\alpha.$$

It will suffice to show that the inclusion $\psi_i : A_{i-1} \hookrightarrow A_i$ induces an isomorphism at the homology level for all $i = 1 \dots s$. Let $\pi_i : Y_i \to Y_{i-1}$ be defined as

$$\pi_i(V) = \begin{cases} V & \text{if } |V| < i \\ \pi(V) & \text{if } |V| \geq i \end{cases}$$

So $\psi = \psi_s \circ \cdots \circ \psi_1$ and $\pi = \pi_1 \circ \cdots \circ \pi_s$. Lemma 3 will give the desired isomorphism if π_i is a simplicial retraction such that $\psi_i \circ \pi_i$ is contiguous with the identity map, i.e. that (1) π_i restricts to the identity on Y_{i-1}, (2) $\pi_i(\sigma) \in A_{i-1}$ for all $\sigma \in A_i$, and (3) $(\sigma \cup \pi_i(\sigma)) \in A_i$ for all $\sigma \in A_i$.

Item (1) is obvious from the definitions. To prove (2) and (3), fix a simplex $\sigma = \{V_0, \dots, V_t\} \in A_i$ and let $\sigma' = \sigma \cup \pi_i(\sigma)$. If $\sigma = \sigma'$ then we are done, so we may assume that for some vertex $V_j \in \sigma$, $\pi(V_j) \notin \sigma$. Recall that the simplices of $\tilde{\mathcal{N}}_k$ (and also A_i) are strictly nested sequences of vertices. So, there is at most one vertex V_j such that $\pi(V_j) \notin \sigma$, namely the one with cardinality i. We may therefore express σ' as $\sigma \cup \{\pi(V_j)\}$. Since $\sigma \in A_i \subset \tilde{\mathcal{N}}_k$, $V_0 \subset \cdots \subset V_t$. Observe that $V \subseteq \pi(V)$ for all $V \in Y^\alpha$ and moreover that $U \subset V$ implies $\pi(U) \subseteq \pi(V)$. So, it follows that

$$V_0 \subset \cdots \subset V_j \subset \pi(V_j) \subset \pi(V_{j+1}) = V_{j+1} \subset \cdots \subset V_t.$$

The inclusion of $\pi(V_j) \subset \pi(V_{j+1})$ is strict because of the assumption that $\pi(V_j) \notin \sigma$. This is a strictly nested sequence of the vertices of σ' so $\sigma' \in A_i$, proving (3). Moreover, $\pi_i(\sigma) = \sigma' \setminus \{V_j\}$ so $\pi_i(\sigma) \in \tilde{\mathcal{N}}_k$ as well. Since $\pi_i(\sigma) \subset Y_{i-1}$, we conclude that $\pi_i(\sigma) \in A_{i-1}$, proving (2). □

Next, we prove that $\tilde{\mathcal{J}}_k$ and $\pi(\tilde{\mathcal{N}}_k)$ have identical persistence diagrams.

Lemma 9 $\operatorname{Dgm} \tilde{\mathcal{J}}_k = \operatorname{Dgm} \pi(\tilde{\mathcal{N}}_k)$

Proof. We will show that $\tilde{\mathcal{J}}_k$ and $\pi(\tilde{\mathcal{N}}_k)$ are isomorphic filtered simplicial complexes and so the result will follow from Lemma 2. It will suffice to show that for all $\alpha \geq 0$, $\tilde{\mathcal{J}}_k^\alpha$ and $\tilde{\mathcal{N}}_k^\alpha$ are isomorphic and that the isomorphism does not depend on α.

The desired isomorphism is the map $\phi : X^\alpha \to \pi(Y^\alpha)$ defined as $\phi(U) = \binom{U}{k}$. The inverse of this map is $\phi^{-1}(V) = \bigcup V$. So, ϕ takes subsets $U \subset \mathcal{F}^\alpha$ of size at least k such that $\bigcap U \neq \emptyset$ to the family of k-element subsets of U. It is easy to check that ϕ is a bijection.

To show that ϕ is an isomorphism, we will prove that σ is a simplex of $\tilde{\mathcal{J}}_k^\alpha$ if and only if $\phi(\sigma)$ is a simplex of $\pi(\tilde{\mathcal{N}}_k)$. Let $\sigma = (U_0, \dots, U_j) \in \tilde{\mathcal{J}}_k^\alpha$ be any simplex. By the definition of $\tilde{\mathcal{J}}_k^\alpha$, $U_0 \subset \cdots \subset U_j$. For any pair of vertices U_a and U_b, $U_a \subset U_b$ if and only if $\phi(U_a) \subset \phi(U_b)$. So, $\sigma \in \tilde{\mathcal{J}}_k^\alpha$ if and only if $\phi(U_0) \subset \cdots \subset \phi(U_j)$, which holds if and only if $\phi(\sigma) \in \pi(\tilde{\mathcal{N}}_k)$. □

We are now ready to prove the main theorem relating the persistence diagrams of k-Bary (Nerve $\mathcal{F}$) and $\bigcup k$-Cover $\mathcal{F}$. The basic strategy is illustrated in Figure 3.

Theorem 10 *If $\mathcal{F}$ is a good filtered cover and $k \in \mathbb{N}$ then* $\operatorname{Dgm}(k\text{-Bary}(\operatorname{Nerve}\mathcal{F})) = \operatorname{Dgm}\left(\bigcup k\text{-Cover}\,\mathcal{F}\right)$.

Proof. Recall the notations $\tilde{\mathcal{J}}_k$, $\mathcal{N}_k$, and $\tilde{\mathcal{N}}_k$ defined above.

$$\begin{aligned}
\operatorname{Dgm} \tilde{\mathcal{J}}_k &= \operatorname{Dgm} \pi(\tilde{\mathcal{N}}_k) && \text{[by Lemma 9]} \\
&= \operatorname{Dgm} \tilde{\mathcal{N}}_k && \text{[by Lemma 8]} \\
&= \operatorname{Dgm} \mathcal{N}_k && \text{[by Lemma 4]} \\
&= \operatorname{Dgm}(\operatorname{Nerve}(k\text{-Cover}\,\mathcal{F})) && \text{[by definition]} \\
&= \operatorname{Dgm}\left(\bigcup k\text{-Cover}\,\mathcal{F}\right) && \text{[by Theorem 6]}
\end{aligned}$$

□

The Barycentric Vietoris-Rips Filtration One drawback of the Čech filtration is that it requires testing sets of balls for common intersections. An alternative approach is to construct the edges only and include simplices for every clique. This is known as the Vietoris-Rips filtration $\mathcal{R} = \{\mathcal{R}^\alpha\}$, where

$$\mathcal{R}^\alpha := \{Q \subseteq P : \operatorname{diameter}(Q) \leq 2\alpha\}.$$

This can be computed using only the pairwise distances between points and therefore is well-defined for any metric space. We can apply the same barycentric bifiltration approach used above to yield a bifiltration $\tilde{\mathcal{R}}_k = \{\tilde{\mathcal{R}}_k^\alpha\} := k\text{-Bary}\,\mathcal{R}$.

Given filtrations $\mathcal{F}$ and $\mathcal{G}$, we say $\operatorname{Dgm}\mathcal{F}$ is c-approximation for $\operatorname{Dgm}\mathcal{G}$ if there is a 1-1 correspondence that maps each $(x, y) \in \operatorname{Dgm}\mathcal{F}$ to $(u, v) \in \operatorname{Dgm}\mathcal{G}$

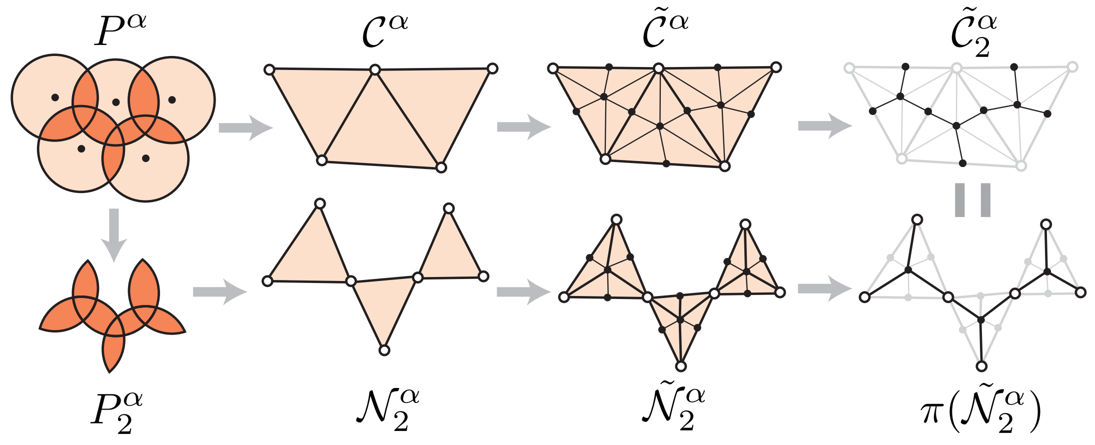

P^α
$\mathcal{C}^\alpha$
$\tilde{\mathcal{C}}^\alpha$
$\tilde{\mathcal{C}}_2^\alpha$
P_2^α
$\mathcal{N}_2^\alpha$
$\tilde{\mathcal{N}}_2^\alpha$
$\pi(\tilde{\mathcal{N}}_2^\alpha)$

Figure 3: We transform the collection of balls in two different ways to get equivalent complexes, $\tilde{\mathcal{C}}_k^\alpha$ (top) and $\pi(\tilde{\mathcal{N}}_k^\alpha)$ (bottom) for $k = 2$.

such that $u/c \le x \le cu$ and $v/c \le y \le cv$. A sufficient condition for $\operatorname{Dgm}\mathcal{F}$ to be a c-approximation to $\operatorname{Dgm}\mathcal{G}$ is that $\mathcal{F}^{\alpha/c} \subseteq \mathcal{G}^\alpha \subseteq \mathcal{F}^{c\alpha}$ for all $\alpha \ge 0$. This is a simple corollary to the Strong Stability Theorem of Chazal et al. [3].

The Vietoris-Rips filtration gives a good approximation to the Čech filtration. It was shown by de Silva and Ghrist that $\mathcal{C}^\alpha \subseteq \mathcal{R}^\alpha \subseteq \mathcal{C}^{c\alpha}$, where $c = 2$ for general metric spaces and $c = \sqrt{2}$ for Euclidean spaces[6]. So, the Vietoris-Rips filtration gives a c-approximation to the Čech filtration for persistent homology. The interleaving also implies the following extension to the Vietoris-Rips bifiltration $\{\tilde{\mathcal{R}}_k^\alpha\}$, where $\tilde{\mathcal{R}}_k = k\text{-Bary}\,\mathcal{R}$.

Theorem 11 *For any fixed k, the persistence diagram of the k-barycentric Rips filtration, $\{\tilde{\mathcal{R}}_k^\alpha\}$, is a $\sqrt{2}$-approximation to the persistence diagram of the (k, α)-offsets $\{P_k^\alpha\}$ when the underlying space is Euclidean, and is a 2-approximation for general metrics.*

Proof. It suffices to observe that $\mathcal{C}^\alpha \subseteq \mathcal{R}^\alpha \subseteq \mathcal{C}^{c\alpha}$ implies $k\text{-Bary}\,\mathcal{C}^\alpha \subseteq k\text{-Bary}\,\mathcal{R}^\alpha \subseteq k\text{-Bary}\,\mathcal{C}^{c\alpha}$ for all $\alpha \ge 0$. □

4 Conclusions and Future Work

We have presented a nerve construction to capture the topology of the k-covered regions of a collection of well-behaved sets. Our focus was on guaranteeing the correct persistent homology, when the sets are filtrations, but it is also possible to consider the case of just a single good open cover $\mathcal{S}$. In that case, using a slightly stronger version of Lemma 3, it is possible to prove that the $k\text{-Bary}(\operatorname{Nerve}\mathcal{S})$ is homotopy equivalent to $\bigcup k\text{-Cover}\,\mathcal{S}$.

In practice, it is common to truncate Čech filtrations at some maximum scale to avoid the huge complexity blowup. The method of barycentric bifiltrations naturally adapts to this setting. In recent work, we proposed an alternative approach to controlling the complexity of distance-based filtrations using hierarchical net-trees [12]. It may be possible to combine those ideas with those presented in this paper to give sparse approximations of the (k, α)-offsets. This is the subject of future work.

5 Acknowledgements

This work was partially supported by the National Science Foundation under grant number CCF-1065106, by GIGA grant ANR-09-BLAN-0331-01, and by the European project CG-Learning No. 255827.

References

[1] G. Carlsson. Topology and data. *Bull. Amer. Math. Soc.*, 46:255–308, 2009.

[2] G. Carlsson, T. Ishkhanov, V. de Silva, and A. Zomorodian. On the local behavior of spaces of natural images. *International Journal of Computer Vision*, 76(1):1–12, 2008.

[3] F. Chazal, D. Cohen-Steiner, M. Glisse, L. J. Guibas, and S. Y. Oudot. Proximity of persistence modules and their diagrams. In *Proceedings of the 25th ACM Symposium on Computational Geometry*, pages 237–246, 2009.

[4] F. Chazal and A. Lieutier. The "λ-medial axis". *Graphical Models*, 67(4):304–331, 2005.

[5] F. Chazal and S. Y. Oudot. Towards persistence-based reconstruction in Euclidean spaces. In *Proceedings of the 24th ACM Symposium on Computational Geometry*, pages 232–241, 2008.

[6] V. de Silva and R. Ghrist. Coverage in sensor networks via persistent homology. *Algorithmic & Geometric Topology*, 7:339–358, 2007.

[7] V. de Silva and R. Ghrist. Homological sensor networks. *Notices Amer. Math. Soc.*, 54(1):10–17, 2007.

[8] T. K. Dey. *Curve and Surface Reconstruction : Algorithms with Mathematical Analysis*. Cambridge University Press, 2007.

[9] H. Edelsbrunner and J. L. Harer. *Computational Topology: An Introduction*. Amer. Math. Soc., 2009.

[10] J. R. Munkres. *Elements of Algebraic Topology*. Addison-Wesley, 1984.

[11] P. Niyogi, S. Smale, and S. Weinberger. A topological view of unsupervised learning from noisy data. 2008.

[12] D. R. Sheehy. Linear-size approximations to the vietoris-rips filtration. In *Proceedings of the 28th ACM Symposium on Computational Geometry*, 2012.

[13] A. Zomorodian and G. Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, 2005.

Index