Proceedings of the 30th Canadian Conference on Computational Geometry (CCCG 2018)

August 8-10, 2018
University of Manitoba
Winnipeg, Manitoba
Canada

Logo designed using Sliceform Studio App (https://www.sliceformstudio.com/)

Preface

This volume contains the proceedings of the 30th Canadian Conference on Computational Geometry (CCCG 2018), which took place on August 8–10, 2018, at the University of Manitoba, in Winnipeg, Manitoba, Canada. These proceedings will be made available electronically after the conclusion of the conference on the CCCG website: `http://www.cccg.ca/`.

We are grateful to the CCCG 2018 Program Committee and external reviewers, for their time and effort carefully reviewing all submissions. Each submission was reviewed by a minimum of three program committee members. The program committee accepted 46 papers out of 65 papers submitted. We thank the authors of all submitted papers and all conference attendees. We thank the invited speakers: Dr. Matthew (Matya) Katz (Paul Erdős Memorial Lecture), Dr. Marc van Kreveld (Ferran Hurtado Memorial Lecture), and Dr. Carola Wenk. In addition, we are grateful for the tremendous efforts of the CCCG 2018 Local Organizing Committee for their assistance; in particular, we would like to acknowledge Avery Miller and Victoria Harris.

We acknowledge the generous financial support from our sponsors: the Pacific Institute for the Mathematical Sciences (PIMS), Elsevier, the Fields Institute for Research in Mathematical Sciences, and the University of Manitoba.

Stephane Durocher
Shahin Kamali
CCCG 2018 Program Committee Co-Chairs

Sponsored by

Invited Speakers

Matthew J. Katz	Ben-Gurion University of the Negev
Marc van Kreveld	Utrecht University
Carola Wenk	Tulane University

Program Committee

David Bremner	University of New Brunswick
Jean-Lou De-Carufel	University of Ottawa
Stephane Durocher (Co-chair)	University of Manitoba
David Eppstein	University of California, Irvine
Will Evans	The University of British Columbia
Ruy Fabila-Monroy	Cinvestav-IPN
Zachary Friggstad	University of Alberta
Konstantinos Georgiou	Ryerson University
Ellen Gethner	University of Colorado Boulder
Shahin Kamali (Co-chair)	University of Manitoba
Akitoshi Kawamura	Kyushu University
Erik Krohn	University of Wisconsin Oshkosh
Jason S. Ku	Massachusetts Institute of Technology
Maarten Loffler	Utrecht University
Anna Lubiw	University of Waterloo
Tamara Mchedlidze	Karlsruhe Institute of Technology
Saeed Mehrabi	Carleton University
Debajyoti Mondal	University of Saskatchewan
Lata Narayanan	Concordia University
Michael Payne	Monash University
Zahed Rahmati	Amirkabir University of Technology
Don Sheehy	University of Connecticut
Bettina Speckmann	Eindhoven University of Technology
Andrew Winslow	University of Texas Rio Grande Valley

Additional Reviewers

Mohammad Ali Abam, Ahmed Abdelkader, Arash Ahadi, Carlos Alegría-Galicia, Aritra Banik, John Bowers, Guido Brueckner, Luis Evaristo Caraballo de La Cruz, Farah Chanchary, Mansoor Davoodi, Frank Duque, Mohammad Farshi, Kyle Fox, Kirk Gardner, Noah Giansiracusa, Barbara Gutiérrez, Aldo Guzmán-Sáenz, Carlos Hidalgo-Toscano, Ivor V.D. Hoog, Mahmoodreza Jahanseir, Mohammad Reza Kazemi, Kamyar Khodamoradi, Bernhard Kilgus, Chih-Hung Liu, Benjamin Niedermann, Tim Ophelders, Jeff Phillips, Marcel Roeloffzen, Erika Roldan, Rasoul Shahsavarifar, Naghmeh Shahverdi Zade Shargh, Willem Sonke, Frank Staals, Katharine Turner, Jérôme Urhausen, Mees van de Kerkhof, Marc Van Kreveld, Erik Jan van Leeuwen, Jordi L. Vermeulen, Lionov Wiratma, Matthias Wolf, Alexander Wolff

Local Organizers

Yeganeh Bahoo	Helen Cameron	Stephane Durocher (Co-chair)
Victoria Harris	Shahin Kamali	Avery Miller (Co-chair)
Jason Morrison	Sameer Naib	Nima Sheibani

(All at University of Manitoba)

Table of Contents

Wednesday, August 8

Paul Erdős Memorial Lecture

Session 1A

Session 1B

Session 2A

Session 2B

Session 4B

Session 5A

Session 5B

Session 6A

Geometric Problems and Structures Arising from the Study of Wireless Networks

Matthew J. Katz*

The study of wireless networks has motivated the formulation of interesting geometric optimization problems such as the power assignment problem, as well as the definition of new geometric data structures such as the bounded-angle spanning tree (related to networks with angular constraints) and the SINR diagram (induced by the Signal to Interference plus Noise Ratio equation). This talk will discuss some of these problems and structures, mentioning a few open problems along the way.

*Ben-Gurion University of the Negev, matya@cs.bgu.ac.il

Low Ply Drawings of Trees and 2-Trees

Michael T. Goodrich* Timothy Johnson†

Abstract

Ply number is a recently developed graph drawing metric inspired by studying road networks. Informally, for each vertex v, which is associated with a point in the plane, a disk is drawn centered on v with a radius that is α times the length of the longest edge incident to v, for some constant $\alpha \in (0, 0.5]$. The *ply number* is the maximum number of disks that overlap at a single point. We show that any tree with maximum degree Δ has a 1-ply drawing when $\alpha = O(1/\Delta)$. We also show that trees can be drawn with logarithmic ply number (for $\alpha = 0.5$), with an area that is polynomial for bounded-degree trees. Lastly, we show that this logarithmic upper bound does not apply to 2-trees, by giving a lower bound of $\Omega(\sqrt{n/\log n})$ ply.

1 Introduction

A useful paradigm for drawing graphs involves visualizing them as maps or road networks, allowing a visualizer to "zoom" in and out of the graph based on known techniques that apply to maps. For example, Gansner *et al.* [10] describe a GMap system for visualizing clusters in graphs as countries with nearby clusters drawn as neighboring countries. In addition, Nachmanson *et al.* [18, 19] describe a GraphMaps system for visualizing graphs as embedded road networks, so as to leverage the drawing and zooming capabilities of a roadmap viewer to explore the graph. Thus, a natural question arises as to which graphs are amenable to being drawn as road networks.

To answer this question, we formulate a precise definition of what we mean by a graph that could be drawn as a road network. One might at first suggest that graph planarity would be a good choice for such a formalism. But the class of planar graphs includes several graph instances that are difficult to visualize as road networks, such as the so-called "nested triangles" graph (e.g., see [6, 9, 12]). In addition, as shown by Eppstein and Goodrich [7], the class of planar graphs is not general enough to include all real-world road networks, as road networks are often not planar. For example, the California highway system alone has over 6,000 crossings. Instead of using planarity, then, Eppstein and Goodrich [7] introduce the concept of the *ply number* of an embedded graph, and they demonstrate experimentally that real-world road networks tend to have small ply. Intuitively, the ply concept tries to capture how road networks have features that are well-separated at multiple scales. The formal definition of the ply number of a graph is derived from the definition of ply for a set of disks (which captures the depth of coverage for such a set of disks) [17]; hence, the ply number of an embedded graph is defined in terms of the ply of a set of disks defined with respect to this embedding.

Let us therefore formally define the *ply number* of an embedded geometric graph. Let Γ be a straight-line drawing of a graph G. For every vertex $v \in G$, let C_v^α be the open disk centered at v and whose radius r_v^α is α times the length of the longest edge incident to v. The set of ply disks containing a point q is then $S_q^\alpha = \{C_v^\alpha \mid \|v - q\| < r_v^\alpha\}$. The *$\alpha$-ply number* of this drawing is defined as

$$\mathrm{pn}(\Gamma) = \max_{q \in \mathbb{R}^2} \|S_q^\alpha\|.$$

Usually, α is chosen in the range $(0, 0.5]$. In this range, a graph with two vertices and a single edge connecting them has ply number 1, because the ply disks for the two vertices will not overlap. If not otherwise specified, the default value for α is 0.5, and if the value of α is taken as this default value or known from the context, then we refer to the α-ply number simply as the ply number.

Previous related work. As an empirical justification of the use of ply numbers, De Luca *et al.*'s experimental study [4] found that some force-directed algorithms, including Kamada-Kawai [16], stress majorization [11], and the fast multipole method [13] all tend to produce drawings with low ply number. Their experiments also suggest that trees with at most three children per node can have unbounded ply number.

The problem of drawing graphs with ply number equal to 1 is related to that of constructing circle-contact representations. A circle-contact representation for a graph is a collection of interior-disjoint circles, in which each circle represents a single vertex, and two vertices are adjacent if and only if their circles are tangent to one another [14, 15]. Di Giacomo *et al.* [5] show that graphs with ply number 1 are equivalent to graphs

*Dept. of Computer Science, University of California, Irvine, goodrich@uci.edu

†Dept. of Computer Science, University of California, Irvine, tujohnso@uci.edu

with weak unit disk contact representations, which are known to be NP-hard to recognize [3]. They also show that binary trees have drawings with ply number 2, or with ply number 1 when α is reduced to 1/3. One such drawing is reproduced in the appendix in Figure 10.

Angelini *et al.* [2] relax our definition of ply number to define the *vertex-ply* of a drawing, which is the maximum number of intersecting disks at any vertex of the drawing. Graphs with vertex-ply number 1 can then be interpreted as a new variant of proximity drawings.

In an earlier paper, Angelini *et al.* [1] show that 10-ary trees have unbounded ply number. Furthermore, they prove that 5-ary trees can be drawn with logarithmic ply number and polynomial area. The ply number of drawings of trees with between three and nine children per node remains an interesting and surprisingly daunting open problem.

Our results. In this paper, we study a number of related problems concerning low-ply drawings of bounded-degree trees. We first answer an open question proposed by Di Giacomo *et al.* [5], which asks whether all trees with maximum degree Δ have 1-ply drawings for a sufficiently small α. We show in Section 2 that a simple fractal drawing pattern can achieve this when $\alpha = O(1/\Delta)$.

In Section 3, we show that all trees (not just 5-ary trees) can be drawn with logarithmic ply number, for $\alpha = 0.5$. Furthermore, the area is polynomial for trees with bounded degree. These results depend on some careful arguments about geometric configurations and fractal-like geometric constructions, as well as yet another use of the heavy-path decomposition technique of Sleator and Tarjan [20].

It is then natural to consider whether any planar graph classes larger than trees can be drawn with logarithmic ply number for $\alpha = 0.5$. In Section 4, we show that this is not the case for 2-trees, by constructing a family of 2-trees that require a ply number of $\Omega(\sqrt{n/\log n})$. Previous lower bounds have only applied for planar drawings, while non-planar drawings are known to sometimes have better ply number.

2 1-ply Drawings

In this section, we provide conditions on α and related constructions for producing 1-ply drawings of trees of any bounded degree. At a high level, our drawings are constructed as follows. For a tree with maximum degree Δ, we divide the area around each parent vertex radially into Δ equal wedges. Then we draw one subtree inside each wedge. The distance from each node to its children is chosen to be a constant fraction f of its distance from its own parent.

This produces a drawing that is highly symmetric, in a fashion that would produce a fractal if continued in the limit.[1] Thus, any bounded-degree tree is a subtree of this infinite tree; hence, this drawing algorithm can produce a drawing of any bounded-degree tree. Filling in the details of this construction requires setting the values of two parameters: f, the ratio between outgoing and incoming edge lengths; and α, the ratio between the radius of a ply disk for a vertex and the length of its longest incident edge. We provide constraints for the following three cases, which taken together ensure that there are no overlaps, so that the ply number of our drawings is 1. We then maximize α such that all of these constraints are satisfied.

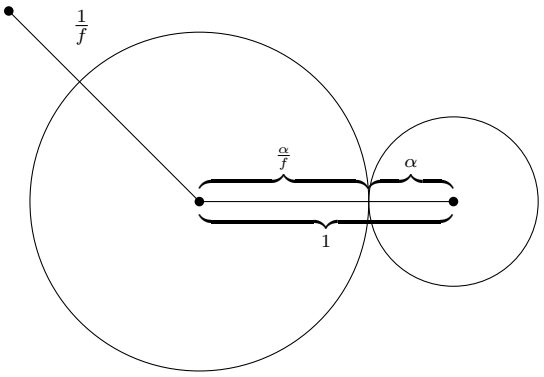


Figure 1: Our edges decrease by a factor of f at each level, and the ply disks have radius α times the length of the incoming edge.

1. Ply disks for adjacent vertices must not overlap.

2. Ply disks for vertices on separate subtrees must not overlap.

3. A ply disk for a vertex must never overlap a ply disk for one of its descendants.

It is easily verified that these three conditions are necessary and sufficient for a tree to have a 1-ply drawing.

Condition 1: Separate adjacent vertices. Except for the root vertex, which has no incoming edge, we proportion the lengths of the edges for each vertex as shown in Figure 1.

That is, taking the length of a reference edge as 1 (illustrated in Figure 1 going from parent to child in a left-to-right orientation), then, based on our definition of the α-ply number, the radius of the larger circle is α/f, the radius of the smaller circle is α, and their distance is 1. Thus, we have our first condition relating α and f:

$$\alpha \leq \frac{f}{1+f}. \tag{1}$$

[1] See Falconer [8] for further reading about fractal geometry.

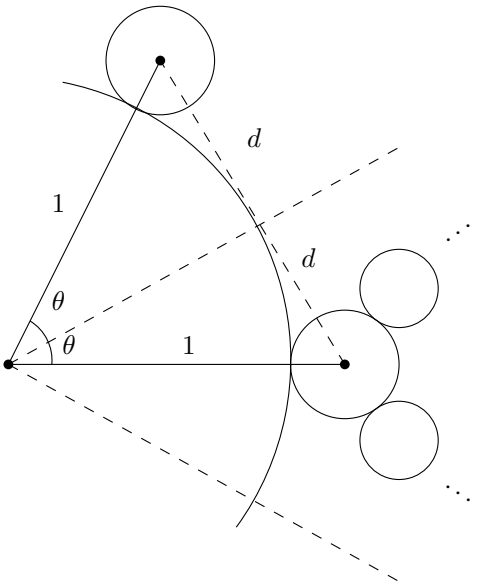


Figure 2: Our default constraint on a wedge containing a subtree of a central vertex.

Condition 2: Separate subtrees with the same root. We require that the ply disks for any subtree all be contained within a wedge of angle $\theta = \frac{2\pi}{\Delta}$ around its parent vertex, where Δ is the degree. Since our wedges for each subtree are disjoint, this ensures that the ply disks for two adjacent subtrees cannot overlap.

As illustrated in Figure 2, the distance from a child vertex to the boundary of its containing wedge is $d = \sin\left(\frac{\pi}{\Delta}\right)$. Note also that the lengths of edges along a path in this subtree form a geometric sequence with ratio f. So the maximum distance from a child vertex to any vertex in its subtree is $\sum_{i=1}^{\infty} f^i = \frac{f}{1-f}$.

Therefore, to confine each subtree within its wedge, we must set

$$\frac{f}{1-f} \leq \sin\left(\frac{\pi}{\Delta}\right).$$

Solving for f, we get

$$f \leq \frac{\sin\left(\frac{\pi}{\Delta}\right)}{1+\sin\left(\frac{\pi}{\Delta}\right)}. \tag{2}$$

Condition 3: Separate each vertex from its descendants. Our last condition is that the ply disk for a vertex cannot overlap any of its descendants. The closest descendants will be those in the wedges on either side of the edge between their parent and grandparent, which are at an angle of $\frac{2\pi}{\Delta}$ from their parent, as in Figure 3.

Normalizing a grandparent-to-parent edge, (u, v), as having length 1 and performing a rigid transformation that takes the grandparent, u, to the origin so that the edge (u, v) is along the x-axis, u's closest grandchild, which we call w, is located at the point

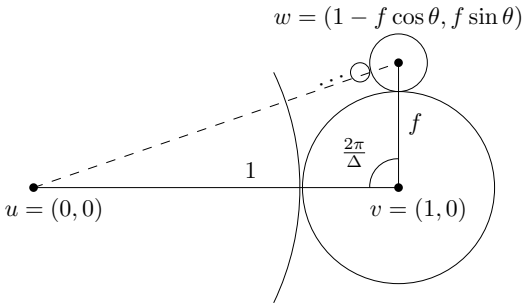


Figure 3: Our layout leaves a gap of angle $\theta = \frac{2\pi}{\Delta}$ for the edge from the parent vertex. The descendants on either side must not be able to overlap their ancestors.

$\left(1 - f\cos\left(\frac{2\pi}{\Delta}\right), f\sin\left(\frac{2\pi}{\Delta}\right)\right)$. We require that the distance from w to its descendants be no greater than the distance from w to the boundary of the ply disk for u. Recall that our wedge angle $\theta = \frac{2\pi}{\Delta}$. We apply the following constraint:

$$\sqrt{(1 - f\cos\theta)^2 + (f\sin\theta)^2} \geq \frac{\alpha}{f} + \sum_{i=2}^{\infty} f^i$$

After simplifying and solving for α, our condition is

$$\alpha \leq f\sqrt{1 - 2f\cos\theta + f^2} - \frac{f^3}{1-f}$$

Let us now compare our three conditions. We see that equation 2 gives us an upper bound for f, while equations 1 and 3 give us upper bounds for α that both increase as f gets larger. So to maximize α, we let f be equal to its upper bound. This gives us the following theorem, and a corollary that is proved in the appendix.

Theorem 1 *Let T be a tree with maximum degree Δ, and let*

$$f = \frac{\sin\left(\frac{\pi}{\Delta}\right)}{1+\sin\left(\frac{\pi}{\Delta}\right)}.$$

T has a 1-ply drawing if

$$\alpha \leq \min\left(\frac{f}{1+f},\ f\sqrt{1 - 2f\cos(2\pi/\Delta) + f^2} - \frac{f^3}{1-f}\right).$$

Corollary 2 *A tree with maximum degree Δ has a 1 ply drawing when $\alpha = O(1/\Delta)$.*

Note, however, that some of our conditions are not tight. For condition 2, we assumed that the branches of our subtrees would approach the sides of their wedge directly. But when the degree of our tree is 4, the angle between two subtrees is 90°. Therefore, every edge in our tree is either horizontal or vertical, so we can measure the distance to the boundary of the wedge using Manhattan distance instead of Euclidean distance. (See Figure 4.)

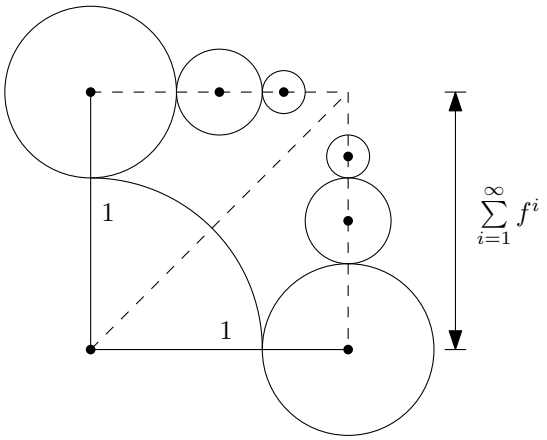


Figure 4: An improved bound for Condition 2. The Manhattan distance is sufficient to confine subtrees within a wedge when all edges are either horizontal or vertical.

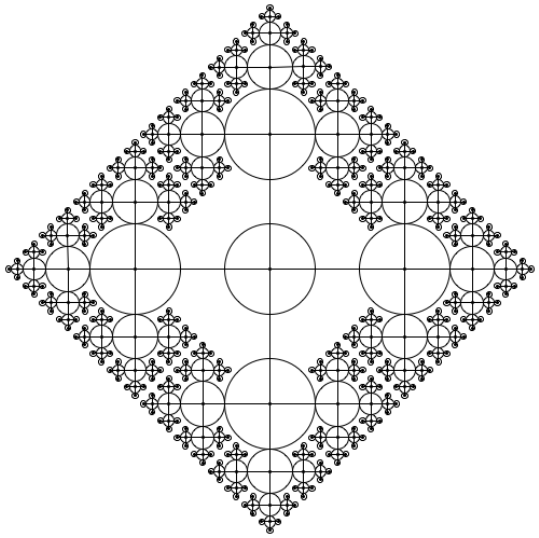


Figure 5: A 1-ply drawing of a tree with maximum degree four, for which $f = 1/2$, $\alpha = 1/3$.

So for a tree with degree 4, we replace condition 2 with $\sum_{i=1}^{\infty} f^i \leq 1$. This implies $f = 1/2$, and our other conditions imply $\alpha = 1/3$. In this case, our bound is tight. (See Figure 5.)

3 Polynomial area, logarithmic ply number

In this section, we prove the following theorem.

Theorem 3 *For $\alpha = 0.5$, a tree with maximum degree Δ can be drawn with ply number $O(\log n)$ in area $n^{O(\Delta)}$.*

Note that for a bounded-degree tree, Δ is a constant, so our area is polynomial in n. We first give a simple fractal layering algorithm that proves our theorem for balanced trees. Then we extend it to all trees by using a heavy path decomposition. A similar approach was used by Angelini *et. al.* [1] for drawing trees up to maximum degree six, but we add our layering technique to make their algorithm work for all trees.

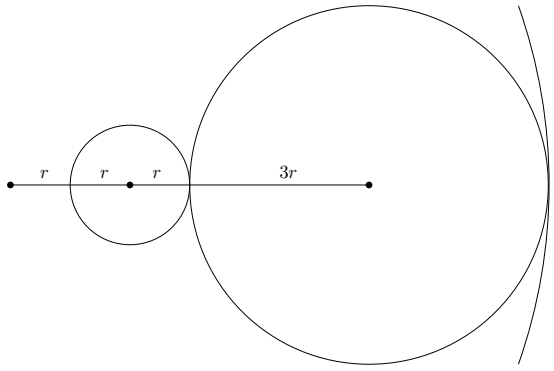


Figure 6: If each layer in a tree drawing is at least three times as far as the previous layer, the ply disks for the layers will not overlap. In this figure, $d_1 = 2r$ and $d_2 = 6r$, so our condition holds.

Radially layered drawings. We begin with a simple algorithm for drawing trees by layering their children. For each vertex, we choose a sequence of distances d_i for the layers, such that vertices in adjacent layers have disjoint ply disks.

Lemma 4 *Suppose that r is the root of a star graph. Let v_1, v_2 be children at distances d_1, d_2, respectively. If $d_2 \geq 3d_1$, then the ply disks for v_1 and v_2 are disjoint.*

Proof. The distance to v_1 is d_1, so its ply disk will have radius $0.5d_1$, and will be contained within an open disk of radius $1.5d_1$ centered at r. The distance to v_2 is d_2, so its ply disk will have radius $0.5d_2$. Its closest approach to r will be at distance $0.5d_2 \geq 1.5d_1$. Thus, the ply disks for v_1 and v_2 are disjoint. (See Figure 6.) □

Next, note that we can put up to six vertices in each layer without overlaps. So for a tree with degree Δ, we need $\lceil \Delta/6 \rceil$ layers. We pick any desired size for the initial layer around our root, then draw the subtrees for each child vertex recursively within their own ply disks. Therefore, the size of the smallest layer must shrink by a factor of $3^{\lceil \Delta/6 \rceil}$ each time we add a level to our tree.

Since our tree is balanced, its total height is $O(\log n)$. Thus the ratio of the longest to the smallest edge is $3^{O(\Delta \log n)} = n^{O(\Delta)}$. The area will then also be $n^{O(\Delta)}$, for a larger constant.

This completes our proof for balanced trees. Figure 11 in the appendix provides an example drawing of a tree with degree 18 using three layers.

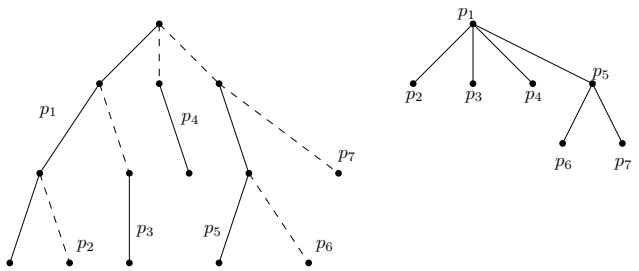


Figure 7: A tree and its heavy path decomposition.

Heavy path decomposition. When our trees are not balanced, we will use the heavy path decomposition [20] to still produce drawings with logarithmic ply number. This decomposition partitions the vertices in our tree into paths that each end at a leaf. To choose the first path, we begin at the root. Then from its child subtrees, we choose the largest one and add its root to our path. We continue downward until we reach a leaf.

We next remove the vertices on this path from our tree, creating a new set of subtrees, and repeat the same process for each subtree. That is, the root vertex for each of these subtrees will become the starting point for a new path constructed by the same process. We recurse until every vertex in our tree is assigned to some path. The subtrees that are rooted at a child of a vertex v and whose root is not on the same path as v are are said to be *anchored* at v. The path containing the root of each of those subtrees is also said to be anchored at v.

The set of paths constructed by this process now itself forms a new tree (see Figure 7), in which the path P_i is a parent of P_j if one of the vertices in P_i is an anchor for P_j. We will show that the ply number of our drawings is proportional to the height of this decomposition tree, which is known to be $O(\log n)$.

Now we describe how to draw each path in the decomposition tree. First, we define a *2-drawing* of a path $P = (v_1, \ldots, v_m)$ as a straight-line drawing of P along a single segment that satisfies the following properties.

- All of the vertices appear in the line segment in the same order as they appear in P.
- For each $i = 2, \ldots, m-1$ we have $\frac{l(v_{i-1},v_i)}{2} \leq l(v_i, v_{i+1}) \leq 2l(v_{i-1}, v_i)$.

Lemma 5 *A 2-drawing of a path has ply number at most 2.*

Proof. See Lemma 5 in Angelini *et al.* [1]. □

Now suppose that we have a path $P = (v_1, v_2, \ldots, v_k)$ in our heavy path decomposition, and let P be anchored at vertex v, so that v is the parent of v_1. Let n be the total size of the subtrees anchored at v, and let n_i be the total size of the subtrees anchored at v_i (Figure 8). Lastly, we denote the length of the edge (v, w) as $l(v, w)$.

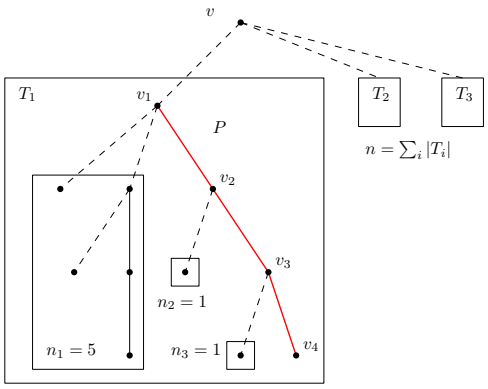


Figure 8: Labels for different sizes in a heavy path decomposition tree.

Intuitively, we want to draw each path so that more space is available for vertices that have larger subtrees. At the same time, we want to ensure that the lengths of the two edges for a vertex are within a factor of two, so that our path is a 2-drawing. This can be achieved using the following algorithm DrawPath.

To draw the path P, we first set $l(v, v_1) = n_1$ and $l(v_i, v_{i+1}) = n_i + n_{i+1}$, for each $i = 1, \ldots, k-1$. Next we visit the edges of our path in decreasing order of length. When an edge (v_i, v_{i+1}) is visited, we make sure that both of its neighboring edges are at least half as long. That is, we set:

- $l(v_{i-1}, v_i) = \max\{\frac{l(v_i,v_{i+1}}{2}, l(v_{i-1}, v_i)\}$
- $l(v_{i+1}, v_{i+2}) = \max\{\frac{l(v_i,v_{i+1}}{2}, l(v_{i+1}, v_{i+2})\}$

Lemma 6 *The algorithm* DrawPath *constructs a 2-drawing Γ of P such that $l(v, v_1) \geq n_1$, $l(v_i, v_{i+1}) \geq n_i + n_{i+1}$, and for each $i = 1, \ldots, m-1$, and $l(P) \leq 6n$.*

Proof. See Lemma 6 in Angelini *et al.* [1]. □

We now perform a bottom-up construction of our tree, drawing each path using the DrawPath algorithm. Once all of the paths anchored at vertices in P have been drawn, we construct a drawing of P with each path in a separate layer (Figure 9). This translation may increase the ply radius of the first vertex in each of these paths, so the ply number of the drawing for each path may increase from 2 to 3.

The drawing is described in more detail in the appendix, and the following properties are shown.

Lemma 7 *For each vertex v we can associate a drawing disk D_v (which is distinct from the ply disk for v) that satisfies the following properties.*

1. *If v, w are two distinct vertices on the same path, then their disks D_v, D_w are disjoint.*

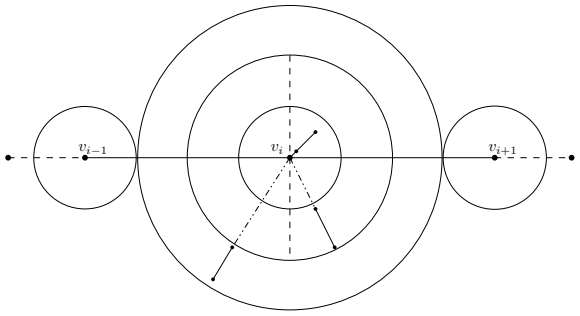


Figure 9: Three vertices along a path in our decomposition, along with their drawing disks (not the ply disks). For the center vertex v_i, we show three paths in different layers around it, which would be drawn recursively.

2. *The ply disks for the subtrees anchored at v are all contained within D_v, and are within disjoint layers.*

3. *Each path is scaled by a factor of $O(3^{\Delta})$ larger than the paths that are anchored at its vertices.*

Together, these properties imply that the ply disks for a path can only overlap with ply disks for their ancestor paths in the heavy path decomposition tree. Therefore, since each path is drawn with ply number at most 3, the total ply number is at most $3(h+1)$, where h is the height of the heavy path decomposition tree. Since $h = O(\log n)$, the ply number is $O(\log n)$.

Lastly, if Δ is a constant, then the total scaling for our largest disk is $3^{O(\Delta \log n)}$, which simplifies to $n^{O(\Delta)}$. This completes our proof of Theorem 4.

4 Lower bound for 2-trees

Since all trees can be drawn with $O(\log n)$ ply number, it is natural to consider larger planar graph classes. We show that a 2-tree can require at least $\Omega(\sqrt{n/\log n})$ ply.

We know that a star can be drawn with ply number 2 when the distance to successive vertices increases exponentially [1]. A tree can be drawn with $O(\log n)$ ply number when the distances from parents to their children decrease exponentially as we move down the tree. Intuitively, combining these two graphs produces a graph that requires large ply, since it is impossible to satisfy both conditions simultaneously.

Accordingly, we begin with m disjoint complete binary trees of height h, which we label T_i, $1 \le i \le m$, where m and h will be determined later. Then we add one vertex v connected to every vertex in each tree. Let $d(w)$ be the distance from v to w, for any tree vertex w.

Suppose that w_1 is a tree vertex in the tree T_i, and w_2 is its child. Then if $d(w_2) > 3d(w_1)$, the ply radius for w_1 is larger than $d(w_1)$. Therefore, the ply disk for w_1 contains v. Similarly, if $d(w_1) > 3d(w_2)$, then the ply disk for w_2 contains v. Assume without loss of generality that the distance from v to the root of T_i is 1. We can then show by induction that if no ply disk in T_i contains v, then the nodes at the jth level of our tree are at distance at most 3^j from v, and at least 3^{-j}.

Next suppose that some tree T_i has no vertices whose ply disk contains v. Then partition our drawing into annuli S_l, where the inner radius of S_l is 3^l, and the outer radius is 3^{l+1}. Next choose $\bar{l}$ to be the index of the annulus containing the maximum number of vertices. $S_{\bar{l}}$ must contain at least $2^h/2h$ vertices, each with a ply radius at least $3^{\bar{l}}/2$.

Let D be the disk centered at v with a radius of $r_D = 3^{\bar{l}+2}/2$, so that all of the ply disks for vertices in $S_{\bar{l}}$ are contained in D. Now we compute the ratio of the areas of the ply disks in D to its own area, which is a lower bound for the ply number. Note that D contains $2^h/2h$ ply disks that each have a radius of at least $r_D/9$. Therefore, this ratio is at least:

$$\underbrace{\frac{2^h}{2h}}_{\text{vertices}} \overbrace{\frac{\pi r_D^2}{81}}^{\text{ply area per vertex}} \underbrace{\frac{1}{\pi r_D^2}}_{\text{inverse disk area}} = \frac{2^h}{162h} = \Omega(2^h/h)$$

Now let $h = (\log n + \log\log n)/2$, and let $m = \sqrt{n/\log n}$. Note that the total number of vertices in each tree is $2^{(\log n + \log\log n)/2} = \sqrt{n \log n}$. The total number of vertices overall is then $m \cdot 2^h + 1 = O(n)$.

If every tree T_i has a vertex whose ply disk contains v, then the ply number is at least $\Omega(m) = \Omega(\sqrt{n/\log n})$. Otherwise, if some tree does not have such a vertex, then that tree's ply number is $\Omega(2^h/h) = \Omega(\sqrt{n/\log n})$. This gives us the following theorem.

Theorem 8 *There is a 2-tree with $O(n)$ vertices for which any drawing has ply number $\Omega(\sqrt{n/\log n})$.*

5 Conclusion

We have shown that all trees have 1-ply drawings when $\alpha = O(1/\Delta)$, or logarithmic ply number when $\alpha = 0.5$, and that 2-trees may require $\Omega(\sqrt{n/\log n})$ ply.

There are many open questions left to resolve, but we are especially interested in closing the gap between constant and logarithmic ply for trees with between three and nine children per node. We would also like to consider intermediate planar graph classes between trees and 2-trees, such as outerplanar graphs, and determine whether they can be drawn with $O(\log n)$ ply.

Acknowledgements

This research was supported by DARPA agreement no. AFRL FA8750-15-2-0092 and NSF grants 1526631 and 1815073. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

[1] Patrizio Angelini, Michael A Bekos, Till Bruckdorfer, Jaroslav Hančl, Michael Kaufmann, Stephen Kobourov, Antonios Symvonis, and Pavel Valtr. Low ply drawings of trees. In *International Symposium on Graph Drawing and Network Visualization*, pages 236–248. Springer, 2016.

[2] Patrizio Angelini, Steve Chaplick, Felice De Luca, Jiri Fiala, Jan Hancl Jr, Niklas Heinsohn, Michael Kaufmann, Stephen Kobourov, Jan Kratochvil, and Pavel Valtr. On vertex-and empty-ply proximity drawings. *arXiv preprint arXiv:1708.09233*, 2017.

[3] Heinz Breu and David G Kirkpatrick. Unit disk graph recognition is NP-hard. *Computational Geometry*, 9(1-2):3–24, 1998.

[4] F. De Luca, E. Di Giacomo, W. Didimo, S. Kobourov, and G. Liotta. An experimental study on the ply number of straight-line drawings. In *International Workshop on Algorithms and Computation (to appear)*. Springer, 2017.

[5] Emilio Di Giacomo, Walter Didimo, Seok-hee Hong, Michael Kaufmann, Stephen G Kobourov, Giuseppe Liotta, Kazuo Misue, Antonios Symvonis, and Hsu-Chun Yen. Low ply graph drawing. In *Information, Intelligence, Systems and Applications (IISA), 2015 6th International Conference on*, pages 1–6. IEEE, 2015.

[6] Christian A. Duncan, David Eppstein, Michael T. Goodrich, Stephen G. Kobourov, and Martin Nllenburg. Lombardi drawings of graphs. *Journal of Graph Algorithms and Applications*, 16(1):85–108, 2012.

[7] David Eppstein and Michael T Goodrich. Studying (non-planar) road networks through an algorithmic lens. In *Proceedings of the 16th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 16. ACM, 2008.

[8] Kenneth Falconer. *Fractal Geometry: Mathematical Foundations and Applications*. John Wiley & Sons, 2004.

[9] Fabrizio Frati and Maurizio Patrignani. A note on minimum-area straight-line drawings of planar graphs. In Seok-Hee Hong, Takao Nishizeki, and Wu Quan, editors, *15th Int. Symp. on Graph Drawing (GD)*, pages 339–344, 2008.

[10] Emden R Gansner, Yifan Hu, and Stephen Kobourov. Gmap: Visualizing graphs and clusters as maps. In *Visualization Symposium (PacificVis), 2010 IEEE Pacific*, pages 201–208. IEEE, 2010.

[11] Emden R Gansner, Yehuda Koren, and Stephen North. Graph drawing by stress majorization. In *International Symposium on Graph Drawing, LNCS 3383*, pages 239–250. Springer, 2004.

[12] Ashim Garg and Roberto Tamassia. Planar drawings and angular resolution: Algorithms and bounds. In Jan van Leeuwen, editor, *2nd European Symp. on Algorithms (ESA)*, pages 12–23, 1994.

[13] Stefan Hachul and Michael Jünger. Drawing large graphs with a potential-field-based multilevel algorithm. In *International Symposium on Graph Drawing, LNCS 3383*, pages 285–295. Springer, 2004.

[14] Petr Hliněnỳ. Contact graphs of curves. In *International Symposium on Graph Drawing*, pages 312–323. Springer, 1995.

[15] Petr Hliněnỳ. Classes and recognition of curve contact graphs. *Journal of Combinatorial Theory, Series B*, 74(1):87–103, 1998.

[16] Tomihisa Kamada and Satoru Kawai. An algorithm for drawing general undirected graphs. *Information Processing Letters*, 31(1):7–15, 1989.

[17] Gary L. Miller, Shang-Hua Teng, William Thurston, and Stephen A. Vavasis. Geometric separators for finite-element meshes. *SIAM Journal on Scientific Computing*, 19(2):364–386, 1998.

[18] Debajyoti Mondal and Lev Nachmanson. A new approach to graphmaps, a system browsing large graphs as interactive maps. *arXiv preprint arXiv:1705.05479*, 2017.

[19] Lev Nachmanson, Roman Prutkin, Bongshin Lee, Nathalie Henry Riche, Alexander E Holroyd, and Xiaoji Chen. Graphmaps: Browsing large graphs as interactive maps. In *International Symposium on Graph Drawing and Network Visualization*, pages 3–15. Springer, 2015.

[20] Daniel D Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *Journal of Computer and System Sciences*, 26(3):362–391, 1983.

Appendix

Here we include additional proofs and figures that were postponed from the main paper due to lack of space.

Corollary 9 *A tree with maximum degree Δ has a 1 ply drawing when $\alpha = O(1/\Delta)$.*

Proof. First, recall that we defined:

$$f = \frac{\sin\left(\frac{\pi}{\Delta}\right)}{1 + \sin\left(\frac{\pi}{\Delta}\right)}.$$

Now we will consider the limiting value of $\Delta \cdot f$.

$$\lim_{\Delta\to\infty} \Delta \cdot f = \lim_{\Delta\to\infty} \frac{\Delta \sin(\pi/\Delta)}{1 + \sin(\pi/\Delta)} = \pi$$

Therefore, $f = \Theta(1/\Delta)$. So as $\Delta \to \infty$, $f \to 0$.
Secondly, recall that in our theorem we showed:

$$\alpha \leq \min\left(\frac{f}{1+f},\ f\sqrt{1 - 2f\cos(2\pi/\Delta) + f^2} - \frac{f^3}{1-f}\right)$$

Suppose that we use the first condition, $\alpha = f/(1+f)$. Then $\alpha/f = 1/(1+f)$. So $\lim_{f\to 0} \alpha/f = 1$.
Then suppose that we use the second condition:

$$\alpha = f\sqrt{1 - 2f\cos(2\pi/\Delta) + f^2} - \frac{f^3}{1-f}$$

Again, $\lim_{f\to 0} \alpha/f = 1$, so $\alpha = O(f) = O(1/\Delta)$. □

Lemma 10 *For each vertex v we can associate a drawing disk D_v (which is distinct from the ply disk for v) that satisfies the following properties.*

1. *If v, w are two distinct vertices on the same path, then their disks D_v, D_w are disjoint.*
2. *The ply disks for the subtrees anchored at v are all contained within D_v, and are within disjoint layers.*
3. *Each path is scaled by a factor of $O(3^\Delta)$ larger than the paths that are anchored at its vertices.*

Proof. We prove each part of our lemma as follows.

1. Suppose that our heavy path decomposition tree has a total height of H, and the path P is at height h. Then we use the DrawPath algorithm to construct a drawing of P. We set the drawing disk for a vertex v_i in P to have radius n_i, that is, the size of the subtrees anchored at v_i. Since the length of the edge (v_i, v_{i+1}) is at least $n_i + n_{i+1}$ (by Lemma 7), the drawing disks for any two adjacent vertices in our path will not overlap.
2. Next we scale the drawing of P by $3^{\Delta(H-h)}$. Note that each path anchored at a vertex in P is scaled by $3^{\Delta(H-(h+1))}$, so the difference in the scaling factor is 3^Δ. We show that at least $\Delta - 1$ paths can be anchored in different layers around each vertex v in P.

   From Lemma 7, we know that each path anchored at v has an unscaled length of at most $6n$, where n is the total size of the subtrees anchored at v. We also know by Lemma 5 that the ply disks for vertices in two different paths will not overlap if their distance from v differs by at least a factor of three.

   So we will draw the jth path anchored at v_i is drawn between x_j and x_{j+1}, where x_j satisfies the following recurrence:

$$x_1 = 6n_i$$
$$x_i = 3x_{i-1} + 6n_i$$

   Solving the recurrence, we find that $x_j = 3n(3^j - 1)$. Since we have at most $\Delta - 1$ layers, the largest layer will have an outer radius less than $3^\Delta n_i$. Since the unscaled drawing disk for v_i had a radius of n_i, a relative scaling factor of 3^Δ is sufficient to fit the paths that are anchored at it.
3. Since our heavy path decomposition has height $O(\log n)$, the largest path will be scaled by a factor of $3^{O(\Delta \log n)}$ from its original length of $O(n)$. So the diameter of our drawing is $3^{O(\Delta \log n)}n$, which simplifies to $n^{O(\Delta)}$. The total area is then also $n^{O(\Delta)}$, for a larger constant.

□

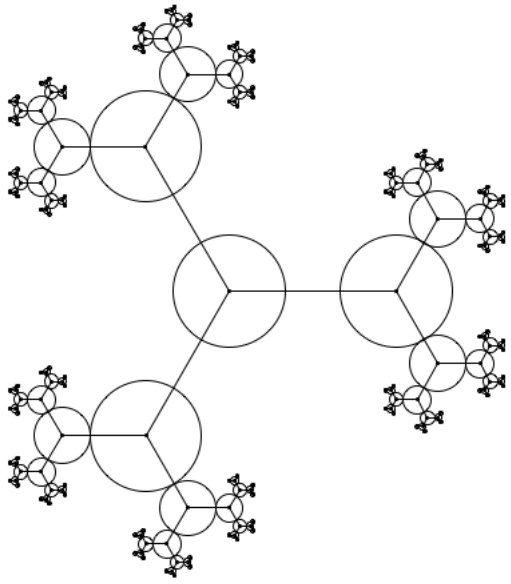


Figure 10: Di Giacomo *et al.*'s drawing of a binary tree with α-ply number 1, for $\alpha = \frac{1}{3}$. The edge lengths decrease by a factor of 2 at each level.

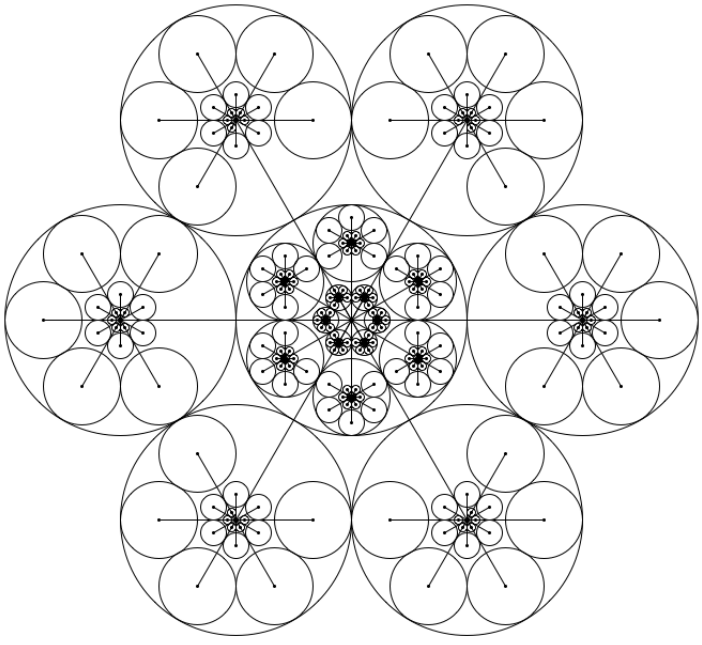


Figure 11: A tree with degree 18, where the children of each vertex are drawn in three layers.

The Crossing Number of Semi-Pair-Shellable Drawings of Complete Graphs

Petra Mutzel * Lutz Oettershagen †

Abstract

The Harary-Hill Conjecture states that for $n \geq 3$ every drawing of K_n has at least

$$H(n) := \frac{1}{4}\left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{n-1}{2} \right\rfloor \left\lfloor \frac{n-2}{2} \right\rfloor \left\lfloor \frac{n-3}{2} \right\rfloor$$

crossings. In general the problem remains unsolved, however there has been some success in proving the conjecture for restricted classes of drawings. The most recent and most general of these classes is seq-shellability [16]. In this work, we improve these results and introduce the new class of *semi-pair-shellable* drawings. We show that each drawing in this new class has at least $H(n)$ crossings using novel results on k-edges. So far, approaches for proving the Harary-Hill Conjecture for specific classes rely on a fixed reference face. We successfully apply new techniques in order to loosen this restriction, which enables us to select different reference faces when considering subdrawings. Furthermore, we introduce the notion of *k-deviations* as the difference between an optimal and the actual number of k-edges. Using k-deviations, we gain interesting insights into the essence of k-edges, and we further relax the necessity of fixed reference faces.

1 Introduction

The crossing number $cr(G)$ of a graph G is the smallest number of edge crossings over all possible drawings of G. In a drawing D of $G = (V, E)$ every vertex $v \in V$ is represented by a point and every edge $uv \in E$ with $u, v \in V$ is represented by a simple curve connecting the corresponding points of u and v. We call an intersection point of the interior of two edges a crossing. The Harary-Hill Conjecture states the following.

Conjecture 1 (Harary-Hill [10]) *Let K_n be the complete graph with n vertices, then*

$$cr(K_n) = H(n) := \frac{1}{4}\left\lfloor \frac{n}{2} \right\rfloor \left\lfloor \frac{n-1}{2} \right\rfloor \left\lfloor \frac{n-2}{2} \right\rfloor \left\lfloor \frac{n-3}{2} \right\rfloor$$

There are construction methods for drawings of K_n that lead to exactly $H(n)$ crossings, for example the class of *cylindrical* drawings first described by Hill [11]. However, there is no proof for the lower bound of the conjecture for arbitrary drawings of K_n with $n \geq 13$. The cases for $n \leq 10$ have been shown by Guy [10] and for $n = 11$ by Pan and Richter [17]. Guy [10] argues that $cr(K_{2n+1}) \geq H(2n+1)$ implies $cr(K_{2(n+1)}) \geq H(2(n+1))$, hence $cr(K_{12}) \geq H(12)$. McQuillan et al. [14] showed that $cr(K_{13}) \geq 219$. Ábrego et al. [1] improved the result to $cr(K_{13}) \in \{223, 225\}$.

Beside these results for arbitrary drawings, there has been success in proving the Harary-Hill Conjecture for different classes of drawings. So far, the conjecture has been verified for 2-page-book [3], cylindrical [5], x-monotone [8, 4], x-bounded [5], shellable [5], bishellable [2] and recently seq-shellable drawings [16]. Seq-shellability is the broadest of the beforehand mentioned classes comprising the others. Here, the proof of the Harary-Hill Conjecture makes use of the concept of *k-edges*. Each edge $e \in E$ in a drawing is assigned a specific value between 0 and $\lfloor \frac{n}{2} \rfloor - 1$ with respect to a fixed reference face. The edge e separates the remaining $n-2$ to vertices into two distinct sets, and is assigned the cardinality k of the smaller of the two sets, i.e. is a k-edge (see section 2 for details). We can express the number of crossings in a drawing in terms of the numbers of k-edges for each $k \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor - 2\}$. Therefore, having lower bounds on the (cumulated) number of k-edges implies a lower bound on the crossing number of a drawing. After two cumulations, we obtain *double cumulated k-edges*. However, the possibilities of their usage for further improvements to new classes of drawings seem to be limited.

Our contribution and outline In this work, we resolve the limitations of double cumulated k-edges by applying two new ideas. Firstly, instead of double cumulated k-edges we utilize *triple cumulated* k-edges. Balko et al. introduced these in [8]. Secondly, so far all classes, including seq-shellability, depend on a *globally fixed* reference face. We call a reference face globally fixed if we do not allow to select a different one when considering subdrawings, which constitutes a strong limitation in the proofs. In this work, we show that under certain conditions and/or assumptions, we are able to change

*Department of Computer Science, TU Dortmund University petra.mutzel@tu-dortmund.de

†Department of Computer Science, TU Dortmund University lutz.oettershagen@tu-dortmund.de

the reference face *locally* or even without restrictions. Changing the reference face locally means, given a vertex v incident to an initial reference face F, we select a new reference face F', such that F' is also incident to v. Using the new results, we introduce a new class of drawings for which we show that each drawing in this class has at least $H(n)$ crossings; we call drawings belonging to this class *semi-pair-shellable*. There are semi-pair-shellable drawings that are not seq-shellable. But unlike seq-shellability, semi-pair-shellability does not comprise all previously found classes and only contains drawings with an odd number of vertices. However, every $(\lfloor \frac{n}{2} \rfloor - 1)$-seq-shellable drawing with n odd is semi-pair-shellable. Furthermore, we introduce *k-deviations* of a drawing D of K_n. They are the difference between the numbers of cumulated k-edges in D and reference values corresponding to a drawing with exactly $H(n)$ crossings. They allow us to further relax the necessity of a globally fixed reference face.

The outline of this paper is as follows. In Section 2 we introduce the preliminaries, and in particular the necessary background on (cumulated) k-edges and their usage for verifying the lower bound on the number of crossings. In the following Section 3, we present our novel results for triple cumulated k-edges, followed by the introduction of semi-pair-shellable drawings in Section 4. We show that each drawing in this class has at least $H(n)$ crossings, and discuss the distinctive differences to seq-shellability. In Section 5 we use k-deviations to formulate conditions under which we are able to further loosen the need for a globally fixed reference face. We conjecture these conditions to be true in all good drawings. Assuming our conjecture holds, we prove a lower bound of $H(n)$ crossings for another broad class of drawings. Finally, in Section 6 we draw our conclusions and give an outlook to further possible work. Note that due to the space restrictions some proofs had to be omitted. A full version which contains all proofs and additional figures is available [15].

2 Preliminaries

A *drawing* D of a graph G on the plane is an injection ϕ from the vertex set V into the plane, and a mapping of the edge set E into the set of simple curves, such that the curve corresponding to the edge $e = uv$ has endpoints $\phi(u)$ and $\phi(v)$, and contains no other vertices [19]. We call an intersection point of the interior of two edges a crossing; a shared endpoint of two adjacent edges is not considered a crossing. The crossing number $cr(D)$ of a drawing D equals the number of crossings in D and the crossing number $cr(G)$ of a graph G is the minimum crossing number over all its possible drawings. We restrict our discussions to *good* drawings of K_n, and call a drawing *good* if (1) any two of the curves have finitely many points in common, (2) no two curves have a point in common in a tangential way, (3) no three curves cross each other in the same point, (4) any two edges cross at most once and (5) no two adjacent edges cross. It is known that every drawing with a minimum number of crossings is good [18]. Given a drawing D, we call the points also vertices and the curves edges, V denotes the set of vertices (i.e. points), and E denotes the set of edges (i.e. curves) of D. If we subtract the drawing D from the plane, a set of open regions remains. We call $\mathcal{F}(D) := \mathbb{R}^2 \setminus D$ the set of *faces* of the drawing D. If we remove a vertex v and all its incident edges from D, we get the subdrawing $D - v$. We denote by $f(v)$ the unique face in $D - v$ that contains all the faces that are incident to v in D, and call $f(v)$ the *superface* of v. We might consider the drawing to be on the surface of the sphere S^2, which is equivalent to the drawing on the plane due to the homeomorphism between the plane and the sphere minus one point. Next, we introduce *k-edges*; according to [7] the origins of k-edges lie in computational geometry and problems over n-point set, especially problems on halving lines and k-sets. An early definition in the geometric setting goes back to Erdős et al. [9]. Given a set P of n points in general position in the plane, the authors add a directed edge $e = (p_i, p_j)$ between the two distinct points p_i and p_j, and consider the continuation as line that separates the plane into a left and a right half plane. There is a (possibly empty) point set $P_L \subseteq P$ on the left side of e, i.e. in the left half plane. Erdős et al. assign $k := \min(|P_L|, |P \setminus P_L|)$ to e. Later, the name *k-edge* emerged for any edge that is assigned the value k. Lovász et al. [13] used k-edges for determining a lower bound on the crossing number of rectilinear graph drawings. Finally, Ábrego et al. [3] extended the concept of k-edges from rectilinear to topological graph drawings and used the concept to show that the crossing number of 2-page-book drawings is at least $H(n)$. Every edge in a good drawing D of K_n is a k-edge for a specific value of $k \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor - 1\}$. Let D be on the surface of the sphere S^2, and $e = uv$ be an edge in D and $F \in \mathcal{F}(D)$ be an arbitrary but fixed face; we call F the *reference face*. Together with any vertex $w \in V \setminus \{u, v\}$, the edge e forms a triangle uvw and hence a closed curve that separates the surface of the sphere into two parts. For an arbitrary but fixed orientation of e, one can distinguish between the left part and the right part of the separated surface. If F lies in the left part of the surface, we say the triangle has orientation $+$ else it has orientation $-$. For e there are $n - 2$ possible triangles in total, of which $0 \leq i \leq n - 2$ triangles have orientation $+$ (or $-$) and $n - 2 - i$ triangles have orientation $-$ (or $+$ respectively). We define the *k-value* of e to be the minimum of i and $n - 2 - i$. We say e is an *i-edge* with respect to the reference face F if its k-value equals i. See Figure 1 for an example.

Ábrego et al. [3] showed that the crossing number of a drawing is expressible in terms of the number of k-edges for $0 \leq k \leq \lfloor \frac{n}{2} \rfloor - 1$ with respect to the reference face. The following definitions of the *cumulated* numbers of k-edges are used for determining lower bounds of the crossing number. The double cumulated number of k-edges has been defined by Ábrego et al. [3], and the triple cumulated number of k-edges has been introduced by Balko et al. [8] in the context of the crossing number of x-monotone drawings.

Definition 1 [3, 8] *Let D be a good drawing and $E_k(D)$ be the number of k-edges in D with respect to a reference face $F \in \mathcal{F}(D)$ and for each $k \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor - 1\}$. We denote*

$$\bar{E}_k(D) := \sum_{j=0}^{k} \sum_{i=0}^{j} E_i(D) = \sum_{i=0}^{k} (k+1-i) E_i(D)$$

the double cumulated number of k-edges, *and*

$$\hat{E}_k(D) := \sum_{i=0}^{k} \bar{E}_i(D) = \sum_{i=0}^{k} \binom{k+2-i}{2} E_i(D)$$

the triple cumulated number of k-edges.

We also write *double (triple) cumulated k-edges* or *double (triple) cumulated k-value* instead of double (triple) cumulated number of k-edges. We express the crossing number of a drawing using the triple cumulated k-edges.

Theorem 2 [8] *Let D be a good drawing of K_n and $m = \lfloor \frac{n}{2} \rfloor - 2$. With respect to a reference face $F \in \mathcal{F}(D)$ we have for n odd*

$$cr(D) = 2 \cdot \hat{E}_m(D) - \frac{1}{8} n(n-1)(n-3)$$

and for n even

$$cr(D) = \hat{E}_m(D) + \hat{E}_{m-1}(D) - \frac{1}{8} n(n-1)(n-2).$$

It is an important observation, that for n odd the value $\hat{E}_m(D)$ and n even $\hat{E}_m(D) + \hat{E}_{m-1}(D)$ are identical for all faces of D. Note that this does not apply to the double cumulated case, i.e. $\bar{E}_m(D)$ or $\bar{E}_m(D) + \bar{E}_{m-1}(D)$, respectively. Using the following lower bounds, we are able to verify the Harary-Hill Conjecture.

Corollary 3 [8] *Let D be a good drawing of K_n. If n is odd and*

$$\hat{E}_{\frac{n-1}{2}-2}(D) \geq 3 \binom{\frac{n-1}{2}+2}{4}$$

or n is even and with respect to a face $F \in \mathcal{F}(D)$

$$\hat{E}_{\frac{n}{2}-2}(D) \geq 3 \binom{\frac{n}{2}+2}{4} \text{ and } \hat{E}_{\frac{n}{2}-3}(D) \geq 3 \binom{\frac{n}{2}+1}{4},$$

then $cr(D) \geq H(n)$.

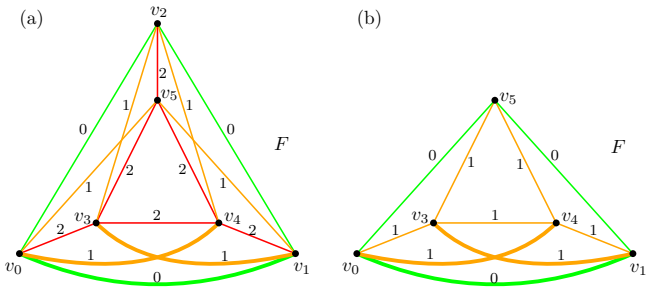


Figure 1: Example (a) shows a crossing optimal drawing D of K_6 with the k-values at the edges. (b) shows the subdrawing $D - v_2$ and its k-values. The fat highlighted edges v_0v_1, v_0v_4 and v_1v_3 are invariant and keep their k-values. The reference face is the outer face F.

If a vertex touches the reference face, it is incident to a certain set of k-edges.

Lemma 4 [3] *Let D be a good drawing of K_n, $F \in \mathcal{F}(D)$ and $v \in V$ be a vertex incident to F. With respect to F, vertex v is incident to two i-edges for $0 \leq i \leq \lfloor \frac{n}{2} \rfloor - 2$. Furthermore, if we label the edges incident to v counter clockwise with $e_0, \ldots, e_{n-2}$ such that e_0 and e_{n-2} are incident to the face F, then e_i is a k-edge with $k = \min(i, n-2-i)$ for $0 \leq i \leq n-2$.*

The definition of semi-pair-shellability uses seq-shellability, which itself is based on simple sequences.

Definition 5 (Simple sequence) [16] *Let D be a good drawing of K_n, $F \in \mathcal{F}(D)$ and $v \in V$ with v incident to F. Furthermore, let $S_v = (u_0, \ldots, u_k)$ with $u_i \in V \setminus \{v\}$ be a sequence of distinct vertices. If u_0 is incident to F and vertex u_i is incident to a face containing F in the subdrawing $D - \{u_0, \ldots, u_{i-1}\}$ for all $1 \leq i \leq k$, then we call S_v a* simple sequence *of v.*

Definition 6 (Seq-Shellability) [16] *Let D be a good drawing of K_n. We call D k-*seq-shellable *for $k \geq 0$ if there exists a face $F \in \mathcal{F}(D)$ and a sequence of distinct vertices $a_0, \ldots, a_k$ such that a_0 is incident to F, and* (1.) *for each $i \in \{1, \ldots, k\}$, vertex a_i is incident to the face containing F in drawing $D - \{a_0, \ldots, a_{i-1}\}$, and* (2.) *for each $i \in \{0, \ldots, k\}$, vertex a_i has a simple sequence $S_i = (u_0, \ldots, u_{k-i})$ with $u_j \in V \setminus \{a_0, \ldots, a_i\}$ for $0 \leq j \leq k-i$ in drawing $D - \{a_0, \ldots, a_{i-1}\}$.*

If a drawing D of K_n is $(\lfloor \frac{n}{2} \rfloor - 2)$-seq-shellable, we omit the $(\lfloor \frac{n}{2} \rfloor - 2)$ part and say that D is seq-shellable. The class of seq-shellable drawings contains all drawings that are $(\lfloor \frac{n}{2} \rfloor - 2)$-seq-shellable.

3 Properties of Triple Cumulated k-Edges

In this section, we present new results for triple cumulated k-edges. First, we introduce the triple cumulated

value of edges incident to v. Having a vertex v incident to the reference face F, we know from Lemma 4 that v is incident to two k-edges for each $k \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor - 2\}$ and it follows that the triple cumulated number of k-edges incident to v is $\hat{E}_k(D, v) = \sum_{i=0}^{k} \binom{k+2-i}{2} \cdot 2 = 2\binom{k+3}{3}$.

Next, we introduce the *double cumulated invariant* edges. Consider removing a vertex $v \in V$ from a good drawing D of K_n, resulting in the subdrawing $D-v$. By deleting v and its incident edges every remaining edge loses one triangle, i.e. for an edge $uw \in E$ there are only $(n-3)$ triangles uwx with $x \in V \setminus \{u, v, w\}$ (instead of the $(n-2)$ triangles in drawing D). The k-value of any edge $e \in E$ is defined as the minimum number of $+$ or $-$ oriented triangles that contain e. If the lost triangle had the same orientation as the minority of triangles, the k-value of e is reduced by one else it stays the same. Therefore, every k-edge in D with respect to $F \in \mathcal{F}(D)$ is either a k-edge or a $(k-1)$-edge in the subdrawing $D - v$ with respect to $F' \in \mathcal{F}(D - v)$ and $F \subseteq F'$. We call an edge e *invariant* if e has the same k-value with respect to F in D as for F' in D'. See Figure 1 for an example.

For $0 \leq k \leq \lfloor \frac{n}{2} \rfloor - 1$ we denote the number of invariant k-edges between D and D' (with respect to F and F' respectively) by $I_k(D, D')$. Furthermore, we define the *double cumulated invariant k-value* as

$$\bar{I}_k(D, D') \coloneqq \sum_{j=0}^{k} \sum_{i=0}^{j} I_i(D, D') = \sum_{i=0}^{k} (k - i + 1) I_i(D, D').$$

We define $\hat{E}_{-1}(D) \coloneqq 0$, and introduce the recursive representation for the triple cumulated k-edges.

Lemma 7 *Let D be a good drawing of K_n, $v \in V$ and $F \in \mathcal{F}(D)$. With respect to the reference face F and for all $k \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor - 2\}$, we have*

$$\hat{E}_k(D) = \hat{E}_{k-1}(D - v) + \hat{E}_k(D, v) + \bar{I}_k(D, D - v).$$

Using the triple cumulated value, we only have to ensure that $\hat{E}_k(D) \geq 3\binom{k+4}{4}$ for $k = \frac{n-1}{2} - 2$ if n is odd, or for each $k \in \{\frac{n}{2} - 2, \frac{n}{2} - 3\}$ if n is even in order to prove that $cr(D) \geq H(n)$ (Theorem 2). Mutzel and Oettershagen [16] showed that any seq-shellable drawing D of K_n has $\bar{E}_i(D) \geq 3\binom{i+3}{3}$ for all $i \in \{0, \ldots, k\}$ with respect to the reference face F. This implies the following corollary.

Corollary 8 *Let D be a drawing of K_n that is seq-shellable for a reference face $F \in \mathcal{F}(D)$, then $\hat{E}_k(D) \geq 3\binom{k+4}{4}$ for all $k \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor - 2\}$ with respect to F.*

The following lemma gives a lower bound on double cumulated invariant edges incident to a vertex that touches the reference face.

Lemma 9 *Let D be a good drawing of K_n with two vertices v and w incident to the reference face $F \in \mathcal{F}(D)$. If v is removed, the double cumulated value of invariant k-edges incident to w with respect to F is at least $\binom{k+2}{2}$ for all $k \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor - 2\}$.*

The following lemma is the gist that allows us to locally change the reference face if we have an odd number of vertices.

Lemma 10 *Let D be a good drawing of K_n and $v \in V$. For n odd, the number of double cumulated invariant edges $\bar{I}_{\lfloor \frac{n}{2} \rfloor - 2}(D, D - v)$ is the same with respect to any face incident to v in D and the superface $f(v)$ in $D - v$.*

Proof. Let $m = \lfloor \frac{n}{2} \rfloor - 2$. Lemma 7 implies that with respect to a face incident to v

$$\bar{I}_m(D, D - v) = \hat{E}_m(D) - \hat{E}_{m-1}(D - v) - \hat{E}_m(D, v).$$

$\hat{E}_m(D)$ is the same for all faces of D, the value $\hat{E}_{m-1}(D - v)$ with respect to face $f(v)$ is fixed and for each face incident to v we have $\hat{E}_m(D, v) = 2\binom{m+3}{3}$. Therefore, it follows that also the value of $\bar{I}_m(D, D - v)$ has to be the same for every face incident to v. □

4 Semi-Pair-Shellability

Basis for the new class of semi-pair-shellable drawings are pair-sequences.

Definition 11 (Pair-sequence) *Let D be a good drawing of K_n, $v \in V$ and $P_v = (u_0, \ldots, u_{\lfloor \frac{n}{2} \rfloor - 2})$ be a sequence of distinct vertices $u_i \in V \setminus \{v\}$ for $0 \leq i \leq \lfloor \frac{n}{2} \rfloor - 2$.*

We call P_v a pair-sequence *of v if for $j \in \{1, \ldots, \lfloor \frac{n}{2} \rfloor - 3\}$ and $(n - j)$ odd, the vertex u_j in the drawing $D - \{u_0, \ldots, u_{j-1}\}$ is incident to a face $F' \in \mathcal{F}(D - \{u_0, \ldots, u_{j-1}\})$, where F' is also incident to v, and in the drawing $D - \{u_0, \ldots, u_j\}$ vertex u_{j+1} is incident to face $f(u_j)$, and vertex u_0 is incident to $F \in \mathcal{F}(D)$, where F is also incident to v.*

For example, in Figure 2 vertex v in the drawing of K_{11} has the pair-sequence (u_0, u_1, u_2, u_3). The pair-sequence of vertex v ensures that if we remove v from D, there are enough double cumulated invariant k-edges. Therefore, we are able to guarantee a lower bound on $\hat{E}_{\lfloor \frac{n}{2} \rfloor - 2}(D)$ using Lemma 7.

Lemma 12 *Let D be a good drawing of K_n, $v \in V$ and $(u_0, \ldots, u_{\lfloor \frac{n}{2} \rfloor - 2})$ a pair-sequence of v, then $\bar{I}_{\lfloor \frac{n}{2} \rfloor - 2}(D, D - v) \geq \binom{\lfloor \frac{n}{2} \rfloor + 1}{3}$.*

Proof. Without loss of generality let n be odd and let $m = \frac{n-1}{2} - 2$ (for n even we can proceed similarly and start with $m = \frac{n}{2} - 2$). Lemma 9 states that the double cumulated value of invariant edges incident to u_0 equals $\binom{k+2}{2}$ for $0 \leq k \leq m$ with respect to a face F incident to v and u_0, and the removal of v from D. Likewise,

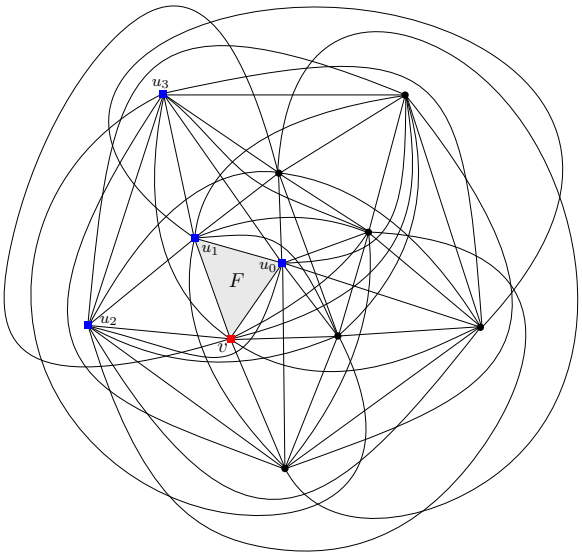


Figure 2: Single-pair-seq-shellable drawing of K_{11}. The initial reference face is F, vertex v has the pair-sequence (u_0, u_1, u_2, u_3).

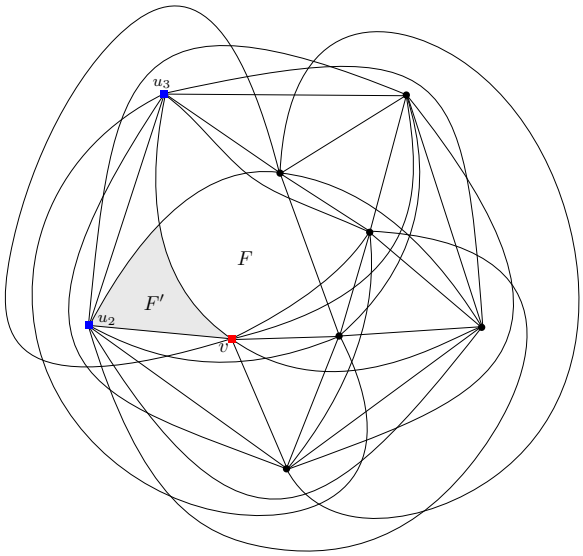


Figure 3: Subdrawing $D-\{u_0, u_1\}$ of the drawing shown in Figure 2. The reference face is now F', which is incident to v and u_2.

the double cumulated value of invariant edges incident to u_1 is at least $\binom{k+2}{2}$ for $0 \leq k \leq m-1$ if we remove v from $D - u_0$ with respect to F. The edge u_0u_1 may be invariant or non-invariant in D with respect to removing v. Now consider the drawing $D - \{u_0, u_1\}$ with $n-2$ vertices and $\frac{n-3}{2} - 2 = \frac{n-1}{2} - 3 = m-1$. Because $n-2$ is odd, we know that for all faces incident to v the value of $\bar{I}_{m-1}(D-\{u_0,u_1\}, D-\{v,u_0,u_1\})$ is the same (Lemma 10). We may select a new reference face F', such that v and u_3 are incident to F', and we can argue again, using Lemma 9, that removing v leads to at least $\binom{k+2}{2}$ for $0 \leq k \leq m-2$ double cumulated value of invariant edges incident to u_2, since u_2 is incident to F'. The double cumulated value of invariant edges incident to u_3 is at least $\binom{k+2}{2}$ for $0 \leq k \leq m-3$ with respect to F' if we remove v from $D - \{u_0, u_1, u_2\}$. Again, the edge u_2u_3 may be invariant or non-invariant in $D - \{u_0, u_1\}$ with respect to removing v.

In general, we are able to change the reference face incident to v if a subdrawing K_r of K_n with $0 < r \leq n$ has an odd number of vertices because the number of double cumulated invariant $(\lfloor \frac{r}{2} \rfloor - 2)$-edges does not change (see Lemma 10). Furthermore, since vertex u_i for $0 \leq i \leq \lfloor \frac{n}{2} \rfloor - 2$ is incident to the (current) reference face, u_i contributes at least $\binom{m-i+2}{2}$ to the value of the double cumulated invariant m-value with respect to removing v from D. Thus, $\bar{I}_m(D, D-v) \geq \sum_{i=1}^{m+2} \binom{i}{2} = \binom{m+3}{3}$. □

In Figure 2, both vertices u_0 and u_1 are incident to the initial reference face F. Figure 3 shows the drawing after removing the first pair (i.e. u_0 and u_1). The face F is not incident to any vertex except v. Changing the reference face to F' allows to proceed with u_2 and u_3. Notice that in a drawing D of K_n with n odd, only the value of $\bar{I}_{\lfloor \frac{n}{2} \rfloor - 2}(D, D-v)$ is invariant with respect to changing the reference face. The values $\bar{I}_k(D, D-v)$ for $k \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor - 3\}$ may change when selecting a different reference face.

Lemma 13 *Let D be a good drawing of K_n with n odd and $v \in V$. If v has a pair-sequence and for the subdrawing $D-v$ we have $\hat{E}_{\lfloor \frac{n}{2} \rfloor - 3}(D-v) \geq 3\binom{\lfloor \frac{n}{2} \rfloor + 1}{4}$ with respect to $f(v)$, then $cr(D) \geq H(n)$.*

Proof. We have $\hat{E}_{\lfloor \frac{n}{2} \rfloor - 2}(D, v) \geq 2\binom{\lfloor \frac{n}{2} \rfloor + 1}{3}$ for any face that is incident to v in D, and because v has a pair-sequence and due to Lemma 12, it follows that $\bar{I}_{\lfloor \frac{n}{2} \rfloor - 2}(D, D-v) \geq \binom{\lfloor \frac{n}{2} \rfloor + 1}{3}$. Using Lemma 7, it follows for every face incident to v $\hat{E}_{\lfloor \frac{n}{2} \rfloor - 2}(D) \geq 3\binom{\lfloor \frac{n}{2} \rfloor + 2}{4}$. Since n is odd, the result follows with Corollary 3. □

Next, we define semi-pair-shellability.

Definition 14 *Let D be a good drawing of K_n with n odd. If there exists a vertex $v \in V$ that has a pair-sequence and the subdrawing $D-v$ is seq-shellable for $f(v)$, then we call D* semi-pair-shellable.

Using Lemma 13, we show that semi-pair-shellable drawings have at least $H(n)$ crossings. By definition the subdrawing $D-v$ is seq-shellable, hence $\hat{E}_{\lfloor \frac{n}{2} \rfloor - 3}(D-v) \geq 3\binom{\lfloor \frac{n}{2} \rfloor + 1}{4}$ for $f(v)$ (see Corollary 8). Consequently, Theorem 15 follows.

Theorem 15 *If D is a semi-pair-shellable drawing of K_n, then $cr(D) \geq H(n)$.*

The drawing D in Figure 2 is semi-pair-shellable but not seq-shellable. It is impossible to find a vertex sequence

and corresponding simple sequences to apply the definition of seq-shellability. However, the subdrawing $D-v$ is seq-shellable for face $f(v)$ and v has a pair-sequence. Consequently, D is semi-pair-shellable.

We are not aware of a crossing optimal semi-pair-shellable drawing that is not seq-shellable. Every $(\lfloor \frac{n}{2} \rfloor - 1)$-seq-shellable drawing D with n odd is also semi-pair-shellable: By definition, D has a vertex sequence $a_0, \ldots, a_{\lfloor \frac{n}{2} \rfloor - 1}$, and each a_i has a simple sequence S_i with $i \in \{0, \ldots, \lfloor \frac{n}{2} \rfloor - 1\}$. The first $\lfloor \frac{n}{2} \rfloor - 2$ vertices of S_0 are a pair-sequence for a_0. Moreover, the drawing $D - a_0$ is $(\lfloor \frac{n}{2} \rfloor - 2)$-seq-shellable with the vertex sequence $a_1, \ldots, a_{\lfloor \frac{n}{2} \rfloor - 1}$ and its corresponding simple sequences. However, there exist $(\lfloor \frac{n}{2} \rfloor - 2)$-seq-shellable drawings that are not semi-pair-shellable. Thus, semi-pair-shellability is a new distinct class that intersects but does not contain the class of seq-shellable drawings.

5 k-Deviations

In the following, we introduce k-deviations, which we use to represent the difference between (cumulated) k-edges and optimal values; k-deviations allow us to formulate conditions under which we are able to change the reference face even more freely. Note that if for a drawing D of K_n it holds that $E_k(D) = 3(k+1)$ for all $0 \leq k \leq \lfloor \frac{n}{2} \rfloor - 2$, then $cr(D) = H(n)$. We define k-deviations as the difference between this value and the number of k-edges in a drawing.

Definition 16 *Let D be a good drawing of K_n, $F \in \mathcal{F}(D)$ and $E_k(D)$ the number of k-edges for $0 \leq k \leq \lfloor \frac{n}{2} \rfloor - 2$ with respect to F. We denote by $\Delta_k(D) \coloneqq E_k(D) - 3(k+1)$ the* k-deviation *of the drawing D for $0 \leq k \leq \lfloor \frac{n}{2} \rfloor - 2$ with respect to F. Moreover, we define the cumulated versions of the k-deviation for F as*

$$\bar{\Delta}_k(D) \coloneqq \sum_{i=0}^{k} \sum_{j=0}^{i} \Delta_j(D) = \sum_{i=0}^{k} (k+1-i)\Delta_i(D) \text{ and}$$

$$\hat{\Delta}_k(D) \coloneqq \sum_{i=0}^{k} \bar{\Delta}_i(D) = \sum_{i=0}^{k} \binom{k+2-i}{2} \Delta_i(D).$$

Finally, we define the deviation of the crossing number of D from the Harary-Hill optimal number of crossings as $\Delta_{cr}(D) \coloneqq cr(D) - H(n)$.

We can express k-deviations in the following ways.

Lemma 17 *Let D be a good drawing of K_n. For a reference face $F \in \mathcal{F}(D)$ and $0 \leq k \leq \lfloor \frac{n}{2} \rfloor - 2$, we have $\hat{\Delta}_k(D) = \hat{\Delta}_{k-1}(D) + \bar{\Delta}_k(D)$.*

Corollary 18 *Let D be a good drawing of K_n. For n odd we have $\Delta_{cr}(D) = 2\hat{\Delta}_{\frac{n-1}{2}-2}(D)$, and for a reference face $F \in \mathcal{F}(D)$ and n even $\Delta_{cr}(D) = \hat{\Delta}_{\frac{n}{2}-2}(D) + \hat{\Delta}_{\frac{n}{2}-3}(D)$.*

Notice, that Corollary 18 implies Kleitman's parity theorem for complete graphs [12]. The following lemma gives a lower bound on $\hat{\Delta}_{\lfloor \frac{n}{2} \rfloor - 3}(D)$.

Lemma 19 *Let D be a good drawing of K_n with $cr(D) \geq H(n)$. For each $F \in \mathcal{F}(D)$ with $\hat{\Delta}_{\lfloor \frac{n}{2} \rfloor - 2}(D) \geq \bar{\Delta}_{\lfloor \frac{n}{2} \rfloor - 2}(D)$, it holds that $\hat{\Delta}_{\lfloor \frac{n}{2} \rfloor - 3}(D) \geq 0$.*

With the following proposition, we are able to select a new reference face for the subdrawing $D - v$.

Proposition 20 *Let D be a good drawing of K_n with n odd and $v \in V$, such that the subdrawing $D - v$ is seq-shellable for any face $F \in \mathcal{F}(D-v)$. If v has a pair-sequence and in subdrawing $D - v$ for $f(v)$ it holds that $\hat{\Delta}_{\frac{n-1}{2}-2}(D-v) \geq \bar{\Delta}_{\frac{n-1}{2}-2}(D-v)$, then $cr(D) \geq H(n)$.*

So far, for all drawings and all faces we inspected, the condition of Lemma 19 has been fulfilled. We conjecture it to be true for all good drawings of K_n.

Conjecture 2 *Let D be a good drawing of K_n. With respect to any face $F \in \mathcal{F}(D)$, we have*

$$\hat{\Delta}_{\lfloor \frac{n}{2} \rfloor - 2}(D) \geq \bar{\Delta}_{\lfloor \frac{n}{2} \rfloor - 2}(D).$$

Under the assumption that Conjecture 2 holds, we are able to prove the Harary-Hill Conjecture for another new class of drawings. Here, we can select a different reference face for each vertex.

Theorem 21 *Let D be a good drawing of K_n and $v_1, \ldots, v_n$ a sequence of the vertices, such that every vertex v_i with $i \in \{1, \ldots, n\}$ and i odd has a pair-sequence, and every vertex v_i with $i \in \{1, \ldots, n\}$ and i even has a simple sequence. If Conjecture 2 holds, then $cr(D) \geq H(n)$.*

6 Conclusions and Outlook

We introduced semi-pair-shellable drawings of complete graphs and verified that each drawing in this class has at least $H(n)$ crossings. For the first time, we used more than a single globally fixed reference face in order to show lower bounds on the triple cumulated k-edges. Semi-pair-shellability is only defined for drawings of K_n with n odd so far. Extending semi-pair-shellability to drawings of K_n with an even number of vertices is an open problem. Here, it would suffice to show that $\hat{\Delta}_{\lfloor \frac{n}{2} \rfloor - 2}(D) + \hat{\Delta}_{\lfloor \frac{n}{2} \rfloor - 3}(D) \geq 0$ implies $\hat{\Delta}_{\lfloor \frac{n}{2} \rfloor - 3}(D) \geq 0$ in order to generalize our results from semi-pair-shellability to *pair-shellability*, i.e. a version of seq-shellability with pair-sequences instead of simple sequences. Moreover, we introduced k-deviations to formulate conditions under which we are able to select a new reference face in each subdrawing. Proving Conjecture 2 would settle the Harary-Hill Conjecture for a very broad class of drawings, comprising seq- and semi-pair-shellability. Still, there are optimal drawings where each face touches a single vertex only [6], thus no vertex has a simple or pair-sequence.

References

[1] B. Ábrego, O. Aichholzer, S. Fernández-Merchant, T. Hackl, J. Pammer, A. Pilz, P. Ramos, G. Salazar, and B. Vogtenhuber. All good drawings of small complete graphs. In *Proc. 31^{st} European Workshop on Computational Geometry (EuroCG)*, pages 57–60, 2015.

[2] B. Ábrego, O. Aichholzer, S. Fernández-Merchant, D. McQuillan, B. Mohar, P. Mutzel, P. Ramos, R. Richter, and B. Vogtenhuber. Bishellable drawings of K_n. In *Proc. XVII Encuentros de Geometría Computacional (EGC)*, pages 17–20, Alicante, Spain, 2017.

[3] B. Ábrego, O. Aichholzer, S. Fernández-Merchant, P. Ramos, and G. Salazar. The 2-page crossing number of K_n. In *Proceedings of the Twenty-eighth Annual Symposium on Computational Geometry*, SoCG '12, pages 397–404, New York, NY, USA, 2012. ACM.

[4] B. M. Ábrego, O. Aichholzer, S. Fernández-Merchant, P. Ramos, and G. Salazar. More on the crossing number of K_n: Monotone drawings. *Electronic Notes in Discrete Mathematics*, 44:411–414, 2013.

[5] B. M. Ábrego, O. Aichholzer, S. Fernández-Merchant, P. Ramos, and G. Salazar. Shellable drawings and the cylindrical crossing number of K_n. *Discrete & Computational Geometry*, 52(4):743–753, 2014.

[6] B. M. Ábrego, O. Aichholzer, S. Fernández-Merchant, P. Ramos, and B. Vogtenhuber. Non-shellable drawings of K_n with few crossings. In *Proceedings of the 26th Canadian Conference on Computational Geometry, CCCG 2014, Halifax, Nova Scotia, Canada*, 2014.

[7] B. M. Ábrego, M. Cetina, S. Fernández-Merchant, J. Leaños, and G. Salazar. On $\leq k$-edges, crossings, and halving lines of geometric drawings of K_n. *Discrete & Computational Geometry*, 48(1):192–215, 2012.

[8] M. Balko, R. Fulek, and J. Kynčl. Crossing numbers and combinatorial characterization of monotone drawings of K_n. *Discrete & Computational Geometry*, 53(1):107–143, 2015.

[9] P. Erdös, L. Lovász, A. Simmons, and E. G. Straus. Dissection graphs of planar point sets. In *A Survey of Combinatorial Theory*, pages 139–149. Elsevier, 1973.

[10] R. K. Guy. A combinatorial problem. *Nabla, Bulletin of the Malayan Mathematical Society*, 7:68–72, 1960.

[11] F. Harary and A. Hill. On the number of crossings in a complete graph. *Proceedings of the Edinburgh Mathematical Society*, 13(4):333–338, 1963.

[12] D. J. Kleitman. A note on the parity of the number of crossings of a graph. *Journal of Combinatorial Theory, Series B*, 21(1):88–89, 1976.

[13] L. Lovász, K. Vesztergombi, U. Wagner, and E. Welzl. Convex quadrilaterals and k-sets. *Contemporary Mathematics*, 342:139–148, 2004.

[14] D. McQuillan, S. Pan, and R. B. Richter. On the crossing number of K_{13}. *J. Comb. Theory, Ser. B*, 115:224–235, 2015.

[15] P. Mutzel and L. Oettershagen. The crossing number of semi-pair-shellable drawings of complete graphs. *CoRR*, abs/1805.06780, 2018.

[16] P. Mutzel and L. Oettershagen. The crossing number of seq-shellable drawings of complete graphs. In C. S. Iliopoulos, H. W. Leong, and W. Sung, editors, *Combinatorial Algorithms - 29th International Workshop, IWOCA 2018, Singapore, July 16-19, 2018, Proceedings*, volume 10979 of *Lecture Notes in Computer Science*, pages 273–284. Springer, 2018.

[17] S. Pan and R. B. Richter. The crossing number of K_{11} is 100. *Journal of Graph Theory*, 56(2):128–134, 2007.

[18] M. Schaefer. The graph crossing number and its variants: A survey. *The Electronic Journal of Combinatorics (Dec 22, 2017)*, 1000:DS21–May, 2013.

[19] L. A. Székely. A successful concept for measuring nonplanarity of graphs: The crossing number. *Electronic Notes in Discrete Mathematics*, 5:284–287, 2000.

Learning Simplicial Complexes from Persistence Diagrams

Robin Lynne Belton* Brittany Terese Fasy*† Rostik Mertz† Samuel Micka† David L. Millman†
Daniel Salinas† Anna Schenfisch* Jordan Schupbach* Lucia Williams†

Abstract

Topological Data Analysis (TDA) studies the "shape" of data. A common topological descriptor is the persistence diagram, which encodes topological features in a topological space at different scales. Turner, Mukherjee, and Boyer showed that one can reconstruct a simplicial complex embedded in $\mathbb{R}^3$ using persistence diagrams generated from all possible height filtrations (an uncountably infinite number of directions). In this paper, we present an algorithm for reconstructing plane graphs $K = (V, E)$ in $\mathbb{R}^2$, i.e., a planar graph with vertices in general position and a straight-line embedding, from a quadratic number height filtrations and their respective persistence diagrams.

1 Introduction

Topological data analysis (TDA) is a promising tool in fields as varied as materials science, transcriptomics, and neuroscience [8, 11, 14]. Although TDA has been quite successful in the analysis of point cloud data [13], its purview extends to any data that can be encoded as a topological space. Topological spaces can be described in terms of their homology, e.g., connected components and "holes." Simplicial complexes, in particular, are the most common representation of topological spaces. In this work, we focus our attention on a subset of simplicial complexes, namely, plane graphs embedded in $\mathbb{R}^2$, with applications to shape reconstruction.

In this paper, we explore the question, *Can we reconstruct embedded simplicial complexes from a finite number of directional persistence diagrams?* Our work is motivated by [15], which proves that one can reconstruct simplicial complexes from an uncountably infinite number of diagrams. Here, we make the first step towards providing a polynomial-time reconstruction for simplicial complexes. In particular, the main contributions of this paper are to set a bound on the number of persistence diagrams required to reconstruct a plane graph and to provide a polynomial-time algorithm for reconstructing the graph.

*Depart. of Mathematical Sciences, Montana State U.
†School of Computing, Montana State U.
{robin.belton, brittany.fasy, david.millman, annaschenfisch, jordan.schupbach}@montana.edu {samuel.micka, daniel.salinas, lucia.williams}@msu.montana.edu rostik.mertz@student.montana.edu

2 Related Work

The problem of manifold and stratified space learning is an active research area in computational mathematics. For example, Zheng et al. study the 3D reconstruction of plant roots from multiple 2D images [16]. Their method uses persistent homology to ensure the resulting 3D root model is connected.

Map construction algorithms reconstruct street maps as an embedded graph from a set of input trajectories. Three common approaches are Point Clustering, Incremental Track Insertion, and Intersection Linking [1]. Ge, Safa, Belkin, and Wang develop a point clustering algorithm using Reeb graphs to extract the skeleton graph of a road from point-cloud data [6]. The original embedding can be reconstructed using a principal curve algorithm [10]. Karagiorgou and Pfoser give an incremental track insertion algorithm to reconstruct a road network from vehicle trajectory GPS data [9]. Ahmed et al. provide an incremental track insertion algorithm to reconstruct road networks from point could data [2]. The reconstruction is done incrementally, using a variant of the Fréchet distance to add curves to the current basis. Ahmed, Karagiorgou, Pfoser, and Wenk describe all these methods in [1]. Finally, Dey, Wang, and Wang use persistent homology to reconstruct embedded graphs. This research has also been applied to input trajectory data [4]. Dey et al. use persistence to guide the Morse cancellation of critical simplices. In contrast, the work presented here uses persistence to generate the diagrams that encode the underlying graph.

Our work extends previous work on the persistent homology transform (PHT) [15]. As detailed in Section 3, persistent homology summarizes the homological changes for a filtered topological space. When applied to a simplicial complex embedded in $\mathbb{R}^d$, we can compute a different filtration for every direction in $\mathbb{S}^{d-1}$; this family of persistence diagrams is referred to as the persistent homology transform (PHT). The map from a simplicial complex to PHT is injective [15]. Hence, knowing the PHT of a simplicial complex uniquely identifies that complex. The proof presented in [15] relies on the continuity of persistence diagrams as the direction of filtration varies *continuously*.

Our paper bounds the number of directions by presenting an algorithm for reconstructing the simplicial

complex, when we are able to obtain persistence diagrams for a given set of directions. Simultaneous to our investigation, others have also observed that the number of directions can be bounded using the Radon transform; see [3, 7]. In the work presented in the current paper, we seek to reconstruct graphs from their respective persistence diagrams, using a geometric approach. We bound the number of directional persistence diagrams since computing the PHT, as presented in [15], requires the computation of filtrations from an infinite number of possible directions. Our work provides a theoretical guarantee of correctness for a finite subset of directions by providing the reconstruction algorithm.

3 Preliminary

In this paper, we explore the question, *Can we reconstruct embedded simplicial complexes from a finite number of directional persistence diagrams?* We begin by summarizing the necessary background information, but refer the reader to [5] for a more comprehensive overview of computational topology.

Simplices and Simplicial Complexes Intuitively, a k-simplex is a k-dimensional generalization of a triangle, i.e., a zero-simplex is a vertex, a one-simplex is an edge connecting two vertices, a two-simplex is a triangle, etc. In this paper, we focus on a subset of simplicial complexes embedded in $\mathbb{R}^2$ consisting of only vertices and edges. Specifically, we study *plane graphs* with straight-line embeddings (referred to simply as *plane graphs* throughout this paper). Furthermore, we assume that the embedded vertices are in general position, meaning that no three vertices are collinear and no two vertices share an x- or y-coordinate.

Height Filtration Let K be a plane graph and denote $\mathbb{S}^1$ as the unit sphere in $\mathbb{R}^2$. Consider $s \in \mathbb{S}^1$; we define the *lower star filtration* with respect to direction s in two steps. First, we let $h_s : K \to \mathbb{R}$ be defined for a simplex $\sigma \subseteq K$ by $h_s(\sigma) = \max_{v \in \sigma} v \cdot s$, where $x \cdot y$ is the inner (dot) product and measures height in the direction of y, if y is a unit vector. Intuitively, the height of σ from s is the maximum height of all vertices in σ. Then, for each $t \in \mathbb{R}$, the subcomplex $K_t := h_s^{-1}([-\infty, t))$ is composed of all simplices that lie entirely below or at the height t, with respect to the direction s. Notice $K_r \subseteq K_t$ for all $r \leq t$ and $K_r = K_t$ if no vertex has height in the interval $[r, t]$. The sequence of all such subcomplexes, indexed by $\mathbb{R}$, is the *height filtration* with respect to s, notated as $F_s(K)$. Often, we simplify notation and define $F_s := F_s(K)$.

Persistence Diagrams The persistence diagram is a summary of the homology groups $H_*(K_t)$ as the height parameter t ranges from $-\infty$ to ∞; in particular, the persistence diagram is a set of birth-death pairs (b_i, d_i). Each pair represents an interval $[b_i, d_i)$ corresponding to a homology generator. For example, a birth event may occur when the height filtration discovers a new vertex, representing a new component, and the corresponding death represents the vertex joining another connected component. By definition [5], all points in the diagonal $y = x$ are also included with infinite multiplicity. However, in this paper, we consider only those points on the diagonal that are explicitly computed in the persistence algorithm found in [5], which correspond to features with the same birth and death time. For a direction $s \in \mathbb{S}^1$, let the *directional persistence diagram* $\mathcal{D}_i(F_s(K))$ be the set of birth-death pairs for the i-th homology group from the height filtration $F_s(K)$. As with the height filtration, we simplify notation and define $\mathcal{D}_i(s) := \mathcal{D}_i(F_s(K))$ when the complex is clear from context. We conclude this section with a remark relating birth-death pairs in persistence diagrams to the simplices in K; a full discussion of this remark is found in [5, pp. 120–121 of §V.4].

Remark 1 (Adding a Simplex) *Let K be a simplicial complex and σ a k-simplex whose faces are all in K. Let β_i refer to the i-th Betti number, i.e., the rank of the i-th homology group. Then, the addition of σ to K will either increase β_k by one or decrease β_{k-1} by one.*

Thus, we can form a bijection between simplices of K and birth-death events in a persistence diagram. This observation is the crux of the proofs of Theorem 5 in Section 4 and Lemma 7 in Section 5.

4 Vertex Reconstruction

In this section, we present an algorithm for recovering the locations of vertices of a simplicial complex K using three directional persistence diagrams. Intuitively, for each direction, we identify the lines on which the vertices of K must lie. We show that by choosing the three directions such that they satisfy a simple property, we can identify all vertex locations by searching for points in the plane where three lines intersect. We call these lines *filtration lines*:

Definition 2 (Filtration Lines) *Given a direction vector $s \in \mathbb{S}^1$, and a height $h \in \mathbb{R}$ the* filtration line at height *h is the line, denoted $\ell(s, h)$, through point $h * s$ and perpendicular to direction s, where $*$ denotes scalar multiplication. Given a finite set of vertices $V \subset \mathbb{R}^2$, the* filtration lines of *V are the set of lines*

$$\mathbb{L}(s, V) = \{\ell(s, h_s(v))\}_{v \in V}.$$

Notice that all lines in $\mathbb{L}(s, V)$ are parallel. Intuitively, if v is a vertex in a simplicial complex K, then the

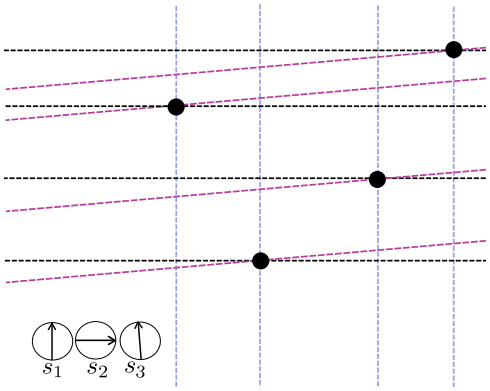


Figure 1: A vertex set, V, of size 4 with filtration lines that satisfy the Vertex Existence Lemma. Here, $s_1, s_2 \in \mathbb{S}^1$ are linearly independent and the filtration lines are colored so that $\mathbb{L}(s_1, V)$ are the black horizontal lines, $\mathbb{L}(s_2, V)$ are the blue vertical lines, and $\mathbb{L}(s_3, V)$ are the magenta diagonal lines. An intersection of three colored lines corresponds to the location of a vertex in V.

line $\ell(s, h_s(v))$ occurs at the height where the filtration $F_s(K)$ includes v for the first time. If the height is known but the complex is not, the line $\ell(s, h_s(v))$ defines all potential locations for v. By Remark 1, the births in the zero-dimensional persistence diagram are in one-to-one correspondence with the vertices of the simplex complex K. Thus, we can construct $\mathbb{L}(s, V)$ from a single directional diagram in $O(n)$ time. Given filtration lines for three carefully chosen directions, we next show a correspondence between intersections of three filtration lines and vertices in K.

In what follows, given a direction $s_i \in \mathbb{S}^1$ and a point $p \in \mathbb{R}^2$, define $\ell_i(p) := \ell(s_i, h_{s_i}(p))$ as a way to simplify notation.

Lemma 3 (Vertex Existence Lemma) *Let K be a simplicial complex with vertex set V of size n. Let $s_1, s_2 \in \mathbb{S}^1$ be linearly independent and further suppose that $\mathbb{L}(s_1, V)$ and $\mathbb{L}(s_2, V)$ each contain n lines. Let A be the collection of vertices at the intersections of lines in $\mathbb{L}(s_1, V) \cup \mathbb{L}(s_2, V)$. Let $s_3 \in \mathbb{S}^1$ such that for all $u, v \in A$, $\ell_3(u) = \ell_3(v) \iff u = v$. Then, the following two statements hold true:*

(1) *$v \in V \iff \ell_3(v) \in \mathbb{L}(s_3, V)$ and $A \cap \ell_3(v) = \{v\}$*
(2) *For all $\ell \in \mathbb{L}(s_3, V)$, $A \cap \ell \neq \emptyset$.*

Proof. First, we prove Part (1).

($\Rightarrow$) Let $v \in V$. Then, $\ell_i(v) \in \mathbb{L}(s_i, V)$ and $v \in \ell_i(v)$ for $i = \{1, 2, 3\}$. Hence, $\ell_1(v) \cap \ell_2(v) \cap \ell_3(v) = \{v\}$, as desired.

($\Leftarrow$) Assume, for the sake of contradiction, that $\ell_3(v) \in \mathbb{L}(s_3, V)$ and $A \cap \ell_3(v) = \{v\}$, yet $v \notin V$. Since $\ell_3(v) \in \mathbb{L}(s_3, V)$ and $v \notin V$, some other vertex $u \in V$ must have height $h_{s_3}(v)$. Since $u \in V$, we know $\ell_i(u) \in \mathbb{L}(s_i, V)$ for $i \in \{1, 2, 3\}$. And, by ($\Rightarrow$) applied to u, we know $u \in A$. Since $\ell_3(u) = \ell_3(v)$, both u and v are in A and on the line $\ell_3(v)$, but $u \neq v$, which is a contradiction.

Next, we prove Part (2) of the lemma. Assume, for contradiction, that there exists $\ell \in \mathbb{L}(s_3, V)$ such that $A \cap \ell = \emptyset$. As $\ell \in \mathbb{L}(s_3, V)$, a vertex $v \in V$ exists such that $\ell = \ell_3(v)$ and v lies on ℓ. However, $v \in \ell_1(v) \cap \ell_2(v) \subset A$, which is a contradiction. □

In the previous lemma, we needed to find a third direction with specific properties. If we use horizontal and vertical lines for our first two directions, then we can use the geometry of the boxes formed from these lines to pick the third direction. More specifically, we look at the box with the largest width and smallest height and pick the third direction so that if one of the corresponding lines intersects the bottom left corner of the box then it will also intersect the box somewhere along the right edge. In Figure 1, the third direction was computed using this procedure with the second box from the left in the top row. Next, we give a more precise description of the vertex localization procedure.

Lemma 4 (Vertex Localization) *Let L_H and L_V be n horizontal and n vertical lines, respectively. Let w (and h) be the largest (and smallest) distance between two lines of L_V (and L_H, respectively). Let B be the smallest axis-aligned bounding box containing the intersections of lines in $L_H \cup L_V$. For $0 < \varepsilon < h$, let $s = (w, h - \varepsilon)$. Any line parallel to s can intersect at most one line of L_H in B.*

Proof. Note that, by definition, s is a vector in the direction that is at a slightly smaller angle than the diagonal of the box of size w by h. Assume, by contradiction, that a line parallel to s may intersect two lines of L_H within B. Specifically, let $\ell_1, \ell_2 \in L_H$ and let ℓ_s be a line parallel to s such that the points $\ell_i \cap \ell_s = (x_i, y_i)$ for $i = \{1, 2\}$ are the two such intersection points within B. Notice since the lines of L_H are horizontal and by the definition of h, we observe that $|y_1 - y_2| \geq h$. Let $w' = |x_1 - x_2|$, and observe $w' \leq w$. Since the slope of ℓ_s is $(h - \varepsilon)/w$, we have $|y_1 - y_2| < h$, which is a contradiction. □

We conclude this section with an algorithm to determine the coordinates of the vertices of the original graph in $\mathbb{R}^2$, using only three height filtrations.

Theorem 5 (Vertex Reconstruction) *Let K be a plane graph. We can can compute the coordinates of all n vertices of K in $O(n \log n)$ time from three directional persistence diagrams.*

Proof. Let $s_1 = (1, 0)$ and $s_2 = (0, 1)$, which are linearly independent. We compute the filtration

lines $\mathbb{L}(s_i, V)$ for $i = 1, 2$ in $O(n)$ time by Remark 1. By our general position assumption, no two vertices of K share an x- or y-coordinate. Thus, the sets $\mathbb{L}(s_1, V)$ and $\mathbb{L}(s_2, V)$ each contain n distinct lines. Let A be the set of intersection points of the lines in $\mathbb{L}(s_1, V)$ and $\mathbb{L}(s_2, V)$. The next step is to identify a direction s_3 such that each line in $\mathbb{L}(s_3, V)$ intersects with only one point A, so that we can use Lemma 3.

Let w (and h) be the greatest (and least) distance between two adjacent lines in $\mathbb{L}(s_1, V)$ (and $\mathbb{L}(s_2, V)$, respectively). Let B be the smallest axis-aligned bounding box containing A, and let $s_* = (w, \frac{h}{2})$. By Lemma 4, any line parallel to s_* will intersect at most one line of $\mathbb{L}(s_2, V)$ in B. Thus, we choose $s_3 \in \mathbb{S}^1$ that is perpendicular to s_*. By the second part of Lemma 3, we now have that each line in $\mathbb{L}(s_3, V)$ intersects A. Thus, there are n intersections between $\mathbb{L}(s_2, V)$ and $\mathbb{L}(s_3, V)$ in B, each of which also intersects with $\mathbb{L}(s_1, V)$.

The previous paragraph leads us to a simple algorithm for finding the third direction and identifying all the triple intersections. In the analysis below, steps that do not mention a number of diagrams use no diagrams. First, we construct $\mathbb{L}(s_1, V)$ and $\mathbb{L}(s_2, V)$ in $O(n)$ time using two directional persistence diagrams. Second, we sort the lines of $\mathbb{L}(s_1, V)$ and $\mathbb{L}(s_2, V)$ by their x- and y-intercepts respectively in $O(n \log n)$ time. Third, we find s_3 by computing w and h from our sorted lines in $O(n)$ time. Fourth, we construct $\mathbb{L}(s_3, V)$ in $O(n)$ time using one directional persistence diagram. Fifth, we sort the lines in $\mathbb{L}(s_3, V)$ by their intersection with the leftmost line of $\mathbb{L}(s_1, V)$ in $O(n \log n)$ time. Finally, we compute coordinates of the n vertices by intersecting the i-th line of $\mathbb{L}(s_2, V)$ with the i-th line of $\mathbb{L}(s_3, V)$ in $O(n)$ time. (Observe, this last step works since the vertices correspond to the n intersections in B, as described above).

Therefore, we use three directional diagrams (two in the first step and one in the fourth step) and $O(n \log n)$ time (sorting of lines in the second and fifth steps) to reconstruct the vertices. □

5 Edge Reconstruction

Given the vertices constructed in Section 4, we describe how to reconstruct the edges in a plane graph using $n(n-1)$ persistence diagrams. The key to determining whether an edge exists or not is counting the degree of a vertex, for edges "below" the vertex with respect to a given direction. We begin this section by defining necessary terms, and then explicitly describing our algorithm for constructing edges.

Definition 6 (Indegree of Vertex) *Let K be a plane graph with vertex set V. Then, for every vertex $v \in V$ and every direction $s \in \mathbb{S}^1$, we define:*

$$\textsc{Indeg}(v, s) = |\{(v, v') \in E \mid s \cdot v' \leq s \cdot v\}|.$$

In other words, the indegree of v is the number of edges incident to v that lie below v, with respect to direction s; see Figure 2.

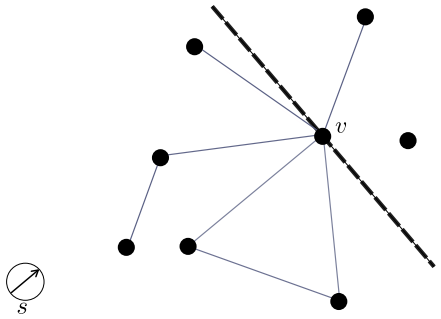


Figure 2: A plane graph with a dashed line drawn intersecting v in the direction perpendicular to s. Since four edges incident to v lie below v, with respect to direction s, $\textsc{Indeg}(v, s) = 4$.

Next, we prove that given a direction, we can determine the indegree of a vertex:

Lemma 7 (Indegree from Diagram) *Let K be a plane graph with vertex set V. Let $s \in \mathbb{S}^1$ be such that no two vertices are at the same height with respect to s, i.e., $|\mathbb{L}(s, V)| = n$. Let $\mathcal{D}_0(s)$ and $\mathcal{D}_1(s)$ be the zero- and one-dimensional persistence diagrams resulting from the height filtration F_s on K. Then, for all $v \in V$,*

$$\begin{aligned}\textsc{Indeg}(v, s) =&|\{(x, y) \in \mathcal{D}_0(s) \mid y = v \cdot s\}|+\\ &|\{(x, y) \in \mathcal{D}_1(s) \mid x = v \cdot s\}|.\end{aligned}$$

Proof. Let $v, v' \in V$ such that $s \cdot v' < s \cdot v$, i.e., the vertex v' is lower in direction s than v. Then, by Remark 1, if $(v, v') \in E$, it must be one of the following in the filter of K defined by s: (1) an edge that joins two disconnected components; or (2) an edge that creates a one-cycle. Since edges are added to a filtration at the height of the higher vertex, we see (1) as a death in $\mathcal{D}_0(s)$ and (2) as a birth in $\mathcal{D}_1(s)$, both at height $s \cdot v$. In addition, each finite death in $\mathcal{D}_0(s)$ and every birth in $\mathcal{D}_1(s)$ at time $s \cdot v$ must correspond to an edge, i.e., edges are the only simplices that can cause these events. Then, the set of edges of types (1) and (2) is $\{(x, y) \in \mathcal{D}_0(s) \mid y = v \cdot s\}$ and $\{(x, y) \in \mathcal{D}_1(s) \mid x = v \cdot s\}$, respectively. The size of the union of these two multi-sets is equal to the number of edges starting at v' lower than v in direction s and ending at v, as required. □

In order to decide whether an edge (v, v') exists between two vertices, we look at the degree of v as seen by two close directions such that v' is the only vertex in what we call a *bow tie at v*:

Definition 8 (Bow Tie) *Let $v \in V$, and choose $s_1, s_2 \in \mathbb{S}^1$. Then, a bow tie at v is the symmetric difference between the half planes below v in directions s_1*

and s_2. The width *of the bow tie is half of the angle between s_1 and s_2.*

Because no three vertices in our plane graph are collinear, for each pair of vertices $v, v' \in V$, we can always find a bow tie centered at v that contains the vertex v' and no other vertex in V; see Figure 3. We

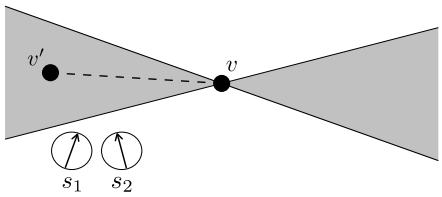


Figure 3: A bow tie B at v, denoted by the gray shaded area. B contains exactly one vertex, v', so the only potential edge in B is (v, v').

use bow tie regions that only contain one vertex, v' other than the center, v to determine if there exists an edge between v and v'; see Figure 4. We then use Lemma 9 to decide if the edge (v, v') exists in our plane graph.

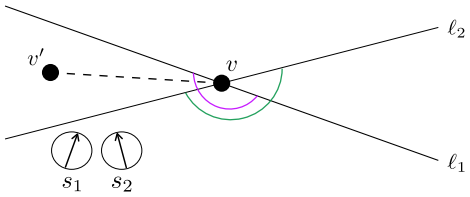


Figure 4: A bow tie at v that contains the vertex v' and no other vertices. In order to determine if there exists an edge between v and v', we compute $\textsc{Indeg}(v, s_1)$ and $\textsc{Indeg}(v, s_2)$, i.e., the number of edges incident to v in the purple and green arcs, respectively. An edge exists between v and v' if and only if $|\textsc{Indeg}(v, s_1) - \textsc{Indeg}(v, s_2)| = 1$.

Lemma 9 (Edge Existence) *Let K be a plane graph with vertex set V and edge set E. Let $v, v' \in V$. Let $s_1, s_2 \in \mathbb{S}^1$ such that the bow tie B at v defined by s_1 and s_2 satisfies: $B \cap V = v'$. Then,*

$$|\textsc{Indeg}(v, s_1) - \textsc{Indeg}(v, s_2)| = 1 \iff (v, v') \in E.$$

Proof. Since edges in K are straight lines, any edge incident to v will either fall in the bow tie region B or will be on the same side (above or below) of both lines. Let A be the set of edges incident on v and below both lines; that is $A = \{(v, v_*) \in E \mid s_i \cdot v_* < s_i \cdot v\}$. Furthermore, suppose we split the bowtie into the two infinite cones. Let B_1 be the set of edges in one cone and B_2 be the set of edges in the other cone. We note that $||B_1| - |B_2||$ is equal to one if there is an edge $(v, v') \in E$ with $v' \in B_1$ or $v' \in B_2$ and zero otherwise. Then, by definition of indegree,

$$\begin{aligned} |\textsc{Indeg}(v, s_1) - \textsc{Indeg}(v, s_2)| &= ||A| + |B_1| - |A| - |B_2|| \\ &= ||B_1| - |B_2|| \\ &= |V \cap B|, \end{aligned}$$

which holds if and only if $(v, v') \in E$. Then $|\textsc{Indeg}(v, s_1) - \textsc{Indeg}(v, s_2)| = 1 \iff (v, v') \in E$, as required. □

Next, we prove that we can find the embedding of the edges in the original graph using $O(n^2)$ directional persistence diagrams.

Theorem 10 (Edge Reconstruction) *Let K be a plane graph, with vertex set V and edge set E. If V is known, then we can compute E using $O(n^2)$ directional persistence diagrams.*

Proof. We prove this theorem constructively. Intuitively, we construct a bow tie for each potential edge and use Lemma 9 to determine if the edge exists or not. Our algorithm has three steps for each pair of vertices in V: Step 1 is to determine a global bow tie width, Step 2 is to construct suitable bow ties, and Step 3 is to compute indegrees. See Appendix A for an example of walking through the reconstruction.

Step 1: Determine bow tie width. For each vertex $v \in V$, we consider the cyclic ordering of the points in $V \setminus \{v\}$ around v. We define $\theta(v)$ to be the minimum angle between all adjacent pairs of lines through v; see Figure 5, where the angles between adjacent lines are denoted θ_i. Finally, we choose θ less than $\min_{v \in V} \theta(v)$. By Lemmas 1 and 2 of [12], we compute the cyclic orderings for all vertices in V in $O(n^2)$ time. Since computing each $\theta(v)$ is $O(n)$ time once we have the cyclic ordering, the runtime for this step is $O(n^2)$.

Step 2: Constuct bow ties. For each pair of vertices $(v, v') \in V \times V$ such that $v \neq v'$, let s be a unit vector perpendicular to vector $(v' - v)$, and let s_1, s_2 be the two unit vectors that form angles $\pm\theta$ with s. Let B be the bow tie between $\ell(s_1, h_{s_1}(v))$ and $\ell(s_2, h_{s_2}(v))$. Note that by the construction, B contains exactly one point from V, namely v'.

Step 3: Compute indegrees. Using B as the bow tie in Lemma 7, compute $\textsc{Indeg}(v, s_1)$ and $\textsc{Indeg}(v, s_2)$. Then, using Lemma 9, we determine whether (v, v') exists by checking if $|\textsc{Indeg}(v, s_1) - \textsc{Indeg}(v, s_2)| = 1$. If it does, the edge exists; if not, the edge does not.

Repeating for all vertex pairs requires $O(n^2)$ diagrams and discovers the edges of K. □

The implications of Theorem 5 and Theorem 10 lead to our primary result. We can find the embedding of the

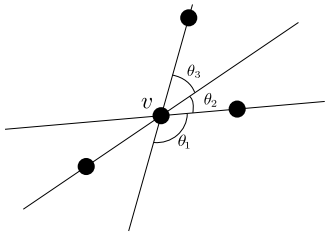


Figure 5: Using a vertex set of a plane graph to construct a bow tie at vertex, v. Lines are drawn through all vertices and then angles are computed between all adjacent pairs of lines. The smallest angle is chosen as $\theta(v)$. Here, $\theta(v) = \theta_2$.

vertices V by Theorem 5 using three directional persistence diagrams. Furthermore, we can discover edges E with $O(n^2)$ directional persistence diagrams by Theorem 10. Thus, we can reconstruct all edges and vertices of a one-dimensional simplicial complex:

Theorem 11 (Plane Graph Reconstruction) *Let K be a plane graph with vertex set V and edge set E. The vertices, edges, and exact embedding of K can be determined using persistence diagrams from $O(n^2)$ different directions.*

6 Discussion

In this paper, we provide an algorithm to reconstruct a plane graph with n vertices embedded in $\mathbb{R}^2$. Our method uses $O(n^2)$ persistence diagrams by first determining vertex locations using only three directions, and, second, determining edge existence based on height filtrations and vertex degrees. Moreover, if we have an oracle that can return a diagram given a direction in $O(T)$ time, then constructing the vertices takes $O(T+n\log n)$ and reconstructing the edges takes takes $O(Tn^2)$ time.

This approach extends to several avenues for future work. First, we plan to generalize these reconstruction results to higher dimensional simplicial complexes. We can show that the vertices of a simplicial complex K in $\mathbb{R}^d$ can be reconstructed in $O(dT + n^d)$ time using the complete arrangement of hyperplanes and $(d + 1)$ directional persistence diagrams. We conjecture that this bound can be improved to $O(dT + dn\log n)$ using the same observation that allows us to do the final step of the vertex reconstruction in linear time. We have a partial proof in this direction, and can likewise extend the bow tie idea to higher dimensions, but the number of directions grows quite quickly. Second, we conjecture that we can reconstruct these plane graphs with a subquadratic number of height filtrations by utilizing more information from each height filtration. Third, we suspect a similar approach can be used to infer other graph metrics, such as classifying vertices into connected components. Intuitively, determining such metrics should require fewer persistence diagrams than required for a complete reconstruction. Finally, we plan to provide an implementation for reconstruction that integrates with existing TDA software.

Acknowledgements This material is based upon work supported by the National Science Foundation under the following grants: CCF 1618605 (BTF, SM), DBI 1661530 (BTF, DLM, LW), DGE 1649608 (RLB), and DMS 1664858 (RLB, BTF, AS, JS). Additionally, RM thanks the Undergraduate Scholars Program. All authors thank the CompTaG club at Montana State University and the reviewers for their thoughtful feedback on this work.

References

[1] Mahmuda Ahmed, Sophia Karagiorgou, Dieter Pfoser, and Carola Wenk. Map construction algorithms. In *Map Construction Algorithms*, pages 1–14. Springer, 2015.

[2] Mahmuda Ahmed and Carola Wenk. Constructing street networks from GPS trajectories. In *European Symposium on Algorithms*, pages 60–71. Springer, 2012.

[3] Justin Curry, Sayan Mukherjee, and Katharine Turner. How many directions determine a shape and other sufficiency results for two topological transforms. arXiv:1805.09782, 2018.

[4] Tamal K. Dey, Jiayuan Wang, and Yusu Wang. Graph reconstruction by discrete Morse theory. arXiv:1803.05093, 2018.

[5] Herbert Edelsbrunner and John Harer. *Computational Topology: An Introduction*. American Mathematical Society, 2010.

[6] Xiaoyin Ge, Issam I Safa, Mikhail Belkin, and Yusu Wang. Data skeletonization via Reeb graphs. In *Advances in Neural Information Processing Systems*, pages 837–845, 2011.

[7] Robert Ghrist, Rachel Levanger, and Huy Mai. Persistent homology and Euler integral transforms. arXiv:1804.04740, 2018.

[8] Chad Giusti, Eva Pastalkova, Carina Curto, and Vladimir Itskov. Clique topology reveals intrinsic geometric structure in neural correlations. *Proceedings of the National Academy of Sciences*, 112(44):13455–13460, 2015.

[9] Sophia Karagiorgou and Dieter Pfoser. On vehicle tracking data-based road network generation.

In *SIGSPATIAL '12: Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, pages 89–98. ACM, 2012.

[10] Balázs Kégl, Adam Krzyzak, Tamás Linder, and Kenneth Zeger. Learning and design of principal curves. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(3):281–297, 2000.

[11] Yongjin Lee, Senja D. Barthel, Paweł Dłotko, S. Mohamad Moosavi, Kathryn Hess, and Berend Smit. Quantifying similarity of pore-geometry in nanoporous materials. *Nature Communications*, 8:15396, 2017.

[12] David L. Millman and Vishal Verma. A slow algorithm for computing the Gabriel graph with double precision. *CCCG '11: Proceedings of the 23rd Annual Canadian Conference on Computational Geometry*, 2011.

[13] Monica Nicolau, Arnold J. Levine, and Gunnar Carlsson. Topology based data analysis identifies a subgroup of breast cancers with a unique mutational profile and excellent survival. *Proceedings of the National Academy of Sciences*, 108(17):7265–7270, 2011.

[14] Abbas H. Rizvi, Pablo G. Camara, Elena K. Kandror, Thomas J. Roberts, Ira Schieren, Tom Maniatis, and Raul Rabadan. Single-cell topological RNA-seq analysis reveals insights into cellular differentiation and development. *Nature Biotechnology*, 35(6):551, 2017.

[15] Katharine Turner, Sayan Mukherjee, and Doug M. Boyer. Persistent homology transform for modeling shapes and surfaces. *Information and Inference: A Journal of the IMA*, 3(4):310–344, 2014.

[16] Ying Zheng, Steve Gu, Herbert Edelsbrunner, Carlo Tomasi, and Philip Benfey. Detailed reconstruction of 3d plant root shape. *Proceedings of the IEEE International Conference on Computer Vision*, pages 2026–2033, 11 2011.

Appendix

A Example of Reconstructing a Plane Graph

We give an example of reconstructing a plane graph. Consider the complex given in Figure 6.

Vertex Reconstruction First, we find vertex locations using the algorithm described in Section 4. We need to choose pairwise linearly independent vectors s_1, s_2 and s_3 such that only n three-way intersections in $A = \mathbb{L}(s_1, V) \cup \mathbb{L}(s_2, V) \cup \mathbb{L}(s_3, V)$ exist; note that in this example, $n = 4$. Using the persistence diagrams from height filtrations in directions $s_1 = (0, 1)$ and $s_2 = (1, 0)$, we construct the set of lines $\mathbb{L}(s_1, V) \cup \mathbb{L}(s_2, V)$. This results in $n^2 = 16$ possible locations for the vertices at the intersections in A. We show these filtration lines and intersections in Figure 6b. Next, we compute the third direction s_3 using the algorithm outlined in Theorem 5. To do this, we need to find the greatest horizontal distance between two vertical lines, $d_1 = 2$ and the least vertical distance between two horizontal lines, $d_2 = 1$. Then, we use these to choose a direction s_3 perpendicular to $s_* = (d_1, \frac{d_2}{2}) = (2, \frac{1}{2})$ (e.g., $s_3 = (\frac{-1}{\sqrt{17}}, \frac{4}{\sqrt{17}}) \in \mathbb{S}^1$). Then, the four three-way intersections in $\mathbb{L}(s_1, V) \cup \mathbb{L}(s_2, V) \cup \mathbb{L}(s_3, V)$ identify all Cartesian coordinates of the original complex. We show filtration lines from all three directions in Figure 6c.

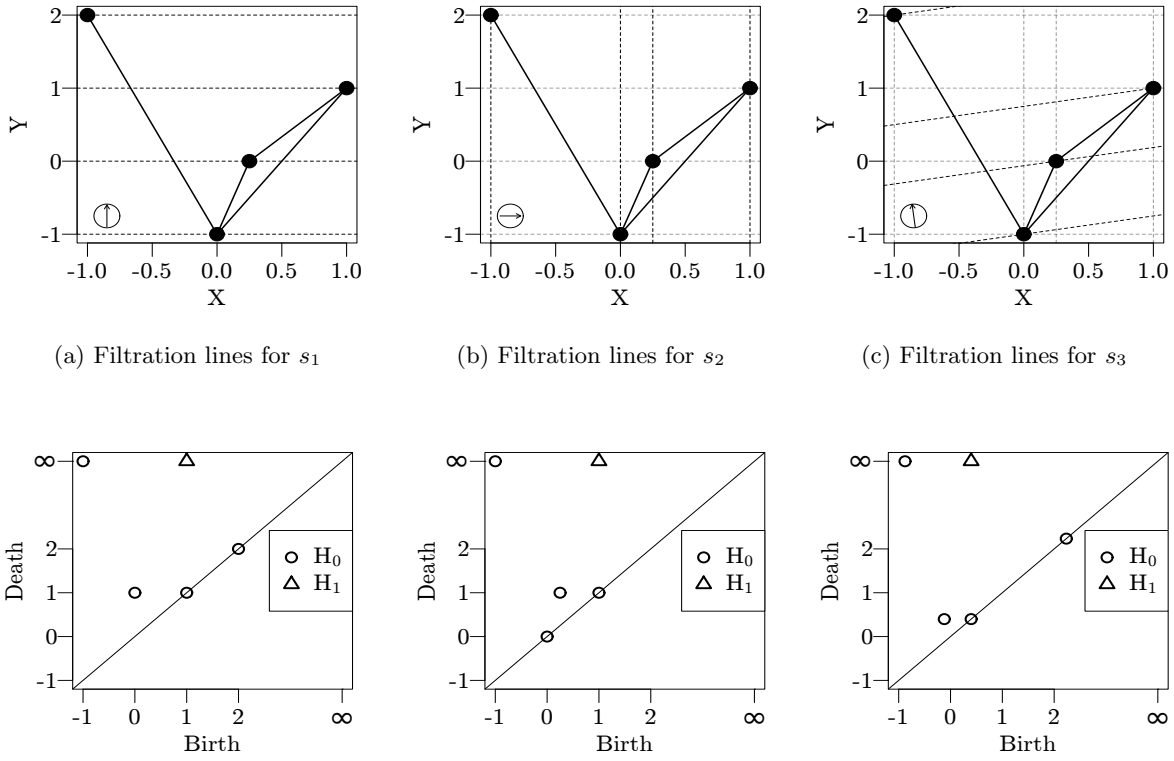
Edge Reconstruction Next, we reconstruct all edges as described in Section 5. In order to do so, we first find the θ we will use to construct bow ties. To do this, we examine each vertex v in turn, finding $\theta(v)$, the minimum angle between adjacent pairs of lines through v and $v' \in V - \{v\}$. Ordering v by increasing x-coordinate, we find $\theta(v)$ to be approximately $0.237, 0.219, 0.399$, and 0.180 radians, respectively. Then, we take θ to be less than the minimum of these, i.e. $< 0.180 radians$.

Now, for each of the $\frac{n(n-1)}{2}$ pairs of vertices $(v, v') \in V^2$, we construct a bow tie B and then use this bow tie to determine whether an edge exists between the two vertices. We go through two examples: one for a pair of vertices that does have an edge between, and one for a pair that does not. First, consider the pair $v = (0.25, 0)$ and $v' = (1, 1)$. To construct their bow tie, we first find the unit vector perpendicular to the vector that points from v to v', which is $s = (-0.8, 0.6)$. Now, we find s_1, s_2 such that they make angles θ with s. We choose $s_1 = (-0.956, 0.293)$ and $s_2 = (-0.433, 0.902)$. Now, by Lemma 7, we can use the persistence diagrams from these two directions to compute $\textsc{Indeg}(v, s_1)$ and $\textsc{Indeg}(v, s_2)$. We observe that $\mathcal{D}_0(s_1)$ contains exactly one birth-death pair (x, y) such that $y = v \cdot s_1$ and $\mathcal{D}_1(s_1)$ has one birth-death pair such that $x = v \cdot s_1$. Thus, $\textsc{Indeg}(v, s_1) = 2$. On the other hand, $\mathcal{D}_0(s_2)$ contains exactly one birth-death pair (x, y) such that $y = v \cdot s_2$, but $\mathcal{D}_1(s_2)$ contains no birth-death pair such that $x = v \cdot s_2$. So $\textsc{Indeg}(v, s_2) = 1$. Now, since $|\textsc{Indeg}(v, s_1) - \textsc{Indeg}(v, s_2)| = 1$, we know that $(v, v') \in E$, by Lemma 9.

For the second example, consider the pair of vertices $v = (0.25, 0)$ and $v' = (-1, 2)$. Again, we construct their bow tie by finding a unit vector perpendicular to the vector pointing from v to v'. We choose this $s = (0.848, 0.530)$. Then, the s_1 and s_2 which form angle $\theta < 0.180$ radians (e.g $\theta = .170$)

with s are $s_1 = (0.968, 0.248)$ and $s_2 = (0.472, 0.882)$. Again by Lemma 7, we examine the zero- and one-dimensional persistence diagrams from these two directions to compute the indegree from each direction for vertex v. In $\mathcal{D}_0(s_1)$, we have one pair (x, y) which dies at $y = v \cdot s_1$, but in $\mathcal{D}_1(s_1)$, no pair is born at $x = v \cdot s_1$. So $\textsc{Indeg}(v, s_1) = 1$. We see the exact same for s_2, which means that $|\textsc{Indeg}(v, s_1) - \textsc{Indeg}(v, s_2)| = 0$. Since Lemma 9 tells us that we have an edge between v and v' only if the absolute value of the difference of indegrees is one, we know that there is no edge between vertices $(0.25, 0)$ and $(-1, 2)$.

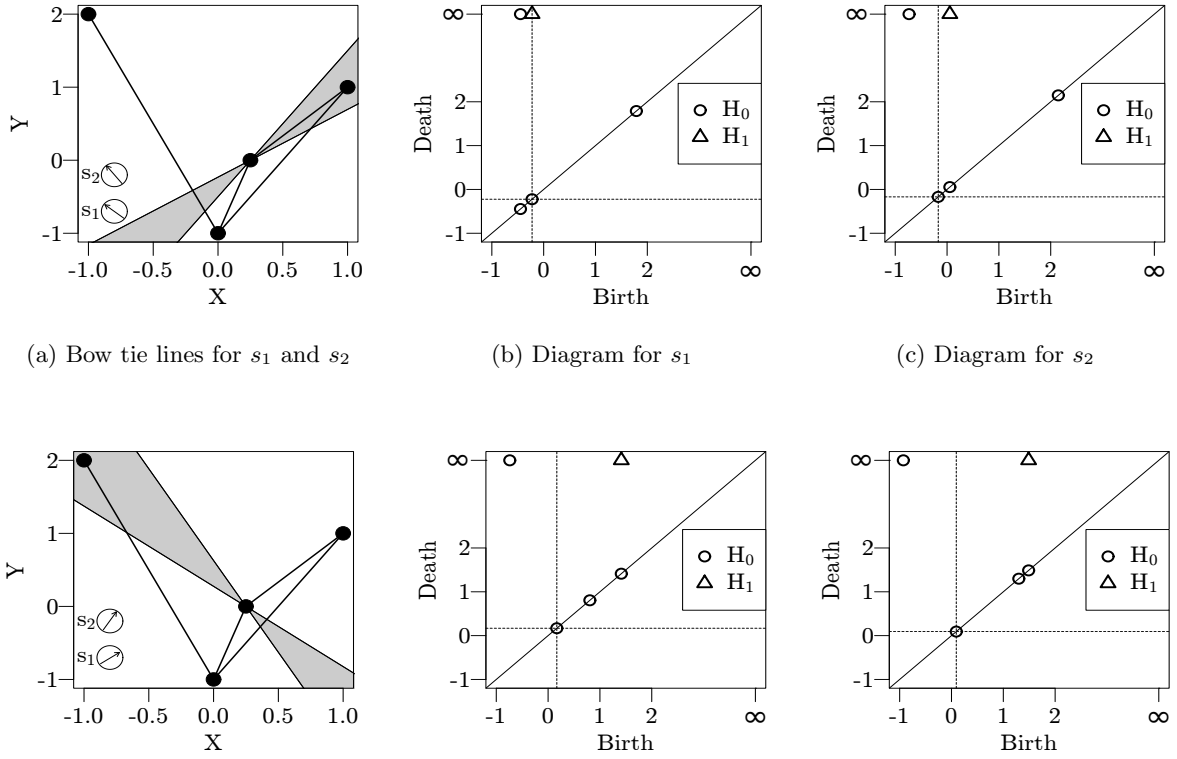
In order to reconstruct all edges, we perform the same computations for all pairs of vertices.



(a) Filtration lines for s_1 (b) Filtration lines for s_2 (c) Filtration lines for s_3

(d) Diagrams for s_1 (e) Diagrams for s_2 (f) Diagrams for s_3

Figure 6: Example of vertex reconstruction from three directions, s_1, s_2 and s_3 with corresponding persistence diagrams built for height filtrations from these directions. The filtration lines are the dotted lines superimposed over the complex.



(a) Bow tie lines for s_1 and s_2 (b) Diagram for s_1 (c) Diagram for s_2

(d) Bow tie lines for s_1 and s_2 (e) Diagram for s_1 (f) Diagram for s_2

Figure 7: Example of edge reconstruction for two edges. The first edge (top row) exists while the second edge (bottom row) does not. The bow tie is given on the left while the persistence diagrams $\mathcal{D}_0(s_1)$ and $\mathcal{D}_1(s_1)$ are given in the middle and the persistence diagrams $\mathcal{D}_0(s_2)$ and $\mathcal{D}_1(s_2)$ are given on the right. The dotted lines indicate $v \cdot s_1$ and $v \cdot s_2$ in diagrams for s_1 and s_2 respectively.

Sto-Stone is NP-Complete

Addison Allen * Aaron Williams †

Abstract

Sto-Stone is a paper-and-pencil puzzle created by Japanese publisher Nikoli. A puzzle consists of an m-by-n grid whose squares are partitioned into connected 'rooms' each of which may have an associated number. The solver shades in squares of the grid, which form maximal 'stones' based on orthogonal connectivity. The goal is to shade squares so that (a) each room contains one stone, (b) individual stones do not cross between rooms, (c) numbered rooms contain a stone with exactly that number of squares, and (d) when the stones are "dropped" downward they perfectly fill the bottom half of the grid. We show that Sto-Stone is NP-complete. This is also true when rule (d) is weakened or omitted.

1 Introduction

This article proves the NP-completeness of a new paper-and-pencil puzzle by Japanese publisher Nikoli. The puzzle is *Sto-Stone* (ストストン) and it was introduced in *Puzzle Communication magazine* Volume 156 [1].

When discussing individual grid squares we use *adjacent* and *connected* to mean orthogonally adjacent and orthogonally connected, respectively.

1.1 Rules of the Puzzle

Sto-Stone is played on an m-by-n grid where m is even. The grid's squares are partitioned into connected "rooms" and the *size* of a room is its number of squares. A room may have a positive number w written in one of its squares, and in this case its *required weight* or *requirement* is w. A grid with these properties is a *board*.

The solver interacts with the puzzle by shading individual squares. The shaded squares partition into *stones* based on connectivity. In other words, any two shaded squares that are adjacent belong to the same stone. The *weight* of a stone is its number of shaded squares. The goal is to create stones subject to the following rules:

(S1) There is exactly one stone in each room. That is, in each room there are shaded squares and these squares are connected.

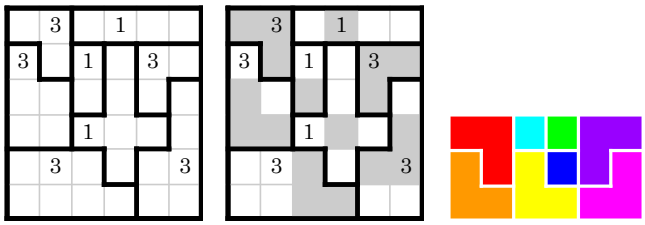
(S2) Shaded squares in different rooms are not adjacent. That is, stones can't be inside more than one room.

(S3) Rooms with requirement w have a weight w stone. That is, a room labeled w has w shaded squares.

(S4) When all stones are "dropped" downward they fill the bottom half of the grid with no gaps.

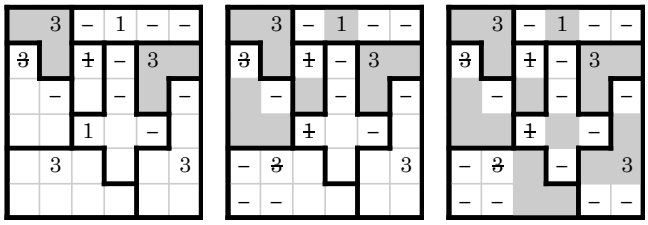
Rule (S4) requires clarification. When stones are dropped they move down as if influenced by gravity. Stones do not change shape when they are dropped, and all room boundaries are ignored during this time.

Figure 1 has a sample puzzle and Figure 2 illustrates the solving process. Figure 1 (c) visually verifies (S4), and - denotes a square that cannot be shaded.



(a) Board. (b) Solution. (c) Drop check.

Figure 1: The corrected version of Sto-Stone Puzzle 4 from Puzzle Communication Volume 162 [2].



(a) (S3) forces the above stones; (S2) contributes the −. (b) (S3) forces the above stones; (S2) and (S4) contribute the −. (c) (S3) forces the bottom-middle stones; (S4) forces the bottom-right.

Figure 2: Solving the Sto-Stone puzzle in Figure 1.

1.2 Drop Rules: Stone, Sand, Silt

We refer to (S4) as the *stone drop rule*. We also define a weaker *sand drop rule* as follows.

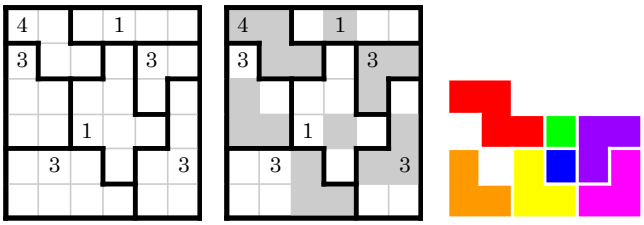
(s4) There are $\frac{m}{2}$ shaded squares in every column.

*Bard College at Simon's Rock, Massachusetts, aallen15@simons-rock.edu

†Bard College at Simon's Rock, Massachusetts, awilliams@simons-rock.edu

Notice that (s4) differs from (S4) in that it ignores the shape of the stones. In other words, the shaded squares are dropped independently like individual grains of sand. We refer to the lack of a drop rule as the *silt drop rule*. In other words, the shaded squares linger in the air like fine grains of silt.

The drop rule in Sto-Stone is somewhat unusual among Nikoli puzzles, and it has led to some initial confusion among puzzle designers, solvers, and academics. Figure 1 actually contains a *corrected* version of Sto-Stone Puzzle 4 from Puzzle Communication Volume 162 [2]. The originally published puzzle shown in Figure 3 can only be solved with the weaker drop rules.



(a) Board. (b) Solution with respect to SAND but not STONE. (c) Drop rule (S4) fails, but (s4) would pass.

Figure 3: The original version of Sto-Stone Puzzle 4 from Puzzle Communication Volume 162 [2].

The error was announced by `@nikoli_official` on twitter [8]. In response, `@postpostdoc` posted an early version of this article [9]. However, the authors did not properly understand rule (S4) at that time, and only established the hardness of (S1)-(S3) with (s4). The early version also allowed 'empty' rooms with no stone, and $w = 0$ requirements, which we now believe are invalid.

1.3 Decision Problems

We formalize three different puzzles based on the type of drop rule that is used. Each of these puzzles has an associated decision problems that takes a board B as input and answers 'yes' or 'no' depending on whether it can be solved using the rules for that puzzle.

- Nikoli's *Sto-Stone puzzle* uses rules (S1)-(S3) and drop rule (S4). The decision problem is STONE(B).
- The *Sto-Sand puzzle* uses rules (S1)-(S3) and drop rule (s4). The decision problem is SAND(B).
- The *Sto-Silt puzzle* uses rules (S1)-(S3) and no drop rule. The decision problem is SILT(B).

If STONE(B) is 'yes', then SAND(B) is 'yes'. Similarly, if SAND(B) is 'yes', then SILT(B) is 'yes'. Figure 3 gave an example board B in which STONE(B) is 'no' and both SAND(B) and SILT(B) are 'yes'.

All three decision problems are in NP because shading an m-by-n board can be done with $m \cdot n$ binary guesses, and each rule can be checked in $O(mn)$-time.

Remark 1 *The decision problems* STONE, SAND, *and* SILT *are all in NP.*

1.4 Popularity

Nikoli is currently promoting three new puzzles including *Sun or Moon* (月か太), *Pencils* (ペンシルズ), and Sto-Stone (ストストン). During a November 2017 poll held on twitter by `@nikoli_official`, the Sto-Stone puzzle ranked behind Sun or Moon in popularity. However, this has changed in a more recent poll from May 2018, as seen in Figure 4.

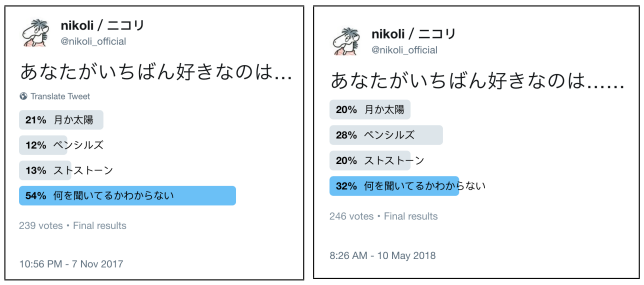


Figure 4: The popularity of three new Nikoli puzzles, where the bottom option translates to "I do not know".

Establishing the hardness of Nikoli puzzles has also been a popular pursuit in academia. An excellent resource on this general topic is *Games, Puzzles, and Computation* by Hearn and Demaine [6].

1.5 Outline

The article is organized as follows. Section 2 defines the NP-complete problem that we will use as a source problem. Section 3 introduces our gadgets and other preliminaries. Sections 4, 5, and 6 proves that SILT, SAND, and STONE are NP-complete, respectively. Section 7 concludes with final remarks and open problems.

2 Source Problem

This section defines the satisfiability problem used in our reduction. We also describe a slight variation to its standard representation.

2.1 Planar Monotone Rectilinear 3SAT

A *(Boolean) variable* is a variable that can be assigned TRUE or FALSE. If x_i is a variable, then its *positive literal* is x_i, and its *negative literal* is $\neg x_i$. A Boolean formula ϕ is in *3 conjunctive normal form (3CNF)* if it equals $C_1 \wedge C_2 \wedge \ldots \wedge C_m$ where each *clause* C_i has the form $(\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$ and each $\ell_{i,j}$ is a literal.

A clause is *positive* or *negative* if it has only positive or negative literals, respectively. The 3CNF formula ϕ is *monotone* if each clause is either positive or negative.

A 3CNF formula is *planar* if the bipartite incidence graph of variables and clauses is planar. A *rectilinear embedding* of a planar monotone 3CNF formula is a drawing on a grid with the following properties:

- Variables and clauses are horizontal line segments.
- Vertical line segments connect variables to clauses.
- Variable line segments are on the same horizontal line called the *variable line*.
- Positive clauses are above the variable line, and negative clauses are below.

Rectilinear embeddings are drawn with their horizontal line segments vertically extended as in Figure 5.

The decision problem PLANAR MONOTONE RECTILINEAR 3SAT (PMR3SAT) takes a rectilinear embedding of a planar monotone 3CNF formula ϕ as input. A *'yes' instance* occurs when the variables can be assigned so that ϕ evaluates to TRUE. In this case, ϕ is *satisfiable*. Otherwise, ϕ is a *'no' instance* and is *unsatisfiable*. For brevity, we often refer to the input of PMR3SAT as the Boolean formula ϕ as opposed to a rectilinear embedding of it. Theorem 1 is by de Berg and Khosravi [4].

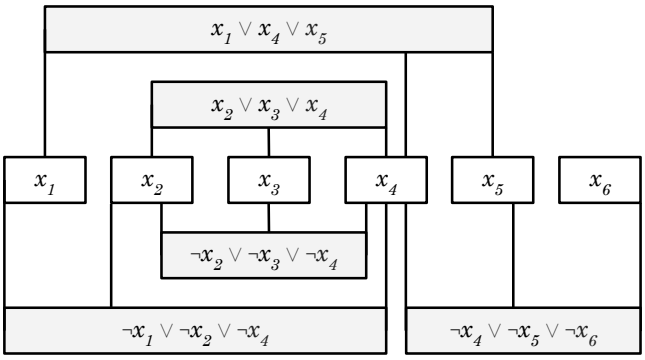


Figure 5: A 'yes' instance of PMR3SAT with $\phi = (x_1 \vee x_4 \vee x_5) \wedge (x_2 \vee x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_4) \wedge (\neg x_4 \vee \neg x_5 \vee \neg x_6)$.

Theorem 1 ([4]) PMR3SAT *is NP-complete.*

When working with PMR3SAT we assume that the variables are ordered from left-to-right as $x_1, x_2, \ldots, x_n$ in the embedding. We also arrange each clause C as $(x_i \vee x_j \vee x_k)$ or $(\neg x_i \vee \neg x_j \vee \neg x_k)$ with the distinct indices satisfying $i < j < k$, and we refer to x_i, x_j, and x_j as the *left*, *middle*, and *right* literals in C, respectively.

2.2 Bent Representation

We will find it helpful to make the following cosmetic adjustments to the input to the PMR3SAT problem:

- Shrink each clause line by moving its left end and right end closer together by any small amount;
- Connections from clauses to positive left literals are redrawn as ⌜ lines. Similarly, negative left literals, positive right literals, and negative right literals are redrawn with ⌞, ⌝, and ⌟ lines, respectively.

We refer to this modified embedding as *bent rectilinear representation* since two-thirds of the connecting lines have a 90^o bend. Figure 6 shows the result of adjusting Figure 5 in this way.

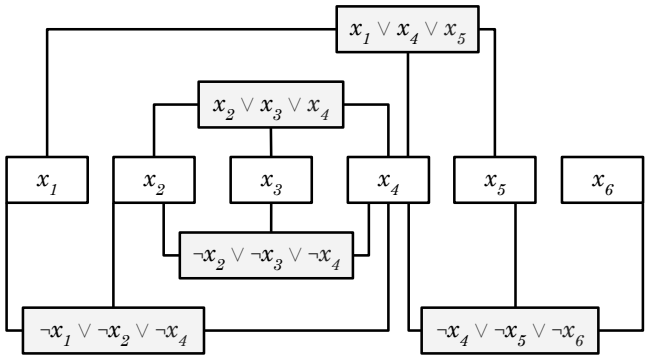


Figure 6: A bent embedding of Figure 5.

3 Gadgets and Preliminaries

In this section we introduce some conventions and terminology, and then present gadgets for Sto-Silt.

3.1 Grid Parity

A square in location (x,y) of the grid is *even* or *odd* based on the sum $x + y$, where the top-left square is in location (1, 1). In other words, the grid has an underlying even/odd checkerboard pattern.

Each room we create will have size at least two, so it will contain at least one even and one odd square. Therefore, we can use the following convention to make our figures more readable: If a room has required weight w, then w is written in a square whose parity is the same as w. In other words, odd requirements are written in odd locations, and even requirements are written in even locations. This convention extends back to Figure 1.

3.2 Rooms

In a partially shaded board B a room with a requirement w is *satisfied* if it has a stone of weight w, and otherwise it is *unsatisfied*. Furthermore, a room is *unsatisfiable* if it is impossible to satisfy the room by shading in additional squares while respecting the rules.

Rooms with size s and requirement w have *type* $s.w$. Our reductions will be primarily restricted to the following special room types.

- A room of type 2.1 is a *binary room*. A binary room can be satisfied in two ways.
- A room of type 3.2 is a *trinary room*. A trinary room can be satisfied in two ways.

- A room of type 3.1 is a *ternary room*. A ternary room can be satisfied in three ways.
- A room with no requirement is a *wild-card room*. These rooms do require at least one shaded square.

In the following subsections we will create gadgets that propagate decisions made at certain binary rooms to other binary rooms. When discussing these gadgets we use the following terminology and conventions.

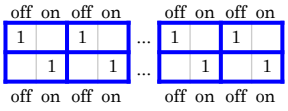
- An *input room* is a horizontal binary room with an *'on' square* to the right of an *'off' square*. A *positive* or *negative* input room has its 'on' square in an even or odd position, respectively.
- An *output room* is a horizontal or vertical binary room with specified 'on' and 'off' squares.

Input and output rooms are coloured red and blue, respectively. These rooms are 'on' if a stone is in their 'on' square, and are 'off' if a stone is in their 'off' square.

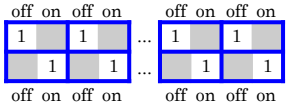
To simplify our figures we assume that empty regions on a board are wild-card rooms which are not drawn.

3.3 Variable Gadget

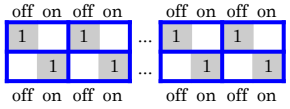
A variable gadget is designed to be satisfiable in one of two ways. Furthermore, this choice must be duplicatable so that it can be passed to any number of clause gadgets. To accomplish these goals we create a cycle of binary rooms. Our *variable gadget of width w* consists of a *positive row* with w positive input rooms, and below it is a *negative row* with w negative input rooms, as shown in Figure 7 (a). The top-left square is always placed on an even grid location.



(a) Variable gadget.



(b) Positive state.



(c) Negative state.

Figure 7: The variable gadget in (a) can be satisfied in exactly two ways (b)–(c).

Shading a single square anywhere in the gadget forces the entire gadget to be satisfied in a particular manner. The precise behavior and state of the gadget is defined Remark 2 and illustrated in Figure 7 (b)–(c).

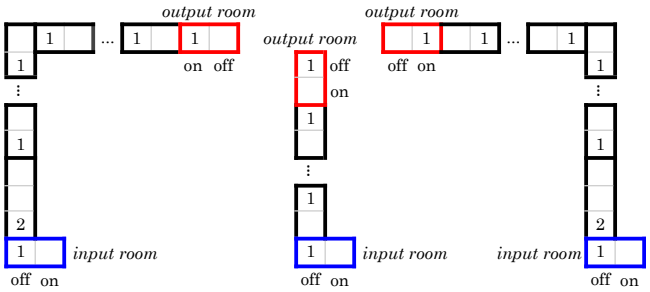
Remark 2 *In a solved Sto-Silt board, a variable gadget must be satisfied in one of two ways:*

- *Its* positive state *has positive input rooms 'on' and negative input rooms 'off'.*
- *Its* negative state *has positive input rooms 'off' and negative input rooms 'on'.*

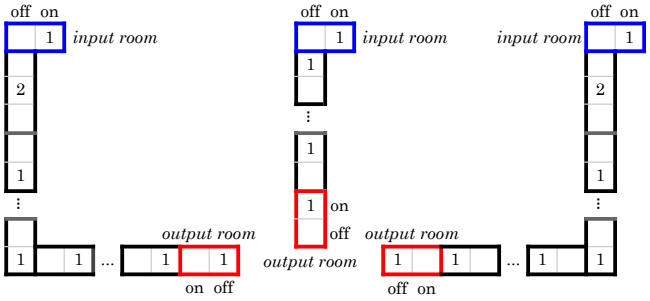
3.4 Wire Gadget

A wire gadget propagates the choice made in one binary room to another binary room. More specifically, the wire ensures a relationship between two specific squares on the board. Remark 3 outlines the main property of the wire gadget that we will construct.

Remark 3 *Binary input and output rooms that are connected by a wire in a solved Stone-Silt board have the following properties. If the input room is 'off', then the output room is also 'off'. If the input room is 'on', then the output room can be 'on' or 'off'.*



(a) Three types of positive wires.



(b) Three types of negative wires.

Figure 8: Wire gadgets connect an input room to an output room, and are paths of binary rooms and at most one trinary room. Shading the 'off' square of an input room forces the shading of the 'off' square in the connected output room.

Remark 3 is 'weak' since it only guarantees one direction, but this will be sufficient for our reduction. Now we define our wires with Figure 8 providing illustrations.

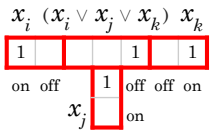
- A *positive/negative wire* is a path of binary and trinary rooms starting from the 'off' square of a positive/negative input room and ending at the 'on' square of a positive/negative output room.

A wire is *straight* if it travels vertically from an input room to a vertical output room. The other wires proceed vertically from an input room, then make a single right-turn or left-turn, and travel horizontally to a horizontal output room. The straight wires only use binary rooms, whereas the turning wires leave their output room with a single trinary room and then consist

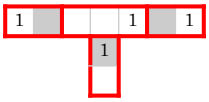
of binary rooms. As a result, the straight and turning wires reach their respective input rooms on opposite parity squares. All wire types are illustrated in Figure 8 and in each case it is easy to verify Remark 3.

3.5 Clause Gadget

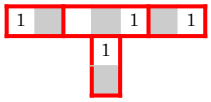
We base our clause gadgets on a ternary *clause room* and three binary output rooms. Each clause room is horizontal with output rooms adjacent to its left and right squares. In the *positive clause gadget* another output room is adjacent to the bottom of its middle square, whereas in the *negative clause gadget* it is adjacent to the top of its middle. The behavior of the gadget is given in Remark 4 and shown in Figures 9 and 10.



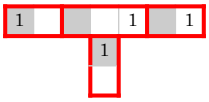
(a) Positive clause.



(b) Unsatisfiable if all 'off'.

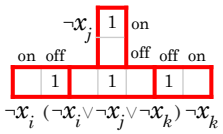


(c) Satisfiable if x_j is 'on'.

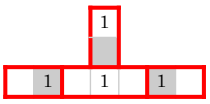


(d) Satisfiable if x_i is 'on'.

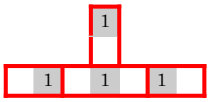
Figure 9: The positive clause gadget is satisfiable if and only if at least one input room is 'on'.



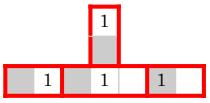
(a) Negative clause.



(b) Unsatisfiable if all 'off'.



(c) Satisfiable if x_j is 'on'.



(d) Satisfiable if x_i is 'on'.

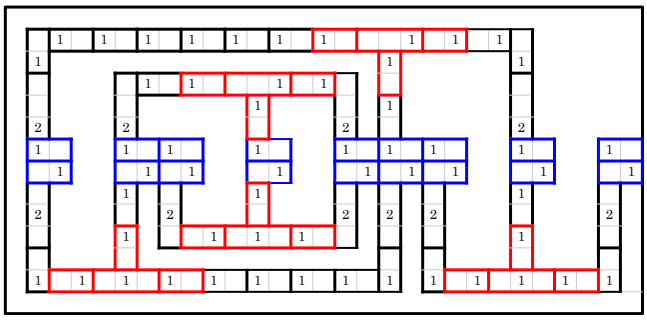
Figure 10: The negative clause gadget is satisfiable if and only if at least one input room is 'on'.

Remark 4 *In a solved Sto-Silt board, a clause gadget is satisfiable if and only if at least one of its adjacent output rooms is 'on'.*

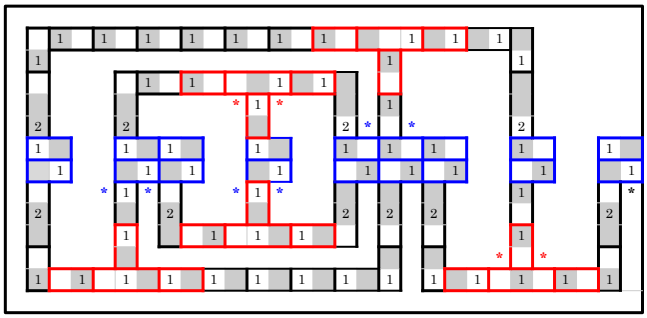
Note: In Theorem 2's proof we always satisfy clauses by shading their middle square if their middle wire is 'on'.

4 NP-Completeness of Sto-Silt

In this section we reduce PMR3SAT to SILT. An example of the reduction based on Figure 6 appears in Figure 11.



(a) The board $S(\phi)$ where wild-card rooms fill the area.



(b) A solution to $S(\phi)$ via $x_4 = x_5 =$ FALSE and $x_1 = x_2 = x_3 = x_6 =$ TRUE. The marked square * can be shaded in the outer wild-card room without violating (S2). Similarly, *'s and *'s mark all suitable squares along the straight wires.

Figure 11: The reduction of the PMR3SAT instance ϕ from Figure 6 to SILT(ϕ).

Suppose ϕ is an instance of PMR3SAT with p positive clauses and z negative clauses. Our reduction creates a board $B = S(\phi)$ whose rows are organized as follows:

- Row 1 is empty.
- Rows $2, 4, \ldots, 2p$ contain positive clause gadgets.
- Rows $2p+3$ and $2p+4$ contain variable gadgets.
- Rows $2p+7, 2p+9, \ldots, 2p+2z+5$ contain negative clause gadgets.
- Row $2p+2z+6$ is empty.

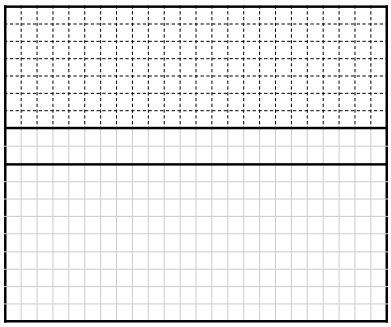
Now suppose that ϕ has p_i clauses with positive literal x_i, and z_i clauses with negative literal $\neg x_i$ for all i. The variable gadgets are sized and positioned as follows:

- Column 1 is empty.
- The variable gadgets are placed side-by-side starting from column 2 with two columns between them. Each x_i gadget is $max(p_i, n_i)$ input rooms wide.

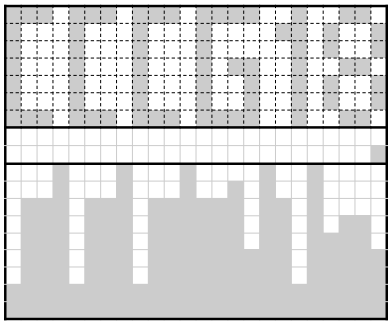
The width of the variable gadgets allow us to connect wires to distinct input rooms for each literal. In particular, the wire connected to the middle literal of a clause travels straight vertically to the middle of the corresponding clause gadget. Similarly, left and right literals enter the left and right sides of their clause gadgets.

Theorem 2 SILT *is NP-complete.*

Proof. Let ϕ be an instance of PMR3SAT. Remarks 3 and 4 imply that the variable, wire, and clause gadgets



(a) A 7-by-24 board B is extended to 16-by-24 board B' with 2-by-24 and 9-by-24 wild-card rooms.



(b) If SILT(B) is 'yes', then fill the added wild-card rooms to satisfy (S4), so SAND(B') and STONE(B') are 'yes'.

Figure 12: Reduction from SILT to SAND to STONE.

of $B = S(\phi)$ can be satisfied if and only if ϕ is satisfiable. The remaining detail is to show that the wild-card rooms of $S(\phi)$ can also be satisfied when ϕ is satisfiable. Since these rooms have no requirement we can satisfy them by shading any single square subject to (S2). See Figure 11 (b) for examples of the arguments below.

By the empty rows and column in B there is a wild-card room surrounding the gadgets. In this outer room we shade the rightmost column in row $2p+2$ or $2p+5$.

All other wild-card rooms border a straight wire.

- If this wire is on, then without loss of generality we can assume that its clause gadget is satisfied by shading its middle cell. Therefore, we can shade a square next to this clause gadget.
- If this wire is off, then we can shade a square next to its variable gadget.

Therefore, B is solvable if and only if ϕ is satisfiable. Theorem 1 and Remark 1 complete the proof. □

5 NP-Completeness of Sto-Sand

Now we prove that SAND is NP-complete by a reduction from SILT. Our strategy is to add rows to a given board so that the sand drop rule (s4) can be satisfied regardless of how the other squares are shaded. See Figure 12.

Theorem 3 SAND *is NP-complete.*

Proof. Suppose B is an m-by-n board that is an input to STONE. We create board B' of size $(2m+2)$-by-n by adding $m+2$ rows to the bottom of B. The additional rows are organized into two wild-card rooms as follows:

- A room of size 2-by-n is added below B.
- A room of size $(m+2)$-by-n is then added below.

Suppose that SILT(B) is a 'yes' instance. We now show that SAND(B') is 'yes'. We shade the top m rows of B' in any way that proves that SILT(B) is 'yes'. Then we shade the additional wild-card rooms as follows:

- The 2-by-n room has a single shaded square in its bottom-right corner.
- The bottom row of the larger room is fully shaded. If there are s shaded squares in kth column of B, then $m-s+1$ additional squares are shaded in its kth column from the bottom up. The only exception is the rightmost column which has one fewer square shaded.

This satisfies (S1)-(S3) and (s4), so SAND(B') is 'yes'.

Suppose that SILT(B) is a 'no' instance. In this case there is no way to satisfy rules (S1) – (S3) in the top m rows of B', hence, SAND(B') is 'no'.

Theorem 2 and Remark 1 complete the proof. □

6 NP-Completeness of Sto-Stone

Now we prove that STONE is NP-complete. We do this by analyzing the previous two reductions and showing that they create boards that can be solved using stones of width 1. In this context (s4) and (S4) are equivalent.

Theorem 4 STONE *is NP-complete.*

Proof. Let ϕ be an instance of PMR3SAT. Let $B = S(\phi)$ and B' be created as in Sections 4–5. We claim that ϕ is satisfiable if and only if STONE(B') is 'yes'.

Suppose that ϕ is satisfiable. By the proof of Theorem 2, SILT(B) is solvable using stones of width 1. By the proof of Theorem 3, this is also true for SAND(B'), except for the bottom stone which is already "bottom justified". Therefore, STONE(B') is also 'yes'.

Conversely, if ϕ is unsatisfiable, then SILT(B) is 'no' by Theorem 2, and so STONE(B') is also 'no'. □

7 Final Remarks

A *numberless Sto-Stone puzzle* is a Sto-Stone puzzle with no requirements. In other words, (S3) is ignored. What is the complexity of numberless Sto-Stone?

Jack Lance Puzzles [7] has several numberless examples. Numberless versions of other Nikoli puzzles have also been considered. For example, Shakashaka [5] and its numberless version [3] are both NP-complete. We note that *numberless Sto-Silt* is in P since (S1) and (S2) are satisfied by an empty board.

We thank the referees, one whom suggested *parameterized Sto-Stone* where the bottom k rows must fill.

References

[1] Puzzle Communication Nikoli Vol. 156. Sto-stone number 1, September 2016.

[2] Puzzle Communication Nikoli Vol. 162. Sto-stone number 4, March 2018.

[3] Aviv Adler, Michael Biro, Erik D. Demaine, Mikhail Rudoy, and Christiane Schmidt. Computational complexity of numberless Shakashaka. In *CCCG*, 2015.

[4] Mark de Berg and Amirali Khosravi. Optimal binary space partitions in the plane. In My T. Thai and Sartaj Sahni, editors, *Computing and Combinatorics*, pages 216–225, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[5] Erik Demaine, Yoshio Okamoto, Ryuhei Uehara, and Yushi Uno. Computational complexity and an integer programming model of Shakashaka. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E97.A(6):1213–1219, 2014.

[6] Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation*. A. K. Peters, Ltd., Natick, MA, USA, 2009.

[7] Jacob Lance. Puzzle 143 – stostone. `https://jacoblance.wordpress.com/2016/09/19/puzzle-143-stostone/`, September 2016.

[8] @nikoli_official. `https://twitter.com/nikoli_official/status/973516077970767873`, March 2018.

[9] @postpostdoc. `https://twitter.com/postpostdoc/status/973539335839469570`, March 2018.

A Paper on Pencils: A Pencil and Paper Puzzle
Pencils is NP-Complete

Daniel Packer * Sophia White † Aaron Williams ‡

Abstract

Pencils is a paper-and-pencil puzzle created by Japanese publisher Nikoli. A puzzle is an m-by-n grid where some squares hold a number or a pencil tip that is pointed in one of the cardinal directions. The goal is to draw 'pencils' that partition the squares of the grid. Each pencil occupies $2k+1$ squares for some $k \geq 1$. A k-pencil has a horizontal or vertical body of length k, a tip pointing away from one end of the body, and a lead that is a path of k squares starting from the tip. In addition, any number inside a body must match the body's size. We show that Pencils is NP-complete even when limited to 1-pencils and 2-pencils.

1 Introduction

This article proves the NP-completeness of a new paper-and-pencil puzzle by Japanese publisher Nikoli. The puzzle is *Pencils* (ペンシルズ) and it was introduced in *Puzzle Communication magazine* Volume 158 [2].

In this article we use *adjacent* and *connected* to mean orthogonally adjacent and orthogonally connected.

1.1 Rules of the Puzzle

Pencils is played on a *board*, which is an m-by-n grid where each square is initially empty or filled with a number or pencil tip pointed in a cardinal direction. A player draws *pencils* which each occupy $2k+1$ connected squares for some $k \geq 1$. A *k-pencil* consists of the following parts:

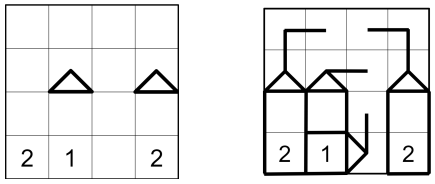
(P1) The *body* is a horizontal or vertical line of k squares.
(P2) The *tip* is 1 square after one end of the body, and it is pointed away from the body.
(P3) The *lead* is a line through the center of $k+1$ connected squares starting from and including the tip.

The goal of Pencils is to draw pencils on the given grid subject to the following rules [3]:

(P4) The pencils partition the $m \cdot n$ squares of the grid.
(P5) If x is a number on the board, then x must be drawn inside of the body of some x-pencil.

*Bard College at Simon's Rock, Massachusetts, dpacker14@simons-rock.edu
†Bard College at Simon's Rock, Massachusetts, swhite15@simons-rock.edu
‡Bard College at Simon's Rock, Massachusetts, awilliams@simons-rock.edu

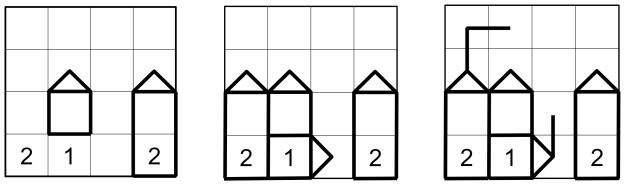
With regard to (P5), an individual x-pencil may have a single x, multiple x's, or no x's inside of it.



(a) Pencils puzzle. (b) Solution.

Figure 1: A 4-by-4 Pencils puzzle that uses 1-pencils and 2-pencils.

A simple puzzle and its solution (originally published in *Puzzle Communication Nikoli* Volume 162 [4]) is displayed in Figure 1, and its solution process is shown in Figure 2. The Pencils decision problem answers 'yes' or 'no' depending on whether an input board is valid and is solvable based on rules (P1)-(P5).



(a) The tip above the 1-pencil forces a pencil body. The tip above the 2-pencil must be part of the 2-pencil due to the 2-pencil position.

(b) Placement of the 1-pencil and 2-pencil bodies and tips are forced.

(c) Leads for the 1-pencil and the left 2-pencil are drawn in, leaving only one solution for the remaining leads.

Figure 2: Solving the Pencils puzzle in Figure 1.

Notice that a solution to an m-by-n board must fill each of the $m \cdot n$ squares with a finite number of possible symbols. More specifically, a square is covered by a horizontal or vertical body, a tip that points in one of four directions, or by a lead that proceeds horizontally, vertically, or turns 90^o. Therefore, we can guess a possible solution in non-deterministic polynomial-time. Rules (P1)–(P5) can then be checked in polynomial-time.

Remark 1 Pencils *is in NP.*

1.2 Outline

The central proof of this paper will be done by reduction from a Boolean satisfiability problem. The specific source problem is included in Section 3 along with an outline of the reduction. Section 4 introduces our gadgets, and then Section 5 proves that Pencils is NP-complete even when restricted to 1-pencils and 2-pencils. Section 6 concludes with open problems. We begin by characterizing the rectangular regions that can be filled with pencils in Section 2.

A preliminary unpublished version of this article was announced on twitter by `@postpostdoc` [5].

2 Empty Rectangles

When solving a pencils puzzle, the solver sometimes faces empty regions of the board that must be completely filled with new pencils. Similarly, we will need to understand how empty space can be filled during our reduction. In this section we provide a full characterization of when rectangular regions can be filled. We formulate this result in terms of solving empty puzzle boards, but we will use the result to solve rectangular "sub-puzzles" inside of larger puzzles.

Define an *empty board* to be an m-by-n grid where each square is empty.

Lemma 1 *Suppose that B is an empty m-by-n board. The decision problem* Pencils(B) *is* True *if and only if $m \cdot n \notin \{1, 2, 4\}$.*

Proof. We begin by considering the negative cases. Observe that the smallest individual pencil (i.e. a 1-pencil) covers 3 squares. Thus, if B has area 1 or 2, then is too small to be filled with a pencil. Similarly, rectangles of area 4 can only admit a 1-pencil, which then leaves one unfillable square.

Now we consider the remaining positive cases. Since the board B can be rotated 90^o without changing the result of Pencils(B), we can assume without loss of generality that $m \leq n$. If the area of B is 3, then it must be that $m = 1$ and $n = 3$, and in this case it can be filled with a single 1-pencil. In the remaining cases the area of B is greater than 4, so we can assume that $n \geq 3$.

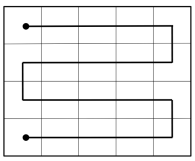


Figure 3: A 5-by-4 grid with a line moving back-and-forth along each row though the centers of the squares in boustrophedon order.

Our strategy is to draw the pencils one after another from end-to-lead in a single line. This line will proceed back-and-forth along each row starting from the top-left square, as illustrated in Figure 3[1]. More specifically, we will primarily draw 1-pencils along the line, since they can turn corners. Since 1-pencils occupy 3 squares, we now proceed in three cases based on the area modulo 3.

- If the area is $3k$, then we draw successive 1-pencils along the line until they fill the entire rectangle.
- If the area $3k + 2$, then recall that our previous assumption that $n \geq 3$. Therefore, we can begin the line with a 2-pencil. This is because the board is wide enough to contain its body and tip, and its lead can bend if $n = 3$ or $n = 4$. Then we fill the remainder of the line with 1-pencils.
- If the area is $3k + 1$, then we consider two cases. If $k = 1$, then the area is $3k + 1 = 7$, and it must be that $n = 1$ and $m = 7$. In this case the rectangle can be filled with a single 3-pencil. Otherwise, if $k > 1$, then the area is $3k + 1 \geq 10$. In this case we can draw two 2-pencils along the line, one starting at the beginning of the line and one starting at the end of the line, and then fill in the remainder of the line with 1-pencils.

□

Now we specialize the previous lemma based on 1-pencils and 2-pencils.

Corollary 1 *If B is an empty m-by-n board, then it can be filled entirely with 1-pencils and 2-pencils if and only if $m \cdot n \notin \{1, 2, 4, 7\}$.*

Proof. Observe that the proof of Lemma 1 uses only 1-pencils and 2-pencils, except in the case that $m \cdot n = 7$. Furthermore, 1-pencils and 2-pencils occupy 3 and 5 squares respectively, so it is impossible for them to fill a board of area 7. □

3 Source Problem

Our hardness proof reduces from a satisfiability problem, and in this section we review relevant terminology and results. Then we give a high-level outline of our reduction.

3.1 Rectilinear Planar 1-in-3SAT

A *(Boolean) variable* can be assigned a truth value of True or False. If x_i is a variable, then its *positive literal* is x_i, and its *negative literal* is $\neg x_i$. A (Boolean) formula is in *3 conjunctive normal form (3CNF)* if it is

[1] This back-and-forth order can be described as *boustrophedonic* which is Greek for "as the ox plows".

written $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_m$ where each *clause* C_i has the form $(\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$ and each $\ell_{i,j}$ is a literal. A clause is *positive* if every one of its literals is positive, and a 3CNF formula is *positive* if every clause is positive. The 3CNF formula ϕ is a *yes* instance of the 3SAT decision problem if its variables can be assigned so that ϕ evaluates to true; otherwise ϕ is a *no* instance. In other words, 3SAT asks if there is an assignment in which every clause has at least one literal that evaluates to true. The 1-IN-3SAT decision problem instead asks if there is a variable assignment in which exactly one literal evaluates to true. A formula is *planar* if the bipartite incidence graph of variables and clauses is planar. A formula is *rectilinear planar* if the graph can be embedded into a grid in such a way that the vertices can be represented by horizontal line segments and the edges can be drawn as vertical lines.

Theorem 2 (Mulzer and Röte [1]) RECTILINEAR POSITIVE PLANAR 1-IN-3SAT *is NP-Complete.*

We will drop the positive condition from Theorem 2 and instead use RECTILINEAR PLANAR 1-IN-3SAT as our source problem. Since every instance of the former problem is an instance of the latter problem, we can easily conclude that the latter is also NP-complete.

3.2 Reduction Outline

Our reduction constructs a planar graph that represents the 1-in-3 satisfiability (or not) of a logical statement in 3CNF. The graph connects variables to their literals in the statement, with not gates appearing along the connections to negative literals. The reduction will use "variable assignment" gadgets—one for each variable—where the player will be able to select whether a variable has a truth value of TRUE or FALSE. Then, wires will carry these truth values to the corresponding literals in each clause. Because a variable can appear more than once in a statement, we include a gadget to duplicate its truth value onto two different wires, thereby ensuring that the choice is consistent in each clause it appears in. Finally, because the statement is in 3CNF, we will also create a gadget that represents an arbitrary 1-in-3 clause, with wire inputs. Using these gadgets, we will reduce the decision problem, PLANAR 1-IN-3 SAT to PENCILS, by transforming a particular logical statement to a pencils board.

4 Gadgets

In this section we present the various gadgets used in our reduction.

Lemma 3 (Wire) *The gadget shown in Figure 4a transmits a truth value from one part of the puzzle to another as an edge in* PLANAR 1-IN-3 SAT.

Proof. Suppose that we have the left 2-pencil already filled in, pointing into the wire (the direction is forced by the variable assignment gadget, shown later). Then the adjacent 2-pencil must point in the same direction as its neighbor, since there is not room for it to point in the opposite direction. Furthermore, the pencil can neither overlap with its neighbor nor leave a gap of size 1 between itself and its neighbor (as this would be unfillable), so the pencil must have the same position relative to the predrawn 2 as its neighboring 2-pencil. Thus, a 2-pencil/predrawn 2 positioning assigned at the front of the wire gets precisely transmitted to all other parts of the wire. □

Using this lemma, we can establish the formalism that if a wire has its 2-pencils with the number 2 in the square adjacent to the tip, then it carries FALSE, and if the 2 is in the other square, then it carries TRUE.

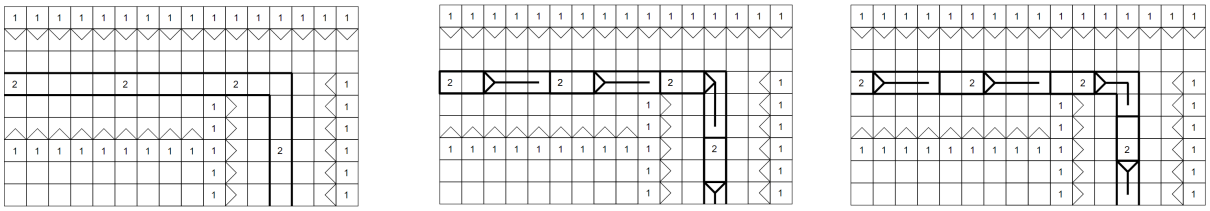
Currently, our wires require that all of our gadgets are a multiple of five squares apart, since the 2's are spaced exactly that far apart in our wire gadget. However, we can deal with this issue with the "modularity switcher" gadget in Figure 5a.

Lemma 4 (Modularity Switch) *The gadget shown in Figure 5a preserves the truth value that a surrounding wire gadget is carrying.*

Proof. Suppose that the incoming truth value is TRUE. Then there will be six unfilled squares between the end of incoming 2's lead and the pair of 2's. Since the 2's on the right are only one square apart from each other, they must both be pointing outward. Thus, the left 2 must have a 2-pencil pointing left, which will occupy either three or four of the empty middle squares. If the 2-pencil fills four middle squares, then there will only be two unfilled middle squares, which cannot be filled by any pencil. Thus, the 2-pencil must fill three middle squares, which must be filled by a 1-pencil. This then forces play on the last 2-pencil, as seen in Figure 5b.

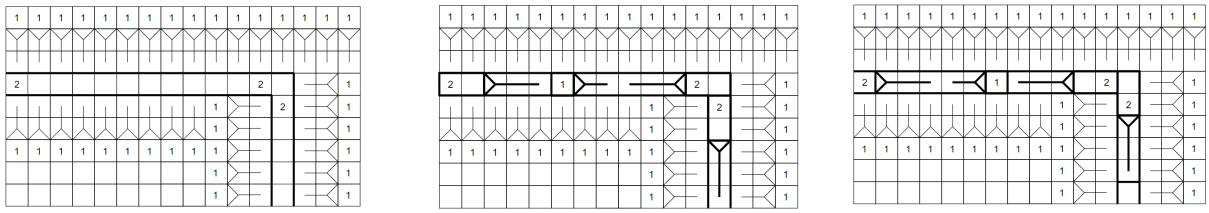
If the incoming truth value is FALSE, then there will seven unfilled squares between the end of the incoming 2's lead and the pair of 2's on the right. Again, the left of the pair of 2-pencils must fill either three or four squares. This 2-pencil cannot occupy three middle squares, for it would leave four squares unoccupied, which cannot be filled. Thus, the 2-pencil must occupy four middle squares, with the remaining three filled by a 1-pencil. This forces the subsequent 2-pencil to play as in Figure 5c. Thus, regardless of the incoming truth value, the modularity switcher does not alter the truth value carried by the wire. □

Lemma 5 (Variable Assignment) *The gadget presented in Figure 6a allows the player to assign a value to a variable that will be transmitted out through the wire on the right.*



(a) The initial wire gadget. (b) The board filled in with TRUE. (c) The board filled in with FALSE.

Figure 4: The initial wire gadget and the manners it can be filled in.



(a) The initial modularity switching gadget. (b) The modularity gadget filled in with value TRUE. (c) The modularity gadget filled in with value FALSE.

Figure 5: The initial modularity gadget and the manners it can be filled in.

Proof. The area in which new pencils may be added is limited to a space of size 9 (there are two possible ways for this to happen, depending on whether the one pencil in the lower right has an upward or leftward pointing lead). Next, the first 2 is located such that its tip must be leftward pointing. If it pointed to the right, there would not be room for the two squares that the line would need to occupy. Thus, the other 2-pencil must be rightward pointing with its body either filling in the square between the 2's or not. Figures 6b and 6c describe how to fill the 3x3 space for TRUE and FALSE variable assignments.

Since the player is not able to play the right 2-pencil another way than the two variable assignments, and the player is able to assign either truth value, our Lemma is proven. □

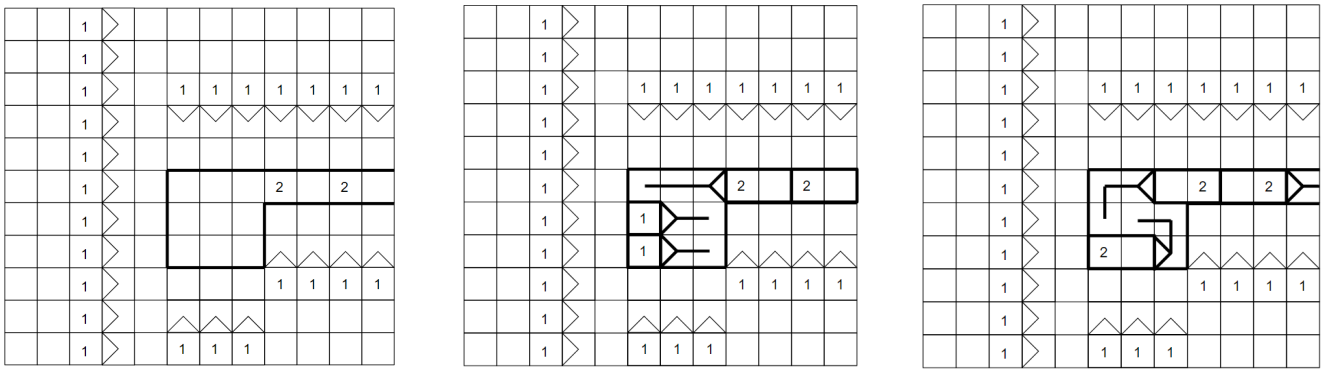
Lemma 6 (Not Gate) *Figure 7a presents a not gate for a leftward facing wire.*

Proof. First, consider the scenario where the initial value of the wire is true. Then, the remaining number of squares up to the next 2 is 7. The next 2 must have its pencil pointing to the left, so it will occupy either three or four of the open spaces. This would leave either three or four consecutive unoccupied spaces. However, we cannot fill four unoccupied spaces by Lemma 1, so we must play the second 2 so that it occupies four of the internal spaces. This forces the last 2 (which must be played to the right) to occupy the empty space between the 2's, making the transmitted value false. On the other hand, if the initial value is false, then there will be eight open squares in the middle. The second 2 can be played so that it occupies three or four spaces. This corresponds to four or five open middle spaces. Of the two options, we can only fill five consecutive middle spaces, so the the second 2 must be played to occupy the empty space between the 2's. This forces the final 2 to be played in the true position. Both of these scenarios are illustrated in Figures 7b and 7c. □

Lemma 7 (Split Gate) *For a given input in the wire on the left, the gadget in Figure 8a assigns truth values to two wires on the right and bottom each carrying the opposite of the given input (to make this a true split gate, we would add a not gate between the input and the gadget or add two more not gates to the ends).*

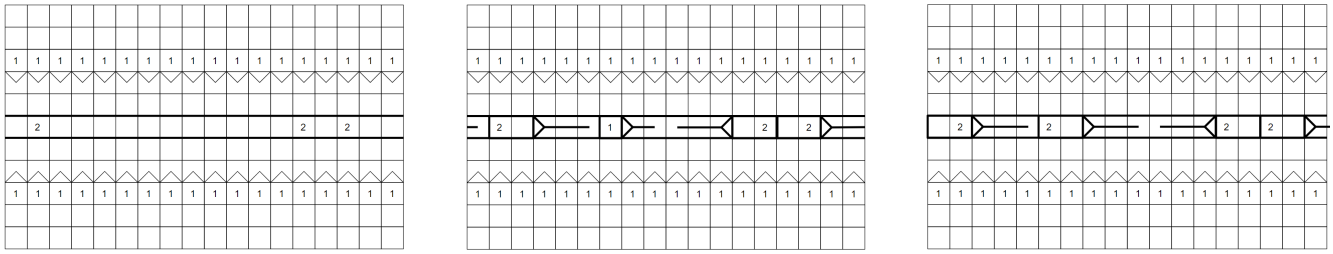
Proof. Of the 2-pencils on the right and on the bottom, the inner pencils of each must be pointing inward; there is not room for them to point outward. If the entering wire is true, then there are 8 remaining spaces within the center of the gadget. The right and bottom inner 2's can occupy either 3 or 4 spaces. If they both occupy 3 squares, then there will be 2 squares unfilled, which cannot be filled by the addition of another pencil. If one occupies 3 squares and the other 4, there will be one square unfilled, which is not fillable by the addition of another pencil. If they both occupy 4 squares, then there are no squares left unfilled, and the gadget is satisfied. This case is forced if the input is true, since there are no other ways to fill the gadget. In this case, the output 2's are both forced to be false, so the input was flipped and placed into two wires, as in the statement of the lemma.

If the entering wire is false, then there are 9 remaining



(a) Unfilled assignment. (b) Filled with TRUE. (c) Filled with FALSE.

Figure 6: The assignment gadget unfilled and filled.



(a) An unfilled not gate. (b) Turning TRUE to FALSE. (c) Turning FALSE to TRUE.

Figure 7: A not gate unfilled and filled with both truth values.

open squares. If either of the inward pointing 2-pencils occupy 4 squares, then the gadget is unfillable, since there will be either 1 or 2 unfilled squares. Thus, the 2 inward pointing pencils must occupy 3 of the inner squares, leaving 3 open squares, which can be filled with a single 1-pencil. This scenario also forces the output 2's to both be true as desired. □

Lemma 8 (1-in-3 Gate) *The gadget of Figure 9 (which takes in input from three wires) is only fillable if exactly one of the input wires is true.*

Proof. Note that each input wire ends its line at either {RT, LT, BT} if it is true or {RF, LF, BF} if it is false. If all the statements are false, then there are four unoccupied squares, so the gadget is unsolvable if all the wires are carrying false values. If all the the statements are true, then there is only one unoccupied square, so the gadget is unsolvable in this case as well. If two of the statements are true, then there exactly two unfilled squares, so the gadget is unsolvable if two statements are true. If only one statement is true, then there are three connected unoccupied squares, which can be filled with a 1-pencil, so the gadget is solvable if and only if exactly one statement is true. Thus, the gadget serves the purpose of a 1-in-3 Gate. □

5 NP-Completeness of Pencils

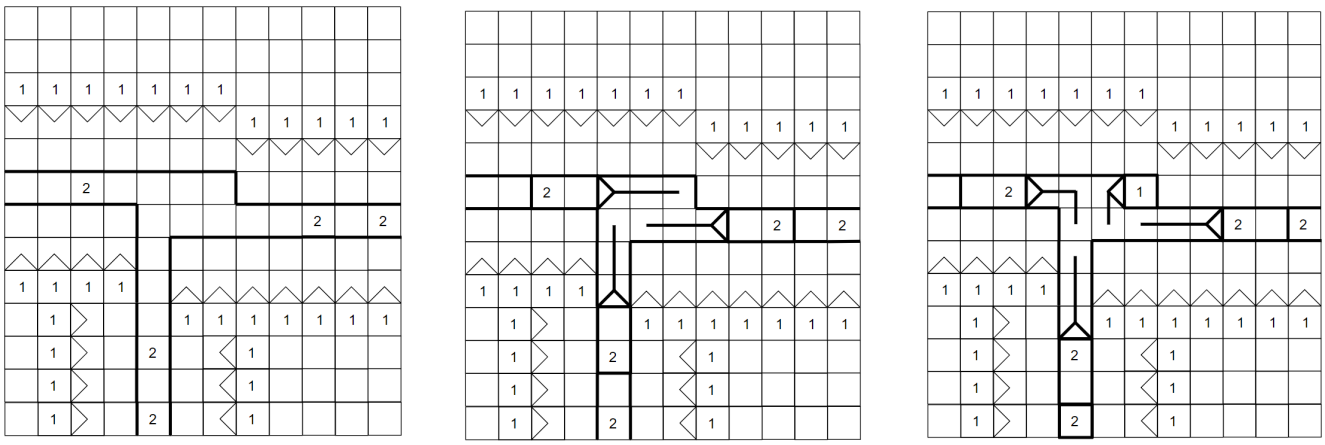
Now we are ready to prove our main result.

Theorem 9 (Pencils is NP-Complete) *For a given board, B, the decision problem* PENCILS(B) *is NP-Complete. Furthermore, this is true when the puzzle designer and solver are restricted to using 1-pencils and 2-pencils.*

Proof. We use Theorem 2 and reduce RECTILINEAR PLANAR 1-IN-3SAT(S) to PENCILS(B). Starting with G_S, the graph corresponding to S, we will encode this graph into a pencils game.

We replace each source variable with the variable assignment gadget and add sufficiently many split and not gates such that each source vector has as many outward going wires as edges leading to literals in the formula. We can then create each clause by leading in the corresponding literals with wires (with not gates if they appear with a $\neg$ modifier in the formula). This is possible since G_S was planar and we can line up the wires to fit in perfectly by inserting modularity switchers sufficiently many times. Call this pencil board B_S.

By the lemmas for each gadget, if RECTILINEAR PLANAR 1-IN-3SAT(S) is true, then by matching up our variable assignment to that which solves S, we can solve the corresponding pencil board, B_S. Thus, RECTILINEAR



(a) An unfilled split gate. (b) TRUE input. (c) FALSE input.

Figure 8: A split gate unfilled and filled with both truth values.

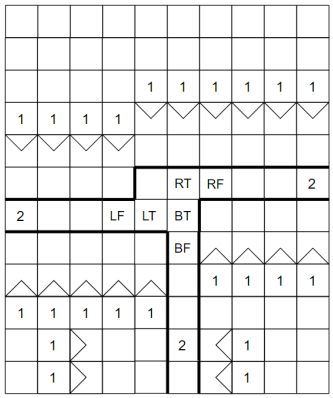


Figure 9: An unfilled 1-in-3 Gate. The non-numeral entries exist to refer to potential pencil endings (and will not affect the actual puzzle).

PLANAR 1-IN-3SAT(S) = TRUE implies PENCILS(B_S) = TRUE.

If PENCILS(B_S) = TRUE, then there must be some assignment of the variable assignment gadgets such that each 1-in-3 gadget was satisfied. However, because this board was derived directly from the graph, it provides a variable assignment for S such that S is true (under 1-in-3 satisfiability rules). Thus, PENCILS(B_S) = TRUE true implies RECTILINEAR PLANAR 1-IN-3SAT(S) = TRUE. So, PENCILS is NP-Hard.

Membership in NP was given in Remark 1. Thus, PENCILS is both NP-Hard and in NP, so it is NP-Complete. □

6 Final Remarks and Open Problems

We proved that a restricted form of the PENCILS decision problem is NP-complete in which only 1-pencils and 2-pencils are used. In this section we provide open problems in several different directions.

Define PENCILS$_S(B)$ as the decision problem in which the puzzle designer and solver are restricted to pencils whose lengths appear in the set S. For example, we proved the NP-completeness of PENCILS$_{\{1,2\}}(B)$, or simply PENCILS$_{1,2}(B)$. This raises the following open problems:

- Is PENCILS$_1(B)$ in P? In other words, is there a polynomial-time algorithm for solving Pencils when only 1-pencils are allowed?
- Is PENCILS$_2(B)$ NP-complete?
- More generally, what is the complexity of PENCILS$_\ell(B)$ for single fixed values of ℓ?

Besides pencil sizes, we could also consider other restrictions to the pencil bodies and leads. For example, we can define a *straight-line pencil* as one in which the lead is a straight line. Similarly, we can define a *horizontal pencil* and a *vertical pencil* based on the orientation of the pencil's body.

- What is the complexity of PENCILS when restricted to straight-line pencils?
- What is the complexity of PENCILS when restricted to horizontal pencils?

Nikoli typically designs individual puzzle instances to have a unique solution. The associated complexity class is *Another Solution Problem* (*ASP*) in which the input is a problem and a solution and the goal is to determine if there is a second solution. This complexity class was popularized by Ueda and Nagao [6]. We pose the question: is PENCILS ASP-hard?

The authors would like to thank the referees whose comments led to many improvements throughout the article.

References

[1] Wolfgang Mulzer and Günter Rote. Minimum-weight triangulation is NP-hard. *J. ACM*, 55(2):Art. 11, 29, 2008. URL: https://doi.org/10.1145/1346330.1346336, doi:10.1145/1346330.1346336.

[2] Nikoli. Pencils number 1. Puzzle Communication Nikoli Vol. 158, March 2017.

[3] Nikoli. Pencils. https://www.nikoli.co.jp/en/puzzles/pencils.html, May 2018.

[4] Nikoli. Pencils number 4. Puzzle Communication Nikoli Vol. 162, March 2018.

[5] @postpostdoc. https://twitter.com/postpostdoc/status/973539335839469570, March 2018.

[6] Nobuhisa Ueda and Tadaaki Nagao. Np-completeness results for nonogram via parsimonious reductions. Technical report, 1996.

Switches are PSPACE-Complete

Jonathan Gabor * Aaron Williams †

Abstract

Switches is a grid-based puzzle game invented by Jonathan Gabor and implemented using MIT's Scratch programming language in 2014. The puzzle is based on the 'switch' mechanism which allows the player to toggle the presence and absence of barriers by walking over a switch of the same color. At first glance the mechanism seems to be similar to previously studied video game mechanisms including pressure plates and doors, but it is in fact quite different. We prove that deciding if a Switches puzzle is solvable is PSPACE-complete and furthermore, this hardness result is true even when the puzzle is only $r = 3$ rows in height. On the other hand, we provide a polynomial-time algorithm for solving Switches puzzles with $r = 1$ row. The computational complexity of the problem with $r = 2$ is open.

1 Introduction

Switches is a puzzle game that was invented by Jonathan Gabor in 2014 while he was a high school student. The puzzle was implemented using MIT's visual programming language called Scratch [8]. This implementation is available online as Switches v2.1 `https://scratch.mit.edu/projects/33587070/`. A new implementation containing playable versions of every level discussed in this paper is available as Switches Remastered `https://scratch.mit.edu/projects/203220688/`.

The puzzle was designed to be played on an r-by-c grid, and each object is placed inside of a single cell. The player's goal on each level is to move their avatar from the start location to the goal location called the portal. The core mechanism involves *switches* and *doors*. Each door is independently on or off and a door is only a barrier to the player's movement when it is on. When a player steps on a switch, then the state of all doors of the same color are toggled. (The player toggles any switch they touch, and they must move to another before toggling it again.) There can be multiple switches of the same color, multiple doors of the same color in either state, and multiple colors that operate independently.

*Bard College at Simon's Rock, Massachusetts, `jgabor16@simons-rock.edu`

†Bard College at Simon's Rock, Massachusetts, `awilliams@simons-rock.edu`

A sample level and its solution are given in Figure 1, along with a legend of graphical symbols.

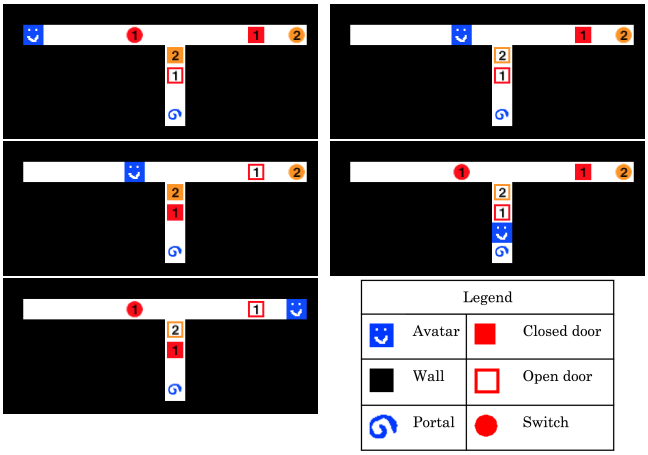


Figure 1: The player solves the 13-by-5 level by walking over the red switch, followed by the orange switch, and then by again walking over the red switch. The images should be read in column-major order.

At first glance, this mechanism may seem to be quite similar to previously studied video game mechanisms such the pressure plate mechanism that was examined by Viglietta [9]. More specifically, a switch behaves like an 'on' pressure plate combined with an 'off' pressure plate, so one might try to simulate pressure plates using switches. However, this is far more difficult than it sounds, because in all but a few cases, levels of switches are reversible. In other words, the player can return to any state they were previously in by performing the previously made moves in reverse[1] order. On the other hand, this is not true for the pressure plate mechanism. Similarly, the mechanisms and hardness results obtained by Aloupis, Demaine, Guo, and Viglietta [1] do not seem to apply to this puzzle. The authors are unaware of any previous puzzle that uses the switch mechanism, but it seems possible that such a puzzle could exist given the mechanism's simplicity.

The Switches decision problem takes a Switches level on an r-by-c grid as input, and the output is 'yes' or 'no'

[1]There is some subtlety to reversibility in Switches levels. If the players avatar is on a blank tile or an open door, then they can return to any previous state by reversing their previous moves. Furthermore, if they can reach a blank tile, they can return to any previous state. However, there are also cases where player can trap themselves.

depending on whether the level is solvable or not. We prove that this decision problem is PSPACE-complete by a reduction from True Quantified Boolean Formula (TQBF). More remarkably, we are able to show that the decision problem remains PSPACE-complete when restricted to levels that have at most $r = 3$ rows. This differentiates it from other PSPACE puzzle games such as Sokoban [2] and Rush Hour [3]. We refer the readers to Hearn and Demaine [6] for further results on the hardness of puzzles and games.

To complement our hardness results, we also prove that Switches puzzles with 1 row can be solved in polynomial-time. The complexity of the decision problem for Switch levels with 2 rows is presently unknown and is a compelling open problem.

The paper is structured as follows. In Section 2 we show how to determine if a level with $r = 1$ rows is solvable in polynomial-time. In Section 3 we show that Switches is NP-hard via a standard 3SAT reduction. In Section 4 we provide a level construction that forces the player to iterate through the well-known binary reflected Gray code. Section 5 then combines the results of Section 3 and 4 to obtain our main PSPACE-hardness result. Section 6 concludes with open problems.

To our knowledge our this article marks the first time that an original Scratch game has been proven to be NP-hard or PSPACE-hard. According to Wikipedia [10],"Scratch has influenced many other programming environments and is now considered a standard for introductory coding experiences for children." As a result, this paper shows that the 'fun' of computational complexity is not just for adults.

2 Polynomial-Time Algorithm for 1 Row

In this section we provide a polynomial-time algorithm for solving Switches levels with $r = 1$ row. Figure 2 gives examples of solvable and unsolvable levels with $r = 1$.

(a) This level is solvable by alternately moving left and right to flip switch 1, switch 2, left switch 4, right switch 4, switch 3, right switch 4, switch 5, switch 2, switch 4, and then moving to the portal.

(b) This level is unsolvable because the player can never reach switch 5 since one of the two 3-doors will always be closed.

Figure 2: Two similar levels with $r = 1$.

Throughout this section we can assume the portal appears on the rightmost square of a level without loss of generality. To understand this assumption, first notice that we can assume that the portal appears to the right of the player's initial position, since otherwise we can instead consider the mirror image of the level. Next notice that any squares to the right of the portal cannot be accessed by the player.

We start this section by considering clusters of adjacent switches, and then by showing how to manipulate their state. Then we consider levels in which the avatar starts on the leftmost square. Finally, we consider levels in which the avatar does not start on the leftmost square.

2.1 Clusters, Configurations, and Traversals

We define a cluster of switches (or simply a cluster) to be a maximal sequence of adjacent switches. In a level with 1 row, membership in a cluster is reflexive, symmetric, and transitive, so the switches partition uniquely into clusters. Figure 3 gives an illustration of clusters.

Figure 3: This level has three clusters.

A left-to-right traversal of a cluster is a sequence of moves in which the player starts at the square immediately to the left of the cluster, ends at the square immediately to the right of the cluster, and at no time moves outside of this region. The traversal ends when the player moves to the next square to the right. We similarly define a right-to-left, right-to-right, and left-to left traversal of a cluster.

When playing a Switches level each color is in one of two states which we call the color's current parity (or parity for short). A cluster containing switches with d distinct colors has 2^d different configurations based on the current parities of these colors. We now show that the player can set a cluster to any configuration during a left-to-right traversal.

Lemma 1 *Suppose the player is standing to the left of a cluster, and the square to the right of the cluster is either open or has the same color as a switch within the cluster. Then the player can set the cluster to any configuration during a left-to-right traversal. Furthermore, the number of steps is linear in terms of the length of the cluster.*

Proof. Suppose that the cluster contains k switches. For convenience let us number the squares from left-to-right starting at 0 from the player's position. In other words, we are focused on the squares numbered 0, 1, 2, . . . , k+1 where 0 and k+1 are immediately outside of the cluster. The pseudocode below uses 'L' and 'R' to denote left and right moves, respectively. The basic idea

is to set the switches to their desired parity from left-to-right. More specifically, if we are standing on square s and the switch on square s-1 has the wrong parity, then we move L then R to correct it, and continue. There are special cases to handle at the right side of the cluster, and we discuss those in more detail below.

```
R
for s = 1, 2, ..., k-1
  R
  if the switch on square s has the wrong parity
    LR
if the switch on square k has the wrong parity
  if square k+1 is a door with switch k's color
    LRRLLRR
  else
    RLR
else
  R
```

After each iteration of the for loop, the avatar will be one space farther to the right, and the color of switch to the players left will be in the correct parity. After completing the for loop, the players avatar will be on the rightmost switch. If this switch is in the correct parity, the player can simply exit the cluster by moving to the right. If it is in the incorrect parity the player will usually be able to fix this by moving right, and then left. Then the player can exit the cluster. Doing this will not affect the parity of any other color because the tile to the right of the player cannot be a switch. However, it is possible that there is a door of the last switchs color directly to the players right. In this case, the player can move left and then right twice (the first time the player moves right, they open the door). However, now the switch on square $k-2$ will be in the wrong parity. This can be fixed by moving left twice and then right twice. This will change the parity of switch $k-2$, but leave the parity of switch $k-1$ constant. □

If the player must end on the same side of the cluster they started on, they can simply move all the way to the left of the cluster, then follow the above algorithm ignoring the leftmost switch. Then, if this switch is in the wrong parity, the player can move all the way to the left, and all the way to the right to fix it.

Lemma 1 also applies to right-to-left and right-to-right traversals, respectively, so long as the player starts on the square to the cluster's immediate right.

2.2 Left-to-Right Levels

Now we focus on $r = 1$ levels in which the player starts on the leftmost square. We refer to these levels as left-to-right levels; when solving these levels we can prove that the player never needs to backtrack to a previously traversed cluster.

For each door, let $s(d)$ be the location of the rightmost switch of its color to its left.

Lemma 2 *A left-to-right level is solvable if and only if the following conditions both hold: If two doors have the same s(d), then they have the same parity; If $s(d)$ is undefined for a door, then that door is initially open.*

Proof. We begin by proving the forward direction. Consider the first point. Obviously, when changing the parity of a door d, the player must either be to the left of location $s(d)$ or to the right of that door. To proceed to the right, they must make that door open. Therefore, to proceed to the right of two doors with the same s(d), they must make both of them open when at position $s(d)$. This is only possible if they have the same parity.

Now consider the second point. If $s(d)$ is undefined, then there is no switch to the left of that door. Then if it is initially closed, the player cannot change its parity, until they go to the right of it, but they cant go to the right of it until they change its parity.

Now consider the reverse direction. We claim that any level satisfying the two points above can be solved using the following linear time algorithm.

Let a clusters ideal configuration be the configuration such that for each switch in it, if that switch is at location $s(d)$ for some door d, the switch is in the parity such that door d is open.

Moving from left to right, we adjust each cluster to its ideal configuration. The only way this algorithm could fail is if the player encounters a closed door which prevents them from traveling farther the right. However, such a door must have a defined $s(d)$. Then it must have been made open. Therefore, encountering a closed door is impossible. □

The algorithms in Lemma 1 runs in linear time in terms of the length of the cluster. Because each cluster is only adjusted once, and the total length of all the clusters is capped by the length of the level, this algorithm runs in linear time.

Theorem 3 *Any solvable level with $r = 1$ row can be solved in polynomial time.*

Proof. Let L_0 be the location of the portal. Let L_{n+1} be the leftmost location the player must reach before reaching location L_n if n is even, and the rightmost such location if n is odd (usually this location will be a switch of the color of a closed door blocking location L_n).

The player can travel from L_{n+1} to L_n in a linear amount of time by Lemma 2.

We will now demonstrate that L_{n+2} is always in between L_{n+1} and L_n (inclusively). Without loss of generality assume that n is even. Because L_{n+2} is the rightmost location the player must reach before L_{n+1}, it is to

the right of L_{n+1} (inclusively). Since L_n is the rightmost location the player must reach before reaching location L_{n-1}, and the player must visit L_{n+2} before visiting L_n, L_{n+2} must to the left of L_n (if it was to the right, then it would be the rightmost location before visiting L_{n-1}).

It follows that if $L_n = L_{n+1}$, for all $m > n$, $L_m = L_n$. Then there is some k which is the lowest value such that $L_k = L_{k+1}$. Then, all L_a with $a < k$ must be distinct. Because there are only a linear number of locations in the level, and moving between each requires a linear amount of time, the level can be solved in $O(n^2)$ time. □

3 NP-Hardness

In this section we prove that the Switches problem is NP-hard by a reduction from 3SAT.

Suppose that we are given an instance of 3SAT ϕ with clauses $c_1, c_2, \ldots, c_m$ and variables $x_1, x_2, x_3, \ldots, x_n$. We construct a Switches level $S(\phi)$ that has 3 rows and $n + 2m + 4$ columns. The level $S(\phi)$ uses n colors in total and there is a single switch of color i. The number of initially open or closed doors of color i is given by the number of positive or negative x_i literals in ϕ, respectively.

The level is organized as follows. The Avatar starts on the left side of the level and to their right is a variable corridor of height 1 and width n+1. The variable corridor contains a variable cluster which is a cluster containing one switch of each color . This corridor leads into a room of height 3 and width 2m+1 called the clause room. Every second column in the clause room is blank, and between these blank columns are columns associated with each of the clauses. Each clause column consists of three doors in a vertical line. The colors of the doors are given by the variable of the literal in the associated clause, and these doors are initially open or closed based on whether the said literals are true or false. The portal is located to the right of the clause room. Figure 4 illustrates this construction.

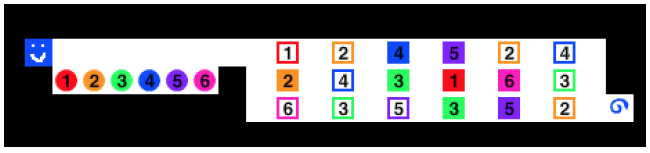


Figure 4: $S(\phi)$ for $\phi = (x_3 \vee \neg x_4 \vee x_5) \wedge \ldots$.

Theorem 4 *Switches is NP-hard.*

Proof. Given an instance of 3SAT ϕ we construct the level $S(\phi)$ as described. Suppose that ϕ is satisfiable by an assignment A which sets variable x_i to a_i for all $1 \leq i \leq n$. By the results of Section 2 the Avatar can perform a left-to-right traversal of the variable cluster with the following property: The switch of color i is switched an even number of times if $a_i = True$ and and odd number of times if $a_i = False$. Now consider a given clause $c_i = (\ell_u \ell_v \ell_w)$ where ℓ_u, ℓ_v, and ℓ_w are either positive or negative literals of variables x_u, x_v, x_w, respectively. Observe that the ith clause column is traversable if and only at least one of its three doors are open. Also recall that the doors associated with positive literals start open, and doors associated with negative literals start closed within $S(\phi)$. Since clause c_i is satisfied the assignment A, it must be that the Avatar's left-to-right traversal of the variable cluster results in the ith clause column being traversable. Therefore, the Avatar can traverse the entire clause room and reach the portal. Therefore, if ϕ is satisfiable, then $S(\phi)$ is solvable.

Suppose that $S(\phi)$ is solvable and consider a particular solution. When the Avatar reaches the portal let $p_i \in \{0, 1\}$ denote the parity of the number of times that switch i was switched during this solution. That is, $p_i = 0$ if switch i was switched an even number of times during the solution, and $p_i = 1$ if switch i was switched an odd number of times during the solution. We construct an assignment A for ϕ as follows: $x_i = True$ if $p_i = 0$, and $x_i = False$ if $p_i = 1$. Due to the structure of the $S(\phi)$ level, when the Avatar reaches the portal, it must be that each one of the clause columns is traversable. Therefore, in each clause in ϕ there must be at least one literal that evaluates to true with respect to assignment A. Therefore, ϕ is solvable.

The reduction is completed by noting that the size of $S(\phi)$ is polynomially bounded by the size of ϕ. □

4 Exponentially Long Levels

In this section we construct Switches levels that require an exponential number of moves to solve. More specifically, the levels contain n distinct colors, and the level forces the player to iterate over all 2^n different states or parities for these colors.

The binary reflected Gray code was previously used in a similar manner in a paper by Greenblatt, Kopinsky, North, Tyrrell, and Williams [5] for the puzzle game MazezaM. The presentation here closely resembles a similar section in that paper.

4.1 Binary Reflected Gray Code

Let $\mathcal{B}(n)$ be the set of n-bit binary strings. The *weight* of $b_1 b_2 \cdots b_n \in \mathcal{B}(n)$ is its bitwise sum $\sum_{i=1}^{n} b_i$. We use exponents to denote bitwise concatenation. For example, $1^4 = 1111$ is the only string of weight four in $\mathcal{B}(4)$.

The *binary reflected Gray code* (BRGC) is an ordering of $\mathcal{B}(n)$ attributed to Gray [4]. In the order each pair of consecutive strings have Hamming distance one (i.e.

they differ in exactly one bit). The order starts with 0^n and ends with $0^{n-1}1$. The BRGC for $n = 4$ is below with overlines showing the bit that changes to create the next string:

$$\bar{0}000, 1\bar{0}00, \bar{1}100, 01\bar{0}0, \bar{0}110, 1\bar{1}10, \bar{1}010, 001\bar{0},$$
$$\bar{0}011, 1\bar{0}11, \bar{1}111, 01\bar{1}1, \bar{0}101, 1\bar{1}01, \bar{1}001, 0001.$$

Now we explain how to create each successive string in the BRGC starting from the initial string 0^n.

Definition 1 *Each $b_1b_2\cdots b_n \in \mathcal{B}(n)$ has up to two* active bits*: (a) its leftmost bit b_1, and (b) its bit immediately to the right of its leftmost* 1*.*

For example, the leftmost 1 in $b_1b_2b_3b_4b_5b_6 = 000111$ is $b_4 = 1$; therefore, its active bits are b_1 and b_5. Every binary string has two active bits except 0^n and $0^{n-1}1$.

The following theorem is well-known (see Knuth [7]).

Theorem 5 *If $b_1b_2\cdots b_n$ has even weight, then complementing active bit (a) gives the next string in the BRGC. Otherwise, if $b_1b_2\cdots b_n$ has odd weight, then complementing active bit (b) gives the next string.*

On the other hand, complementing the 'other' active bit of $b_1b_2\cdots b_n$ gives the previous string in the BRGC.

For example, 000111 has odd weight, so 000101 is the next string in the BRGC and 100111 is the previous.

4.2 Gray Code Level

Now we construct a level **Gray**(n) based on the BRGC for n-bit binary strings. The construction is illustrated in Figure 5. Remarkably, **Gray**(n) has only $r = 3$ rows $c = 2n + 1$ in general.

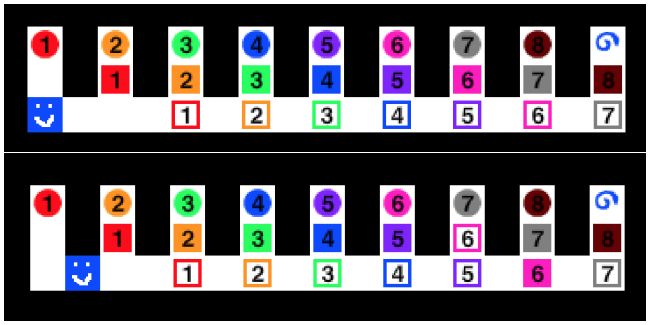


Figure 5: The level **Gray**(8) in state $b_1b_2\cdots b_8 = 00000000$ (top) and $b_1b_2\cdots b_8 = 00000100$ (bottom). In the latter case observe that only those columns with switches 1 and 7 at the top are accessible as per Theorem 5.

The level a corridor along the bottom row, and then $n+1$ columns protruding upwards from the bottom row. The tops of these columns include switches for colors $1, 2, \ldots, n$ and the portal, respectively. The switch for color 1 is not protected, and the column with switch i at the top is protected by a door of color $i - 1$ which is initially on. Furthermore, the corridor below the column with switch i at the top is protected by a door of color $i - 2$ which is initially off.

We associated a binary string $b_1b_2\cdots b_n$ with the state of **Gray**(n)'s switches that is initially $00\cdots0$. Due to the structure of the level and Theorem 5 we have the following theorem.

Theorem 6 *To complete* **Gray**(n) *the player must iterate over all 2^n states of* **Gray**(n) *according to the binary reflected Gray code.*

5 PSPACE Hardness

In this section we prove that Switches is PSPACE-hard. We do this by combining the results of Sections 3 and 4 to create a level $Q(\phi)$ that models a True Quantified Boolean Formula (TQBF) ϕ. More specifically, we model the unquantified Boolean 3SAT formula within ϕ as in Section 3, and then we force the player to iterate overall all possible 2^u states of the u universally quantified variables according Section 4.

When creating the level we utilize the fact that the leftmost bit is changed every second time in the binary reflected Gray code. We leverage this fact by horizontally separating the leftmost bit and the remaining bits in the level, and placing the 3SAT construction between them. In other words, we force the player to traverse the 3SAT formula (from left-to-right or right-to-left) every time they wish to change one of the quantified variable bits. Thus, the player must ensure that the 3SAT formula is satisfied before they can make progress in the higher-level problem that is iterating through the binary reflected Gray code.

A sample construction involving $n = 8$ variables is shown in Figure 6. Due to width restrictions we illustrate the sample level with 9 rows; the taller rows can easily be turned (at the expense of making the level much wider) to create a level with 3 rows.

Theorem 7 *The Switches decision problem is PSPACE-hard even when restricted to levels with $r = 3$ rows.*

Finally, we prove membership in PSPACE below.

Lemma 8 *The Switches decision problem is in PSPACE.*

Proof. Consider an r-by-c level with d distinct colors of switches. There are at most $2^d \cdot rc$ possible states for this level, where 2^d counts the number of different states for the colors, and rc is an upper bound on the number of

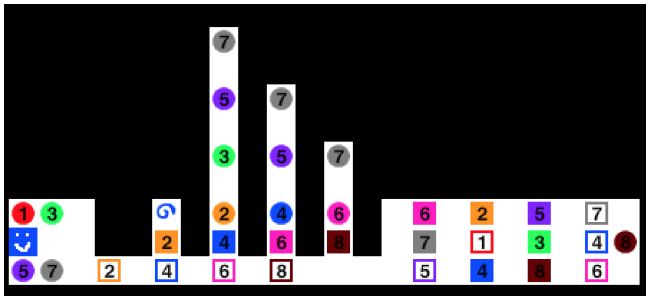


Figure 6: Level $P(\phi)$ which models the TQBF formula ϕ with quantified variables $\exists x_1 \forall x_2 \exists x_3 \forall x_4 \exists x_5 \forall x_6 \exists x_7 \forall x_8$. The bottom of the level contains clause gadgets including the rightmost clause $x_7 \vee x_4 \vee x_6$.

locations for the player's avatar. Therefore, we can represent an individual state of the level with $d \cdot log_2(rc)$ bits. Similarly, $d \cdot log_2(rc)$ bits bits are sufficient for counting from 0 to $2^d \cdot rc - 1$. Therefore, we can now establish membership in NPSPACE by nondeterministically moving the avatar in one of the four cardinal directions, and it keeping a counter for the number of times we have done this. If the avatar reaches the portal, then we stop the algorithm and answer yes. Otherwise, once the counter exceeds the number of possible states then we terminate the algorithm and answer no. If there is a solution to the level, then there will be at least one path through the computation that answers yes. Since we used only $d \cdot log_2(rc)$ bits of storage, this establishes membership in NPSPACE, and by Savitch's theorem, PSPACE. □

Corollary 1 *The Switches decision problem is PSPACE-Complete.*

6 Final Remarks

In this paper we investigated the computational complexity of the Switches puzzle game. There are a number of interesting open problems:

- What is the computational complexity of solving Switches levels with r = 2 rows?
- Our PSPACE-hardness reduction uses an arbitrarily large number of colors. What is the computational complexity if the number of colors is a constant?
- Our PSPACE-hardness reduction uses an arbitrarily large number of doors per color. What is the computational complexity if each doors color can be used only a constant number of times?
- Our PSPACE-hardness reduction uses up to two switches per color. What is the computational complexity if no two switches have the same color?
- Are there any other geometric puzzle games created on Scratch that are NP-hard or PSPACE-hard?

Regarding the 2-row case, there are several reasons to think that it should be solvable in polynomial time. First, when there are two rows, if the player's avatar is in the same column as a square, then the player can immediately reach that square. This means that the player cannot "pass by" an object while being unable to interact with it, something essential to constructing exponential orderings. Second, with only two rows, there is no obvious way to implement clause gadgets. Third, if there are no switches between 2 points in a level, determining whether there exists a set of parities to travel between those points can be done in polynomial time, since it can be reduced to a 2-sat problem.

We also mention that the original Switches implementation has an additional dual switch mechanism in which two switches are toggled simultaneously. We did not require its use to establish PSPACE-hardness, and its inclusion does not change the problem's inclusion in PSPACE.

References

[1] G. Aloupis, Erik D. Demaine, A. Guo, and G. Viglietta. Classic Nintendo games are (computationally) hard. *Theoretical Computer Science*, 586:135–160, 2015.

[2] J. Culbertson. Sokoban is PSPACE-complete. In *Fun With Algorithms (FUN 1998)*, Lecture Notes in Computer Science, pages 65–76. Carleton Scientific, 1998.

[3] G. W. Flake and E. B. Baum. Rush hour is PSPACE-complete, or "why you should generously tip parking lot attendants". *Theor. Comput. Sci.*, 270:895–911, 2002.

[4] F. Gray. Pulse code communication. *U.S. Patent 2,632,058*, 1947.

[5] A. Greenblatt, J. Kopinsky, B. North, M. Tyrrell, and A. Williams. MazezaM levels with exponentially long solutions. In *20th Japan Conference on Discrete and Computational Geometry, Graphs, and Games (JCDCGGG 2017)*, page 2 pages, 2017.

[6] Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation*. A. K. Peters, Ltd., Natick, MA, USA, 2009.

[7] D. E. Knuth. *The Art of Computer Programming*, volume 4: Generating All Tuples and Permutations. Addison-Wesley, 2005.

[8] Scratch. About scratch, 2018. URL: `https://scratch.mit.edu/about`.

[9] G. Viglietta. Gaming is a hard job, but someone has to do it! *Theory of Computing Systems*, 54(4):595–621, 2014.

[10] Wikipedia. Scratch (programming language), 2018. URL: `https://en.wikipedia.org/wiki/Scratch_(programming_language)`.

Packing Plane Spanning Trees into a Point Set

Ahmad Biniaz* Alfredo García†

Abstract

Let P be a set of n points in the plane in general position. We show that at least $\lfloor n/3 \rfloor$ plane spanning trees can be packed into the complete geometric graph on P. This improves the previous best known lower bound $\Omega(\sqrt{n})$. Towards our proof of this lower bound we show that the center of a set of points, in the d-dimensional space in general position, is of dimension either 0 or d.

1 Introduction

In the two-dimensional space, a *geometric graph* G is a graph whose vertices are points in the plane and whose edges are straight-line segments connecting the points. A subgraph S of G is *plane* if no pair of its edges cross each other. Two subgraphs S_1 and S_2 of G are *edge-disjoint* if they do not share any edge.

Let P be a set of n points in the plane. The *complete geometric graph* $K(P)$ is the geometric graph with vertex set P that has a straight-line edge between every pair of points in P. We say that a sequence $S_1, S_2, S_3, \ldots$ of subgraphs of $K(P)$ is *packed into* $K(P)$, if the subgraphs in this sequence are pairwise edge-disjoint. In a packing problem, we ask for the largest number of subgraphs of a given type that can be packed into $K(P)$. Among all subgraphs, plane spanning trees, plane Hamiltonian paths, and plane perfect matchings are of interest. Since $K(P)$ has $n(n-1)/2$ edges, at most $\lfloor n/2 \rfloor$ spanning trees, at most $\lfloor n/2 \rfloor$ Hamiltonian paths, and at most $n-1$ perfect matchings can be packed into it.

A long-standing open question is to determine whether or not it is possible to pack $\lfloor n/2 \rfloor$ plane spanning trees into $K(P)$. If P is in convex position, the answer in the affirmative follows from the result of Bernhart and Kanien [3], and a characterization of such plane spanning trees is given by Bose et al. [5]. In CCCG 2014, Aichholzer et al. [1] showed that if P is in general position (no three points on a line), then $\Omega(\sqrt{n})$ plane spanning trees can be packed into $K(P)$; this bound is obtained by a clever combination of crossing family (a set of pairwise crossing edges) [2] and double-stars (trees with only two interior nodes) [5]. Schnider [12] showed that it is not always possible to pack $\lfloor n/2 \rfloor$ plane spanning double stars into $K(P)$, and gave a necessary and sufficient condition for the existence of such a packing. As for packing other spanning structures into $K(P)$, Aichholzer et al. [1] and Biniaz et al. [4] showed a packing of 2 plane Hamiltonian cycles and a packing of $\lceil \log_2 n \rceil - 2$ plane perfect matchings, respectively.

The problem of packing spanning trees into (abstract) graphs is studied by Nash-Williams [11] and Tutte [13] who independently obtained necessary and sufficient conditions to pack k spanning trees into a graph. Kundu [10] showed that at least $\lceil (k-1)/2 \rceil$ spanning trees can be packed into any k-edge-connected graph.

In this paper we show how to pack $\lfloor n/3 \rfloor$ plane spanning trees into $K(P)$ when P is in general position. This improves the previous $\Omega(\sqrt{n})$ lower bound.

2 Packing Plane Spanning Trees

In this section we show how to pack $\lfloor n/3 \rfloor$ plane spanning tree into $K(P)$, where P is a set of $n \geqslant 3$ points in the plane in general position (no three points on a line). If $n \in \{3, 4, 5\}$ then one can easily find a plane spanning tree on P. Thus, we may assume that $n \geqslant 6$.

The *center* of P is a subset C of the plane such that any closed halfplane intersecting C contains at least $\lceil n/3 \rceil$ points of P. A *centerpoint* of P is a member of C, which does not necessarily belong to P. Thus, any halfplane that contains a centerpoint, has at least $\lceil n/3 \rceil$ points of P. It is well known that every point set in the plane has a centerpoint; see e.g. [7, Chapter 4]. We use the following corollary and lemma in our proof of the $\lfloor n/3 \rfloor$ lower bound; the corollary follows from Theorem 4 that we will prove later in Section 3.

Corollary 1 *Let P be a set of $n \geqslant 6$ points in the plane in general position, and let C be the center of P. Then, C is either 2-dimensional or 0-dimensional. If C is 0-dimensional, then it consists of one point that belongs to P, moreover n is of the form $3k+1$ for some integer $k \geqslant 2$.*

Lemma 1 *Let P be a set of n points in the plane in general position, and let c be a centerpoint of P. Then, for every point $p \in P$, each of the two closed halfplanes, that are determined by the line through c and p, contains at least $\lceil n/3 \rceil + 1$ points of P.*

*University of Waterloo, Canada. Supported by NSERC Postdoctoral Fellowship. ahmad.biniaz@gmail.com

†Universidad de Zaragoza, Spain. Partially supported by H2020-MSCA-RISE project 734922 - CONNECT and MINECO project MTM2015-63791-R. olaverri@unizar.es

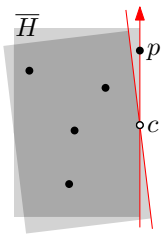


Figure 1: Illustration of the proof of Lemma 1.

Proof. For the sake of contradiction assume that a closed halfplane $\overline{H}$, that is determined by the line through c and p, contains less than $\lceil n/3 \rceil + 1$ points of P. By symmetry assume that $\overline{H}$ is to the left side of this line oriented from c to p as depicted in Figure 1. Since c is a centerpoint and $\overline{H}$ contains c, the definition of centerpoint implies that $\overline{H}$ contains exactly $\lceil n/3 \rceil$ points of P (including p and any other point of P that may lie on the boundary of $\overline{H}$). By slightly rotating $\overline{H}$ counterclockwise around c, while keeping c on the boundary of $\overline{H}$, we obtain a new closed halfplane that contains c but misses p. This new halfplane contains less than $\lceil n/3 \rceil$ points of P; this contradicts c being a centerpoint of P. □

Now we proceed with our proof of the lower bound. We distinguish between two cases depending on whether the center C of P is 2-dimensional or 0-dimensional. First suppose that C is 2-dimensional. Then, C contains a centerpoint, say c, that does not belong to P. Let $p_1, \ldots, p_n$ be a counter-clockwise radial ordering of points in P around c. For two points p and q in the plane, we denote by $\overrightarrow{pq}$, the ray emanating from p that passes through q.

Since every integer $n \geqslant 3$ has one of the forms $3k$, $3k+1$, and $3k+2$, for some $k \geqslant 1$, we will consider three cases. In each case, we show how to construct k plane spanning directed graphs $G_1, \ldots, G_k$ that are edge-disjoint. Then, for every $i \in \{1, \ldots, k\}$, we obtain a plane spanning tree T_i from G_i. First assume that $n = 3k$. To build G_i, connect p_i by outgoing edges to $p_{i+1}, p_{i+2}, \ldots, p_{i+k}$, then connect p_{i+k} by outgoing edges to $p_{i+k+1}, p_{i+k+2}, \ldots, p_{i+2k}$, and then connect p_{i+2k} by outgoing edges to $p_{i+2k+1}, p_{i+2k+2}, \ldots, p_{i+3k}$, where all the indices are modulo n, and thus $p_{i+3k} = p_i$. The graph G_i, that is obtained this way, has one cycle $(p_i, p_{i+k}, p_{i+2k}, p_i)$; see Figure 2. By Lemma 1, every closed halfplane, that is determined by the line through c and a point of P, contains at least $k+1$ points of P. Thus, all points $p_i, p_{i+1}, \ldots, p_{i+k}$ lie in the closed halfplane to the left of the line through c and p_i that is oriented from c to p_i. Similarly, the points $p_{i+k}, \ldots, p_{i+2k}$ lie in the closed halfplane to the left of the oriented line from c to p_{i+k}, and the points $p_{i+2k}, \ldots, p_{i+3k}$ lie in the closed halfplane to the left of the oriented line from c to p_{i+2k}. Thus, all the k edges outgoing from p_i are in the convex wedge bounded by the rays $\overrightarrow{cp_i}$ and $\overrightarrow{cp_{i+k}}$, all the edges outgoing from p_{i+k} are in the convex wedge bounded by $\overrightarrow{cp_{i+k}}$ and $\overrightarrow{c_{i+2k}}$, and all the edges from p_{i+2k} are in the convex wedge bounded by $\overrightarrow{cp_{i+2k}}$ and $\overrightarrow{c_{i+3k}}$. Therefore, the spanning directed graph G_i is plane. As depicted in Figure 2, by removing the edge (p_{i+2k}, p_i) from G_i we obtain a plane spanning (directed) tree T_i. This is the end of our construction of k plane spanning trees.

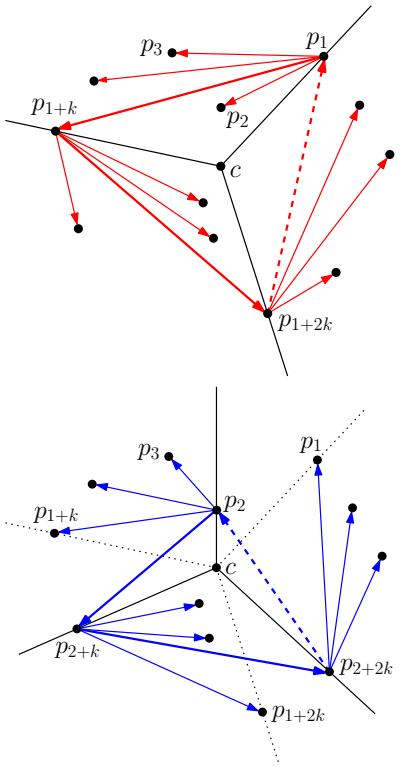


Figure 2: The plane spanning trees T_1 (the top) and T_2 (the bottom) are obtained by removing the edges (p_{1+2k}, p_1) and (p_{2+2k}, p_2) from G_1 and G_2, respectively.

To verify that the k spanning trees obtained above are edge-disjoint, we show that two trees T_i and T_j, with $i \neq j$, do not share any edge. Notice that the tail of every edge in T_i belongs to the set $I = \{p_i, p_{i+k}, p_{i+2k}\}$, and the tail of every edge in T_j belongs to the set $J = \{p_j, p_{j+k}, p_{j+2k}\}$, and $I \cap J = \emptyset$. For contrary, suppose that some edge (p_r, p_s) belongs to both T_i and T_j, and without loss of generality assume that in T_i this edge is oriented from p_r to p_s while in T_j it is oriented from p_s to p_r. Then $p_r \in I$ and $p_s \in J$. Since $(p_r, p_s) \in T_i$ and the largest index of the head of every outgoing edge from p_r is $r+k$, we have that $s \leqslant (r+k) \mod n$. Similarly, since $(p_s, p_r) \in T_j$ and the largest index of the head of every outgoing edge from p_s is $s+k$, we have that $r \leqslant (s+k) \mod n$. However, these two inequalities cannot hold together; this contradicts our assumption

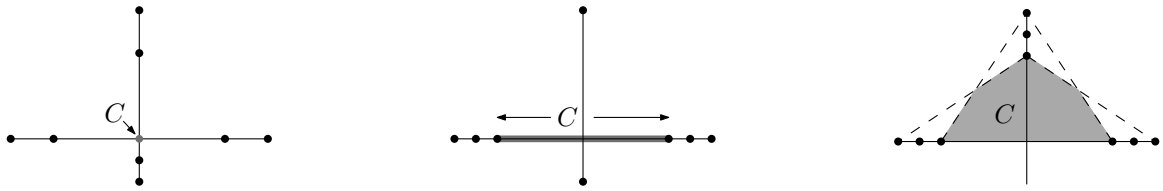


Figure 3: The dimension of a point set in the plane, that is not in general position, can be any number in $\{0, 1, 2\}$.

that (p_r, p_s) belongs to both trees. Thus, our claim, that $T_1, \ldots, T_k$ are edge-disjoint, follows. This finishes our proof for the case where $n = 3k$.

If $n = 3k+1$, then by Lemma 1, every closed halfplane that is determined by the line through c and a point of P contains at least $k + 2$ points of P. In this case, we construct G_i by connecting p_i to its following $k + 1$ points, i.e., $p_{i+1}, \ldots, p_{i+k+1}$, and then connecting each of p_{i+k+1} and p_{i+2k+1} to their following k points. If $n = 3k + 2$, then we construct G_i by connecting each of p_i and p_{i+k+1} to their following $k + 1$ points, and then connecting p_{i+2k+2} to its following k points. This is the end of our proof for the case where C is 2-dimensional.

Now we consider the case where C is 0-dimensional. By Corollary 1, C consists of one point that belongs to P, and moreover $n = 3k + 1$ for some $k \geqslant 2$. Let $p \in P$ be the only point of C, and let $p_1, \ldots, p_{n-1}$ be a counterclockwise radial ordering of points in $P \setminus \{p\}$ around p. As in our first case (where C was 2-dimensional, c was not in P, and n was of the form $3k$) we construct k edge-disjoint plane spanning trees $T_1, \ldots, T_k$ on $P\setminus\{p\}$ where p playing the role of c. Then, for every $i \in \{1, \ldots, k\}$, by connecting p to p_i, we obtain a plane spanning tree for P. These plane spanning trees are edge-disjoint. This is the end of our proof. In this section we have proved the following theorem.

Theorem 2 *Every complete geometric graph, on a set of n points in the plane in general position, contains at least $\lfloor n/3 \rfloor$ edge-disjoint plane spanning trees.*

3 The Dimension of the Center of a Point Set

The *center* of a set P of $n \geqslant d + 1$ points in $\mathbb{R}^d$ is a subset C of $\mathbb{R}^d$ such that any closed halfspace intersecting C contains at least $\alpha = \lceil n/(d+1) \rceil$ points of P. Based on this definition, one can characterize C as the intersection of all closed halfspaces such that their complementary open halfspaces contain less than α points of P. More precisely (see [7, Chapter 4]) C is the intersection of a finite set of closed halfspaces $\overline{H_1}, \overline{H_2}, \ldots, \overline{H_m}$ such that for each $\overline{H_i}$

1. the boundary of $\overline{H_i}$ contains at least d affinely independent points of P, and
2. the complementary open halfspace H_i contains at most $\alpha - 1$ points of P, and the closure of H_i contains at least α points of P.

Being the intersection of closed halfspaces, C is a convex polyhedron. A *centerpoint* of P is a member of C, which does not necessarily belong to P. It follows, from the definition of the center, that any halfspace containing a centerpoint has at least α points of P. It is well known that every point set in the plane has a centerpoint [7, Chapter 4]. In dimensions 2 and 3, a centerpoint can be computed in $O(n)$ time [9] and in $O(n^2)$ expected time [6], respectively.

A set of points in $\mathbb{R}^d$, with $d \geqslant 2$, is said to be in *general position* if no $k+2$ of them lie in a k-dimensional flat for every $k \in \{1, \ldots, d-1\}$.[1] Alternatively, for a set of points in $\mathbb{R}^d$ to be in general position, it suffices that no $d + 1$ of them lie on the same hyperplane. In this section we prove that if a point set P in $\mathbb{R}^d$ is in general position, then the center of P is of dimension either 0 or d. Our proof of this claim uses the following result of Grünbaum.

Theorem 3 (Grünbaum, 1962 [8]) *Let $\mathcal{F}$ be a finite family of convex polyhedra in $\mathbb{R}^d$, let I be their intersection, and let s be an integer in $\{1, \ldots, d\}$. If every intersection of s members of $\mathcal{F}$ is of dimension d, but I is $(d - s)$-dimensional, then there exist $s + 1$ members of $\mathcal{F}$ such that their intersection is $(d-s)$-dimensional.*

Before proceeding to our proof, we note that if P is not in general position, then the dimension of C can be any number in $\{0, 1, \ldots, d\}$; see e.g. Figure 3 for the case where $d = 2$.

Observation 1 *For every $k \in \{1, \ldots, d+1\}$ the dimension of the intersection of every k closed halfspaces in $\mathbb{R}^d$ is in the range $[d - k + 1, d]$.*

Theorem 4 *Let P be a set of $n \geqslant d + 1$ points in $\mathbb{R}^d$, and let C be the center of P. Then, C is either d-dimensional, or contained in a $(d - s)$-dimensional polyhedron that has at least $n - (s+1)(\alpha-1)$ points of P for some $s \in \{1, \ldots, d\}$ and $\alpha = \lceil n/(d+1) \rceil$. In the latter case if P is in general position and $n \geqslant d + 3$, then C consists of one point that belongs to P, and n is of the form $k(d + 1) + 1$ for some integer $k \geqslant 2$.*

[1] A flat is a subset of d-dimensional space that is congruent to a Euclidean space of lower dimension. The flats in 2-dimensional space are points and lines, which have dimensions 0 and 1.

Proof. The center C is a convex polyhedron that is the intersection of a finite family $\mathcal{H}$ of closed halfspaces such that each of their complementary open halfspaces contains at most $\alpha-1$ points of P [7, Chapter 4]. Since C is a convex polyhedron in $\mathbb{R}^d$, its dimension is in the range $[0,d]$. For the rest of the proof we consider the following two cases.

(a) The intersection of every $d+1$ members of $\mathcal{H}$ is of dimension d.

(b) The intersection of some $d+1$ members of $\mathcal{H}$ is of dimension less than d.

First assume that we are in case (a). We prove that C is d-dimensional. Our proof follows from Theorem 3 and a contrary argument. Assume that C is not d-dimensional. Then, C is $(d-s)$-dimensional for some $s \in \{1,\ldots,d\}$. Since the intersection of every s members of $\mathcal{H}$ is d-dimensional, by Theorem 3 there exist $s+1$ members of $\mathcal{H}$ whose intersection is $(d-s)$-dimensional. This contradicts the assumption of case (a) that the intersection of every $d+1$ members of $\mathcal{H}$ is d-dimensional. Therefore, C is d-dimensional in this case.

Now assume that we are in case (b). Let s be the largest integer in $\{1,\ldots,d\}$ such that every intersection of s members of $\mathcal{H}$ is d-dimensional; notice that such an integer exists because every single halfspace in $\mathcal{H}$ is d-dimensional. Our choice of s implies the existence of a subfamily $\mathcal{H}'$ of $s+1$ members of $\mathcal{H}$ whose intersection is d'-dimensional for some $d'<d$. Let s' be an integer such that $d'=d-s'$. By Observation 1, we have that $d' \geqslant d-s$, and equivalently $d-s' \geqslant d-s$; this implies $s' \leqslant s$. To this end we have a family $\mathcal{H}'$ with $s+1$ members for which every intersection of s' members is d-dimensional (because $s' \leqslant s$ and $\mathcal{H}' \subseteq \mathcal{H}$), but the intersection of all members of $\mathcal{H}'$ is $(d-s')$-dimensional. Applying Theorem 3 on $\mathcal{H}'$ implies the existence of $s'+1$ members of $\mathcal{H}'$ whose intersection is $(d-s')$-dimensional. If $s'<s$, then this implies the existence of $s'+1 \leqslant s$ members of $\mathcal{H}' \subseteq \mathcal{H}$, whose intersection is of dimension $d-s'<d$. This contradicts the fact that the intersection of every s members of $\mathcal{H}$ is d-dimensional. Thus, $s'=s$, and consequently, $d'=d-s'=d-s$. Therefore C is contained in a $(d-s)$-dimensional polyhedron I which is the intersection of the $s+1$ closed halfspaces of $\mathcal{H}'$. Let $H_1,\ldots,H_{s+1}$ be the complementary open halfspaces of members of $\mathcal{H}'$, and recall that each H_i contains at most $\alpha-1$ points of P. Let $\overline{I}$ be the complement of I. Then,

$$\begin{aligned} n &= |I \cup \overline{I}| = |I \cup H_1 \cup \cdots \cup H_{s+1}| \\ &\leqslant |I| + |H_1| + \cdots + |H_{s+1}| \leqslant |I| + (s+1)(\alpha-1), \end{aligned}$$

where we abuse the notations I, $\overline{I}$, and H_i to refer to the subset of points of P that they contain. This inequality implies that I contains at least $n-(s+1)(\alpha-1)$ points of P. This finishes the proof of the theorem except for the part that P is in general position.

Now, assume that P is in general position and $n \geqslant d+3$. By the definition of general position, the number of points of P in a $(d-s)$-dimensional flat is not more than $d-s+1$. Since I is $(d-s)$-dimensional, this implies that

$$n-(s+1)(\alpha-1) \leqslant d-s+1.$$

Notice that n is of the form $k(d+1)+i$ for some integer $k \geqslant 1$ and some $i \in \{0,1,\ldots,d\}$. Moreover, if i is 0 or 1, then $k \geqslant 2$ because $n \geqslant d+3$. Now we consider two cases depending on whether or not i is 0. If $i=0$, then $\alpha=k$. In this case, the above inequality simplifies to $k(d-s) \leqslant d-2s$, which is not possible because $k \geqslant 2$ and $d \geqslant s \geqslant 1$. If $i \in \{1,\ldots,d\}$, then $\alpha=k+1$. In this case, the above inequality simplifies to $(k-1)(d-s)+i \leqslant 1$, which is not possible unless $d=s$ and $i=1$. Thus, for the above inequality to hold we should have $d=s$ and $i=1$. These two assertions imply that $n=k(d+1)+1$, and that I is 0-dimensional and consists of one point of P. Since $C \subseteq I$ and C is not empty, C also consists of one point of P. □

References

[1] O. Aichholzer, T. Hackl, M. Korman, M. J. van Kreveld, M. Löffler, A. Pilz, B. Speckmann, and E. Welzl. Packing plane spanning trees and paths in complete geometric graphs. *Information Processing Letters*, 124:35–41, 2017. Also in CCCG'14, pages 233–238.

[2] B. Aronov, P. Erdös, W. Goddard, D. J. Kleitman, M. Klugerman, J. Pach, and L. J. Schulman. Crossing families. *Combinatorica*, 14(2):127–134, 1994. Also in SoCG'91, pages 351–356.

[3] F. Bernhart and P. C. Kainen. The book thickness of a graph. *Journal of Combinatorial Theory, Series B*, 27(3):320–331, 1979.

[4] A. Biniaz, P. Bose, A. Maheshwari, and M. H. M. Smid. Packing plane perfect matchings into a point set. *Discrete Mathematics & Theoretical Computer Science*, 17(2):119–142, 2015.

[5] P. Bose, F. Hurtado, E. Rivera-Campo, and D. R. Wood. Partitions of complete geometric graphs into plane trees. *Computational Geometry: Theory and Applications*, 34(2):116–125, 2006.

[6] T. M. Chan. An optimal randomized algorithm for maximum tukey depth. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 430–436, 2004.

[7] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer, 1987.

[8] B. Grünbaum. The dimension of intersections of convex sets. *Pacific Journal of Mathematics*, 12(1):197–202, 1962.

[9] S. Jadhav and A. Mukhopadhyay. Computing a centerpoint of a finite planar set of points in linear time. *Discrete & Computational Geometry*, 12:291–312, 1994.

[10] S. Kundu. Bounds on the number of disjoint spanning trees. *Journal of Combinatorial Theory, Series B*, 17(2):199–203, 1974.

[11] C. St. J. A. Nash-Williams. Edge-disjoint spanning trees of finite graphs. *Journal of the London Mathematical Society*, 36(1):445–450, 1961.

[12] P. Schnider. Packing plane spanning double stars into complete geometric graphs. In *Proceedings of the 32nd European Workshop on Computational Geometry, EuroCG*, pages 91–94, 2016.

[13] W. T. Tutte. On the problem of decomposing a graph into n connected factors. *Journal of the London Mathematical Society*, 36(1):221–230, 1961.

Compatible Paths on Labelled Point Sets

Elena Arseneva* Yeganeh Bahoo† Ahmad Biniaz‡ Pilar Cano§ Farah Chanchary§ John Iacono¶
Kshitij Jain‡ Anna Lubiw‡ Debajyoti Mondal‖ Khadijeh Sheikhan** Csaba D. Tóth††

Abstract

Let P and Q be finite point sets of the same cardinality in $\mathbb{R}^2$, each labelled from 1 to n. Two noncrossing geometric graphs G_P and G_Q spanning P and Q, respectively, are called *compatible* if for every face f in G_P, there exists a corresponding face in G_Q with the same clockwise ordering of the vertices on its boundary as in f. In particular, G_P and G_Q must be straight-line embeddings of the same connected n-vertex graph G. No polynomial-time algorithm is known for deciding whether two labelled point sets admit compatible geometric graphs. The complexity of the problem is open, even when the graphs are constrained to be triangulations, trees, or simple paths.

We give polynomial-time algorithms to find compatible paths or report that none exist in three scenarios: $O(n)$ time for points in convex position; $O(n^2)$ time for two simple polygons, where the paths are restricted to remain inside the closed polygons; and $O(n^2 \log n)$ time for points in general position if the paths are restricted to be monotone.

1 Introduction

Computing noncrossing geometric graphs on finite point sets that are in some sense 'compatible' is an active area of research in computational geometry. The study of compatible graphs is motivated by applications to shape animation and simultaneous graph drawing [4, 12].

Let P and Q be finite point sets, each containing n points in the plane labelled from 1 to n. Let G_P and G_Q be two noncrossing geometric graphs spanning P and Q, respectively. G_P and G_Q are called *compatible*, if for every face f in G_P, there exists a corresponding face in G_Q with the same clockwise ordering of the vertices on its boundary as in f. It is necessary, but not sufficient, that G_P and G_Q represent the same connected n-vertex graph G. Given a pair of labelled point sets, it is natural to ask whether they have compatible graphs, and if so, to produce one such pair, G_P, G_Q. The question can also be restricted to specific graph classes such as paths, trees, triangulations, and so on; previous work (described below) has concentrated on compatible triangulations. Compatible triangulations of polygons are also of interest, which motivated us to examine compatible paths inside simple polygons.

In this paper we examine the problem of computing compatible paths on labelled point sets. Equivalently, we seek a permutation of the labels $1, 2, \ldots, n$ that corresponds to a noncrossing (plane) path in P and in Q. Figures 1(a)–(b) show a positive instance of this problem, and Figures 1(c)–(d) depict an affirmative answer.

Our results. We describe a quadratic-time dynamic programming algorithm that either finds compatible paths for two simple polygons, where the paths are restricted to remain inside the closed polygons, or reports that no such path exists. For the more limited case of two point sets in convex position, we give a linear time algorithm to find compatible paths (if they exist). For two general point sets, we give an $O(n^2 \log n)$-time algorithm to find compatible monotone paths (if they exist). Finding (unrestricted) compatible paths of point sets remains open.

1.1 Background

Saalfeld [11] first introduced compatible triangulations of labelled point sets, which he called "joint" triangulations. In Saalfeld's problem, each point set is enclosed inside an axis-aligned rectangle, and the goal is to compute compatible triangulations (possibly using Steiner points). Although not every pair of labelled point sets admit compatible triangulations, Saalfeld showed that one can always construct compatible triangulations using (possibly an exponential number of) Steiner points.

Aronov et al. [2] proved that $O(n^2)$ Steiner points are always sufficient and sometimes necessary to compatibly

*Université libre de Bruxelles (ULB), Belgium. ea.arseneva@gmail.com
†Department of Computer Science, University of Manitoba, Canada. bahoo@cs.umanitoba.ca
‡Cheriton School of Computer Science, University of Waterloo, Canada. ahmad.biniaz@gmail.com, {k22jain,alubiw}@uwaterloo.ca
§Department of Computer Science, Carleton University, Canada. pilukno@gmail.com, farah.chanchary@carleton.ca
¶Université libre de Bruxelles, Belgium & NYU, USA. jiacono@ac.ulb.be
‖Department of Computer Science, University of Saskatchewan, Canada. dmondal@cs.usask.ca
**NYU Tandon School of Engineering, Brooklyn, USA. khadijeh@nyu.edu
††Department of Mathematics, California State University Northridge, Los Angeles, CA, USA. csaba.toth@csun.edu

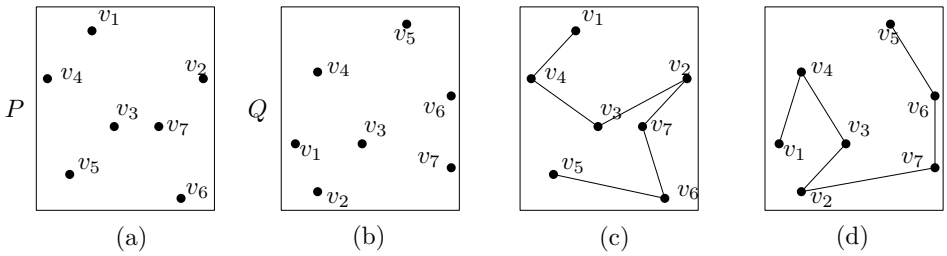


Figure 1: (a)–(b) A pair of labelled point sets P and Q. (c)–(d) A pair of compatible paths.

triangulate two polygons when the vertices of the polygons are labelled $1, 2, \ldots, n$ in clockwise order. Babikov et al. [3] extended the $O(n^2)$ bound to *polygonal regions* (i.e., polygons with holes), where the holes are also labelled 'compatibly' (with the same clockwise ordering of labels). The holes may be single points, so this includes Saalfeld's "joint triangulation" problem. Pach et al. [10] gave an $\Omega(n^2)$ lower bound on the number of Steiner points in such scenarios.

Lubiw and Mondal [9] proved that finding the minimum number of Steiner points is NP-hard for the case of polygonal regions. The complexity status is open for the case of polygons, and also for point sets. Testing for compatible triangulations without Steiner points may be an easier problem. Aronov et al. [2] gave a polynomial-time dynamic programming algorithm to test whether two polygons admit compatible triangulations without Steiner points. But testing whether there are compatible triangulations without Steiner points is open for polygonal regions, as well as for point sets.

The compatible triangulation problem seems challenging even for unlabelled point sets (i.e., when a bijection between P and Q can be chosen arbitrarily). Aichholzer et al. [1] conjectured that every pair of unlabelled point sets (with the same number of points on the convex hull) admit compatible triangulations without Steiner points. So far, the conjecture has been verified only for point sets with at most three interior points.

Let G_S be a complete geometric graph on a point set S. Let $H(S)$ be the *intersection graph* of the edges of G_S, i.e., each edge of G_S corresponds to a vertex in $H(S)$, and two vertices are adjacent in $H(S)$ if and only if the corresponding edges in G_S properly cross (i.e., the open line segments intersect). Every plane triangulation on S has $3n-3-h$ edges, where h is the number of points on the convex hull of S, and thus corresponds to a maximum independent set in $H(S)$. In fact, $H(S)$ belongs to the class of *well-covered graphs*. (A graph is well covered if every maximal independent set of the graph has the same cardinality). A rich body of research attempts to characterize well-covered graphs [6, 13]. Deciding whether two point sets, P and Q, admit compatible triangulations is equivalent to testing whether $H(P)$ and $H(Q)$ have a common independent set of size $3n-3-h$.

2 Paths in Polygons and Convex Point Sets

In this section we describe algorithms to find compatible paths on simple polygons and convex point sets. By compatible paths on polygons, we mean: given two polygons, find two compatible paths on the vertices of the polygons that are constrained to be non-exterior to the polygons. (See Figures 2(a)–(b).) Note that convex point sets correspond to a special case, where the polygons are the convex hulls. Not every two convex point sets admit compatible paths, e.g., 5-point sets where the points are labelled (1,2,3,4,5) and (1,3,5,2,4), resp., in counterclockwise order (Appendix A).

We first give a quadratic-time dynamic programming algorithm for simple polygons, and then a linear time algorithm for convex point sets.

We begin with two properties of any noncrossing path that visits all vertices of a simple polygon. Let P be a simple polygon with vertices $p_1, p_2, \ldots, p_n$ in some order (so the vertices have labels $1, 2, \ldots, n$). Let σ be a label sequence corresponding to a noncrossing path that lies inside P and visits all vertices of P. Define an *interval* on P to be a sequence of labels that appear consecutively around the boundary of P (in clockwise or counterclockwise order). For example, in Figure 2(a), one interval is $(2, 1, 7, 6)$. Define an *interval set* on P to be the unordered set of elements of an interval.

Claim 1 *The set of labels of every prefix of σ is an interval set on P. Furthermore, if the prefix does not contain all the labels, then the last label of the prefix corresponds to an endpoint of the interval.*

Proof. We proceed by induction on t, the length of the prefix, with the base case $t = 1$ being obvious. So assume the first $t-1$ labels form an interval set corresponding to interval I. Let ℓ be the t-th element of σ. Suppose vertex p_ℓ is not contiguous with the interval I on P. Let u and v be the two neighbors of p_ℓ around the polygon P. Then u and v do not belong to I, and so the path must visit both of them after p_ℓ. But then the subpath between u and v crosses the edge of the path that arrives at p_ℓ, contradicting the assumption that the path is noncrossing. Thus vertex p_ℓ must appear just

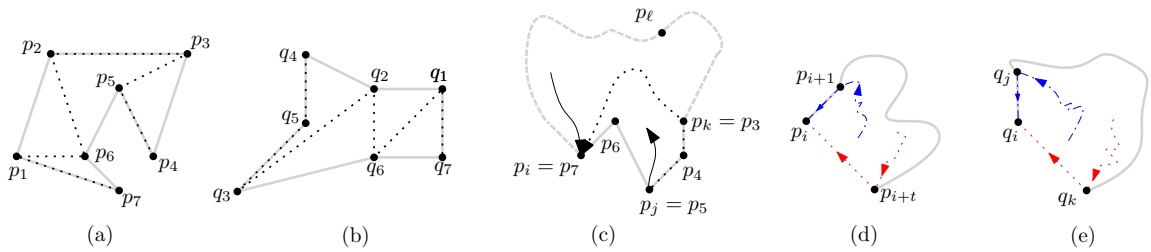


Figure 2: (a)–(b) Compatible paths on a pair of labelled polygons. The paths are drawn with dotted lines. (c) Illustration for Claim 2, where $I = (p_7, p_6, p_5, p_4, p_3)$. (d)–(e) Illustration for the dynamic programming algorithm.

before or after I, forming a longer interval with p_ℓ as an endpoint of the interval. □

Claim 2 *If I is an interval on P and σ does not start or end in I, then the labels of I appear in the same order in σ and in I (either clockwise or counterclockwise). Note that the labels need not appear consecutively in σ.*

Proof. Consider three labels i, j, k that appear in this order in I. Assume, for a contradiction, that these labels appear in a different order in σ and suppose, without loss of generality, that they appear in the order i, k, j in σ. Let ℓ be the last label of σ. Because ℓ does not lie in I, the order of vertices around P is p_i, p_j, p_k, p_ℓ. See, e.g., Figure 2(c) where $i, j, k = 7, 5, 3$. Then the subpath of σ from p_i to p_k crosses the subpath from p_k to p_ℓ, a contradiction. □

2.1 An $O(n^2)$-time dynamic programming algorithm

Let P, Q be two n-vertex simple polygons with labelled vertices. Let p_i (resp., q_i) be the vertex of P (resp., Q) with the label i.

Two vertices of a polygon are *visible* if the straight line segment connecting the vertices lies entirely inside the polygon. We precompute the visibility graph of each polygon in $O(n^2)$ time [8] such that later we can answer any visibility query in constant time.

Notation for our dynamic programming algorithm will be eased if we relabel so that polygon P has labels $1, 2, \ldots, n$ in clockwise order. For each label $i = 1, \ldots, n$ and each length $t = 1, \ldots, n$ let $I_Q(i, t, cw)$ denote the interval on Q of t vertices that starts at q_i and proceeds clockwise. Define $I_Q(i, t, ccw)$ similarly, but proceed counterclockwise from q_i. Define $I_P(i, t, cw)$ and $I_P(i, t, ccw)$ similarly. Note that $I_P(i, t, cw)$ goes from p_i to p_{i+t-1} (index addition modulo n).

We say that a path *traverses* interval $I_Q(i, t, d)$ (where $d = cw$ or ccw), if the path is noncrossing, lies inside Q, visits exactly the vertices of $I_Q(i, t, d)$ and ends at q_i. We make a similar definition for a path to traverse an interval $I_P(i, t, d)$.

Our algorithm will solve subproblems $A(i, t, d_P, d_Q)$ where i is a label from 1 to n, t is a length from 1 to n, and d_P and d_Q take on the values cw or ccw. This subproblem records whether there is a path that traverses $I_Q(i, t, d_Q)$ and a path with the same sequence of labels that traverses $I_P(i, t, d_P)$. If this is the case, we say that the two intervals are *compatible*. Observe that P and Q have compatible paths if and only if $A(i, n, d_P, d_Q)$ is true for some i, d_P, d_Q.

We initialize by setting $A(i, 1, d_P, d_Q)$ to TRUE for all i, d_P, d_Q, and then solve subproblems in order of increasing t. In order for intervals $I_Q(i, t+1, d_Q)$ and $I_P(i, t+1, d_P)$ to be compatible, the intervals of length t formed by deleting the last label, i, must also be compatible, with an appropriate choice of direction (cw or ccw) on those intervals. There are two choices in P and two in Q. We try all four combinations. For a particular combination to 'work' (i.e., yield compatible paths for the original length $t+1$ intervals), we need the last labels of the length t intervals to match, and we need appropriate visibility edges in the polygons for the last edge of the paths.

We give complete details for $A(i, t+1, cw, cw)$. See Figure 2(d)-(e). (The other four possibilities are similar.) Deleting label i from $I_P(i, t+1, cw)$ gives $I_P(i+1, t, cw)$ and $I_P(i+t, t, ccw)$. Let q_j be the vertex following q_i in clockwise order around Q and let q_k be the other endpoint of $I_Q(i, t+1, cw)$ (in practice, for efficiency, we would store k with the subproblem). Deleting label i from $I_Q(i, t+1, cw)$ gives $I_Q(j, t, cw)$ and $I_Q(k, t, ccw)$. The two possibilities for P and Q are shown by blue dash-dotted and red dotted lines in Figures 2(d) and (e), respectively. We set $A(i, t+1, cw, cw)$ TRUE if any of the following four sets of conditions hold:

1. Conditions for $I_P(i+1, t, cw)$ and $I_Q(j, t, cw)$: $i+1 = j$ and $A(i+1, t, cw, cw)$.
2. Conditions for $I_P(i+1, t, cw)$ and $I_Q(k, t, ccw)$: $i+1 = k$ and q_k sees q_i in Q and $A(i+1, t, cw, ccw)$. Note that the last edge of the path in Q must be (q_k, q_i) which is why we impose the visibility condition.
3. Conditions for $I_P(i+t, t, ccw)$ and $I_Q(j, t, cw)$: $i+t = j$ and p_{i+t} sees p_i in P and $A(i+t, t, ccw, cw)$.

4. Conditions for $I_P(i+t, t, ccw)$ and $I_Q(k, t, ccw)$: $i+t = k$ and p_{i+t} sees p_i in P and q_k sees q_i in Q and $A(i+t, t, ccw, ccw)$.

Since there are a quadratic number of subproblems, each taking constant time to solve, this algorithm runs in time $O(n^2)$, which proves:

Theorem 1 *Given two n-vertex polygons, each with points labelled from 1 to n in some order, one can find a pair of compatible paths or determine that none exist in $O(n^2)$ time.*

2.2 A linear-time algorithm for convex point sets

In this section we assume that the input is a pair of convex point sets P, Q, along with their convex hulls.

Given a label x, we first define a *greedy construction* to compute compatible paths starting at x. The output of the construction is an ordered sequence σ_x of labels. Using Claim 1 we keep track of the intervals in P and Q corresponding to σ_x. Initially σ_x contains the label x. Each subsequent step attempts to add a new label to σ_x, maintaining intervals in P and Q. Suppose the intervals corresponding to the current σ_x are I_P and I_Q in P and Q respectively. Let a and b be the labels of the vertices just before and just after interval I_P on the boundary of P. Similarly, let c and d be the labels of the vertices just before and just after interval I_Q on the boundary of Q. If $\{a, b\} = \{c, d\}$, then we add a and b to σ_x in arbitrary order. Otherwise, if there is one label in common between the two sets, we add that label to σ_x. Finally, if there are no common labels, then the construction ends. Let σ_x be a maximal sequence constructed as above.

Lemma 2 *P and Q have compatible paths starting at label x if and only if σ_x includes all n labels.*

Proof. If P and Q have compatible paths with label sequence σ starting at label x then by Claim 1 every prefix of σ corresponds to an interval in P and in Q, and we can build σ_x in exactly the same order as σ.

For the other direction, we claim to construct noncrossing paths in P and Q corresponding to σ_x. Observe that when we add one or two labels to σ_x, we can add the corresponding vertices to our paths because the point sets are convex, so every edge is allowable. Furthermore, the paths constructed in this way are noncrossing because the greedy construction of σ_x always maintains intervals in P and Q. Hence the new edges are outside the convex hull of the paths so far. □

Lemma 2 allows us to find compatible paths (if they exist) in $O(n^2)$ time by trying each label x as the initial label of the path. In order to improve this to linear time, we first argue that when σ_x does not provide compatible paths, then we need not try any of its other labels as the initial label.

Lemma 3 *If σ_x has length less than n, then no label in σ_x can be the starting label for compatible paths of P and Q.*

Proof. Suppose that there are compatible paths with label sequence s_y starting at a label y in σ_x. Let z be the first label that appears in s_y but not in σ_x. Let I_P and I_Q be the intervals corresponding to σ_x in P and Q respectively. By Claim 1 the prefix of s_y before z corresponds to intervals, say I'_P and I'_Q on P and Q, respectively. Then $I'_P \subseteq I_P$ and $I'_Q \subseteq I_Q$ (by our assumption that z is the first label of s_y not in σ_x). Since the vertex with label z must be adjacent to I'_P on the boundary of P and to I'_Q on the boundary of Q, and z does not appear in σ_x, therefore the vertex with label z must be adjacent to I_P on the boundary of P and to I_Q on the boundary of Q. But then our construction would add label z to σ_x. □

We will use Lemma 3 to show that we can eliminate some labels entirely when σ_x is found to have length less than n. Suppose σ_x does not include all labels. Let I_P and I_Q be the intervals on P and Q, respectively, corresponding to the set of labels of σ_x. Let a and b be the labels that appear at the endpoints of I_P.

Suppose P and Q have compatible paths (of length n) with label sequence σ. Then by Lemma 2 the initial and final label of σ lie outside of σ_x. Furthermore, by Claim 2, the set of labels of σ_x must appear consecutively and in the same order around P and around Q (either clockwise or counterclockwise). Our algorithm checks whether I_P and I_Q have the same ordered lists of labels. If not, then there are no compatible paths.

So suppose that I_P and I_Q have the same ordered lists of labels. Then the endpoints of I_Q must have labels a and b. We will now reduce to a smaller problem by discarding all internal vertices of I_P and I_Q. Let P' and Q' be the point sets formed from P and Q, respectively, by deleting the vertices with labels in $\sigma_x - \{a, b\}$.

Lemma 4 *Suppose z is a label appearing in P'. P and Q have compatible paths with initial label z if and only if P' and Q' have compatible paths with initial label z.*

Proof. If P and Q have compatible paths (of length n) with initial label z, then we claim that deleting from those paths the vertices with labels in $\sigma_x - \{a, b\}$ yields compatible paths of P' and Q' with initial label z. It suffices to show that if we delete one vertex from a noncrossing path on points in convex position then the resulting path is still noncrossing. The two edges incident to the point to be deleted form a triangle, and the new path will use the third side of the triangle. Since the points are in convex position, the triangle is empty of

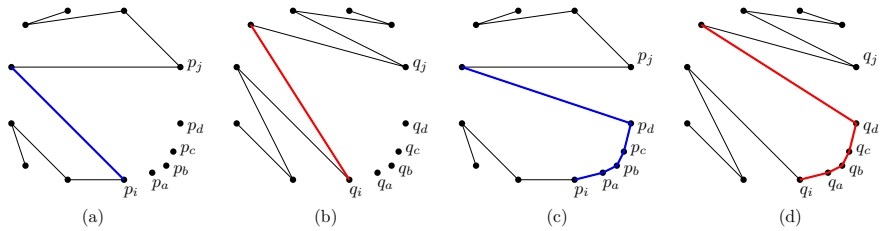


Figure 3: (a)–(b) Compatible paths on the point sets $P \setminus \{p_a, p_b, p_c, p_d\}$ and $Q \setminus \{q_a, q_b, q_c, q_d\}$. (c)–(d) Insertion of the deleted points keeps the paths compatible.

other points, and so the new edge does not cross any other edge of the path.

For the other direction, suppose that σ' is a label sequence of compatible paths of P' and Q' with initial label z. Suppose without loss of generality that label a comes before label b in σ'. Construct a sequence σ by adding the labels of $\sigma_x - \{a, b\}$ after a in σ' in the order that they appear in I_P. It remains to show that the corresponding paths in P and Q are noncrossing. This follows from the fact that in both P and Q the added points appear consecutively around the convex hull following the point with label a. □

We can now prove the main result of this section.

Theorem 5 *Given two sets of n points in convex position (along with their convex hulls) each with points labelled from 1 to n, one can find a pair of compatible paths or determine that none exist in linear time.*

Proof. The algorithm is as described above. At each stage we try some label x to be the initial label of compatible paths, by computing σ_x using the greedy construction. If σ_x has length n we are done. Otherwise if σ_x has length 1 or 2, then we have ruled out the labels in σ_x as initial labels. Finally, if σ_x has length less than n and at least 3 then we test whether the intervals corresponding to σ_x in P and Q have the same ordering, and if they do, then we apply the reduction described above and recurse on the smaller instance as justified by Lemma 4.

The running time of the algorithm is determined by the length of all the σ-sequences we compute. Define a σ-sequence to be 'long' or 'short' depending on whether it contains at least three labels or not. Every long sequence of length ℓ reduces the number of points by $(\ell-2)$ and requires $O(\ell)$ time. Thus, long sequences take $O(n)$ time in total. Computing any short sequence takes $O(1)$ time. Since for each label, we compute σ at most once, the short sequences also take $O(n)$ time in total. □

3 Monotone Paths in General Point Sets

In this section we examine arbitrary point sets in general position, but we restrict the type of path.

Let P be a point set in general position. An ordering σ of the points of P is called *monotone* if there exists some line ℓ such that the orthogonal projection of the points on ℓ yields the order σ. A *monotone path* is a path that corresponds to a monotone ordering. Note that every monotone path is noncrossing.

Two points sets P and Q each labelled $1, 2, \ldots, n$ have *compatible monotone paths* if there is an ordering of the labels that corresponds to a monotone path in P and a monotone path in Q. To decide whether compatible monotone paths exist, we can enumerate all the monotone orderings of P, and for each of them check in linear time whether it determines a monotone path in Q.

A method for enumerating all the monotone orderings of a point set P was developed by Goodman and Pollack:

Theorem 6 (Goodman and Pollack [7]) *Let ℓ_0 be a line not orthogonal to any line determined by two points of P. Starting with $\ell = \ell_0$, rotate the line ℓ through 360° counter-clockwise about a fixed point. Projecting the points onto ℓ as it rotates gives all the possible monotone orderings of P. There are $2\binom{n}{2} = n(n-1)$ orderings, and each successive ordering differs from the previous one by a swap of two elements adjacent in the ordering.*

Furthermore, the sequence of swaps that change each ordering to the next one can be found in $O(n^2 \log n)$ time by sorting the $O(n^2)$ lines (determined by all pairs of points) by their slopes.

This gives a straight-forward $O(n^3)$ time algorithm to find compatible monotone paths, since we can generate the $O(n^2)$ monotone orderings of P in constant time per ordering, and check each one for monotonicity in Q in linear time.

We now present a more efficient $O(n^2 \log n)$ time algorithm. For ease of notation, relabel the points so that the order of points P along ℓ_0 is $1, 2, \ldots, n$. As the line ℓ rotates, let $L_0^P, L_1^P, \ldots L_{t-1}^P$, where $t = n(n-1)$, be the monotone orderings of P, and let S_P be the corresponding swap sequence. Similarly, let $L_0^Q, L_1^Q, \ldots L_{t-1}^Q$ be the monotone orderings of Q and let S_Q be the corresponding swap sequence (Figure 4). We need to find whether there exist some i and j such that $L_i^P = L_j^Q$.

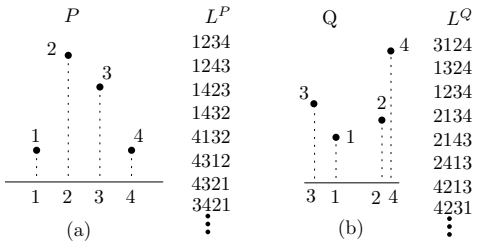


Figure 4: Illustration for computing compatible monotone paths.

As noted above, S_P and S_Q have size $O(n^2)$ and can be computed in time $O(n^2 \log n)$.

Recall that the *inversion number*, $I(L)$ of a permutation L is the number of pairs that are out of order. It is easy to see that the inversion numbers of the L_i^P's progress from 0 to $\binom{n}{2}$ and back again. In particular, $I(L_i^P) = i$ for $0 \leq i \leq \binom{n}{2}$. Our algorithm will compute the inversion numbers of the L_j^Q's, which also have some structure. Let I_j be the inversion number of L_j^Q. Note that we can compute I_0 in $O(n \log n)$ time—sorting algorithms can be modified to do this [5].

Claim 3 *For all j, $1 \leq j \leq n(n-1)$, I_j differs from I_{j-1} by ± 1, and can be computed from I_{j-1} in constant time.*

Proof. L_j^Q is formed by swapping one pair of adjacent elements in L_{j-1}^Q. If this swap moves a smaller element after a larger one then $I_j = I_{j-1}+1$. Otherwise, it is $I_{j-1}-1$. □

The main idea of our algorithm is as follows. If $L_j^Q = L_i^P$, then they must have the same inversion number, I_j. There is one value of i in the range $0 \leq i < \binom{n}{2}$ that gives this inversion number, namely $i = I_j$. There is also one value of i in the second half of the range that gives this inversion number, but we can ignore the second half of the range based on the following:

Remark 1 *If there exist i, j such that $L_i^P = L_j^Q$, then there is such a pair with i in the first half of the index range, i.e., $0 \leq i < \binom{n}{2}$.*

Proof. The second half of each list of orderings contains the reversals of the orderings in the first half [7]. Thus if there is a match $L_i^P = L_j^Q$ then the reversals of the two orderings also provide a match, say $L_{i'}^P = L_{j'}^Q$, and either i or i' is in the first half of the index range. □

Our plan is to iterate through the orderings L_j^Q for $0 \leq j < n(n-1)$. Since each ordering differs from the previous one by a single swap, we can update from one to the next in constant time. For each j, we will check if L_j^Q is equal to $L_{I_j}^P$, i.e., for each j, $0 \leq j < n(n-1)$ we will compute the following four things:

- L_j^Q, I_j, $L_{I_j}^P$, and
- H_j, which is the *Hamming distance*—i.e., the number of mismatches—between L_j^Q and $L_{I_j}^P$

If we find a j with $H_j = 0$ then we output L_j^Q and $L_{I_j}^P$ as compatible monotone paths. Otherwise, we declare that no compatible monotone paths exist. Correctness of this algorithm follows from Remark 1 and the discussion above:

Claim 4 *P and Q have compatible monotone paths if and only if $H_j = 0$ for some j, $0 \leq j < n(n-1)$.*

We now give the details of how to perform the above computations. For $j = 0$ we will compute everything directly, and for each successive j, we will show how to update efficiently. We initialize the algorithm at $j = 0$ by computing L_0^Q and I_j in $O(n \log n)$ time, $L_{I_j}^P$ in $O(n^2)$ time, and H_j in linear time.

Now consider an update from $j-1$ to j. As already mentioned, L_j^Q differs from L_{j-1}^Q by one swap of adjacent elements, so we can update in constant time. By Lemma 3, I_j differs from I_{j-1} by ± 1 and we can compute it in constant time. This also means that $L_{I_j}^P$ differs from $L_{I_{j-1}}^P$ by one swap of adjacent elements, so we can update it in constant time.

Finally, we can update the Hamming distance in a two-step process as the two orderings change. When we update from L_{j-1}^Q to L_j^Q, two positions in the list change, and we can compare them to the same positions in $L_{I_{j-1}}^P$ to update from H_{j-1} to obtain the number of mismatches between L_j^Q and $L_{I_{j-1}}^P$. When we update to $L_{I_j}^P$, two positions in this list change, and we can compare them to the same positions in $L_{I_j}^Q$ to update to H_j. This two-step process takes constant time.

In total, we spend $O(n^2)$ time on initialization and constant time on each of $O(n^2)$ updates, for a total of $O(n^2)$ time. We thus obtain the following theorem.

Theorem 7 *Given two point sets, each containing n points labelled from 1 to n, one can find a pair of compatible monotone paths or determine that none exist in $O(n^2 \log n)$ time.*

Acknowledgement: We thank the organizers of the Fields Workshop on Discrete and Computational Geometry, held in July 2017 at Carleton University. E. Arseneva is supported in part by the SNF Early Postdoc Mobility grant P2TIP2-168563 and by F.R.S.-FNRS; A. Biniaz, K. Jain, A. Lubiw, and D. Mondal are supported in part by NSERC.

References

[1] O. Aichholzer, F. Aurenhammer, F. Hurtado, and H. Krasser. Towards compatible triangulations. *Theoretical Computer Science*, 296(1):3–13, 2003.

[2] B. Aronov, R. Seidel, and D. L. Souvaine. On compatible triangulations of simple polygons. *Computational Geometry*, 3:27–35, 1993.

[3] M. Babikov, D. L. Souvaine, and R. Wenger. Constructing piecewise linear homeomorphisms of polygons with holes. In *Proceedings of the 9th Canadian Conference on Computational Geometry (CCCG)*, 1997.

[4] W. V. Baxter III, P. Barla, and K. Anjyo. Compatible embedding for 2D shape animation. *IEEE Transactions on Visualization and Computer Graphics*, 15(5):867–879, 2009.

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

[6] A. S. Finbow, B. L. Hartnell, and M. D. Plummer. On well-covered pentagonalizations of the plane. *Discrete Applied Mathematics*, 224:91–105, 2017.

[7] J. E. Goodman and R. Pollack. On the combinatorial classification of nondegenerate configurations in the plane. *Journal of Combinatorial Theory, Series A*, 29(2):220–235, 1980.

[8] J. Hershberger. An optimal visibility graph algorithm for triangulated simple polygons. *Algorithmica*, 4(1-4):141–155, 1989.

[9] A. Lubiw and D. Mondal. On compatible triangulations with a minimum number of Steiner points. In *Proceedings Canadian Conference on Computational Geometry (CCCG)*, pages 101–106, 2017. http://arxiv.org/abs/1706.09086.

[10] J. Pach, F. Shahrokhi, and M. Szegedy. Applications of the crossing number. *Algorithmica*, 16(1):111–117, 1996.

[11] A. Saalfeld. Joint triangulations and triangulation maps. In *Proceedings of the Third Annual Symposium on Computational Geometry (SoCG)*, pages 195–204. ACM, 1987.

[12] V. Surazhsky and C. Gotsman. High quality compatible triangulations. *Engineering with Computers*, 20(2):147–156, 2004.

[13] D. Tankus and M. Tarsi. The structure of well-covered graphs and the complexity of their recognition problems. *J. Comb. Theory, Ser. B*, 69(2):230–233, 1997.

Appendix A

In this section we show that for every $n \geq 5$, there exist two convex labelled point sets, each containing n points, that do not admit compatible trees. Note that this also rules out the existence of compatible paths.

Claim 5 *Let P and Q be point sets in convex position, each containing $n \geq 2$ points labelled by $\{1, 2, \ldots, n\}$. If they admit a compatible tree that is not a star, then there exists a partition $\{1, 2, \ldots, n\} = A \cup B$ such that $2 \leq |A| \leq |B| \leq n-2$ such that A and B are interval sets for both P and Q.*

Proof. Suppose that P and Q admit a compatible tree T, which is not a star. Then T has an edge e between two vertices of degree two or higher. The deletion of e decomposes T into two subtrees, say T_1 and T_2, each with at least two vertices. The vertex sets of T_1 and T_2, resp., correspond to an interval set in P and Q. □

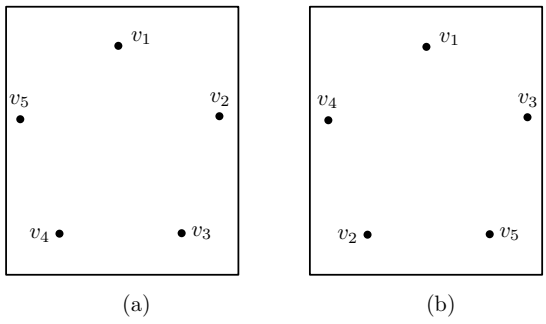


Figure 5: Illustration for Lemma 8.

Theorem 8 *For every integer $n \geq 5$, there exist two sets, P_n and Q_n, each of n labelled points in convex position, such that P_n and Q_n do not admit any compatible tree.*

Proof. For $n = 5$, let P_5 and Q_5 be point sets labelled $(1, 2, 3, 4, 5)$ and $(1, 3, 5, 2, 4)$, respectively, in counterclockwise order (Figure 5). If a compatible star exists, then the four leaves would appear in the same counterclockwise order in both P_5 and Q_5 (by the definition of compatibility). However, the two convex sets have distinct counterclockwise 4-tuples. If there is a compatible tree that is not a star, then by Claim 5, a 2-element set $A \subset \{1, 2, 3, 4, 5\}$ is an interval set for both P_5 and Q_5. However, all five consecutive pairs along the convex hull of P_5 are nonconsecutive in the convex hull of Q_5. Therefore, P_5 and Q_5 do not admit any compatible tree.

For $n > 5$, we can construct P_n and Q_n analogously. Let P_n be labelled $(1, 2 \ldots, n)$ in counterclockwise order. For $i = 0, 1, 2, 3, 4$, let N_i be the sequence of labels in $\{1, 2, \ldots, n\}$ congruent to i modulo 5 in increasing order. Now let Q_n be labelled by the concatenation of the sequences N_1, N_3, N_0, N_2, N_4 in counterclockwise order.

If a compatible star exists, then the $n-1$ leaves would appear in the same counterclockwise order in both P_n and Q_n (by the definition of compatibility). However, the both neighbors of a vertex in P_n are different from the two neighbors in Q_n, consequently P_n and Q_n do not share any counterclockwise $(n-1)$-tuple. If there is a compatible tree that is not a star, then by Claim 5, there is a partition $\{1, 2, \ldots, n\} = A \cup B$ into interval sets, where $|A|, |B| \geq 2$. However, A and B cannot partition any subset of 5 consecutive elements in sequence $(1, 2, \ldots, n)$, similarly to the case when $n = 5$. Consequently, P_n and Q_n do not admit any compatible tree. □

Ladder-Lottery Realization

Katsuhisa Yamanaka* Takashi Horiyama† Takeaki Uno‡ Kunihiro Wasa§

Abstract

A *ladder lottery* of a permutation $\pi = (p_1, p_2, \ldots, p_n)$ is a network with n vertical lines and zero or more horizontal lines each of which connects exactly two consecutive vertical lines. The top ends and the bottom ends of the vertical lines correspond to the identity permutation and π, respectively. Each horizontal line corresponds to an intersection of two vertical lines. Suppose that we are given a permutation π of $[n] = \{1, 2, \ldots, n\}$ and a multi-set S of intersections each of which is a pair of elements in $[n]$. Then Ladder-Lottery Realization problem asks whether or not there is a ladder-lottery of π in which each intersection in S appears exactly once. We show that Ladder-Lottery Realization problem is NP-complete. We also present some positive results of Ladder-Lottery Realization and its variant.

1 Introduction

A *ladder lottery*, known as the "Amidakuji" in Japan, is a very common way to obtain a "random" assignment. Japanese kids often use ladder lotteries to determine an assignment in a group. Let us show an example of how to use ladder lotteries. Suppose that, in an elementary school, we have to determine a group leader among n classmates. First, a teacher draws n vertical lines in a notebook and ticks off one of the bottom ends of the vertical lines so that any student cannot predict where the tick-mark is. See Figure 1(a). Second, the teacher covers the bottom ends of all vertical lines, then the teacher draws some horizontal lines connecting adjacent vertical lines (Figure 1(b)). Third, each student chooses the top end of a vertical line (Figure 1(c)). Finally, the teacher takes off the cover. The obtained figure gives an assignment (Figure 1(d)).

Formally, for a permutation $\pi = (p_1, p_2, \ldots, p_n)$ of $[n] = \{1, 2, \ldots, n\}$, a ladder lottery is a network with n vertical lines (*lines* for short) and zero or more horizontal lines (*bars* for short) each of which connects exactly two consecutive vertical lines. The top ends of lines correspond to the identity permutation $(1, 2, \ldots, n)$. The bottom ends of lines correspond to π. See Figure 2(a).

*Iwate University, Japan. yamanaka@cis.iwate-u.ac.jp
†Saitama University, Japan. horiyama@al.ics.saitama-u.ac.jp
‡National Institute of Informatics, Japan. uno@nii.ac.jp
§National Institute of Informatics, Japan. wasa@nii.ac.jp

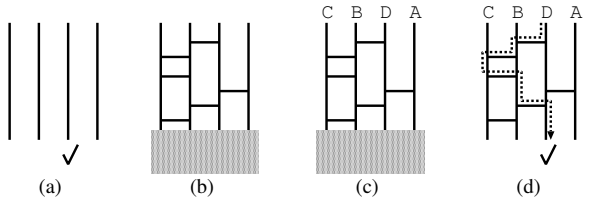


Figure 1: An example of how to use a ladder lottery. Imagine the situation that we choose a leader among four students A, B, C, and D. (a) four vertical lines and a tick-mark. (b) The tick-mark is hided and six horizontal lines are drawn by a teacher according to his or her intuition. (c) Each student chooses a top end of a vertical line. (d) The result of the obtained assignment. In this assignment, D is a leader.

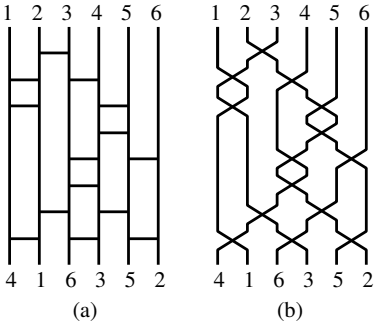


Figure 2: (a) A ladder lottery of (4,1,6,3,5,2) and (b) its pseudoline drawing.

Each element i in $[n]$ starts from the top end of ith line from the left, and goes down along the line, then whenever i comes to an end of a bar, i goes horizontally along the bar to the other end, then goes down again. Finally, i reaches the bottom end of jth line from the left such that $i = p_j$. We can regard a bar as a modification of the current permutation, and a sequence of such modifications in a ladder lottery always results in the identity permutation.

Ladder lotteries of the reverse permutations have a one-to-one correspondence to pseudoline arrangements [12]. The route of an element from a top end to a bottom end corresponds to a pseudoline and a bar corresponds to an intersection of two pseudolines. To calculate the number of pseudoline arrangements, some enumeration and counting algorithms of ladder lotteries

were presented in [5, 12]. The history of the counting results is shown in the Online Encyclopedia of Integer Sequences [7]. In the area of algebra, a ladder lottery is regarded as a decomposition of a permutation into adjacent transpositions. The top ends of lines correspond to the identity permutation. The bottom ends of lines correspond to a permutation. Each bar corresponds to an adjacent transposition. From these viewpoints, ladder lotteries have been studied as mathematically attractive objects. In recent years, from the viewpoint of theoretical computer science, some problems on ladder lotteries are considered: counting [11], random generation [11], enumeration [5, 12, 10, 11], reconfiguration [3].

A few years ago, Yamanaka et al. [8] proposed the puzzle, called Token Swapping problem: We are given a permutation and a set of allowable transpositions. The Token Swapping problem asks to find a minimum-length decomposition using only transpositions in the set.[1] Recently, this puzzle and its variants have been actively studied [1, 4, 6, 9].

In this paper, we propose a new puzzle regarding ladder lotteries. The purpose of Token Swapping problem is to find a shortest decomposition of a permutation. On the other hand, we consider the problem, called Ladder-Lottery Realization, of constructing a target permutation using compositions of designated transpositions. Let us describe our problem more formally. We are given a target permutation π of and a multi-set S of transpositions. The problem asks whether one can construct the target permutation by composing each transposition in the set exactly once. In this paper, we investigate the computational complexity of Ladder-Lottery Realization problem. We show the NP-completeness of the problem and give some positive results for the problem and its variant.

Due to page limitation, all proofs are omitted.

2 Preliminaries

A *ladder lottery* of a permutation $\pi = (p_1, p_2, \ldots, p_n)$ is a network with n vertical lines (*lines* for short) and zero or more horizontal lines (*bars* for short) each of which connects two consecutive vertical lines. The top ends of the n lines correspond to the identity permutation. The bottom ends of the n lines correspond to π. See Figure 2(a). Each element i in the identity permutation starts the top end of ith line from the left, and goes down along the line, then whenever i comes to an end of a bar, i goes to the other end and goes down again, then finally i reaches the bottom end of jth line such that $i = p_j$. By representing the route for each element i as a pseudoline and each bar as an intersection of two pseudolines, one can represent a ladder lottery as a drawing of pseudolines. In this paper, for convenience of descriptions, we use the pseudoline drawing to represent a ladder lottery. For example, Figure 2(b) is the pseudoline drawing of the ladder lottery in Figure 2(a). From now on, if it is clear from the context, we call the route of an element as a pseudoline. Clearly, we can regard that a pseudoline in the pseudoline drawing of a ladder lottery forms a y-monotone curve. Hence, in the following, we assume that any pseudoline is y-monotone.

Now, let us define Ladder-Lottery Realization problem. Suppose that we are given a permutation $\pi = (p_1, p_2, \ldots, p_n)$ of $[n]$ and a multi-set S of intersections each of which is a pair of elements in $[n]$. Then Ladder-Lottery Realization asks whether or not there is a ladder-lottery of π in which each intersection in S appears exactly once. For example, suppose that we are given the permutation (4,1,6,3,5,2) and the multi-set

$$\{\{1,3\}^2, \{1,4\}, \{2,3\}, \{2,4\}^3, \{2,5\}^3, \{2,6\}, \{3,4\}, \{3,6\}, \{5,6\}^3\}$$

of intersections, where $\{i,j\}^k$ means k $\{i,j\}$s. Then, the answer is yes, since the ladder lottery in Figure 2(a) is a solution.

3 Hardness of ladder-lottery realization

We give a reduction from a well-known NP-complete problem One-in-Three 3SAT:

Problem: One-in-Three 3SAT [2]
Instance: Set X of variables, collection C of clauses over X such that each clause in C contains exactly three literals.
Question: Is there a truth assignment for X such that each clause in C has exactly one true literal?

Let $I_S = (X, C)$ be an instance of One-in-Three 3SAT, where $X = \{x_1, x_2, \ldots, x_n\}$ is a set of variables and $C = \{C_1, C_2, \ldots, C_m\}$ is a collection of clauses. We may assume without loss of generality that any clause $C_i \in C$ does not contain both the positive and the negative literals of any variable in X. We denote by n and m the numbers of variables and clauses, respectively. We are going to construct an instance $I_R = (\pi, S)$ of Ladder-Lottery Realization from I_S, where π is a permutation and S is a multi-set of intersections.

To reduce I_S to I_R, we prepare the gadgets: a room gadget, a drawer gadget, a variable gadget, a clause gadget, and an assignment gadget. Let us explain these gadgets one by one.

[1] Actually, the Token Swapping problem is defined as a puzzle consisting of n tokens on n-vertex graph where each token has a distinct starting vertex and a distinct target vertex it wants to reach, and the only allowed transformation is to swap the tokens on adjacent vertices [8].

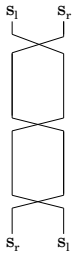


Figure 3: Room gadget with 4 rooms.

Room gadget

First, we define a room gadget. The *room gadget* consists of two pseudolines s_ℓ, s_r and a multi-set $S_R(I_S)$ of intersections. The top ends of the two pseudolines appear in the order s_ℓ, s_r and their bottom ends appear in the reverse order. We define the multi-set of intersections so that the two pseudolines form $4n$ regions:

$$S_R(I_S) := \{s_l, s_r\}^{4n-1}.$$

Then the two pseudolines intersect $4n-2$ closed regions and the top and bottom regions enclosed by s_ℓ and s_r. See Figure 3. We call the ith region from the top the *ith room*.

Later, we use two rooms to represent an assignment of each variable. More precisely, we use the $(4i-3)$rd and $(4i-1)$th rooms to represent the assignment of the variable x_i for $i = 1, 2, \ldots, n$.

Drawer gadget

We next define a *drawer gadget*, which consists of $4n$ pseudolines $d_1, d'_1, d_2, d'_2, \ldots, d_{2n}, d'_{2n}$ and a multi-set $S_D(I_S)$ of intersections. The top ends of the pseudolines are arranged in the order $d'_{2n}, d_{2n}, d'_{2n-1}, d_{2n-1}, \ldots, d'_1, d_1$ in the left region of the pseudolines of the room gadget and their bottom ends are arranged in the reverse order, namely $d_1, d'_1, d_2, d'_2, \ldots, d_{2n}, d'_{2n}$ (see Figure 4).

We define $S_D(I_S)$ such that d_i and d'_i for each $i = 1, 2, \ldots, 2n$ come to the $(2i-1)$th and $(2i)$th rooms and leave the rooms, respectively. Besides, every pseudoline in the drawer gadget crosses with all other pseudolines except itself in the gadget exactly once. The formal definition of $S_D(I_S)$ is as follows:

$$\begin{aligned}
S_D(I_S) := \\
&\{\{d_i, d_{i'}\}, \{d_i, d'_{i'}\} \mid i, i' = 1, 2, \ldots, 2n \text{ and } i < i'\} \\
&\cup \{\{d_i, d'_i\} \mid i = 1, 2, \ldots, 2n\} \\
&\cup \{\{d'_i, d_{i'}\}, \{d'_i, d'_{i'}\} \mid i, i' = 1, 2, \ldots, 2n \text{ and } i < i'\} \\
&\cup \{\{d_i, s_\ell\}^2 \mid i = 1, 2, \ldots, 2n\}. \\
&\cup \{\{d'_i, s_r\}^2 \mid i = 1, 2, \ldots, 2n\}.
\end{aligned}$$

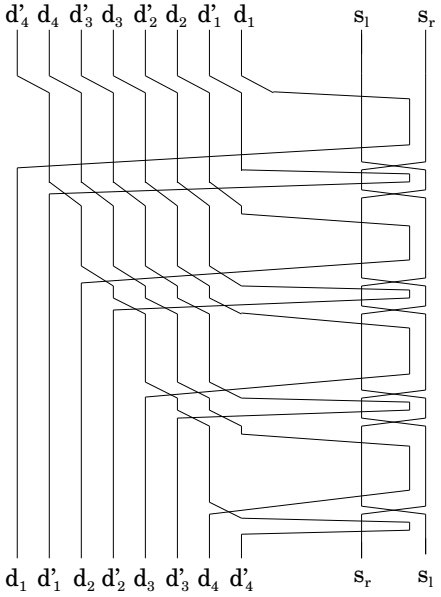


Figure 4: Drawer gadget.

Figure 4 shows an example of pseudolines of a drawer gadget and a room gadget. From the definition of $S_D(I_S)$, one can observe the form of a pseudoline in the drawer gadget, as follows. First, d_i for each $i = 1, 2, \ldots, 2n$ crosses with every $d_{i'}$ and $d'_{i'}$ with $i' < i$. Then d_i crosses with s_ℓ two times. That is, d_i comes to $(2i-1)$th room and leaves it. Then d_i crosses with every $d_{i'}$ with $i' > i$ and every $d'_{i''}$ with $i'' \geq i$. As a result, the bottom end of d_i is $(2i-1)$th one from the left among the pseudolines of the drawer gadget. The shape of d'_i for each $i = 1, 2, \ldots, 2n$ is similar.

Now, we explain why d_i and d'_i for $i = 1, 2, \ldots, 2n$ form the above shape more formally. For any y-coordinate, a pseudoline d_i (and d'_i) is *rightmost* if, in the y-coordinate, the x-coordinate of d_i (and d'_i) is the largest among all the pseudolines in a drawer gadget. The *rightmost y-coordinate set* of d_i (and d'_i) is the set of the y-coordinates in which d_i (and d'_i) is rightmost. From the definition of a drawer gadget, the pseudolines in the drawer gadget cross each other exactly once and the order of the bottom ends of the pseudolines is the reverse order of their top ends. Hence, it can be observed that a rightmost y-coordinate set of a pseudoline always forms an open interval. Since s_ℓ crosses with $d_1, d_2, \ldots, d_{2n}$ and does not cross with $d'_1, d'_2, \ldots, d'_{2n}$, s_ℓ crosses with d_i in a y-coordinate in the rightmost y-coordinate set of d_i. Similarly, s_r crosses with d'_i in a y-coordinate in the rightmost y-coordinate set of d'_i. Therefore, the drawing of the pseudolines of a drawer gadget and a room gadget is unique, as shown in Figure 4.

Variable gadget

Here, let us define a *variable gadget* consisting of n pseudolines and a multi-set $S_X(I_S)$ of intersections. We create a pseudoline $p(x_i)$ for each variable x_i, and arrange their top ends in the order $p(x_1), p(x_2), \ldots, p(x_n)$, and all the top ends appear in the right of s_r. We also define the order of their bottom ends as the same one.

Let us explain the outline of the form of $p(x_i)$ (Figure 5). $p(x_i)$ crosses with d_{2i-1} and d_{2i} but does not cross with s_ℓ. Hence, $p(x_i)$ crosses the two pseudolines in only the coresponding rooms. First, the pseudoline $p(x_i)$ crosses with other pseudolines to approach the room gadget. Then, $p(x_i)$ comes to and leaves two rooms one by one. In the rooms, $p(x_i)$ crosses with d_{2i-1} and d_{2i}. Finally, $p(x_i)$ crosses with other pseudolines to go back to its the original position. Now, we define the multi-set $S(p(x_i))$ of intersections for $p(x_i)$ as follows:

$$S(p(x_i)) := \{p(x_i), s_r\}^4 \cup \{p(x_i), d_{2i-1}\}^2 \cup \{p(x_i), d_{2i}\}^2 \cup \bigcup_{i'=1}^{i-1} \{p(x_i), p(x_{i'})\}^2.$$

Let us explain the shape of $p(x_i)$ more carefully. The multi-set $S(p(x_i))$ does not include $\{p(x_i), s_\ell\}$, and hence $p(x_i)$ cannot enter the left region of s_ℓ. However, $S(p(x_i))$ includes both $\{p(x_i), d_{2i-1}\}^2$ and $\{p(x_i), d_{2i}\}^2$. Hence, $p(x_i)$ comes to the $(4i-3)$rd and $(4i-1)$th rooms to cross with d_{2i-1} and d_{2i}, respectively. To approach the rooms, $p(x_i)$ crosses with $p(x_{i-1})$, $p(x_{i-2})$, $\ldots$, $p(x_1)$. Then, $p(x_i)$ arrives at the region next to the target rooms. First, $p(x_i)$ comes to the $(4i-3)$rd room, crosses with d_{2i-1} two times in the room, and leaves the room. Next, $p(x_i)$ comes to the $(4i-1)$th room, crosses with d_{2i} two times in the room, and leaves the room. Then, to go back to the original position, $p(x_i)$ crosses with $p(x_1)$, $p(x_2)$, $\ldots$, $p(x_{i-1})$ again.

We show an example in Figure 5. Note that, since $p(x_i)$ does not cross with s_ℓ, it has to cross with pseudolines of a drawer gadget only in the rooms to which the pseudolines come.

Now, let us define the multi-set of intersections of a variable gadget:

$$S_X(I_S) := \bigcup_{i=1}^{n} S(p(x_i)).$$

Clause gadget

A *clause gadget* consists of m pseudolines corresponding to the clauses in C and a multi-set $S_C(I_S)$ of intersections. We create a pseudoline $p(C_j)$ for each clause $C_j \in C$. The order of the top ends of the pseudolines is $p(C_1), p(C_2), \ldots, p(C_m)$ between the top ends of s_r and $p(x_1)$ (See Figure 6). The order of the bottom ends of the pseudolines is the same as the top ends. The bottom ends are arranged between the bottom ends of s_r and $p(x_1)$ (See Figure 6).

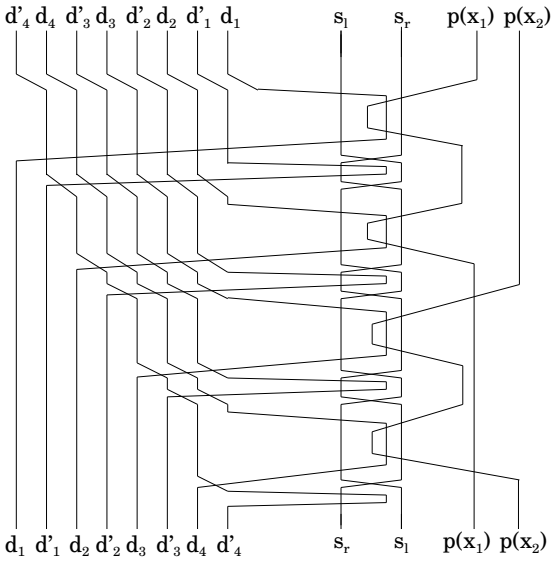


Figure 5: An example of a variable gadget for $n = 2$.

We design a multi-set of intersections for $p(C_j)$ for $j = 1, 2, \ldots, m$ so that $p(C_j)$ forms the shape below. If C_j includes a positive literal of x_i, then $p(C_j)$ comes to and leaves the $(4i-3)$rd room. If C_j includes a negative literal of x_i, $p(C_j)$ comes to and leaves the $(4i-1)$th room. Otherwise, C_j includes neither the positive nor negative literals of x_i, $p(C_j)$ comes to neither the $(4i-3)$rd nor $(4i-1)$th rooms. To force $p(C_j)$ to be such a shape, we define a multi-set of intersections, as follows. We denote by $L(C_j)$ the set of literals in C_j. Let $L(C_j) = \{\ell_{j,1}, \ell_{j,2}, \ell_{j,3}\}$. For each literal $\ell_{j,p}$, $p \in \{1, 2, 3\}$, we define the following multi-set of intersections.

$$\begin{aligned} S(\ell_{j,p}, C_j) := &\{\{p(C_j), p(C_{j'})\}^2 \mid j' < j \wedge \ell_{j,p} \notin L(C_{j'})\} \\ &\cup \{\{p(C_j), d_{2i-1}\}^2 \mid \ell_{j,p} = x_i\} \\ &\cup \{\{p(C_j), d_{2i}\}^2 \mid \ell_{j,p} = \overline{x_i}\} \\ &\cup \{\{p(C_j), s_r\}^2\} \end{aligned}$$

The intersections in the first set of $S(\ell_{j,p}, C_j)$ are used to approach the room gadget corresponding to $\ell_{j,p}$. If $\ell_{j,p} \in L(C_{j'})$ holds, $p(C_j)$ and $p(C_{j'})$ has no intersection. The intersections in the second and third sets are used to force $p(C_j)$ to come to the rooms corresponding to the literals of x_i.

Besides, we define the following multi-set of intersections for $p(C_j)$ and $p(x_i)$:

$$\begin{aligned} &S(\ell_{j,p}, C_j, x_i) := \\ &\quad \{\{p(C_j), p(x_i)\}^4 \mid \ell_{j,p} \neq x_i \wedge \ell_{j,p} \neq \overline{x_i}\} \\ &\quad \cup \{\{p(C_j), p(x_i)\}^2 \mid \ell_{j,p} = x_i \vee \ell_{j,p} = \overline{x_i}\} \end{aligned}$$

The intersections above are used so that $p(x_i)$ comes to the corresponding the room gadget.

Now we define the set of intersections for clauses, as follows:

$$S_C(I_S) := \left(\bigcup_{j=1}^{m}\bigcup_{p=1}^{3} S(\ell_{j,p}, C_j)\right) \cup \left(\bigcup_{i=1}^{n}\bigcup_{j=1}^{m}\bigcup_{p=1}^{3} S(\ell_{j,p}, C_j, x_i)\right).$$

We give an example shown in Figure 6. The example shows an reduced instance from the One-in-Three 3SAT instance (X, C), where $X = \{x_1, x_2, x_3, x_4\}$, $C = \{C_1, C_2\}$, $C_1 = (\overline{x_1} \vee x_2 \vee \overline{x_3})$, and $C_2 = (x_2 \vee x_3 \vee x_4)$.

Assignment gadget

The last gadget is the one for representing a truth-false assignment of variables. We define an *assignment gadget* consisting of a pseudoline a and a set of intersections for a. The top and bottom ends of a are respectively located in the left of d'_{2n} and d_1 (see Figure 6). We define that a crosses with each $p(x_i)$ twice for $i = 1, 2, \ldots, n$ and a crosses with s_ℓ $2n$ times but does not cross with s_r to make a cross with each $p(x_i)$ in either $(4i-3)$th or $(4i-1)$th room. If a crosses with $p(x_i)$ in $(4i-3)$rd room, then it means that x_i is assigned true. Otherwise, if a crosses with $p(x_i)$ in $(4i-1)$th room, then it means that x_i is assigned false. Besides, we force that a crosses with each $p(C_j)$ two times. This corresponds to make the clause C_j true. The pseudoline a touches each C_j exactly once, and hence this assignment corresponds to a solution of an instance of One-in-Three 3SAT. We can define the multi-set of intersections which implements such shape of a:

$$S_A(I_S) := \left(\bigcup_{i=1}^{n}\{a, p(x_i)\}^2\right) \cup \left(\bigcup_{i=1}^{2n}(\{a, d_i\}^{2n}, \{a, d'_i\}^{2n})\right) \cup \left(\bigcup_{j=1}^{m}\{a, C_j\}^2\right) \cup \{a, s_\ell\}^{2n}.$$

The first term is for the intersections with $p(x_i)$ for each $i = 1, 2, \ldots, n$. The second term is the intersections with the pseudolines in the drawer gadget to approach the rooms and to go back to the original position. Note that a does not have to go back to the leftmost region for each entrance to a room. In Figure 6, a goes back to the leftmost region immediately after each entrance to a room. This is just an example of the form of a. The third term is for the intersections with the pseudolines in the clause gadget. The last term is for the intersections with s_ℓ to come to rooms. The pseudoline a cannot go inside the right region of s_ℓ since there is no intersection $\{a, s_\ell\}$. Hence, a has to cross with the pseudolines of the variables and the clauses in the rooms.

Now, we are ready to describe a reduced instance of Ladder-Lottery Realization. Given an instance $I_S = (X, C)$ of One-in-Three 3SAT, we construct an instance $I_R = (\pi(I_S), S(I_S))$, where

$$\begin{aligned}\pi(I_S) = (&a, d_1, d'_1, d_2, d'_2, \ldots, d'_{2n}, d_{2n}, s_r, s_\ell,\\ &p(C_1), p(C_2), \ldots, p(C_m),\\ &p(x_1), p(x_2), \ldots, p(x_n))\end{aligned}$$

and

$$S(I_S) = S_R(I_S) \cup S_D(I_S) \cup S_X(I_S) \cup S_C(I_S) \cup S_A(I_S).$$

Using the reduction above, one can show NP-completeness of Ladder-Lottery Realization.

Theorem 1 Ladder-Lottery Realization *is NP-complete.*

4 Positive results

In this section, we give positive results. Let $I_R = (\pi, S)$ be an instance of Ladder-Lottery Realization, where π is a permutation of $[n]$ and S is a multi-set of intersections. If $\{i, j\}^k \in S$, we say that the *multiplicity* of $\{i, j\}$ in S is k.

Theorem 2 *Let $I_R = (\pi, S)$ be an instance of* Ladder-Lottery Realization. *If the multiplicity of every intersection in S is 1, one can determine whether or not I_R is a yes-instance in polynomial time.*

Now, let us consider a variant of Ladder-Lottery Realization problem. Suppose that we are given only a multi-set S of intersections each of which is a pair of elements in $[n]$. Then, AnyPerm-Ladder-Lottery Realization asks whether or not there is a ladder-lottery of a permutation in which each intersection in S appears exactly once. Note that, in this problem, we have no permutation as an input. The problem simply asks whether or not there is a ladder-lottery of "some permutation" for S.

Theorem 3 *Let S be a multi-set of intersections. If the multiplicity of every intersection in S is 1, one can solve* AnyPerm-Ladder-Lottery Realization *for S in polynomial time.*

In the case that the multiplicity of every intersection is odd, we can solve Ladder-Lottery Realization in polynomial time.

Theorem 4 *Let $I_R = (\pi, S)$ be an instance of* Ladder-Lottery Realization. *If the multiplicity of every intersection in S is odd, one can determine whether I_R is a yes-instance in polynomial time.*

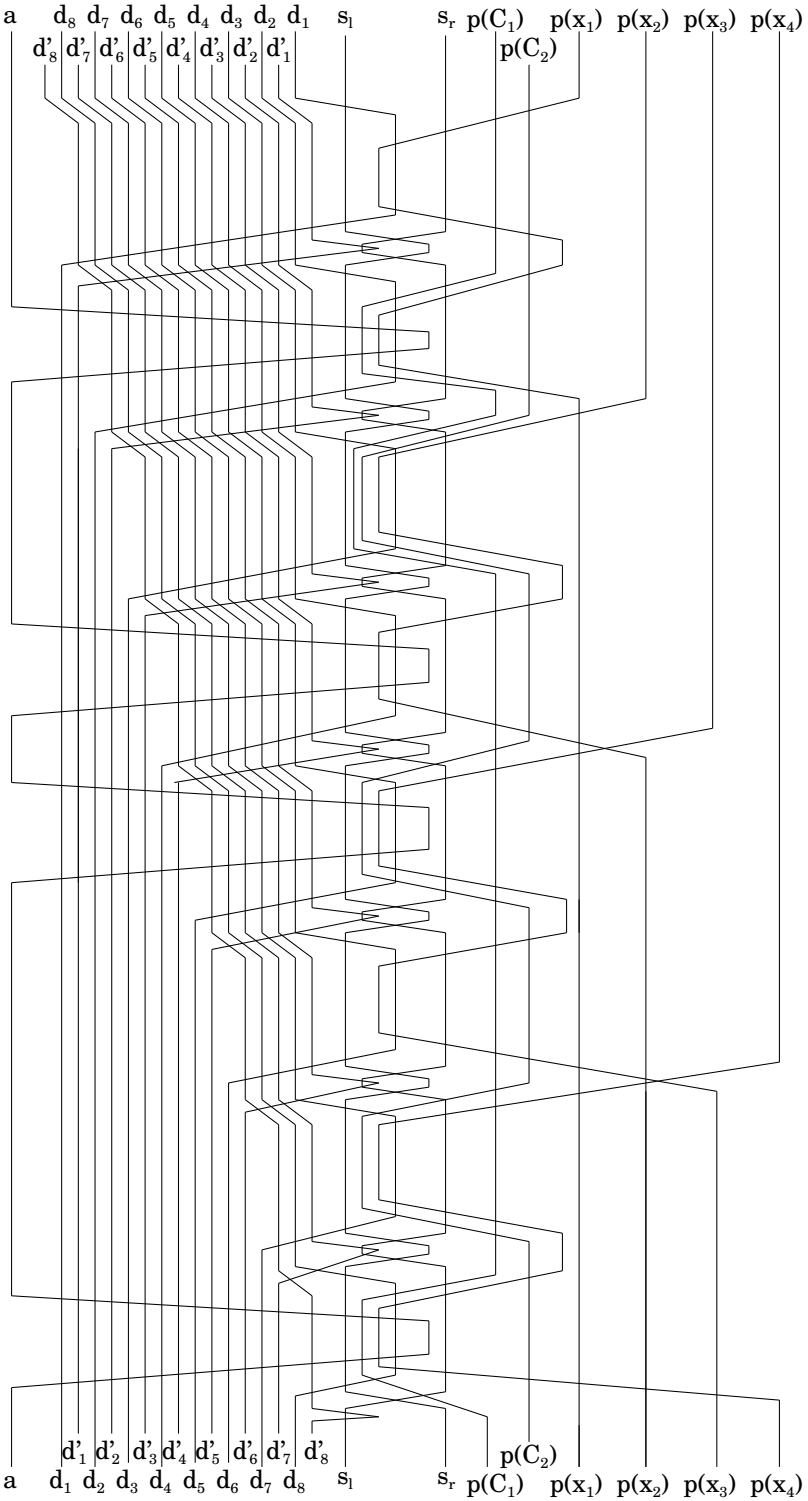


Figure 6: Reduced instance from (X, C) of a One-in-Three 3SAT instance, where $X = \{x_1, x_2, x_3, x_4\}$, $C = \{C_1, C_2\}$, $C_1 = (\overline{x_1} \vee x_2 \vee \overline{x_3})$, and $C_2 = (x_2 \vee x_3 \vee x_4)$. The assignment gadget represents $(x_1, x_2, x_3, x_4) = (0, 0, 1, 0)$.

References

[1] É. Bonnet, T. Miltzow, and P. Rzążewski. Complexity of token swapping and its variants. *Algorithmica*, pages 1–27, Oct 2017.

[2] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.

[3] T. Horiyama, K. Wasa, and K. Yamanaka. Reconfiguring optimal ladder lotteries. In *Proceedings of the 10th Japanese-Hungarian Symposium on Discrete Mathematics and Its Applications*, pages 217–224, May 2017.

[4] J. Kawahara, T. Saitoh, and R. Yoshinaka. The time complexity of the token swapping problem and its parallel variants. In *Proceeding of The 11th International Conference and Workshops on Algorithms and Computation*, volume 10167 of *Lecture Notes in Computer Science*, pages 448–459, 2017.

[5] J. Kawahara, T. Saitoh, R. Yoshinaka, and S. Minato. Counting primitive sorting networks by πdds. *Hokkaido University, Division of Computer Science, TCS Technical Reports*, TCS-TR-A-11-54, 2011.

[6] T. Miltzow, L. Narins, Y. Okamoto, G. Rote, A. Thomas, and T. Uno. Approximation and hardness of token swapping. In *Proceeding of 24th Annual European Symposium on Algorithms, ESA 2016*, pages 66:1–66:15, 2016.

[7] N. Sloane. The on-line encyclopedia of integer sequences. Published electronically at `https://oeis.org/A006245`. Accessed: 2018-05-02.

[8] K. Yamanaka, E. D. Demaine, T. Ito, J. Kawahara, M. Kiyomi, Y. Okamoto, T. Saitoh, A. Suzuki, K. Uchizawa, and T. Uno. Swapping labeled tokens on graphs. *Theoretical Computer Science*, 586:81–94, 2015.

[9] K. Yamanaka, T. Horiyama, D. Kirkpatrick, Y. Otachi, T. Saitoh, R. Uehara, and Y. Uno. Swapping colored tokens on graphs. *Theoretical Computer Science*, 729:1–10, 2018.

[10] K. Yamanaka and S. Nakano. Efficient enumeration of all ladder lotteries with k bars. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 97-A(6):1163–1170, 2014.

[11] K. Yamanaka and S. Nakano. Enumeration, counting, and random generation of ladder lotteries. *IEICE Transactions Information and Systems*, 100-D(3):444–451, 2017.

[12] K. Yamanaka, S. Nakano, Y. Matsui, R. Uehara, and K. Nakada. Efficient enumeration of all ladder lotteries and its application. *Theoretical Computer Science*, 411:1714–1722, 2010.

Away from Rivals

Kazuyuki Amano * Shin-ichi Nakano †

Abstract

Let P be a set of n points, and $d(p, q)$ be the distance between a pair of points p, q in P. We assume the distance is symmetric and satisfies the triangle inequality. For a point $p \in P$ and a subset $S \subset P$ with $|S| \geq 3$, the 2-dispersion cost $cost_2(p, S)$ of p with respect to S is the sum of (1) the distance from p to the nearest point in $S \setminus \{p\}$ and (2) the distance from p to the second nearest point in $S \setminus \{p\}$. The 2-dispersion cost $cost_2(S)$ of $S \subset P$ with $|S| \geq 3$ is $\min_{p \in S}\{cost_2(p, S)\}$.

In this paper we give a simple $1/8-$approximation algorithm for the 2-dispersion problem.

1 Introduction

Many facility location problems compute locations minimizing some cost or distance [4, 5]. While in this paper we consider a dispersion problem which computes locations maximizing some cost or distance [1, 2, 3, 6, 9, 10, 11].

Dispersion problems has an important application for information retrieval. It is desirable to find a small subset of a large data set, so that the small subset have a certain diversity. Such a small subset may be a good sample to overview the large data set [2], and diversity maximization has became an important concept in information retrieval.

A typical dispersion problem is as follows. Given a set P of points and an integer k, find k points subset S of P maximizing a designated cost. If the cost is the minimum distance between a pair of points in S then it is called the max-min dispersion problem, and if the cost is the sum of the distances between all pair of points in S then it is called the max-sum dispersion problem. Unfortunately both problems are NP-hard, even the distance satisfies the triangle inequality [9].

In this paper we consider a recently proposed related problem called the 2-dispersion problem [7, 8]. We give a simple approximation algorithm for the 2-dispersion problem, where the cost of a point in S is the sum of the distances to the nearest two points in S, and the cost of S is the minimum among the cost of points in S. Intuitively we wish to locate our k chain stores so that each store is located far away from the nearest two "rival" stores to avoid self-competition. We call the problem 2-dispersion problem. In [7, 8] more general variants, including max-min and max-sum dispersion problems are studied.

In this paper we give a simple approximation algorithm for the 2-dispersion problem defined above. Our algorithm computes a 1/8-approximate solution for the 2-dispersion problem. This is the first approximation algorithm for the 2-dispersion problem.

The remainder of the paper is organized as follows. Section 2 gives some definitions. Section 3 gives our simple approximation algorithm for the 2-dispersion problem. In Section 4 we consider more general problem called c-dispersion problem. Finally Section 4 is a conclusion.

2 Definitions

Let P be a set of n points, and $d(p, q)$ be the distance between a pair of points p, q in P. We assume that the distance is symmetric and satisfies the triangle inequality, meaning $d(p, q) = d(q, p)$ and $d(p, q) + d(q, r) \geq d(p, r)$.

For a point $p \in P$ and a subset $S \subset P$ with $|S| \geq 3$, the 2-dispersion cost $cost_2(p, S)$ of p with respect to S is the sum of (1) the distance from p to the nearest point in $S \setminus \{p\}$ and (2) the distance from p to the second nearest point in $S \setminus \{p\}$. The 2-dispersion cost $cost_2(S)$ of $S \subset P$ with $|S| \geq 3$ is $\min_{p \in S}\{cost_2(p, S)\}$.

Given P, d and an integer $k \geq 3$, the 2-dispersion problem is the problem to find the subset S of P with $|S| = k$ such that the 2-dispersion cost $cost_2(S)$ is maximized.

3 Greedy Algorithm

Now we give an approximation algorithm to solve the 2-dispersion problem. See **Algorithm 1**. The algorithm is a simple greedy algorithm.

Now we consider the approximation ratio of the solution obtained by the algorithm.

Let $S^* \subset P$ be the optimal solution for a given 2-dispersion problem, and $S \subset P$ the solution obtained by the algorithm above. We are going to show $cost_2(S) \geq cost_2(S^*)/8$, namely the approximation ratio of our algorithm is at least 1/8.

*Department of Computer Science, Gunma University, amano@cs.gunma-u.ac.jp

†Department of Computer Science, Gunma University, nakano@cs.gunma-u.ac.jp

Algorithm 1 greedy(P, d, k)

compute $S_3 \subset P$ consisting of the three points p_1, p_2, p_3 with maximum cost $cost_2(S_3)$
for $i = 4$ to k **do**
  find a point $p_i \in P \setminus S_{i-1}$ such that $cost_2(p_i, S_{i-1})$ is maximized
  $S_i = S_{i-1} \cup \{p_i\}$
end for
output S

Let D_p be the disk with center at p and the radius $r^* = cost_2(S^*)/4$. Let $D^* = \{D_p | p \in S^*\}$. We have the following three lemmas.

Lemma 1 *For any $p \in P$, D_p properly contains at most two points in S^*.*

Proof. Assume for a contradiction that D_p properly contains three points $p_1, p_2, p_3 \in S^*$. Now $d(p_1, p_2) < 2r^*$ and $d(p_1, p_3) < 2r^*$ hold, then $cost_2(p_1, S^*) < d(p_1, p_2) + d(p_1, p_3) < 4r^* = cost_2(S^*)$, a contradiction. □

Lemma 2 *For each $i = 3, 4, \ldots, k$, $cost_2(p_i, S_{i-1}) \geq r^*$ holds.*

Proof. Clearly the claim holds for $i = 3$. Assume $j - 1 < k$ and the claim holds for each $i = 3, 4, \ldots, j-1$. Now we consider for $i = j$. We have the following two cases.

Case 1: There is a point p^* in S^* such that D_{p^*} properly contains at most one point in S_{j-1}. Note that D_{p^*} is the disk with center at p^* and the radius $r^* = cost_2(S^*)/4$.

Then the distance from p^* to the 2nd nearest point in S_{j-1} is at least r^* so $cost_2(p^*, S_{j-1}) \geq r^*$. Since the algorithm choose p_j in a greedy manner, $cost_2(p_j, S_{j-1})$ is also at least r^*. Thus $cost_2(p_j, S_{j-1}) \geq r^*$ holds.

Case 2: Otherwise. (For each point p^* in S^*, D_{p^*} contains at least two points in S_{j-1}.)

We now count the number N of distinct pairs (p^*, q) with (1) $p^* \in S^*$, (2) $q \in S_{j-1}$ and (3) $d(p^*, q) < r^*$.

By Lemma 1 each D_q with $q \in S_{j-1}$ contains at most two points in S^*. Thus $N \leq 2(j-1) < 2k$. Since Case 1 does not occur, each D_{p^*} with $p^* \in S^*$ contains two or more points in S_{j-1}, so $N \geq 2k$. A contradiction.

Thus Case 2 never occurs. □

Lemma 3 *For each $i = 3, 4, \ldots, k$, $cost_2(S_i) \geq r^*/2$ holds.*

Proof. Clearly the claim holds for $i = 3$. Assume that $j-1 < k$ and the claim holds for each $i = 3, 4, \ldots, j-1$. Now we consider for $i = j$.

To prove $cost_2(S_j) \geq r^*/2$ we only need to show for any three points u, v, w in S_j, $d(u, v) + d(u, w) \geq r^*/2$. We have the following four cases.

If none of u, v, w is p_j, then $d(u, v) + d(u, w) \geq r^*/2$ is clearly held as it was held in S_{j-1}.

If u is p_j, then by Lemma 2 $d(p_j, v) + d(p_j, w) \geq cost_2(p_j, S_{j-1}) \geq r^*$. Thus $d(u, v) + d(u, w) \geq r^*/2$ holds.

If v is p_j, assume for a contradiction that $d(u, p_j) + d(u, w) < r^*/2$. Then clearly $d(u, p_j) = d(p_j, u) < r^*/2$ and by the triangle inequality $d(p_j, w) \leq d(p_j, u) + d(u, w) = d(u, p_j) + d(u, w) < r^*/2$. Then $cost_2(p_j, S_{j-1}) \leq d(p_j, u) + d(p_j, w) < r^*$, contradiction to Lemma 2. Thus if v is p_j then $d(u, p_j) + d(u, w) \geq r^*/2$ holds.

If w is p_j, then we can prove the claim in a similar manner to the case v is p_j. □

Since $S_k = S$, we have the following theorem.

Theorem 4 $cost_2(S) \geq cost_2(S^*)/8$.

Thus the approximation ratio of **Algorithm 1** is at least 1/8.

Is the approximation ratio above best possible? We now provide an example for which our algorithm computes a solution with approximation ratio asymptotically 1/4. See an example in Fig.1. $P = \{q_1, q_2, q_3, q_4, q_5, q_6, r, s\}$ and $k = 6$ for which our algorithm computes a solution $S = \{q_1, q_2, \ldots, q_6\}$, where the points are chosen in this order. The distances between points are as follows. $d(q_1, q_2) = d(q_2, q_3) = d(q_3, q_1) = 1$. q_5 is the midpoint between q_1 and q_2. q_6 is on the line segment between q_1 and q_3 and $d(q_1, q_6) = 0.75$ and $d(q_3, q_6) = 0.25$. Finally we set $d(q_1, r) = d(q_2, s) = d(q_3, q_4) = \epsilon$, where ϵ is small enough.

Note that $cost_2(S) = cost_2(q_3, S) \leq 0.25 + \epsilon$ while $cost_2(S^*) = 1$ for $S^* = \{q_1, q_2, q_3, q_4, r, s\}$. Thus the approximation ratio is 1/4.

Thus we still have a chance to improve the approximation ratio of our simple greedy algorithm, or we can find an example of P for which our algorithm generates a solution with approximation ratio smaller than 1/4.

4 Generalization

The 2-dispersion problem can be naturally generalized to the c-dispersion problem as follows.

For a point $p \in P$ and a subset $S \subset P$ with $|S| \geq c+1$, the c-dispersion cost $cost_c(p, S)$ of $p \in S$ with respect to S is the sum of the distances from p to the nearest c points in $S \setminus \{p\}$. The c-dispersion cost $cost_c(S)$ of $S \subset P$ with $|S| \geq c+1$ is $\min_{p \in S}\{cost_c(p, S)\}$. Given P, d and an integer $k \geq c+1$, the c-dispersion problem is the problem to find the subset S of P with $|S| = k$ such that the c-dispersion cost $cost_c(S)$ is maximized.

We can naturally generalize our greedy algorithm in Section 3 to the algorithm to solve the c-dispersion problem. See **Algorithm 2**.

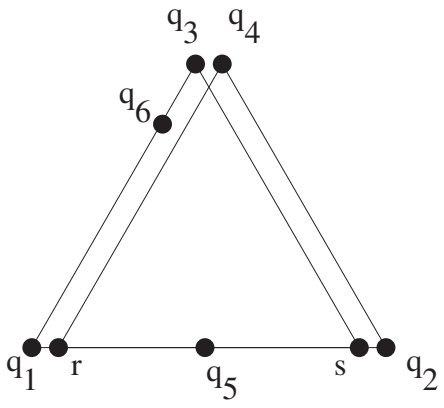


Figure 1: An example of a solution $S = \{q_1, q_2, \ldots, q_6\}$ with approximation ratio 1/4.

Algorithm 2 greedy-c(P, d, k)

compute $S_{c+1} \subset P$ consisting of the $c+1$ points $p_1, p_2, \ldots, p_{c+1}$ with maximum cost $cost_c(S_c)$
for $i = c+2$ to k **do**
  find a point $p_i \in P \setminus S_{i-1}$ such that $cost_c(p_i, S_{i-1})$ is maximized
  $S_i = S_{i-1} \cup \{p_i\}$
end for
output S

Let S^* be the optimal solution for a given c-dispersion problem, and $S \subset P$ the solution obtained by the greedy algorithm above. We now consider the approximation ratio of the solution obtained by the greedy algorithm.

Let D_p be the disk with center at p and the radius $r^{**} = cost_c(S^*)/(2c)$. Let $D^{**} = \{D_p | p \in S^*\}$. We have the following three lemmas.

Lemma 5 *For any $p \in P$, D_p properly contains at most c points in S^*.*

Proof. Assume for a contradiction that D_p properly contains $c+1$ points, say $q_1, q_2, \ldots, q_{c+1} \in S^*$. Now $d(q_{c+1}, q_t) < 2r^{**}$ holds for each $t = 1, 2, \ldots, c$. Then $cost_2(q_{c+1}, S^*) < d(q_{c+1}, q_1) + d(q_{c+1}, q_2) + \cdots + d(q_{c+1}, q_c) < 2cr^{**} = cost_c(S^*)$, a contradiction. □

Lemma 6 *For each $i = c+1, c+2, \ldots, k$, $cost_c(p_i, S_{i-1}) \geq r^{**}$ holds.*

Proof. Clearly the claim holds for $i = c+1$. Assume $j - 1 < k$ and the claim holds for each $i = c+1, c+2, \ldots, j-1$. Now we consider for $i = j$. We have the following two cases.
Case 1: There is a point p^* in S^* such that D_{p^*} properly contains at most $c-1$ point in S_{j-1}.

Then the distance from p^* to the c-th nearest point in S_{j-1} is at least r^{**} so $cost_c(p^*, S_{j-1}) \geq r^{**}$. Since the algorithm choose p_j in a greedy manner, $cost_c(p_j, S_{j-1})$ is also at least r^{**}. Thus $cost_c(p_j, S_{j-1}) \geq r^{**}$ holds.
Case 2: Otherwise.

We now count the number N of distinct pairs (p^*, q) with (1) $p^* \in S^*$, (2) $q \in S_{j-1}$ and (3) $d(p^*, q) < r^{**}$.

By Lemma 5 each D_q with $q \in S_{j-1}$ contains at most c points in S^*. Thus $N \leq c(j-1) < ck$. Since Case 1 does not occur, each D_{p^*} with $p^* \in S^*$ contains c or more points in S_{j-1}, so $N \geq ck$. A contradiction.

Thus Case 2 never occurs. □

Lemma 7 *For each $i = c+1, c+2, \ldots, k$, $cost_c(S_i) \geq r^{**}/c$ holds.*

Proof. Clearly the claim holds for $i = c+1$. Assume that $j-1 < k$ and the claim holds for each $i = c+1, c+2, \ldots, j-1$. Now we consider for $i = j$.

For any point u in S_j we show $cost_c(u, S_j) \geq r^{**}/c$ holds. We have three cases. Let $S(u)$ be the set of point in $S_j \setminus \{u\}$ consisting of the nearest c points to u.

If $p_j \notin \{u\} \cup S(u)$, then clearly $cost_c(u, S_j) \geq r^{**}/c$ holds, since $cost_c(u, S_{j-1}) \geq r^{**}/c$ holds.

If $p_j = u$, then by Lemma 6 $cost_c(u, S_j) \geq r^{**}$ holds, so $cost_c(u, S_j) \geq r^{**}/c$ holds.

If $p_j \in S(u)$, then assume for a contradiction that $cost_c(u, S_j) < r^{**}/c$. Let $S(u) = \{q_1, q_2, \ldots, q_c\}$ and $q_x = p_j$. Then clearly $d(u, p_j) < cost_c(u, S_j) < r^{**}/c$ and by the triangle inequality for each $t \neq x$ $d(p_j, q_t) \leq d(p_j, u) + d(u, q_t) = cost_c(u, S_j) < r^{**}/c$. Then $cost_c(p_j, S_j) \leq d(p_j, q_1) + d(p_j, q_2) + \cdots + d(p_j, q_c) < r^{**}$, contradiction to Lemma 6. □

Since $S_k = S$, we have the following theorem.

Theorem 8 $cost_c(S) \geq cost_c(S^*)/(2c^2)$.

5 Conclusion

In this paper we have presented a simple 1/8−approximation algorithm to solve the 2-dispersion problem. The running time of the algorithm is $O(n^3)$. Similarly we have presented a simple $1/(2c^2)$−approximation algorithm to solve the c-dispersion problem. The running time of the algorithm is $O(n^{c+1})$.

References

[1] C. Baur and S.P. Fekete, Approximation of Geometric Dispersion Problems, Pro. of APPROX '98, Pages 63-75 (1998).

[2] A. Cevallos, F. Eisenbrand and R. Zenklusen, Local search for max-sum diversification, Proc. of SODA '17, pp.130-142 (2017).

[3] B. Chandra and M. M. Halldorsson, Approximation Algorithms for Dispersion Problems, J. of Algorithms, 38, pp.438-465 (2001).

[4] Z. Drezner, Facility Location: A Survey of Applications and Methods, Springer (1995).

[5] Z. Drezner and H.W. Hamacher, Facility Location: Applications and Theory, Springer (2004).

[6] R. Hassin, S. Rubinstein and A. Tamir, Approximation Algorithms for Maximum Dispersion, Operation Research Letters, 21, pp.133-137 (1997).

[7] T. L. Lei, R. L. Church, A unified model for dispersing facilities, Geographical Analysis, 45, pp.401-418 (2013).

[8] T. L. Lei, R. L. Church, On the unified dispersion problem: Efficient formulations and exact algorithms, European Journal of Operational Research, 241, pp.622-630 (2015).

[9] S. S. Ravi, D. J. Rosenkrantz and G. K. Tayi, Heuristic and Special Case Algorithms for Dispersion Problems, Operations Research, 42, pp.299-310 (1994).

[10] M. Sydow, Approximation Guarantees for Max Sum and Max Min Facility Dispersion with Parameterised Triangle Inequality and Applications in Result Diversification, Mathematica Applicanda, 42, pp.241-257 (2014).

[11] D. W. Wang and Yue-Sun Kuo, A study on Two Geometric Location Problems, Information Processing Letters, 28, pp.281-286 (1988).

An efficient approximation for point-set diameter in higher dimensions

Mahdi Imanparast* Seyed Naser Hashemi* Ali Mohades*†

Abstract

In this paper, we study the problem of computing the diameter of a set of n points in d-dimensional Euclidean space for a fixed dimension d, and propose a new $(1+\varepsilon)$-approximation algorithm with $O(n+1/\varepsilon^{d-2})$ time and $O(n)$ space, where $0 < \varepsilon \leqslant 1$. We also show that the proposed algorithm can be modified to a $(1+O(\varepsilon))$-approximation algorithm with $O(n+1/\varepsilon^{\frac{2d}{3}-\frac{1}{2}})$ running time. These results provide some improvements in comparison with existing algorithms in terms of simplicity, and data structure.

1 Introduction

Given a finite set $\mathcal{S}$ of n points, the diameter of $\mathcal{S}$, denoted by $D(\mathcal{S})$ is the maximum distance between two points of $\mathcal{S}$. Namely, we want to find a diametrical pair p and q such that $D(\mathcal{S}) = \max_{p,q\in\mathcal{S}}(||p-q||)$. Computing the diameter of a set of points has a large history, and it may be required in various fields such as database, data mining, and vision. A trivial brute-force algorithm for this problem takes $O(dn^2)$ time, but this is too slow for large-scale data sets that occur in the fields. Hence, we need a faster algorithm which may be exact or is an approximation.

By reducing from the set disjointness problem, it can be shown that computing the diameter of n points in $\mathbb{R}^d$ requires $\Omega(n\log n)$ operations in the algebraic computation-tree model [1]. It is shown by Yao that it is possible to compute the diameter in sub-quadratic time in each dimension [2]. There are well-known solutions in two and three dimensions. In the plane, this problem can be computed in optimal time $O(n\log n)$, but in three dimensions, it is more difficult. Clarkson and Shor [3] present an $O(n\log n)$-time randomized algorithm. Their algorithm needs to compute the intersection of n balls (with the same radius) in $\mathbb{R}^3$. It may be slower than the brute-force algorithm for the most practical data sets, and it is not an efficient method for higher dimensions because the intersection of n balls with the same radius has a large size. Some deterministic algorithms with running time $O(n\log^3 n)$ and $O(n\log^2 n)$ are found for this problem in three dimensions. Finally, Ramos [4] introduced an optimal deterministic $O(n\log n)$-time algorithm in $\mathbb{R}^3$.

In the absence of fast algorithms, many attempts have been made to approximate the diameter in low and high dimensions. A 2-approximation algorithm in $O(dn)$ time can be found easily by selecting a point of $\mathcal{S}$ and then finding the farthest point of it by brute-force manner for the dimension d. The first non-trivial approximation algorithm for the diameter is presented by Egecioglu and Kalantari [5] that approximates the diameter with factor $\sqrt{3}$ and operations cost $O(dn)$. They also present an iterative algorithm with $t \leq n$ iterations and the cost $O(dn)$ for each iteration that has approximate factor $\sqrt{5-2\sqrt{3}}$. Agarwal et al. [6] present a $(1+\varepsilon)$-approximation algorithm in $\mathbb{R}^d$ with $O(n/\varepsilon^{(d-1)/2})$ running time by projection to directions. Barequet and Har Peled [7] present a $\sqrt{d}$-approximation diameter method with $O(dn)$ time. They also describe a $(1+\varepsilon)$-approximation approach with $O(n+1/\varepsilon^{2d})$ time. They show that the running time can be improved to $O(n+1/\varepsilon^{2(d-1)})$. Similarly, Har Peled [8] presents an approach which for the most inputs is able to compute very fast the exact diameter, or an approximation with $O((n+1/\varepsilon^{2d})\log 1/\varepsilon)$ running time. Although, in the worst case, the algorithm running time is still quadratic, and it is sensitive to the hardness of the input. Chan [9] observes that a combination of two approaches in [6] and [7] yields a $(1+\varepsilon)$-approximation with $O(n+1/\varepsilon^{3(d-1)/2})$ time and a $(1+O(\varepsilon))$-approximation with $O(n+1/\varepsilon^{d-\frac{1}{2}})$ time. He also introduces a core-set theorem, and shows that using this theorem, a $(1+O(\varepsilon))$-approximation in $O(n+1/\varepsilon^{d-\frac{3}{2}})$ time can be found [10]. Recently, Chan [11] has proposed an approximation algorithm with $O((n/\sqrt{\varepsilon}+1/\varepsilon^{\frac{d}{2}+1})(\log\frac{1}{\varepsilon})^{O(1)})$ time by applying the Chebyshev polynomials in low constant dimensions, and Arya et al. [12] show that by applying an efficient decomposition of a convex body using a hierarchy of Macbeath regions, it is possible to compute an approximation in $O(n\log\frac{1}{\varepsilon}+1/\varepsilon^{\frac{(d-1)}{2}+\alpha})$ time, where α is an arbitrarily small positive constant.

1.1 Our results

In this paper, we propose a new $(1+\varepsilon)$-approximation algorithm for computing the diameter of a set $\mathcal{S}$ of

*Department of Mathematics and Computer Science, Amirkabir University of Technology, Tehran, Iran, m.imanparast@aut.ac.ir, nhashemi@aut.ac.ir

†Laboratory of Algorithms and Computational Geometry, Amirkabir University of Technology, Tehran, Iran, mohades@aut.ac.ir

Table 1: A summary on the complexity of some non-constant approximation algorithm for the diameter of a point set. Our results are denoted by +.

Ref.	Approx. Factor	Running Time
[6]	$1+\varepsilon$	$O(\frac{n}{\varepsilon^{(d-1)/2}})$
[7]	$1+\varepsilon$	$O(n+1/\varepsilon^{2(d-1)})$
[8]	$1+\varepsilon$	$O((n+1/\varepsilon^{2d})\log\frac{1}{\varepsilon})$
[9]	$1+\varepsilon$	$O(n+1/\varepsilon^{\frac{3(d-1)}{2}})$
+	$1+\varepsilon$	$O(n+1/\varepsilon^{d-2})$
[9]	$1+O(\varepsilon)$	$O(n+1/\varepsilon^{d-\frac{1}{2}})$
[10]	$1+O(\varepsilon)$	$O(n+1/\varepsilon^{d-\frac{3}{2}})$
[11]	$1+O(\varepsilon)$	$O((\frac{n}{\sqrt{\varepsilon}}+1/\varepsilon^{\frac{d}{2}+1})(\log\frac{1}{\varepsilon})^{O(1)})$
[12]	$1+O(\varepsilon)$	$O(n\log\frac{1}{\varepsilon}+1/\varepsilon^{\frac{(d-1)}{2}+\alpha})$
+	$1+O(\varepsilon)$	$O(n+1/\varepsilon^{\frac{2d}{3}-\frac{1}{2}})$

n points in $\mathbb{R}^d$ with $O(n+1/\varepsilon^{d-2})$ time and $O(n)$ space, where $0<\varepsilon\leqslant 1$. Moreover, we show that the proposed algorithm can be modified to a $(1+O(\varepsilon))$-approximation algorithm with $O(n+1/\varepsilon^{\frac{2d}{3}-\frac{1}{2}})$ time and $O(n)$ space. As stated above, two new results have been recently presented for this problem in [11] and [12]. It should be noted that our algorithms are completely different in terms of computational technique. The polynomial technique provided by Chan [11] is based on using Chebyshev polynomials and discrete upper envelope subroutine [10], and the method presented by Arya et al. [12] requires the use of complex data structures to approximately answer queries for polytope membership, directional width, and nearest-neighbor. While our algorithms in comparison with these algorithms are simpler in terms of understanding and data structure. We have provided a summary on the non-constant approximation algorithms for the diameter in Table 1.

2 The proposed algorithm

In this section, we describe our new approximation algorithm to compute the diameter of a point set. In our algorithm, we first find the extreme points in each coordinate and compute the axis-parallel bounding box of $\mathcal{S}$, which is denoted by $B(\mathcal{S})$. We use the largest length side ℓ of $B(\mathcal{S})$ to impose grids on the point set. In fact, we first decompose $B(\mathcal{S})$ to a grid of regular hypercubes with side length ξ, where $\xi=\varepsilon\ell/2\sqrt{d}$. We call each hypercube a cell. Then, each point of $\mathcal{S}$ is rounded to its corresponding central cell-point. See Figure 1. In the following, we impose again an ξ_1-grid to $B(\mathcal{S})$ for $\xi_1=\sqrt{\varepsilon}\ell/2\sqrt{d}$. We round each point of the rounded point set $\hat{\mathcal{S}}$ to its nearest grid-point in this new grid that

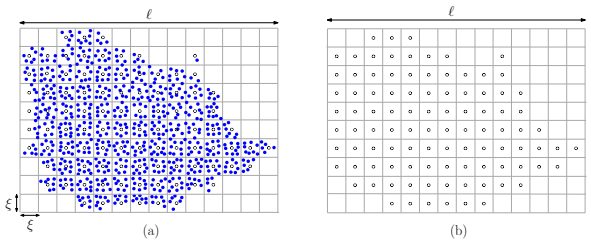


Figure 1: (a) A set of points in $\mathbb{R}^2$ and an ξ-grid. Initial points are shown by blue points and their corresponding central cell-points are shown by circle points. (b) Rounded point set $\hat{\mathcal{S}}$.

results in a point set $\hat{\mathcal{S}}_1$. Let, $\mathcal{B}_\delta(p)$ be a hypercube with side length δ and central-point p. We restrict our search domain for finding diametrical pairs of the first rounded point set $\hat{\mathcal{S}}$ into two hypercubes $\mathcal{B}_{2\xi_1}(\hat{p}_1)$ and $\mathcal{B}_{2\xi_1}(\hat{q}_1)$ corresponding to two diametrical pair points $\hat{p}_1$ and $\hat{q}_1$ in the point set $\hat{\mathcal{S}}_1$. Let us use two point sets $\mathcal{B}_1$ and $\mathcal{B}_2$ for maintaining points of the rounded point set $\hat{\mathcal{S}}$, which are inside two hypercubes $\mathcal{B}_{2\xi_1}(\hat{p}_1)$ and $\mathcal{B}_{2\xi_1}(\hat{q}_1)$, respectively (see Figure 2). Then, it is sufficient to find a diameter between points of $\hat{\mathcal{S}}$, which are inside two point sets $\mathcal{B}_1$ and $\mathcal{B}_2$. We use notation $Diam(\mathcal{B}_1,\mathcal{B}_2)$ for the process of computing the diameter of the point set $\mathcal{B}_1\cup\mathcal{B}_2$. Altogether, we can present the following algorithm.

Algorithm 1: APPROXIMATE DIAMETER $(\mathcal{S},\varepsilon)$

Input: a set $\mathcal{S}$ of n points in $\mathbb{R}^d$ and an error parameter ε.

Output: Approximate diameter $\tilde{D}$.

1: Compute the axis-parallel bounding box $B(\mathcal{S})$ for the point set $\mathcal{S}$.
2: $\ell\leftarrow$ Find the length of the largest side in $B(\mathcal{S})$.
3: Set $\xi\leftarrow\varepsilon\ell/2\sqrt{d}$ and $\xi_1\leftarrow\sqrt{\varepsilon}\ell/2\sqrt{d}$.
4: $\hat{\mathcal{S}}\leftarrow$ Round each point of $\mathcal{S}$ to its central-cell point in a ξ-grid.
5: $\hat{\mathcal{S}}_1\leftarrow$ Round each point of $\hat{\mathcal{S}}$ to its nearest grid-point in a ξ_1-grid.
6: $\hat{D}_1\leftarrow$ Compute the diameter of the point set $\hat{\mathcal{S}}_1$ by brute-force manner, and simultaneously, a list of the diametrical pairs $(\hat{p}_1,\hat{q}_1)$, such that $\hat{D}_1=||\hat{p}_1-\hat{q}_1||$.
7: Find points of $\hat{\mathcal{S}}$ which are in two hypercubes $\mathcal{B}_1=\mathcal{B}_{2\xi_1}(\hat{p}_1)$ and $\mathcal{B}_2=\mathcal{B}_{2\xi_1}(\hat{q}_1)$, for each diametrical pair $(\hat{p}_1,\hat{q}_1)$.
8: $\hat{D}\leftarrow$ Compute $Diam(\mathcal{B}_1,\mathcal{B}_2)$, corresponding to each diametrical pair $(\hat{p}_1,\hat{q}_1)$ by brute-force manner and return the maximum value between them.
9: $\tilde{D}\leftarrow\hat{D}+\varepsilon\ell/2$.
10: Output $\tilde{D}$.

2.1 Analysis

In this subsection, we analyze the proposed algorithm.

Theorem 1 *Algorithm 1 computes an approximate diameter for a set $\mathcal{S}$ of n points in $\mathbb{R}^d$ in $O(n+1/\varepsilon^{d-2})$ time and $O(n)$ space, where $0<\varepsilon\leqslant 1$.*

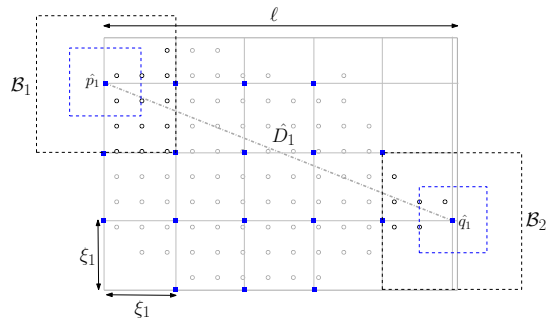


Figure 2: Points of the set $\hat{\mathcal{S}}$ are shown by circle points and their corresponding nearest grid-points in set $\hat{\mathcal{S}}_1$ are shown by blue square points.

Proof. Finding the extreme points in all coordinates and finding the largest side of $B(\mathcal{S})$ can be done in $O(dn)$ time. The rounding step takes $O(d)$ time for each point, and for all of them takes $O(dn)$ time. But for computing the diameter over the rounded point set $\hat{\mathcal{S}}_1$ we need to know the number of points in the set $\hat{\mathcal{S}}_1$. We know that the largest side of the bounding box $B(\mathcal{S})$ has length ℓ and the side length of each cell in ξ_1-grid is $\xi_1 = \sqrt{\varepsilon}\ell/2\sqrt{d}$. On the other hand, the volume of a hypercube of side length L in d-dimensional space is L^d. Since, corresponding to each point in the point set $\hat{\mathcal{S}}_1$, we can take a hypercube of side length ξ_1. Therefore, in order to count the maximum number of points inside the set $\hat{\mathcal{S}}_1$, it is sufficient to calculate the number of hypercubes of length ξ_1 in a hypercube (bounding box) with length $\ell + \xi_1$. See Figure 2. This means that the number of grid-points in an imposed ξ_1-grid to the bounding box $B(\mathcal{S})$ is at most

$$\frac{(\ell+\xi_1)^d}{(\xi_1)^d} = \left(\frac{2\sqrt{d}}{\sqrt{\varepsilon}} + 1\right)^d = O\left(\frac{(2\sqrt{d})^d}{\varepsilon^{\frac{d}{2}}}\right). \quad (1)$$

So, the number of points in $\hat{\mathcal{S}}_1$ is at most $O((2\sqrt{d})^d/\varepsilon^{\frac{d}{2}})$. Hence, by the brute-force quadratic algorithm, we need $O((2\sqrt{d})^d/\varepsilon^{\frac{d}{2}})^2) = O((2\sqrt{d})^{2d}/\varepsilon^d)$ time for computing all distances between grid-points of the set $\hat{\mathcal{S}}_1$, and its diametrical pair list. Then, for a diametrical pair $(\hat{p}_1, \hat{q}_1)$ in the point set $\hat{\mathcal{S}}_1$, we compute two sets $\mathcal{B}_1$ and $\mathcal{B}_2$. This work takes $O(dn)$ time. In addition, for computing the diameter of point set $\mathcal{B}_1 \cup \mathcal{B}_2$, we need to know the number of points in each of them. On the other hand, the number of points in two sets $\mathcal{B}_1$ or $\mathcal{B}_2$ is at most

$$\frac{Vol(\mathcal{B}_{2\xi_1})}{Vol(\mathcal{B}_\xi)} = \frac{(2\sqrt{\varepsilon}\ell/2\sqrt{d})^d}{(\varepsilon\ell/2\sqrt{d})^d} = \frac{(2\sqrt{\varepsilon})^d}{\varepsilon^d} = \frac{(2)^d}{\varepsilon^{\frac{d}{2}}}. \quad (2)$$

Hence, for computing $Diam(\mathcal{B}_1, \mathcal{B}_2)$, we need $O(((2)^d/\varepsilon^{\frac{d}{2}})^2) = O((2)^{2d}/\varepsilon^d)$ time by brute-force manner, but we might have more than one diametrical pair $(\mathcal{B}_1, \mathcal{B}_2)$. Since the point set $\hat{\mathcal{S}}_1$ is a set of grid-points, so we could have in the worst-case $O(2^d)$ different diametrical pairs $(\mathcal{B}_1, \mathcal{B}_2)$ in the point set $\hat{\mathcal{S}}_1$. This means that this step takes at most $O(2^d \cdot (2)^{2d}/\varepsilon^d) = O((2\sqrt{2})^{2d}/\varepsilon^d)$ time. Now, we can present the complexity of our algorithm as follows:

$$T_d(n) = O(dn) + O\left(\frac{(2\sqrt{d})^{2d}}{\varepsilon^d}\right) + O(2^d dn) + O\left(\frac{(2\sqrt{2})^{2d}}{\varepsilon^d}\right),$$

$$\leqslant O\left(2^d dn + \frac{(2\sqrt{d})^{2d}}{\varepsilon^d}\right). \quad (3)$$

Since d is fixed, we have: $T_d(n) = O(n + \frac{1}{\varepsilon^d})$.

We can also reduce the running time of the Algorithm 1 by discarding some internal points which do not have any potential to be the diametrical pairs in rounded point set $\hat{\mathcal{S}}_1$, and similarly, in two point sets $\mathcal{B}_1$ and $\mathcal{B}_2$. By considering all the points which are same in their $(d-1)$ coordinates and keep only highest and lowest [7]. Then, the number of points in $\hat{\mathcal{S}}_1$, and two point sets $\mathcal{B}_1$ and $\mathcal{B}_2$ can be reduced to $O(1/\varepsilon^{\frac{d}{2}-1})$. So, using the brute-force quadratic algorithm, we need $O((1/\varepsilon^{\frac{d}{2}-1})^2)$ time to find the diametrical pairs. Hence, this gives us the total running time $O(n + 1/\varepsilon^{d-2})$. About the required space, we only need $O(n)$ space for storing required point sets. So, this completes the proof. □

Now, we explain the details of the approximation factor.

Theorem 2 *Algorithm 1 computes an approximate diameter $\tilde{D}$ such that: $D \leqslant \tilde{D} \leqslant (1+\varepsilon)D$, where $0 < \varepsilon \leqslant 1$.*

Proof. In line 7 of the Algorithm 1, we compute two point sets $\mathcal{B}_1$ and $\mathcal{B}_2$, for each diametrical pair $(\hat{p}_1, \hat{q}_1)$ in the point set $\hat{\mathcal{S}}_1$. We know that a grid-point $\hat{p}_1$ in point set $\hat{\mathcal{S}}_1$ is formed from points of the set $\hat{\mathcal{S}}$ which are inside hypercube $B_{\xi_1}(\hat{p}_1)$. We use a hypercube $\mathcal{B}_1$ of side length $2\xi_1$ to make sure that we do not lose any candidate diametrical pair of the first rounded point set $\hat{\mathcal{S}}$ around a diametrical point $\hat{p}_1$ (see Figure 2). In the next step, we should find the diametrical pair $(\hat{p}, \hat{q}) \in \hat{\mathcal{S}}$ for points which are inside two point sets $\mathcal{B}_1$ and $\mathcal{B}_2$. Hence, it is remained to show that the diameter, which is computed by two points $\hat{p}$ and $\hat{q}$, is a $(1+\varepsilon)$-approximation of the true diameter. Let $\hat{p}$ and $\hat{q}$ are two central-cell points of the first rounded point set $\hat{\mathcal{S}}$ which are used in line 8 of the Algorithm 1 for computing the approximate diameter $\hat{D}$. Then, we have two cases, either two true points p and q are in far distance from each other in their corresponding cells (Figure 3 (a)), or they are in near distance from each other (Figure 3 (b)). It is obvious that the other cases are between these two cases.

For first case (Figure 3 (a)), let for two projected points $\hat{p}'$ and $\hat{q}'$, we set $d_1 = ||p - \hat{p}'||$ and $d_2 = ||q - \hat{q}'||$.

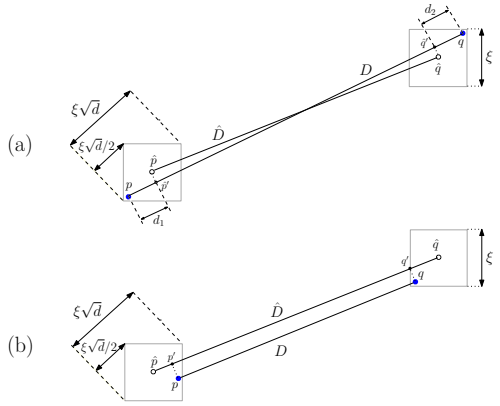


Figure 3: Two cases in proof of the Theorem 2.

We know that the side length of each cell in a grid which is used for $\hat{\mathcal{S}}$ is ξ. So, the hypercube (cell) diagonal is $\xi\sqrt{d}$. From Figure 3 (a) it can be found that $d_1 < \xi\sqrt{d}/2$ and $d_2 < \xi\sqrt{d}/2$. Therefore, we have

$$D = \hat{D} + d_1 + d_2,$$
$$D \leqslant \hat{D} + \xi\sqrt{d},$$
$$D - \xi\sqrt{d} \leqslant \hat{D}. \tag{4}$$

Similarly, for the second case (Figure 3 (b)), we know that $c_1 = ||\hat{p} - p'|| < \xi\sqrt{d}/2$ and $c_2 = ||\hat{q} - q'|| < \xi\sqrt{d}/2$. So,

$$\hat{D} = D + c_1 + c_2,$$
$$\hat{D} \leqslant D + \xi\sqrt{d}. \tag{5}$$

Then, from (4) and (5) we can result:

$$D - \xi\sqrt{d} \leqslant \hat{D} \leqslant D + \xi\sqrt{d}. \tag{6}$$

Since we know that $\xi = \varepsilon\ell/2\sqrt{d}$, we have:

$$D - \varepsilon\ell/2 \leqslant \hat{D} \leqslant D + \varepsilon\ell/2,$$
$$D \leqslant \hat{D} + \varepsilon\ell/2 \leqslant D + \varepsilon\ell. \tag{7}$$

We know that $\ell \leqslant D$. For this reason we can result:

$$D \leqslant \hat{D} + \varepsilon\ell/2 \leqslant (1+\varepsilon)D. \tag{8}$$

Finally, if we assume that $\tilde{D} = \hat{D} + \varepsilon\ell/2$, we have:

$$D \leqslant \tilde{D} \leqslant (1+\varepsilon)D. \tag{9}$$

Therefore, the theorem is proven. □

2.2 The modified algorithm

In this subsection, we present a modified version of our proposed algorithm by combining it with a recursive approach due to Chan [9]. Hence, we first explain Chan's recursive approach. As mentioned before, Agarwal et al. [6] proposed a $(1+\varepsilon)$-approximation algorithm for computing the diameter of a set of points in $\mathbb{R}^d$. Their result is based on the following simple fact that we can find $O(1/\varepsilon^{(d-1)/2})$ numbers of directions in $\mathbb{R}^d$, for example by constructing a uniform grid on a unit sphere, such that for each vector $x \in \mathbb{R}^d$, there is a direction that the angle between this direction and x be at most $\sqrt{\varepsilon}$. In fact, they found a small set of directions which can approximate well all directions. This can be done by forming unit vectors which start from origin to grid-points of a uniform grid on a unit sphere [6], or to grid-points on the boundary of a box [10]. These sets of directions have cardinality $O(1/\varepsilon^{(d-1)/2})$. The following observation explains how we can find these directions on the boundary of a box.

Observation 1 *([10]) Consider a box B which includes origin o such that the boundary of this box (∂B) be in the distance at least 1 from the origin. For a $\sqrt{\varepsilon/2}$-grid on ∂B and for each vector $\vec{x}$, there is a grid point $\boldsymbol{a}$ on ∂B such that the angle between two vectors $\vec{a}$ and $\vec{x}$ is at most $arccos(1-\varepsilon/8) \leqslant \sqrt{\varepsilon}$.*

This observation explains that grid-points on the boundary of a box (∂B) form a set V_d of $O(1/\varepsilon^{(d-1)/2})$ numbers of unit vectors in $\mathbb{R}^d$ such that for each $x \in \mathbb{R}^d$, there is a vector $a \in V_d$ from the origin o to a grid-point a on ∂B, where the angle between two vectors x and a is at most $\sqrt{\varepsilon}$. On the other hand, according to observation 1, there is a vector $a \in V_d$ such that if α be the angle between two vectors x and a, then, $\alpha \leqslant arccos(1-\varepsilon/8)$, and so $cos\alpha \geqslant (1-\varepsilon/8)$. If x' is the projection of the vector x on the vector a, then:

$$||x|| = \frac{||x'||}{cos\alpha} \leqslant ||x'|| \frac{1}{(1-\frac{\varepsilon}{8})}$$
$$\leqslant ||x'||(1 + \frac{\varepsilon}{8} + \frac{\varepsilon^2}{8^2} + \frac{\varepsilon^3}{8^3} + \cdots)$$
$$\leqslant ||x'||(1+\varepsilon). \tag{10}$$

So, we have:

$$||x'|| \leqslant ||x|| \leqslant (1+\varepsilon)||x'||. \tag{11}$$

This means that if pair (p,q) be the diametrical pair of a point set, then there is a vector $a \in V_d$ such that the angle between two vectors pq and a is at most $\sqrt{\varepsilon}$. See Figure 4. Then, pair (p',q') which is the projection of the pair (p,q) on the vector a, is a $(1+\varepsilon)$-approximation of the true diametrical pair (p,q), and we have:

$$||p'-q'|| \leqslant ||p-q|| \leqslant (1+\varepsilon)||p'-q'||. \tag{12}$$

In other words, we can project point set $\mathcal{S}$ on $O(1/\varepsilon^{(d-1)/2})$ directions for all $a \in V_d$, and compute a $(1+\varepsilon)$-approximation of the diameter by finding maximum diameter between all directions. We project n

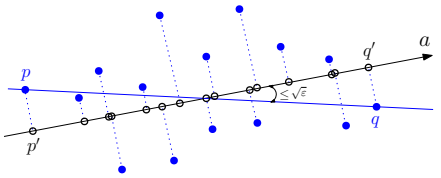


Figure 4: Projecting a point set on a direction a.

points on $|V_d| = O(1/\varepsilon^{(d-1)/2})$ directions. Since, computing the extreme points on each direction $a \in V_d$ takes $O(n)$ time. Consequently, Agarwal et al. [6] algorithm computes a $(1+\varepsilon)$-approximation of the diameter in $O(n/\varepsilon^{(d-1)/2})$ time. Chan [9] proposes that if we reduce the number of points from n to $O(1/\varepsilon^{d-1})$ by rounding to a grid and then apply Agarwal et al. [6] method on this rounded point set, we need $O((1/\varepsilon^{d-1})/\varepsilon^{(d-1)/2}) = O(1/\varepsilon^{3(d-1)/2})$ time to compute the maximum diameter over all $O(1/\varepsilon^{(d-1)/2})$ directions. Taking into account $O(n)$ time for rounding to a grid, this new approach takes $O(n + 1/\varepsilon^{3(d-1)/2})$ time. Moreover, Chan [9] observed that the bottleneck of this approach is the large number of projection operations. Hence, he proposes that we can project points on a set of $O(1/\sqrt{\varepsilon})$ 2-dimensional unit vectors instead of $O(1/\varepsilon^{(d-1)/2})$ d-dimensional unit vectors to reduce the problem to $O(1/\sqrt{\varepsilon})$ numbers of $(d-1)$-dimensional subproblems which can be solved recursively. In fact, according to the relation (11), for a vector $x \in \mathbb{R}^2$, there is a vector a such that:

$$||x'|| \leqslant ||x|| \leqslant (1+\varepsilon)||x'||, \quad x \in \mathbb{R}^2. \tag{13}$$

where x' is the projection of the vector x on vector a. Since a is a unit vector ($||a|| = 1$), therefore, $||x'|| = (a \cdot x)/||a|| = a \cdot x$. Hence, we can rewrite the previous relation as follows:

$$(a \cdot x)^2 \leqslant ||x||^2 \leqslant (1+\varepsilon)^2 (a \cdot x)^2, \; x \in \mathbb{R}^2, a \in V_2, \tag{14}$$

or

$$(a_1x_1+a_2x_2)^2 \leqslant (x_1^2+x_2^2) \leqslant (1+\varepsilon)^2(a_1x_1+a_2x_2)^2, \; a \in V_2. \tag{15}$$

where x_i be the ith coordinate for a point $x \in \mathbb{R}^d$. We can expand (15) to:

$$(a_1x_1 + a_2x_2)^2 + \cdots + x_d^2 \leqslant (x_1^2 + x_2^2 + \cdots + x_d^2) \leqslant$$
$$(1+\varepsilon)^2((a_1x_1 + a_2x_2)^2 + \cdots + x_d^2). \tag{16}$$

Now, define the projection $\pi_a : \mathbb{R}^d \to \mathbb{R}^{d-1} : \pi_a(x) = (a_1x_1 + a_2x_2, x_3, \cdots, x_d) \in \mathbb{R}^{d-1}$. Then, we can rewrite relation (16) for each vector $x \in \mathbb{R}^d$ as follows:

$$||\pi_a(x)||^2 \leqslant ||x||^2 \leqslant (1+\varepsilon)^2||\pi_a(x)||^2, \quad a \in V_2. \tag{17}$$

So, since $||\pi_a(p-q)|| = ||\pi_a(p)|| - ||\pi_a(q)||$ we have for diametrical pair (p, q):

$$||\pi_a(p-q)|| \leqslant ||p-q|| \leqslant (1+\varepsilon)||\pi_a(p-q)||, \quad a \in V_2. \tag{18}$$

Therefore, for finding a $(1+O(\varepsilon))$-approximation for the diameter of point set $P \subseteq \mathbb{R}^d$, it is sufficient that we approximate recursively the diameter of a projected point set $\pi_a(P) \subset \mathbb{R}^{d-1}$ over each of the vectors $a \in V_2$. Then, the maximum diametrical pair computed in the recursive calls is a $(1+O(\varepsilon))$-approximation to the diametrical pair. Now, let us reduce the number of points from n to $m = O(1/\varepsilon^{d-1})$ by rounding to a grid, and we denote the required time for computing the diameter of m points in d-dimensional space with $t_d(m)$. Then, for $m = O(1/\varepsilon^{d-1})$ grid points, this approach breaks the problem into $O(1/\sqrt{\varepsilon})$ subproblems in a $(d-1)$ dimension. Hence, we have a recurrence $t_d(m) = O(m + 1/\sqrt{\varepsilon}t_{d-1}(O(1/\varepsilon^{d-1})))$. By assuming $E = 1/\varepsilon$, we can rewrite the recurrence as:

$$t_d(m) = O(m + E^{\frac{1}{2}}t_{d-1}(O(E^{d-1}))). \tag{19}$$

This can be solved to: $t_d(m) = O(m + E^{d-\frac{1}{2}})$. In this case, $m = O(1/\varepsilon^{d-1})$, so, this recursive takes $O(1/\varepsilon^{d-\frac{1}{2}})$ time. Taking into account $O(n)$ time, we spent for rounding to a grid at the first, Chan's recursive approach computes a $(1+O(\varepsilon))$-approximation for the diameter of a set of n points in $O(n+1/\varepsilon^{d-\frac{1}{2}})$ time [9].

In the following, we use Chan's recursive approach in a phase of our proposed algorithm.

Algorithm 2: APPROXIMATE DIAMETER 2 $(\mathcal{S}, \varepsilon)$

Input: a set $\mathcal{S}$ of n points in $\mathbb{R}^d$ and an error parameter ε.
Output: Approximate diameter $\tilde{D}$.

1: Compute the axis-parallel bounding box $B(\mathcal{S})$ for the point set $\mathcal{S}$.
2: $\ell \leftarrow$ Find the length of the largest side in $B(\mathcal{S})$.
3: Set $\xi \leftarrow \varepsilon\ell/2\sqrt{d}$ and $\xi_2 \leftarrow \varepsilon^{\frac{1}{3}}\ell/2\sqrt{d}$.
4: $\hat{\mathcal{S}} \leftarrow$ Round each point of $\mathcal{S}$ to its central-cell point in a ξ-grid.
5: $\hat{\mathcal{S}}_1 \leftarrow$ Round each point of $\hat{\mathcal{S}}$ to its nearest grid-point in a ξ_2-grid.
6: $\hat{D}_1 \leftarrow$ Compute the diameter of the point set $\hat{\mathcal{S}}_1$ by brute-force, and simultaneously, a list of the diametrical pairs $(\hat{p}_1, \hat{q}_1)$, such that $\hat{D}_1 = ||\hat{p}_1 - \hat{q}_1||$.
7: Find points of $\hat{\mathcal{S}}$ which are in two hypercubes $\mathcal{B}_1 = \mathcal{B}_{2\xi_2}(\hat{p}_1)$ and $\mathcal{B}_2 = \mathcal{B}_{2\xi_2}(\hat{q}_1)$ for each diametrical pair $(\hat{p}_1, \hat{q}_1)$.
8: $\tilde{D} \leftarrow$ Compute $Diam(\mathcal{B}_1, \mathcal{B}_2)$, corresponding to each diametrical pair $(\hat{p}_1, \hat{q}_1)$ using Chan's [9] recursive approach and return the maximum value $||p'-q'||$ over all of them.
9: Output $\tilde{D}$.

Now, we will analyze the Algorithm 2.

Theorem 3 *A $(1+O(\varepsilon))$-approximation for the diameter of a set of n points in d-dimensional Euclidean space can be computed in $O(n+1/\varepsilon^{\frac{2d}{3}-\frac{1}{2}})$ time and $O(n)$ space, where $0 < \varepsilon \leqslant 1$.*

Proof. As it can be seen, lines 1 to 6 of the Algorithm 2 are the same as the Algorithm 1. In this case, the number of points in rounded points set $\hat{\mathcal{S}}_1$ is at most:

$$\frac{(\ell+\xi_2)^d}{(\xi_2)^d} = \left(\frac{2\sqrt{d}}{\varepsilon^{\frac{1}{3}}} + 1\right)^d = O\left(\frac{(2\sqrt{d})^d}{\varepsilon^{\frac{d}{3}}}\right). \tag{20}$$

This can be reduced to $O((2\sqrt{d})^d/\varepsilon^{\frac{d}{3}-1})$, by keeping only highest and lowest points which are the same in their $(d-1)$ coordinates. So, for finding all diametrical pairs of the point set $\hat{\mathcal{S}}_1$, we need $O((2\sqrt{d})^d/\varepsilon^{\frac{d}{3}-1})^2) = O((2\sqrt{d})^{2d}/\varepsilon^{\frac{2d}{3}-2})$ time. Moreover, the number of points in two sets $\mathcal{B}_1$ or $\mathcal{B}_2$ is at most

$$\frac{Vol(\mathcal{B}_{2\xi_2})}{Vol(\mathcal{B}_\xi)} = \frac{(2\varepsilon^{\frac{1}{3}}\ell/2\sqrt{d})^d}{(\varepsilon\ell/2\sqrt{d})^d} = \frac{(2\varepsilon^{\frac{1}{3}})^d}{\varepsilon^d} = \frac{(2)^d}{\varepsilon^{\frac{2d}{3}}}. \quad (21)$$

This can be reduced to $O((2)^d/\varepsilon^{\frac{2d}{3}-1})$. Now, for computing $Diam(\mathcal{B}_1, \mathcal{B}_2)$, we use Chan's [9] recursive approach instead of using the quadratic brute-force algorithm on the point set $\mathcal{B}_1 \cup \mathcal{B}_2$. On the other hand, computing the diameter on a set of $O(1/\varepsilon^{\frac{2d}{3}-1})$ points using Chan's recursive approach takes the following recurrence based on the relation (19): $t_d(m) = O(m + 1/\sqrt{\varepsilon}t_{d-1}(O(1/\varepsilon^{\frac{2d}{3}-1})))$. By assuming $E = 1/\varepsilon$, we can rewrite the recurrence as:

$$t_d(m) = O(m + E^{\frac{1}{2}}t_{d-1}(O(E^{\frac{2d}{3}-1}))). \quad (22)$$

This can be solved to: $t_d(m) = O(m + E^{\frac{2d}{3}-\frac{1}{2}})$. In this case, $m = O(E^{\frac{2d}{3}-1})$, so, this recursive takes $O(E^{\frac{2d}{3}-\frac{1}{2}}) = O(1/\varepsilon^{\frac{2d}{3}-\frac{1}{2}})$ time. Moreover, if we have more than one diametrical pair $(\hat{p}_1, \hat{q}_1)$ in point set $\hat{\mathcal{S}}_1$, then this step takes at most $O((2^d)(2)^d/\varepsilon^{\frac{2d}{3}-\frac{1}{2}}) = O(2^{2d}/\varepsilon^{\frac{2d}{3}-\frac{1}{2}})$ time. So, we have total time:

$$T_d(n) = O(dn)+O\left(\frac{(2\sqrt{d})^{2d}}{\varepsilon^{\frac{2d}{3}-2}}\right)+O(2^d dn)+O\left(\frac{2^{2d}}{\varepsilon^{\frac{2d}{3}-\frac{1}{2}}}\right),$$

$$\leqslant O\left(2^d dn + \frac{(2\sqrt{d})^{2d}}{\varepsilon^{\frac{2d}{3}-\frac{1}{2}}}\right). \quad (23)$$

Since d is fixed, we have: $T_d(n) = O(n + \frac{1}{\varepsilon^{\frac{2d}{3}-\frac{1}{2}}})$.

In addition, Chan's recursive approach in line 8 of the Algorithm 2 returns a diametrical pair (p', q') which is a $(1 + O(\varepsilon))$-approximation for the diametrical pair $(\hat{p}, \hat{q}) \in \hat{\mathcal{S}}$. So, according to relation (12), we have:

$$||p' - q'|| \leqslant ||\hat{p} - \hat{q}|| \leqslant (1 + O(\varepsilon))||p' - q'||. \quad (24)$$

Moreover, the diametrical pair $(\hat{p}, \hat{q})$ is an approximation of the true diametrical pair $(p, q) \in \mathcal{S}$, and according to the relation (8), we have:

$$||p - q|| \leqslant ||\hat{p} - \hat{q}|| + \varepsilon\ell/2 \leqslant (1 + \varepsilon)||p - q||. \quad (25)$$

Hence, from (24) and (25) we can result:

$$\begin{aligned} ||p - q|| &\leqslant ||\hat{p} - \hat{q}|| + \varepsilon\ell/2, \\ &\leqslant ||\hat{p} - \hat{q}|| + \varepsilon||\hat{p} - \hat{q}||, \\ &\leqslant (1 + \varepsilon)||\hat{p} - \hat{q}||, \\ &\leqslant (1 + \varepsilon)((1 + O(\varepsilon))||p' - q'||), \\ &\leqslant (1 + O(\varepsilon))||p' - q'||. \end{aligned} \quad (26)$$

So, Algorithm 2 finds a $(1 + O(\varepsilon))$-approximation in $O(n + 1/\varepsilon^{\frac{2d}{3}-\frac{1}{2}})$ time and $O(n)$ space. □

3 Conclusion

We have presented two new non-constant approximation algorithms to compute the diameter of a point set $\mathcal{S}$ of n points in $\mathbb{R}^d$ for a fixed dimension d, which provide some improvements in terms of simplicity, and data structure.

References

[1] F. P. Preparata, and M. I. Shamos. *Computational Geometry: an Introduction*. New York, Springer-Verlag, pages 176–182, 1985.

[2] A. C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11:721–736, 1982.

[3] K. L. Clarkson, and P. W. Shor. Applications of random sampling in computational geometry. *Discrete and Computational Geometry*, 4:387–421, 1989.

[4] E. A. Ramos. An optimal deterministic algorithm for computing the diameter of a three-dimensional point set. *Discrete and Computational Geometry*, 26:245–265, 2001.

[5] O. Egecioglu, and B. Kalantari. Approximating the diameter of a set of points in the Euclidean space. *Information Processing Letters*, 32(4):205–211, 1989.

[6] P. K. Agarwal, J. Matousek, and S. Suri. Farthest neighbors maximum spanning trees and related problems in higher dimensions. *Computational Geometry: Theory and Applications*, 1:189–201, 1992.

[7] G. Barequet, and S. Har-Peled. Efficiently approximating the minimum-volume bounding box of a point set in three dimensions. *Journal of Algorithms*, 38:91–109, 2001.

[8] S. Har-Peled. A practical approach for computing the diameter of a point set. *In Proceedings of the 17th annual Symposium on Computational Geometry (SoCG'01)*, pages 177–186, 2001.

[9] T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. *International Journal of Computational Geometry and Applications*, 12:67–85, 2002.

[10] T. M. Chan. Faster core-set constructions and data stream algorithms in fixed dimensions. *Computational Geometry: Theory and Applications*, 35:20–35, 2006.

[11] T. M. Chan. Applications of Chebyshev polynomials to low-dimensional computational geometry. *In Proceedings of the 33rd International Symposium on Computational Geometry (SoCG'17)*, 26:1–15, 2017.

[12] S. Arya, G. D. da Fonseca, and D. M. Mount. Near-optimal ε-kernel construction and related problems. *In Proceedings of the 33rd International Symposium on Computational Geometry (SoCG'17)*, 10:1–15, 2017.

Computing the Shift-Invariant Bottleneck Distance for Persistence Diagrams

Don Sheehy* Oliver Kisielius† Nicholas Cavanna‡

Abstract

We define an algorithm that can compute the minimum of the bottleneck distance between two persistence diagrams over all diagonal shifts, in $O(n^{3.5})$ time. When applied to log-scale persistence diagrams, this is a scale-invariant version of bottleneck distance.

1 Introduction

A persistence diagram is a set of points in the plane that describes the changes in topology of the sublevel sets of a function. Each point's coordinates represent the birth and death of a topological feature. Often, persistence diagrams are generated from other geometric data sets and can serve as data summaries. They have risen to prominence in topological data analysis for their ability to capture multi-scale structure in a way that is invariant to distance-preserving transformations.

The stability theory of persistence diagrams implies that for small changes in the inputs, the persistence diagrams will have correspondingly small changes with respect to the bottleneck distance. This distance is defined in terms of a minimal matching between two diagrams that allows points to be matched with the diagonal. This distance is used as the foundation of all approximation results in persistent homology.

Persistence diagrams from metric inputs are sensitive to scaling of the input data. One way to combat this is to use log-scale persistence diagrams, as in [5]. In such a diagram, the prominence of a feature—its distance to the diagonal—is determined by the ratio of the death and birth times of the original diagram. This eliminates the artificial inflation of prominence that would result from a change in units.

Even log-scale persistence diagrams cannot recognize that two diagrams are generated by the same metric input measured in different units. Although the prominence of the features will remain the same, the two diagrams will differ by a shift along the diagonal. To resolve this, we introduce a new pseudometric, the *shifted bottleneck distance* on persistence diagrams that minimizes over all possible shifts, thus adding scale-invariance to the resulting metric space of diagrams. In the language of Euclidean geometry, this makes persistent homology useful not only for congruence but also similarity.

We give the formal definition of the shifted bottleneck distance and the proof of its metric properties. It is stable in the sense proven for bottleneck distance in [2]. Then, we show how to compute the distance in polynomial time.

1.1 Persistent Homology—A Quick Example

The results in this paper do not depend on a deep understanding of persistent homology, and we refer the reader to the accessible survey by Edelsbrunner and Morozov [3] for more background. We give a simple example here to show a common way that geometric points are turned into persistence diagrams that capture multiscale structure.

For the point set P shown in Figure 1, we will compute the persistent homology of the sublevel sets of the function $r_P : \mathbb{R}^2 \to \mathbb{R}$, which is the distance to the set P:

$$r_P(x) = \min_{p \in P} \|x - p\|. \tag{1}$$

The sublevel sets of r_P are topologically equivalent to subcomplexes of the Delaunay triangulation of P. As one considers larger scales (i.e. sublevels of r_P for larger thresholds), one obtains larger and larger subcomplexes of $\mathrm{Del}\,(P)$. The persistence algorithm will convert this growing sequence of complexes into a persistence diagram, $\mathrm{Dgm}(r_P)$, as shown in Figure 2. Each point in $\mathrm{Dgm}(r_P)$ is a pair (b, d) representing the birth and death of a topological feature. In general, for a filtration based on distance from a finite point set, one can use log-scale diagrams for features of any dimension except 0. The eye-catching features of P—two cycles—appear at different scales, and in the original persistence diagram, the inside of the big cycle dwarfs the other features. In the log-scale diagram, both cycles are prominent. Both diagrams are shown in Figure 2.

2 Defining Shifted Bottleneck Distance

The only distance we consider between points in the plane is the infinity metric, d_∞.

$$\mathrm{d}_\infty\left((x, y), (x', y')\right) = \max\{|x - x'|, |y - y'|\} \tag{2}$$

*Computer Science Department, University of Connecticut, don.r.sheehy@gmail.com

†Computer Science Department, University of Connecticut, oliver.kisielius@uconn.edu

‡Computer Science Department, University of Connecticut, nicholas.j.cavanna@uconn.edu

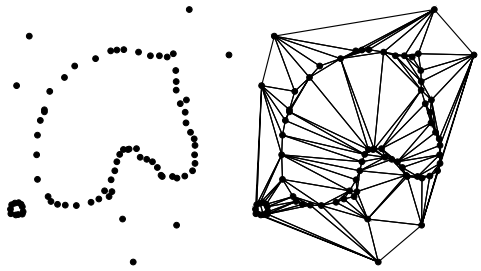


Figure 1: A point set P with its Delaunay triangulation $\mathrm{Del}\,(P)$

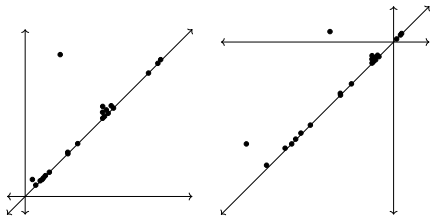


Figure 2: Persistence diagram from the filtration of P. Each point is a (birth, death) pair. On the left is the original persistence diagram. On the right is the log-scale diagram. The two cycles look similarly prominent on the log-scale.

2.1 Shifted Points and Shifted Bottleneck Distance

Fix $p = (x, y)$, a point in the plane, and fix $s \in \mathbb{R}$. Define the *image of p under shift s* as

$$p_s = (x + s, y + s). \tag{3a}$$

Define the image of an entire multiset A of points in $\mathbb{R}^2$ under shift s as the multiset

$$A_s = \{p_s \mid p \in A\}. \tag{3b}$$

If p is an off-diagonal point, then $\delta(p)$ denotes the orthogonal projection of p onto the diagonal.

$$\delta((x, y)) = (\frac{x+y}{2}, \frac{x+y}{2}) \tag{4}$$

Define Δ, *the diagonal*, to be the multiset containing each point (x, x) in $\mathbb{R}^2$ with infinite multiplicity.

Let A be a finite multiset in the plane, with $x < y$ for all (x, y) in A. Denote by $\hat{A}$ the infinite *persistence diagram* $A \cup \Delta$. We assume all persistence diagrams have finitely many off-diagonal points.

For two multisets of points A and B in $\mathbb{R}^d$, the *bottleneck distance* $\mathrm{d_B}\,(A, B)$ is defined as follows:

$$\mathrm{d_B}\,(A, B) = \min_{M} \max_{\langle a,b\rangle \in M} \mathrm{d}_\infty\,(a, b) \tag{5}$$

where M ranges over all perfect matchings between A and B.

For multisets A and B of points in the plane, define the *shifted bottleneck distance* of A and B:

$$\mathrm{d_{SB}}\,(A, B) = \min_{s\in\mathbb{R}} \mathrm{d_B}\,(A_s, B) \tag{6}$$

Given finite multisets A and B, our algorithm computes $\mathrm{d_{SB}}\,(\hat{A}, \hat{B})$.

Lemma 1 *If A and B are finite multisets of points, then $\mathrm{d_{SB}}\,(A, B)$ is well-defined.*

If A and B are finite multisets of points with $x < y$ for all (x, y) in $A \cup B$, then $\mathrm{d_{SB}}\,(\hat{A}, \hat{B})$ is well-defined.

Proof. Let $r = \inf_{s\in\mathbb{R}} \mathrm{d_B}\,(A_s, B)$. We need to show $\mathrm{d_B}\,(A_s, B) = r$ for some real shift s.

It's clear that $\mathrm{d_B}\,(A_s, B)$ is a continuous function of s. This means it is sufficient to demonstrate a closed, bounded set S such that $r = \inf_{s\in S} \mathrm{d_B}\,(A_s, B)$. The set

$$S = \bigcup_{\langle a,b\rangle \in A\times B} \{s \in \mathbb{R} \mid \mathrm{d}_\infty\,(a_s, b) \le r\} \tag{7}$$

will suffice. S is closed and bounded because it is a finite union of closed intervals.

We show that $\mathrm{d_{SB}}\,(\hat{A}, \hat{B})$ is well-defined by a similar argument. Let $\hat{r} = \inf_{s\in\mathbb{R}} \mathrm{d_B}\,(\hat{A}_s, \hat{B})$.

We can assume that A and B are nonempty and that $\hat{r} < \max_{p\in A\cup B} \mathrm{d}_\infty\,(p, \delta(p))$, because otherwise there is nothing to show. With that assumption, we use the same argument as before, with the same set S. □

If (6) used inf instead of min, then $\mathrm{d_{SB}}$ might be well-defined for more inputs. However, it is useful to know that there always exists some s such that

$$\mathrm{d_{SB}}\,(\hat{A}, \hat{B}) = \mathrm{d_B}\,(\hat{A}_s, \hat{B}).$$

Lemma 2 *Let X, Y, and Z be persistence diagrams or finite sets of points.*

$$\mathrm{d_{SB}}\,(X, Z) \le \mathrm{d_{SB}}\,(X, Y) + \mathrm{d_{SB}}\,(Y, Z) \tag{8}$$

In other words, shifted bottleneck distance satisfies the triangle inequality.

Proof. Let s_0 and s_1 be shifts such that

$$\mathrm{d_{SB}}\,(X, Y) = \mathrm{d_B}\,(X_{s_0}, Y) \tag{9}$$
$$\mathrm{d_{SB}}\,(Y, Z) = \mathrm{d_B}\,(Y_{s_1}, Z) \tag{10}$$

Since $\mathrm{d_B}$ is a metric, we have

$$\mathrm{d_B}\,(X_{s_0+s_1}, Z) \le \mathrm{d_B}\,(X_{s_0+s_1}, Y_{s_1}) + \mathrm{d_B}\,(Y_{s_1}, Z) \tag{11}$$
$$= \mathrm{d_B}\,(X_{s_0}, Y) + \mathrm{d_B}\,(Y_{s_1}, Z) \tag{12}$$
$$= \mathrm{d_{SB}}\,(X, Y) + \mathrm{d_{SB}}\,(Y, Z) \tag{13}$$

(We reach (12) by applying (3a), and we get (13) by applying (9) and (10).) Now we apply (6) to get (14).

$$\mathrm{d_{SB}}\,(X, Z) \le \mathrm{d_B}\,(X_{s_0+s_1}, Z) \tag{14}$$

Combining (13) and (14) yields (8). □

It's typical to compare two persistence diagrams using bottleneck distance, thanks to the following stability result proven in [2].

Let $\mathrm{Dgm}(f)$ denote the persistence diagram of sublevel sets of f. The main theorem of [2] states that, assuming some conditions on the topological space X and the continuous functions $f, g : X \to \mathbb{R}$, we have

$$\mathrm{d_B}\left(\mathrm{Dgm}(f), \mathrm{Dgm}(g)\right) \leq \|f - g\|_\infty \tag{15}$$

Since it's obvious that in general $\mathrm{d_{SB}}\left(\hat{A}, \hat{B}\right) \leq \mathrm{d_B}\left(\hat{A}, \hat{B}\right)$, the stability result by [2] must hold for $\mathrm{d_{SB}}$ as well:

Theorem 3 *Let X be a topological space X, and let $f, g : X \to \mathbb{R}$ be functions. If X, f, and g satisfy the conditions for the main stability result of [2], then we have the same result for shifted bottleneck distance.*

$$\mathrm{d_{SB}}\left(\mathrm{Dgm}(f), \mathrm{Dgm}(g)\right) \leq \|f - g\|_\infty \tag{16}$$

2.2 Related Work

We make use of some ideas from prior work concerning a different pseudo-metric, which we'll call *general shifted bottleneck distance*, or $\mathrm{d_{GSB}}$.

$$\mathrm{d_{GSB}}\left(X, Y\right) = \min_{t \in \mathbb{R}^2} \mathrm{d_B}\left(\{x + t \mid x \in X\}, Y\right) \tag{17}$$

Here X and Y are finite sets of points in the plane.

The earliest relevant algorithm for $\mathrm{d_{GSB}}$ is by Alt et al. in [1]. They compute $\mathrm{d_{GSB}}$ in time $O(n^6 \log n)$. To do this, they first make an $O(n^6)$ time decision algorithm that, given X, Y, and r, tests whether $\mathrm{d_{GSB}}\left(X, Y\right) \leq r$. Then they generate and sort all $O(n^6)$ possible answers and find the correct answer with a binary search.

One idea of theirs that we adopt is their subroutine in which a bipartite matching is repeatedly maximized (and pruned) while the set of available edges changes incrementally. This involves $O(n^4)$ invocations of the Hopcroft-Karp augmenting paths algorithm at a cost of $O(n^2)$ time per augmenting path. That's lower than the cost of computing a matching from scratch $O(n^4)$ times.

Efrat et al. improved this result by using geometry to optimize the augmenting-path routine [4]. They use near-neighbor structure to represent edges implicitly during the graph searches of the Hopcroft-Karp, which results in running time of $\log n$ per node of the graph, and thus $O(n \log n)$ per augmenting path. Like Alt et al., they find $O(n^4)$ total augmenting paths, so their algorithm runs in $O(n^5 \log n)$ time.

Efrat et al. also use the optimized Hopcroft-Karp algorithm to compute bottleneck distance between finite point sets [4], and Kerber et al. use the same technique to compute bottleneck distance between persistence diagrams in $O(n^{1.5} \log n)$ time [6].

To minimize $\mathrm{d_B}$ over all two-dimensional shifts, Alt et al. pay quadratic time just to reduce the problem to a one-dimensional problem in polar coordinates. Their key idea is to guess ($O(n^2)$ times) which edge is the bottleneck. Knowing that $\langle x, y\rangle \in X \times Y$ is the bottleneck, you can test whether $\mathrm{d_{GSB}}\left(X, Y\right) \leq r$ by testing only shifts t such that $\mathrm{d_\infty}\left(x + t - y\right) = r$. (This works for Euclidean distance as well.) Then only a one-dimensional value, the angle from y to $x+t$, is unknown. And so they compute $O(n^2)$ *critical angles* at which another edge has value exactly r, and they check for a matching at each critical angle.

Our algorithm is faster. Since we compute $\mathrm{d_{SB}}$, we have a one-dimensional parameter from the beginning, the shift, so we need not spend $O(n^2)$ immediately. Furthermore, in our setting we can process the *critical shifts* only once, reducing the radius and reordering the critical shifts on the fly. Because we needn't perform a binary search, our full algorithm resembles the decision-only version of the other algorithms.

3 Background

Throughout the remaining discussion we refer to A and B, the finite input multisets to our algorithm. These are to be distinguished from the infinite sets $\hat{A}$ and $\hat{B}$.

3.1 Diagonal-Perfect Matchings

Edges with at least one end on the diagonal are called *diagonal edges*. Edges in $A \times B$ are *non-diagonal edges*.

A finite matching M between $\hat{A}$ and $\hat{B}$ is *diagonal-perfect* if the degree in M of each point in $A \sqcup B$ equals the multiplicity of that point. For such a matching M, the *value* of M is the minimum, over all shifts, of the greatest edge length in M.

$$\texttt{value}\left(M\right) = \min_{s \in \mathbb{R}} \max_{\langle a,b\rangle \in M} \mathrm{d_\infty}\left(a_s, b\right)$$

An *r-matching* is a diagonal-perfect matching M between $\hat{A}$ and $\hat{B}$ with $\texttt{value}\left(M\right) = r$. Clearly such a matching is a certificate that $\mathrm{d_{SB}}\left(\hat{A}, \hat{B}\right) \leq r$. A *less-than-$r$-matching* is an r'-matching for some $r' < r$.

If M is a diagonal-perfect matching between $\hat{A}$ and $\hat{B}$, then the union of M with any perfect matching from Δ to Δ is a perfect matching. In particular, if you extend M by adding edges of the form $\langle x, x_s\rangle \in \Delta$, where s is the optimal shift for M, then the value of the resulting perfect matching is $\texttt{value}\left(M\right)$.

As proven in [6], if an r-matching exists, then one exists that contains no "skew" diagonal edges. A *non-skew diagonal edge* is an edge $\langle p, \delta(p)\rangle$ or $\langle \delta(p), p\rangle$. This includes $\langle p, p\rangle$ where $p = \delta(p)$.

3.2 Working with the Diagonal

The addition of points on the diagonal in the definition of persistence diagrams is useful for stability results, but requires special consideration in our algorithm. Bottleneck distance between diagrams can be reduced to bottleneck distance between finite sets using (18).

$$\mathrm{d_B}\left(\hat{A}, \hat{B}\right) = \mathrm{d_B}\left(A \cup \delta(B), B \cup \delta(A)\right) \tag{18}$$

However, this does not work for shifted bottleneck distance. It is not true that $\mathrm{d_{SB}}\left(\hat{A}, \hat{B}\right) = \mathrm{d_{SB}}\left(A \cup \delta(B), B \cup \delta(A)\right)$, because you can't match B to the shifted image $\delta(B)_s$ or match A_s to $\delta(A)$. Instead, we must handle the diagonal as a special case. As noted in [6], it is faster to give the diagonal special treatment, because all but $O(n)$ edges involving the diagonal can be ignored.

Diagonal-to-Diagonal Edges Augmenting paths discovered by the Hopcroft-Karp algorithm can include diagonal-to-diagonal (Δ-to-Δ) edges. Because $\hat{A}$ and $\hat{B}$ include every point of the diagonal with infinite multiplicity, the length of the longest Δ-to-Δ edge in any matching can be made arbitrarily small via augmenting paths in $\Delta \times \Delta$. This lets us consider the length of Δ-to-Δ edges to be zero. The diagonal parts of $\hat{A}$ and $\hat{B}$ form a complete bipartite graph on zero-length edges. As a result, we represent $\hat{A} \cap \Delta$ and $\hat{B} \cap \Delta$ as two nodes Δ_A and Δ_B, both with infinite multiplicity. We identify any edge $\langle a, \delta(a)\rangle$ with $\langle a, \Delta_B\rangle$, and similarly we identify $\langle \delta(b), b\rangle$ with $\langle \Delta_A, b\rangle$.

As noted in [6], the near-neighbor search structure (used in the optimized Hopcroft-Karp algorithm) can be adapted to handle Δ_A and Δ_B.

4 A Kinetic Data Structure Approach

The main algorithm will look at an increasing sequence of shifts. At different shifts, edges will appear or disappear. These are the *events* we want to track. Moreover, as we discover better matchings, our upper bound on the radius r decreases. Changing the radius reorders future events, i.e. the partition P_r. In this section, we will define the events and introduce an *event queue* that provides access to the events in the correct order.

4.1 Searching for a Better Matching

Let r be an upper bound on $\mathrm{d_{SB}}\left(\hat{A}, \hat{B}\right)$. The *left shift* λ of a non-diagonal edge $\langle (a_x, a_y), (b_x, b_y)\rangle$ at radius r is

$$\lambda(\langle (a_x, a_y), (b_x, b_y)\rangle, r) = \max\{b_x - r - a_x, b_y - r - a_y\}.$$

Similarly, the *right shift* ρ is

$$\rho\left(\langle (a_x, a_y), (b_x, b_y)\rangle, r\right) = \min\{b_x + r - a_x, b_y + r - a_y\}.$$

For any edge $e = \langle a, b\rangle$ and shift s such that $\mathrm{d}_\infty(a_s, b) < r$, we have $\lambda(e, r) < s < \rho\left(e, r\right)$. Let P_r denote the set of all left and right shifts, i.e.

$$P_r = \{\lambda(e, r) \mid e \in A \times B\} \cup \{\rho\left(e, r\right) \mid e \in A \times B\}$$

If $s_0, \ldots, s_k$ are the shifts of P_r, where $s_0 < \cdots < s_k$, then $\textsc{intvl}\, P_r$ is the set of open intervals $\{(s_i, s_{i+1}) \mid i \in \{0, \ldots, k-1\}\}$.

The *set of available non-diagonal edges* for a shift s and radius r is:

$$\mathcal{E}(r, s) = \{\langle a, b\rangle \in A \times B \mid \mathrm{d}_\infty\left(a_s, b\right) < r\}$$

At radius r, the *set of available non-skew diagonal edges*, which includes $\langle \Delta_A, \Delta_B\rangle$ when $r > 0$, is:

$$\mathcal{D}(r) = \{e \in A \times \{\Delta_B\} \cup \{\Delta_A\} \times B \cup \{\langle \Delta_A, \Delta_B\rangle\} \mid \mathrm{d}_\infty(e) < r\}$$

For each interval (s_i, s_{i+1}) in $\textsc{intvl}\, P_r$, we have a graph

$$G(r, s_i) = \mathcal{E}(r, \frac{s_i + s_{i+1}}{2}) \cup \mathcal{D}(r).$$

The graph contains the diagonal edges and the available non-diagonal edges at a shift inside the interval (s_i, s_{i+1}). The choice of the midpoint is arbitrary, and indeed, $G(r, s_i) = \mathcal{E}(r, s) \cup \mathcal{D}(r)$ for any $s_i < s < s_{i+1}$. This implies the following lemma. (Proofs of these facts can be found in the appendix.)

Lemma 4 *If M is a less-than-r-matching, then $M \subseteq G(r, \lambda(e, r))$ for some e in $A \times B$.*

Lemma 5 *For any edge e and radii $r' < r$, we have $G(r', \lambda(e, r')) \subseteq G(r, \lambda(e, r))$.*

4.2 The Event Queue

In this section, we will describe the event queue data structure. The events provided by this structure are *L-events* and *R-events*. An event `e` of either type stores an edge `e.edge`. An L-event also stores a shift `e.shift` such that $G(r, \lambda(\texttt{e.edge}, r)) = \mathcal{E}(r, \texttt{e.shift}) \cup \mathcal{D}(r)$.

The event queue `Q` holds three stacks of edges. The stack `Q.D` contains all non-skew diagonal edges, including $\langle \Delta_A, \Delta_B\rangle$, in decreasing order by length. (Longer edges are popped first.) The stacks `Q.L` and `Q.R` contain the edges of $A \times B$ sorted increasing by λ and ρ respectively (at radius 0). The order within those stacks does not depend on the radius, because for any $x, r \in \mathbb{R}$, $\lambda(e, r + x) - \lambda(e, r) = x$ and $\rho\left(e, r - x\right) - \rho\left(e, r\right) = x$.

The method `Q.nextevent`(r) goes like this: If the top of `Q.D` has length r or greater, return an R-event for `Q.D.pop()`. If $\rho\left(\texttt{Q.R}, r\right) \le \lambda(\texttt{Q.L}, r)$, return an R-event for `Q.R.pop()`. Otherwise, return an L-event for `Q.L.pop()`. Also remove from `Q.L` any edges with the same left shift as the edge popped.

Input: Event queue Q, radius r
1 **if** $|\texttt{Q.D}| > 0$ *and* $\mathrm{d}_\infty(\texttt{Q.D.top}()) \geq r$ **then**
2 | **return** RightEvent(edge $\leftarrow$ Q.D.pop())
3 **if** $\rho(\texttt{Q.R.top}(), r) \leq \lambda(\texttt{Q.L.top}(), r)$ **then**
4 | **return** RightEvent(edge $\leftarrow$ Q.R.pop())
5 Let e $\leftarrow$ Q.L.pop()
6 **while** $|\texttt{Q.L}| > 0$ *and* $\lambda(\texttt{e}, 0) = \lambda(\texttt{Q.L.top}(), 0)$ **do**
7 | Q.L.pop()
8 Let t $\leftarrow \rho(\texttt{Q.R.top}(), r)$
9 **if** $|\texttt{Q.L}| > 0$ **then**
10 | Set t $\leftarrow \min\{\texttt{t}, \lambda(\texttt{Q.L.top}(), r)\}$
11 **return**
LeftEvent(edge $\leftarrow$ e, shift $\leftarrow (\lambda(\texttt{e}, \texttt{r}) + \texttt{t}) \div 2$)

Algorithm 1: Q.nextevent(r)

Theorem 6 *Let* Q *be the event queue, and let* $r_1 \geq r_2 \geq \cdots \geq r_n$ *be a nonincreasing sequence of radii. Say that* Q.nextevent(r_i) *is called for* i *from 1 to* n *in order, and suppose the call to* Q.nextevent(r_n) *returns an L-event whose shift is* s*. Then after the sequence of calls, we have the following.*

$$\mathcal{D}(r_n) = \texttt{Q.D} \tag{19}$$
$$\mathcal{E}(r_n, s) = \texttt{Q.R} \setminus \texttt{Q.L} \tag{20}$$

Here, Q.L*,* Q.R*, and* Q.D *are treated as sets.*

The proof is in the appendix.

Corollary 7 *A modified version of Theorem* 6 *holds, where in the sequence of operations,* e $\leftarrow$ Q.nextevent(r) *may be followed immediately by* Q.L.push(e.edge)*, provided* e *is an L-event.*

Proof. Reinserting the edge undoes the previous operation and has no other side effects. □

4.3 Reducing the Radius

To reduce the radius after we find a perfect matching, we use the method Q.newradius(), which returns

$$\max\{\mathrm{d}_\infty(\texttt{Q.D.top}()), \frac{1}{2}(\lambda(\texttt{Q.L.top}(), 0) - \rho(\texttt{Q.R.top}(), 0))\}.$$

Lemma 8 *The invocation* Q.nextevent(r) *returns an R-event if and only if* $r \leq$ Q.newradius()*.*

Proof. Consider Algorithm 1. Q.nextevent(r) returns an R-event from line 2 if and only if $r \leq \mathrm{d}_\infty(\texttt{Q.D.top}())$. Otherwise, Q.nextevent() returns an R-event if and only if $\rho(\texttt{Q.R.top}(), r) \leq \lambda(\texttt{Q.L.top}(), r)$. In fact $\rho(\texttt{Q.R.top}(), r) \leq \lambda(\texttt{Q.L.top}(), r)$ exactly when $r \leq \frac{1}{2}(\lambda(\texttt{Q.L.top}(), 0) - \rho(\texttt{Q.R.top}(), 0))$, with equality only when $r = \frac{1}{2}(\lambda(\texttt{Q.L.top}(), 0) - \rho(\texttt{Q.R.top}(), 0))$. Thus r.nextevent() an L-event is returned, by line 11, if and only if $r >$ Q.newradius(). □

Lemma 9 *Suppose the event queue* Q *is in a state such that* Q.nextevent(r) *would return an L-event with shift* s*, and there is a less-than-*r*-matching* M *in* $G(r, s)$*. Then*

$$r > \texttt{Q.newradius}() \geq \texttt{value}(M) \geq \mathrm{d}_{\mathrm{SB}}(\hat{A}, \hat{B}).$$

Proof. Lemma 8 gives us $r >$ Q.newradius(), since Q.nextevent(r) would return an L-event. For any $r > r' >$ Q.newradius(), call s' the shift of the L-event resulting from Q.nextevent(r'). Because Q.nextevent(r) and Q.nextevent(r') have equivalent effects on the state of Q, Theorem 6 says $G(r', s') = G(r, s)$. Thus $r' >$ value(M) for any $r' >$ Q.newradius(), and so Q.newradius() $\geq$ value(M). The last inequality, value(M) $\geq \mathrm{d}_{\mathrm{SB}}(\hat{A}, \hat{B})$, holds for any diagonal-perfect M. □

5 The Algorithm

Here we state the main algorithm and prove its correctness and running time in the real RAM model.

5.1 Algorithm for Shifted Bottleneck Distance

Algorithm 2 computes the shifted bottleneck distance as follows. Let Q be the event queue. Set the radius r to be an upper bound on $\mathrm{d}_{\mathrm{SB}}(\hat{A}, \hat{B})$, say the length of the longest non-skew diagonal edge. Maintain a bipartite matching M, initially empty, between $\hat{A}$ and $\hat{B}$. While Q is not empty, get the next event e from Q. If e is an R-event, remove e.edge from M. Otherwise, e is an L-event: Augment M using the geometrically-optimized version of Hopcroft-Karp (as in [4] and [6]), and if M is now diagonal-perfect, then reinsert e.edge into Q.L and reduce r to Q.newradius(). Finally, return r, which now equals $\mathrm{d}_{\mathrm{SB}}(\hat{A}, \hat{B})$.

Input: Multisets A and B representing diagrams $\hat{A}$ and $\hat{B}$
1 Let $r \leftarrow \max\{\mathrm{d}_\infty(x, \delta(x)) \mid x \in A \cup B\}$
2 Let Q be the event queue
3 Let M be an empty matching
4 **while** Q.L, Q.R *and* Q.D *are nonempty* **do**
5 | Let e $\leftarrow$ Q.nextevent(r)
6 | **if** e *is an R-event* **then**
7 | | Remove e.edge from M
8 | **else**
9 | | Use augmenting paths to maximize M at shift e.shift and radius r
10 | | **if** M *is diagonal-perfect* **then**
11 | | | Q.L.push(e.edge)
12 | | | $r \leftarrow$ Q.newradius()
13 **return** r

Algorithm 2: $\mathrm{d}_{\mathrm{SB}}(\hat{A}, \hat{B})$

5.2 Correctness in the Real RAM Model

Lemma 10 *After line 9 executes,* M *is a maximum matching in* $G(r, \lambda(\texttt{e.edge}, r))$.

Proof. If $\texttt{M} \subseteq G(r, \lambda(\texttt{e.edge}, r))$ before line 9, then M is a maximum matching in $G(r, \lambda(\texttt{e.edge}, r))$ after line 9 because the augmenting path algorithm .

It will suffice to show that $\texttt{M} \subseteq G(r, \lambda(\texttt{e.edge}, r))$ whenever the execution reaches line 9. We proceed by induction. For the base case, in the first execution of line 9, the matching M is initially empty.

In the inductive case, we have $\texttt{M} \subseteq G(r, \lambda(\texttt{e.edge}, r))$ after line 9. By Corollary 7, this is equivalent to $\texttt{M} \subseteq \texttt{Q.D} \cup \texttt{Q.R} \setminus \texttt{Q.L}$. This still holds just before line 9 next executes, because Q.L has not increased and every edge popped from Q.D or Q.R has been removed from M. □

Theorem 11 *Given persistence diagrams A and B, Algorithm 2 outputs* $\mathrm{d_{SB}}(\hat{A}, \hat{B})$ *in time* $O(n^{3.5})$ *where* $n = |A| + |B|$.

Proof. Each iteration makes progress toward termination. For iterations where we reinsert an edge into Q.L, we set r to Q.newradius(), guaranteeing an R-event will be processed next (Lemma 8). In all other cases, we shrink Q.L, Q.R, or Q.D. Thus the outer loop executes at most $2|A||B| + |A| + |B|$ times. As in [6], line 9 takes time $O((|A| + |B|)^{1.5})$ per augmenting path. We find at most $|A| + |B|$ paths the first time we augment the matching, and subsequently we find at most one path per event, as in [4]. Thus the total running time is $O((|A| + 1)(|B| + 1)(|A| + |B|)^{1.5})$, i.e. $O(n^{3.5})$.

Initially, $r = \texttt{value}(\mathcal{D}(\infty)) \geq \mathrm{d_{SB}}(\hat{A}, \hat{B})$. (Note $\mathcal{D}(\infty)$ is diagonal-perfect.) Because M is diagonal-perfect at line 12, Lemma 9 tells us that $r \geq \mathrm{d_{SB}}(\hat{A}, \hat{B})$ always and that r always decreases at line 12.

We reinsert an edge e in Q.L unless $G(r, \lambda(e, r))$ contains no diagonal-perfect matching. When the radius decreases from r to r', we get $G(r', \lambda(e, r')) \subseteq G(r, \lambda(e, r))$ for each edge e by Lemma 5. So by induction, there is never a diagonal-perfect matching in $G(r, \lambda(e, r))$ for any edge e in $A \times B \setminus \texttt{Q.L}$ at the start of the loop. The base case is vacuous.

When we exit the loop, we have $G(r, \lambda(e, r)) = \mathcal{D}(r)$ for every edge e in Q.L. (This is vacuous if Q.L is empty; if Q.D is empty, then $r = 0$; otherwise, Q.R is empty, and Theorem 6 applies.) So we know there is no diagonal-perfect matching in $G(r, \lambda(e, r))$ for any edge e in $A \times B$. Thus $r \leq \mathrm{d_{SB}}(\hat{A}, \hat{B})$ by Lemma 4. Now $r \leq \mathrm{d_{SB}}(\hat{A}, \hat{B}) \leq r$, and so $r = \mathrm{d_{SB}}(\hat{A}, \hat{B})$. □

5.3 A Constant-Factor Improvement

At line 6 of Algorithm 1, whenever several edges in Q.L have the same left shift, we discard all but one of them. With negligible extra effort, we set e to be the edge maximizing $\rho(e, 0)$. Then before line 9 of Algorithm 2, we test, for the event e, whether $\lambda(\texttt{e.edge}, r) \geq \rho(\texttt{e.edge}, r)$, and if so we skip the rest of the iteration. (In particular, we skip the expensive line 9.)

If $\lambda(\texttt{e.edge}, r) \geq \rho(\texttt{e.edge}, r)$, then e.edge is not in $G(r, \lambda(\texttt{e.edge}, r))$, and neither are any other edges with the same left shift as e.edge. This means $G(r, \lambda(\texttt{e.edge}, r)) \subseteq G(r, \lambda(e', r))$, where e' is the previous edge popped from Q.L. Since we have no hope of finding a diagonal-perfect matching, it is sound to skip the rest of the iteration.

6 Implementation

We have implemented Algorithm 2 with Python 3. Our near-neighbor structure uses a kd-tree. Our implementation is slow for even small point sets. (This is consistent with the $O(n^{3.5})$ running time.) For inputs of sizes 32 (i.e. $|A| = |B| = 32$), 64, and 128, the computation takes about two seconds, 15 seconds, and two minutes. To compute shifted bottleneck distance for medium or large point sets, we will need a faster algorithm.

Our implementation needed some tweaking to account for floating point errors. The research-grade code is available on request.

References

[1] H. Alt, K. Mehlhorn, H. Wagener, and E. Welzl. Congruence, similarity, and symmetries of geometric objects. In *Proceedings of the Third Annual Symposium on Computational Geometry*, SCG '87, pages 308–315, New York, NY, USA, 1987. ACM.

[2] D. Cohen-Steiner, H. Edelsbrunner, and J. Harer. Stability of persistence diagrams. *Comput. Geom*, 37:103–120, 2007.

[3] H. Edelsbrunner and D. Morozov. Persistent homology: Theory and practice. In *European Congress of Mathematics Kraków, 2 - 7 July, 2012*, pages 31 – 50. European Mathematical Society Publishing House, 2012.

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

[5] B. Hudson, G. L. Miller, S. Y. Oudot, and D. R. Sheehy. Topological inference via meshing. In *Proceedings of the 26th ACM Symposium on Computational Geometry*, pages 277–286, 2010.

[6] M. Kerber, D. Morozov, and A. Nigmetov. Geometry helps to compare persistence diagrams. *CoRR*, abs/1606.03357, 2016.

Appendix

Lemma 12 *If two shifts s_0 and s_1 lie in the same (open) interval of* $\textsc{intvl}(P_r)$*, then* $\mathcal{E}(r, s_0) = \mathcal{E}(r, s_1)$.

Proof. Suppose (WLOG) that $s_0 < s_1$ and that we have some edge $e = \langle a, b\rangle \in \mathcal{E}(r, s_0) \setminus \mathcal{E}(r, s_1)$. Then, since $\mathrm{d}_\infty(a_s, b)$ is a continuous function of s, the intermediate value theorem tells us there is a shift $s_0 < s' < s_1$ such that $\mathrm{d}_\infty(a_{s'}, b) = r$.

This s' must be either $\lambda(e, r)$ or $\rho(e, r)$, and so s_0 and s_1 do not lie in the same interval of $\textsc{intvl}(P_r)$. □

There are a few prerequisites to Lemma 4.

Lemma 13 *Let $s_{i-1} < s < s_i < s' < s_{i+1} \in \mathbb{R}$, where s_{i-1}, s_i, s_{i+1} are consecutive elements of P_r. Then $\mathcal{E}(r, s_i) \subseteq \mathcal{E}(r, s) \cap \mathcal{E}(r, s')$.*

Proof. Suppose for contradiction there is an edge $e = \langle a, b\rangle$ in $\mathcal{E}(r, s_i) \setminus \mathcal{E}(r, s)$. Lemma 12 tells us $e \notin \mathcal{E}(r, t)$ whenever $s_{i-1} < t < s_i$. Thus for the continuous function $f(t) = \mathrm{d}_\infty(a_t, b)$, we have $f(t) \geq r$ for $s_{i-1} < t < s_i$ but $f(s_i) < r$, which is impossible. Therefore $\mathcal{E}(r, s_i) \subseteq \mathcal{E}(r, s)$.

A similar argument shows $\mathcal{E}(r, s_i) \subseteq \mathcal{E}(r, s')$. □

Lemma 14 *Let $s_i < s_{i+1} < s_{i+2}$ be consecutive elements of P_r such that s_{i+1} is not the left shift of any edge. If $s_i < s < s_{i+1} < s' < s_{i+2}$, then $\mathcal{E}(r, s') \subseteq \mathcal{E}(r, s)$.*

Consequently, if there is no diagonal-perfect matching in $G(r, s_i)$, then there is no such matching in $G(r, s_{i+1})$.

Proof. The statement $\mathcal{E}(r, s') \subseteq \mathcal{E}(r, s)$ holds because if we have an edge $e \in \mathcal{E}(r, s') - \mathcal{E}(r, s)$, then $\lambda(e, r) = s_{i+1}$, which violates the premise.

The second statement is immediate. □

Proof. [Lemma 4] Let M be a less-than-r-matching. We know $M \subseteq \mathcal{E}(r, s) \cup \mathcal{D}(r)$ for some shift s. Consider two cases:

1. $\min P_r < s < \max P_r$. Lemma 13 lets us assume WLOG that $s_i < s < s_{i+1}$ for consecutive elements s_i, s_{i+1} of P_r. Then Lemma 12 tells us $\mathcal{E}(r, s) \cup \mathcal{D}(r) = G(r, s_i)$.

   If s_i is not a left shift, we can apply Lemma 14 to show that $M \subseteq G(r, s_{i-1})$. We iterate this until we reach a left shift. (If we never reach a left shift, then $M \subseteq \mathcal{D}(r)$.)
2. $s \leq \min P_r$ or $s > \max P_r$. This means $\mathcal{E}(r, s) = \emptyset$, so $M \subseteq \mathcal{D}(r) \subseteq G(r, t)$ for any t in P_r.

□

Proof. [Lemma 5] Pick some small offset t such that $G(r', \lambda(e, r')) = \mathcal{E}(r', \lambda(e, r') + t) \cup \mathcal{D}(r')$ and $G(r, \lambda(e, r)) = \mathcal{E}(r, \lambda(e, r) + t) \cup \mathcal{D}(r)$. It is clear that $\mathcal{D}(r') \subseteq \mathcal{D}(r)$.

Let e' be an edge in $\mathcal{E}(r', \lambda(e, r') + t)$. This means

$$\lambda(e', r') < \lambda(e, r') + t < \rho(e', r').$$

Subtracting $(r - r')$ from all three sides yields

$$\lambda(e', r) < \lambda(e, r) + t < \rho(e', r) - 2(r - r').$$

This means $e' \in \mathcal{E}(r, \lambda(e, r) + t) \subseteq G(r, \lambda(e, r))$, since $r - r' > 0$. Thus $G(r', \lambda(e, r')) \subseteq G(r, \lambda(e, r))$. □

Proof. [Theorem 6] Let $\{\mathtt{e}_i\}_{1 \leq i \leq n}$ be the results of the n successive calls $\mathtt{Q.nextevent}(r_i)$.

Since $\mathtt{e}_n$ is an L-event, we know the condition at line 1 is false during the call $\mathtt{Q.nextevent}(r_n)$. Therefore, there are no edges in $\mathtt{Q.D}$ of length r_n or less, and so $\mathtt{Q.D} \subseteq \mathcal{D}(r_n)$. Because $r_i \geq r_n$ for all $i < n$, we have only popped edges longer than r_n from $\mathtt{Q.D}$. Thus $\mathcal{D}(r_n) \subseteq \mathtt{Q.D}$, and (19) follows.

Because the order of $\mathtt{Q.L}$ is independent of the radius, and because we pop all edges with the same left shift as $\mathtt{e}_n\mathtt{.edge}$, we have

$$\mathtt{Q.L} = \{e \in A \times B \mid \lambda(e, r_n) > \lambda(\mathtt{e}_n\mathtt{.edge}, r_n)\}.$$

If $\mathtt{Q.L}$ is nonempty, then $\mathtt{e}_n\mathtt{.shift} < \lambda(\mathtt{Q.L.top}(), r_n)$. Thus

$$\mathtt{Q.L} = \{e \in A \times B \mid \lambda(e, r_n) > \mathtt{e}_n\mathtt{.shift}\}. \tag{21}$$

Let

$$S = \{e \in A \times B \mid \lambda(\mathtt{e}_n\mathtt{.edge}, r_n) < \rho(e, r_n)\}.$$

We will prove $\mathtt{Q.R} = S$, which implies

$$\mathtt{Q.R} = \{e \in A \times B \mid \mathtt{e}_n\mathtt{.shift} < \rho(e, r_n)\}. \tag{22}$$

Because the call $\mathtt{Q.nextevent}(r_n)$ returns an L-event, we know $\mathtt{Q.R} \subseteq S$ because of line 3. Fix an edge e in $A \times B \setminus \mathtt{Q.R}$. Let $\mathtt{e}_i$ be the R-event such that $e = \mathtt{e}_i\mathtt{.edge}$. At the time when $\mathtt{Q.nextevent}(r_i)$ returns $\mathtt{e}_i$, we have $\rho(e, r_i) \leq \lambda(\mathtt{Q.L.top}(), r_i) \leq \lambda(\mathtt{e}_n\mathtt{.edge}, r_i)$. Because $r_i \leq r_n$, we get $\rho(e, r_n) \leq \lambda(\mathtt{e}_n\mathtt{.edge}, r_n)$, which means e is not in S. This means $S \subseteq \mathtt{Q.R}$, and so $S = \mathtt{Q.R}$.

From (21) and (22), we get

$$\begin{aligned}\mathtt{Q.R} \setminus \mathtt{Q.L} &= \{e \in A \times B \mid \lambda(e, r_n) < \mathtt{e}_n\mathtt{.shift} < \rho(e, r_n)\} \\ &= \mathcal{E}(r_n, \mathtt{e}_n\mathtt{.shift}).\end{aligned}$$

□

Hitting a Set of Line Segments with One or Two Discrete Centers

Xiaozhou He* Zhihui Liu† Bing Su‡ Yinfeng Xu§ Feifeng Zheng¶ Binhai Zhu‖

Abstract

Given the scheduling model of bike-sharing, we consider the problem of hitting a set of n axis-parallel line segments in $\mathbb{R}^2$ by a square (and two squares) whose center(s) must lie on some line segment(s) such that the (maximum) edge length of the square(s) is minimized. Under a different model, we also consider the cases when one needs to compute one (and two) centers on some edge(s) of a tree of size m, where n labeled segments must be hit, and the objective is to minimize the maximum path length from the labeled segments to the nearer center(s). We give three linear-time algorithms and an $O(n^2 \log n)$ algorithm for the four problems in consideration.

1 Introduction

In recent years, the (private) bike-sharing business are booming in China (and in Singapore). To use a shared-bike, a user can use his/her smartphone to scan and unlock the bike. A small amount of fee, about US$0.16 currently, is charged for any use/transaction during that day. It is estimated that there are at least 30 million such transactions in major cities of China alone. Different from the traditional public bike-sharing services, wherein a user must return the bike to specified bike racks at fixed locations, in this bike-sharing service a user can lock and drop a bike anywhere after finishing using it. Of course, a lot of these bikes are dropped on some streets typically near bus/subway stations. In fact, right before and after rush hours, it is not uncommon to notice hundreds of bikes near some major subway stations in big cities like Beijing and Shanghai. This also holds when there is a major event near some site, like an open music show.

*Business School, Sichuan University, Chengdu, Sichuan, China, xiaozhouhe126@qq.com

†School of Computer Science and Technology, Shandong Technology and Business University, Yantai, Shandong, China, dane.zhihui.liu@gmail.com

‡School of Economics and Management, Xi'an Technological University, Xi'an, China, subing684@sohu.com

§School of Management, Xi'an Jiaotong University, Xi'an, China, yfxu@mail.xjtu.edu.cn

¶Glorious Sun School of Business and Management, Donghua University, Shanghai, China, ffzheng@dhu.edu.cn

‖Gianforte School of Computing, Montana State University, Bozeman, MT, 59717, USA, bhz@montana.edu

For the bike-sharing company, the objective is certainly to maximize the profit (i.e., the number of use of the bikes) and minimize the cost (i.e., collecting the scattered bikes quickly, and manually, to re-distribute them in bulk). Our research is motivated by this: given a set of roads (segments) scattered with shared-bikes, distribute these bikes in bulk from a center (or several centers) and transport them to these streets. Hence the problem is to find a center (resp. several centers) on these roads as the stations to store the bikes so that the distance to the farthest target road from the nearest station is minimal. Note that these centers change when the target roads are changed.

In this paper, we give two model of the streets in the cities. One is the classic grid network that is widely used in the urban streets model. In this model, we describe all the n target roads as some axis-parallel line segments and we use the ℓ_∞-norm to measure the distance. Here we also consider a practical restriction: the center (station) is exactly on one line segment (road) for the convenience of storage and scheduling, and we only need to touch every line segment (target road) at any point (position) to manually distribute the bikes. Also, note that a ℓ_∞ circle is an axis-parallel square.

Thus, follow this model the one-hitting-square problem is to find the minimum axis-parallel square whose center is on a line segment, to hit all the line segments, such that the edge length of the square is minimized. Analogously, the two-hitting-square problem can be defined.

In addition to the grid networks, we also consider the geometric tree network, with size m, to model the streets. The target roads are n segments/edges on the tree and the distance between any two points on the tree is the shortest distance between them along the tree edges. In this case we consider the one-center and two-center problems such that the centers must lie on some tree edges and the maximum distance between the target segments to the nearer centers is minimized. We next review some previous works.

When the target are n points and the distance is ℓ_2, the corresponding one-center [9, 12], two-center [2, 6, 11] (and discrete two-center [1]) problems have been well studied. In fact, even under ℓ_∞, the two-center and three-center problem can be solved in $O(n)$ time [4, 7] and a variation of the discrete two-center problem (where the centers of the congruent axis-parallel squares must be on some input points and the area of the squares

is minimized) can be solved in $O(n \log^2 n)$ time [8]. (There are other variations of these problems, like the target to cover is a convex polygon. We refer the readers to [5] for the references.) The research which is the closest to this one is by Sadhu et al., where the problem is to cover/hit a set of line segments using one or two (congruent) axis-parallel squares with the smallest size (edge length) [10]. Linear time algorithms are given for these problems. Our problems can be considered as the discrete version of these problems, where the centers must line on some input segments. We give $O(n)$ and $O(n^2 \log n)$ algorithms respectively. On the tree model, little is known for the corresponding two-center problem when the target is a set of edges, though the one-center solution (for edges) can be adapted to some forklore algorithm on computing the diameter of a tree in linear time.

This paper is organized as follows: In Section 2, we present some definitions and formally describe the four problems. Then, in Section 3-4 we give details for our solutions for the four problems. We conclude the paper in Section 5.

2 Preliminaries

2.1 Notations and Definitions

Coordinates: For every point $p \in \mathbb{R}^2$, we use $x(p)$ and $y(p)$ to denote its x-coordinate and y-coordinate, respectively.

Endpoints: We use $L(l_i)$, $R(l_i)$, $T(l_i)$ and $B(l_i)$ to denote the left endpoint, right endpoint, top endpoint and bottom endpoint of the line segment l_i $(1 \leq i \leq n)$ where $x(L(l_i)) < x(R(l_i))$ and $y(T(l_i)) > y(B(l_i))$.

Remark: A horizontal line segment has only a left endpoint and a right endpoint with the same y-coordinate, this is similar for a vertical line segment.

Distance: **(I)** In the first model, we use $d_\infty(p, l_i)$ to denote the *distance* between a point $p \in \mathbb{R}^2$ and a line segment l_i $(1 \leq i \leq n)$, and it is defined to be the minimum ℓ_∞-distance between p and some point q on l_i, where $d_\infty(p, q) = \max\{|x(p) - x(q)|, |y(p) - y(q)|\}$ and $|x(p) - x(q)|, |y(p) - y(q)|$ are the horizontal and vertical components respectively (also denoted as $d_h(p, l_i)$ and $d_v(p, l_i)$ as shown red in Fig. 1, where $p = s'$).

(II) In the tree network T, the *distance* between two points p, q (denoted as $d(p, q)$) is the length of the (unique) path between p and q along tree edges. And we denote the *distance* between a point p on an edge of T and a target edge (line segment) l_i by $d(p, l_i)$, which is the minimum distance between p and any point in l_i; formally, $d(p, l_i) = \min_{q \in l_j} d(p, q)$. Hence, $d(p, l_i)$ must be the shortest distance between p and an endpoint of l_i. Also, we use $d(l_i, l_j)$ to denote the *distance* between two line segments l_i and l_j as $d(l_i, l_j) = \min_{p \in l_i, q \in l_j} d(p, q)$,

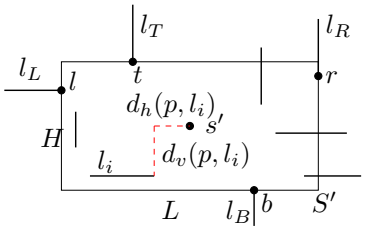


Figure 1: A minimum (unrestricted) rectangle S' hitting all segments.

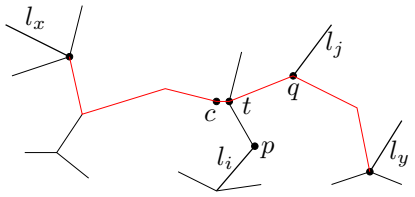


Figure 2: Distance on a tree T: $d(p, q) = d(p, t) + d(t, q)$, and $d(p, l_j) = d(l_i, l_j) = d(p, q)$.

which is the shortest distance between the endpoints of l_i, l_j. (See Fig. 2)

Diameter and radius: The *diameter* D and *radius* R of a set of line segments in a tree network T is defined as follows. For n line segments $U = \{l_1, l_2, ..., l_n\}$ on T, its *diameter* is the longest of the (shortest) path between any two line segments (in Fig. 2 the red path denotes the diameter). And the *diameter* D is also used to denote the length of this path, i.e., $D = \max_{1 \leq i,j \leq n} d(l_i, l_j)$. And naturally the radius is half of diameter $R = D/2$.

2.2 Problems

Throughout this paper, all squares are axis-parallel. Let S be an axis-parallel square with center s and its edge length (or *size*) is ℓ, then S can be defined as $S = \{p | d_\infty(p, s) \leq \ell/2\}$. We say a square S *hits* a segment l_i if there is a point $p \in S$ such that $d_\infty(p, l_i) = 0$. We now define the first two problems on finding one and two hitting squares.

Problem 1 (Discrete One-Hitting-Square): Given a set of axis-parallel line segments $U = \{l_1, l_2, ..., l_n\}$ in $\mathbb{R}^2$, the problem is to find a square S of minimum size such that S hits all the line segments in U and its center s is on a line segment in U. (See Fig. 3)

Problem 2 (Discrete Two-Hitting-Square): Given a set of axis-parallel line segments $U = \{l_1, l_2, ..., l_n\}$ in $\mathbb{R}^2$, the problem is to find two congruent squares S_1 and S_2 of minimum size such that each line segment in U is hit by at least one square and the center s_1 (resp. s_2) of S_1 (resp. S_2) is on a line segment in U.

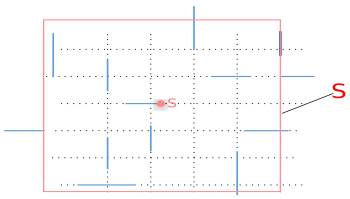


Figure 3: An example for the one hitting-square problem.

We next define the problems on a tree network. Note that in this case a hitting 'square' is virtual.

Problem 3 (One-tree-center): Given a set of edges (line segments) $U = \{l_1, l_2, ..., l_n\}$ on a geometric tree T in $\mathbb{R}^2$, the problem is to find a center c on an edge of T such that the maximum distance from a line segment in U to c is minimized. (See Fig. 2)

Problem 4 (Two-tree-center): Given a set of edges (line segments) $U = \{l_1, l_2, ..., l_n\}$ on a geometric tree T in $\mathbb{R}^2$, the problem is to find two centers c_1 and c_2 on some edges of T such that the maximum distance from a segment in U to the nearer center is minimized.

3 Solutions for Discrete Hitting-Square Problems

3.1 Discrete One-Hitting-Square

We first compute the minimum axis-parallel rectangle S' (with no restriction on its center s') such that all the line segments are hit by S' as in [10]. We then adjust this rectangle by moving s' to s and expanding the edge length to obtain the required square S whose center s must lie on a line segment in U.

To obtain the rectangle S', we present the definition of *boundary line segments* and *boundary points* at first. As shown in Fig. 1, we define four *boundary line segments*: the leftmost segment l_L, the rightmost segment l_R, the topmost segment l_T and the bottommost segment l_B to be the line segments that have *boundary points* l, r, t and b at one of their endpoints, respectively, where l, r, t and b are defined as follows:

$$l = \min_{\forall l_i \in U} x(R(l_i)),\ r = \max_{\forall l_i \in U} x(L(l_i))$$
$$t = \max_{\forall l_i \in U} y(B(l_i)),\ b = \min_{\forall l_i \in U} y(T(l_i))$$

Remark: If a boundary line segment is parallel to the boundary, then any point on it can be recognized as the boundary point since it is fine to hit any point on this line segment.

And we can then construct the rectangle S' by computing its four sides. This is based on the fact that S' hits all the line segments is equivalent to hitting the four boundary line segments l_L, l_R, l_T and l_B.

Now, we focus on how to adjust the rectangle to obtain the desired square. For every line segment l_i, we compute the distance $d_\infty(s', l_i)$, and record the vertical and horizontal components of the distance. Then we compute the expanding lengths γ_i's, i.e., the length that the longer side of the rectangle must be expanded into a square which hits all the segments and whose center lies on l_i. Finally, we choose the minimum γ^* to obtain the target square S.

Without loss of generality, we assume that the horizontal and vertical edge length of the rectangle S' are L and H $(L > H)$ respectively, as shown in Fig. 1.

Lemma 1 *To obtain a feasible square $S(i)$ which hits all the segments in U and whose center lies on some segment l_i, we need to expand the (horizontal) edge length of the minimum hitting rectangle S' by at least γ_i, where*

$$\gamma_i = 2 * \max\{\max\{d_v - \frac{L-H}{2}, 0\}, d_h\}, \qquad (1)$$

and $d_h = d_h(s', l_i)$, $d_v = d_v(s', l_i)$ are the horizontal and vertical components of $d_\infty(s', l_i)$ respectively, with s' being the center of S'.

Proof. As we need to move the center s' of S' horizontally by a value d_h to obtain $S(i)$, the edge length of $S(i)$ would be expanded by at least $2d_h$. (In this proof, imagine that S' is expanded smoothly in both directions, at the same pace.) At the vertical direction, the height of $S(i)$ would not be influenced by d_v if $d_v < \frac{L-H}{2}$. This is because after an expansion by $2d_v$ the height of S' is still shorter than the length. If $d_v > \frac{L-H}{2}$, the edge length of S' would have to be expanded by at least $2(d_v - \frac{L-H}{2})$ to have a feasible $S(i)$ (due to the vertical move of s'). Hence, $\gamma_i = 2 * \max\{\max\{d_v - \frac{L-H}{2}, 0\}, d_h\}$. □

Theorem 2 *The edge length of the smallest discrete hitting square S is $L + \gamma^*$, where $\gamma^* = \min_{\forall l_i \in U}\{\gamma_i\}$; moreover, S can be computed in $O(n)$ time.*

Proof. We first compute γ^*. Then, S can be computed from S' by finding the segment l^* which is γ^* distance away from s', i.e., $d_\infty(s', l^*) = \gamma^*/2$. The point s on l^* realizing $d_\infty(s', l^*) = d_\infty(s', s) = \gamma^*/2$ is the center of S, and the edge length of S is $L + \gamma^*$. □

3.2 Discrete Two-Hitting-Square

It seems hard to solve this 'two-hitting-square' version in the same way because simultaneously moving the two centers of the two hitting rectangles (presumably constructed as in [10]) to the destination is hard, and a brute-force method trying all the $O(n^2)$ combinations of the locations of the two centers s_1, s_2 (which must lie on some segments) would result in a high running time. Hence we use a different method. We try to fix one target square S_1 and then find the other square S_2. And

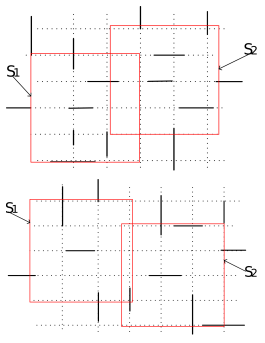


Figure 4: Two configurations for the two centers.

in order to fix S_1, it is natural to consider the corresponding decision problem and then obtain the optimal solution.

For the decision problem, the question is to determine whether there are two congruent (axis-parallel) squares $S_1(\alpha)$ and $S_2(\alpha)$ such that every line segments in U is hit by at least one of them, the center $s_1(\alpha)$ (resp. $s_2(\alpha)$) is on some line segment in U, and the edge length of $S_1(\alpha)$ and $S_2(\alpha)$ is 2α.

As the previous subsection, we find the smallest axis-parallel rectangle S' that hits all the segments in U. We assume that we also have the four *boundary line segments* and the four *boundary points*. Besides, we also assume that the two edge lengths of S' satisfy $L > H$. Then, there are two configurations of $S_1(\alpha)$ and $S_2(\alpha)$ similar to [10], see Fig. 4. Here we only discuss the configuration that $S_1(\alpha)$ (resp. $S_2(\alpha)$) hits l and b (resp. r and t), and the other configuration can be handled symmetrically.

We know that the coordinates of the center $s_1(\alpha)$ must satisfy $x(s_1(\alpha)) \leq x(l)+\alpha$ and $y(s_1(\alpha)) \leq y(b)+\alpha$ since the edge length of $S_1(\alpha)$ is 2α. Then, for every line segment $l_i (1 \leq i \leq n)$, we determine whether $s_1(\alpha)$ can lie on it (to constitute a feasible solution) following Observation 1.

Observation 1 *If $s_1(\alpha)$ is on a horizontal segment l_i, then the left endpoint u of l_i satisfies that $x(s_1(\alpha)) \geq x(u)$. Similarly, if $s_1(\alpha)$ is on a vertical segment l_j, then the bottom endpoint v of l_j satisfies that $y(s_1(\alpha)) \geq y(v)$.*

In fact, Observation 1 implies that, when l_i is horizontal, we could locate $s_1(\alpha)$ on it such that **(A)** $x(s_1(\alpha)) = \min\{x(l)+\alpha, x(R(l_i))\}$ and $y(s_1(\alpha)) = y(l_i)$, where the y-coordinate of l_i is $y(l_i)$. Similarly, when l_j is vertical, we could locate $s_1(\alpha)$ on it such that **(B)** $y(s_1(\alpha)) = \min\{y(b)+\alpha, y(T(l_i))\}$ and $x(s_1(\alpha)) = x(l_i)$, where the x-coordinate of l_i is $x(l_i)$.

Then, the decision procedure is straightforward: we locate $s_1(\alpha)$ (consequently $S_1(\alpha)$) on a candidate segment l_i according to (A) and (B), and for all the segments not covered by $S_1(\alpha)$ we use Theorem 2 to decide whether they can be covered by $S_2(\alpha)$ in $O(n)$ time. As there are n candidate segments l_i's, the decision procedure takes $O(n^2)$ time.

For the optimization problem, notice that the optimal solution value α^* must be in the form $d_\infty(l_i, l_j)$ or $d_\infty(l_i, l_j)/2$ (the corresponding optimal squares have an edge length $2d_\infty(l_i, l_j)$ or $d_\infty(l_i, l_j)$ respectively). Hence we can compute and sort this list of distances in $O(n^2 \log n)$ time. Then, we just use the decision procedure to perform a binary search to find α^* in $O(n^2 \log(n^2)) = O(n^2 \log n)$ time. Consequently, $S_1 \leftarrow S_1(\alpha^*), S_2 \leftarrow S_2(\alpha^*)$.

Theorem 3 *The Discrete Two-Hitting-Square problem can be solved in $O(n^2 \log n)$ time.*

4 Solution for Hitting the Line Segments on a Tree

In this section, we consider the problems on hitting a set of n segments on a tree T. We assume that T contains m edges, with $m > n$. Here a segment l_i is *hit* by a center c on T if $d(c, l_i)$ is bounded from above by some value β. Our problems are to hit all target segments with either one or two centers such that the corresponding β is minimized (Fig. 2).

4.1 One-tree-center

We present the algorithm to find c in the following algorithm. This algorithm is adapted from a folklore algorithm on computing the diameter of a tree.

1. Arbitrarily choose a node r_1 in the tree T as the root and find the line segment l_x that is the farthest from r_1 by breadth-first-search on T. Let $d(r_1, l_x) = d(r_1, x)$, where x is an endpoint of l_x.

2. Find the farthest line segment l_y from x by breadth-first-search on T. Let $d(x, l_y) = d(x, y)$, where y is an endpoint of l_y.

3. Compute the path between x and y as the diameter. The center c is the midpoint on the path between x and y (e.g., l_x and l_y).

Theorem 4 *The One-Tree-Center problem can be solved in $O(m+n)$ time; in fact, the optimal center c is just the midpoint of the diameter D; formally, $D = d(l_x, l_y) = \max_{1 \leq i,j \leq n} d(l_i, l_j)$ and c is on the path between l_x and l_y such that $d(c, l_x) = d(c, l_y)$.*

Proof. The correctness can be proved by contradiction. The details will be given in the full paper. The running time of the algorithm is obviously $O(m+n)$ as the main cost is two runs of the breadth-first-search algorithm [3]. □

4.2 Two-Tree-Center

In this problem, the objective is to find two centers c_1 and c_2 on the tree T such that

$$f \stackrel{def}{=} \max_{l_i \in U} \min\{d(c_1, l_i), d(c_2, l_i)\}$$

is minimized for any line segment l_i on the tree T.

To make our analysis more clean, we initially take c as a virtual root of the tree T and then perform some preprocessing, i.e., remove all the subtrees that do not contain target line segments and denote the position of every line segment by its endpoint (node) that is closer to c. Thus, every leaf node is the endpoint of a line segment in the transformed tree (we still call it T), and we abuse the terminology by calling these line segments as *leaves*.

For the sake of brevity, we use the notation f_1 (resp. f_2) to denote the distance between c_1 (resp. c_2) and the farthest line segment it hits. Thus, $f = \max\{f_1, f_2\}$.

In this subsection, we propose the algorithm first and sketch its correctness a bit later. We first implement the same algorithm as we did in the last subsection to obtain $l_x, l_y \in U$ and c. (Recall that $d(l_x, l_y)$ gives the diameter of the segments in U on T.) Then, we discuss the next steps in the following two cases:

(1) c is not a node of T, i.e., c is between two adjacent nodes in T. Cut T into two parts T' and T'' at c such that l_x, l_y are contained in T', T'' respectively. We can find the center c' (resp. c'') of T' (resp. T'') as done in Section 4.1. In Section 4.2.1 we give details to show that c' and c'' are just the two centers c_1 and c_2 of T' and T'', respectively, and $f = \max\{d(c', l_x), d(c'', l_y)\}$.

(2) c is exactly a node of T. In this case, there are two subcases to be discussed:

(2.1) There are exactly two subtrees (also denoted by T' and T'') of c: One contains l_x while the other contains l_y. Without loss of generality, it is assumed that c is contained in T' but not in T''. We can also compute c' and c'' similar to (1) and they are also the two centers of T', T''. And in this case it can be computed that $f = D/4 = d(l_x, l_y)/4$.

(2.2) There are more than two subtrees of c, denoted by $T^1, T^2, ..., T^k$, respectively. (Suppose c does not belong to any subtree.) Let the two subtrees that contains l_x and l_y be T^1 and T^2 respectively. Compute the centers c^1, c^2, c^{-1}, c^{-2} and radii R^1, R^2, R^{-1}, R^{-2} of T^1, T^2, T^{-1}, T^{-2}, where $T^{-1} = T^2 \cup T^3 \cup ... \cup T^k \cup c$ and $T^{-2} = T^1 \cup T^3 \cup ... \cup T^k \cup c$. In this case, $f = \min\{\max\{R^1, R^{-1}\}, \max\{R^2, R^{-2}\}\}$. We obtain c_1 and c_2 respectively as c^1 and c^{-1}, if $\max\{R^1, R^{-1}\} \leq \max\{R^2, R^{-2}\}$; and vice versa. In summary we have the following theorem.

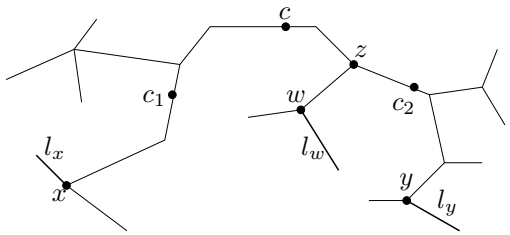


Figure 5: Illustration for the proof of Lemma 9, under the assumption that l_w can be hit by c_1 but cannot be hit c_2.

Theorem 5 *The Two-Tree-Center problem can be solved in $O(m+n)$ time.*

We next give some details for the above theorem, due to space constraints, we leave out some details for the final version of this paper.

4.2.1 Case 1. c is not a node of the tree T

For this case, we have the following properties which are intuitively obvious. Due to space limit, the proofs are omitted in this version.

Observation 2 *l_x and l_y must be two leaf nodes of T and are the farthest nodes from c.*

Lemma 6 *The two centers c_1 and c_2 of T must be in T' and T'' respectively.*

Observation 3 *l_x and l_y in T are hit by c_1 and c_2 respectively.*

Lemma 7 *l_x and l_y are the farthest line segments hit by c_1 and c_2 respectively, i.e., $d(c_1, l_x) = f_1$ and $d(c_2, l_y) = f_2$.*

Lemma 8 *c_1 (resp. c_2) is on the path between c and l_x (resp. l_y).*

Lemma 9 *In an optimal solution, even if there is a line segment l_w in T'' which is hit by c_1, i.e., $d(c_1, l_w) \leq f_1$, we can make a swap to use c_2 to hit it without making the solution worse. Similarly, even if there is a line segment l_v in T' which is hit by c_2, we can make a swap to hit l_v with c_1.*

Proof. Due to space limit, we only give a sketch of the proof, see Fig. 5. Assume that l_w in T'' is hit by c_1, i.e., $d(c_1, l_w) \leq f_1$, but cannot be hit by c_2, i.e., $d(c_2, l_w) > f_2$. We have $f = f_2 \geq f_1$, but we can show $f_1 > f_2$ to lead the needed contradiction. □

Corollary 1 *In an optimal solution of the Two-Tree-Center problem, c_1 hits all the line segments in T' and c_2 hits all the line segments in T''. Thus we can find c_1 (resp. c_2) by solving the One-Tree-Center problem on T' (resp. T'').*

4.2.2 Case 2. c is a node of the tree T

(2.1) *There are exactly two subtrees T' and T'' of c.* It is easy to see that c_1, c_2 are in T', T'' and hit l_x, l_y respectively, similar to Lemma 6 and Observation 3. Moreover, at least one of c_1 and c_2 hits l_x and c (or, l_y and c) simultaneously. (Otherwise the solution is not optimal.) Assume that c_1 hits both l_x and c, then $f_1 = d(l_x, l_y)/4 = D/4$ and f_2 cannot be greater, because l_x and l_y are the two line segments farthest from c. Hence, $f = \max\{f_1, f_2\} = \max\{d(c_1, l_x), d(c_2, l_y)\} = d(c_1, l_x) = D/4 = d(l_x, l_y)/4$.

(2.2) *There are more than two subtrees of c: $T^1, T^2, ..., T^k$.* Assume that l_x and l_y are in T^1 and T^2, respectively. We first claim that c_1 and c_2 must be in T^1 and T^2, respectively. (Otherwise, one of T^1 and T^2, say it is T^1, does not contain any center; thus $f = \max\{d(c_1, l_x), d(c_2, l_x)\} > d(c, l_x) = D/2$ which is even worse than the corresponding one-center solution. A contradiction.)

In this case we can also prove that $d(c_1, l_x) = f_1$, $d(c_2, l_y) = f_2$, and c_1 (resp. c_2) is on the path between c and l_x (resp. l_y) similar to Lemma 7 and Lemma 8. Thus we can also obtain the conclusion that all the line segments in T^1 (resp. T^2) are hit by c_1 and c_2, respectively, as in Lemma 9 and Corollary 1. Now we only need to consider the line segments in $T^3, ..., T^k$: Assume that the tree containing the farthest line segment from c other than T^1 and T^2 is T^3, and suppose that c_1 hits all the line segments in T^3. When we compute f_1 for c_1 to hit all the line segments in T^1 and T^3, it is obvious that all the line segments in $T^4, ..., T^k$ can also be hit by c_1 without increasing f_1. That is to say, the line segments in $T^1, T^3, ..., T^k$ are all hit by c_1. Similarly, if all the line segments in T^3 are hit by c_2, then all the line segments in $T^2, T^3, ..., T^k$ are hit by c_2. Hence we obtain the conclusion that either **(a)** c_1 hits all the line segments in T^1 and c_2 hits all the line segments in $T^2, T^3, ..., T^k$, or **(b)** c_1 hits all the line segments in $T^1, T^3, ..., T^k$ and c_2 hits all the line segments in T^2. Thus, $f = \min\{\max\{R^1, R^{-1}\}, \max\{R^2, R^{-2}\}\}$, and c_1 and c_2 can be computed accordingly. This concludes the correctness proof of Case 2.

5 Concluding Remarks

An extension of this research is to use a more realistic model, i.e., an irregular grid network (a grid network with some edges randomly deleted, e.g., something similar to a wall graph). It seems to take some effort to solve the discrete two-center problem in roughly $O(m^2)$ time or even better, where m is size of the network and there are $n (n < m)$ streets/segments to cover.

Acknowledgments

This research is partially supported by NNSF of China under project 61628207. XH is supported by China Scholarship Council under program 201706240214 and by the Fundamental Research Funds for the Central Universities under project 2012017yjsy219. ZL is supported by a Shandong Government Scholarship.

References

[1] P. Agarwal, M. Sharir and E. Welzl. The discrete 2-center problem, *Discrete and Computational Geometry*, 20(3):287-305, 1998.

[2] T. Chan. More planar two-center algorithms, *Computational Geometry: Theory and Applications*, 13(3):189-198, 1999.

[3] T. Cormen, C. Leiserson, R. Rivest and C. Stein. *Introduction to Algorithms*, second edition, MIT Press, 2001.

[4] Z. Drezner. On the rectangular p-center problem, *Naval Research Logistics*, 34(2):229-234, 1987.

[5] H. Du and Y. Xu. An approximation algorithm for k-center problem on a convex polygon, *J. Combinatorial Optimization*, 27(3):504-518, 2014.

[6] D. Eppstein. Faster construction of planar two-centers, In *Proc. 8th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'97)*, pages 131-138, 1997.

[7] M. Hoffman. A simple linear algorithm for computing rectilinear 3-centers, *Computational Geometry: Theory and Applications*, 31(3):150-165, 2005.

[8] M. Katz, K. Kedem and M. Segal. Discrete rectilinear 2-center problem, *Computational Geometry: Theory and Applications*, 15(4):203-214, 2000.

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

[10] S. Sadhu, S. Roy, S. Nandy and S. Roy. Optimal covering and hitting of line segments by two axis-parallel squares, In *Proc. 23rd International Computing and Combinatorics Conference (COCOON'17)*, LNCS 10392, pages 459-468, 2017.

[11] M. Sharir. A near-linear algorithm for the planar 2-center problem, *Discrete and Computational Geometry*, 18(2):125-134, 1997.

[12] E. Welzl. Smallest enclosing disks (balls and ellipsoids), In *New Results and New Trends in Computer Science* (Ed. H. Maurer), LNCS 555, pages 359-370, 1991.

Finding Intersections of Algebraic Curves in a Convex Region using Encasement

Joseph Masterjohn* Victor Milenkovic† Elisha Sacks‡

Abstract

We present a subdivision based technique for finding the intersections of two algebraic curves inside a convex region. Even though it avoids computing resultants, the technique is guaranteed to find all intersections with bounded backwards error. The subdivision, called an *encasement*, also encodes the arrangement structure of the curves. We implement the encasement algorithm using adaptive precision interval arithmetic. We compare its performance to the CGAL library implementation of resultant based curve intersection techniques. We provide CPU and CPU/GPU versions of the algorithm and implementation. On the CPU, encasement generates all curve intersections, to accuracy 10^{-8}, 10 to 30 times faster than CGAL for degrees 8 to 18, and it handles degrees up to 20 that CGAL cannot handle. The GPU speeds up the calculation by a factor of 3 to 4.

1 Introduction

An algebraic curve f is the zero set of a bivariate polynomial $f(x, y)$. Given a convex polygon B, we find all intersections of curves f and g inside B. Curve intersection is a core geometric calculation. It is a key step in calculating the *arrangement* of n curves $f_1, \ldots, f_n$: a partition of B into intersection vertices, open curve segments, and open regions. We have in mind scientific or industrial applications, which provide the polynomial coefficients as floating-point numbers. Broadly speaking, numerical programs use double-float for calculations and aim for single-float accuracy in the output. An arrangement with accuracy $\delta = 10^{-8}$ would more than satisfy the latter. Nevertheless, exact arrangement computation is required to support CG algorithms that manipulate arrangements. These algorithms require the signs of predicates evaluated on the vertices of the arrangement. An incorrect sign can lead to program failure or to nonsensical output. This is the robustness problem of Computational Geometry.

*Department of Computer Science, University of Miami, joe@cs.miami.edu

†Department of Computer Science, University of Miami, vjm@cs.miami.edu

‡Computer Science Department, Purdue University, eps@cs.purdue.edu

Exact Computational Geometry (ECG) uses extended precision and algebraic algorithms to determine the signs of primitives. This approach can be numerically expensive even when heuristics are used, such as floating point filtering. In the case of curve arrangements, an exact algorithm requires construction of resultant polynomials. These have high degree and bignum coefficients.

Numerical methods, such as subdivision and curve tracing, are often stymied by ill-conditioned inputs. Usually, the subdivision is by axis-parallel lines, which can require a large number of cuts to separate features. Curve tracing is faster but is even less reliable.

1.1 Prior Work

Algebraic methods compute the turning points and the intersection points of bivariate polynomial curves via resultants and other algebraic computation. For example, the CGAL arrangement package [5, 15] implements a sweep algorithm for plane algebraic curves using Exacus [4].

Subdivision methods [7, 1] provide a faster means to isolate the intersection points of algebraic curves and to trace algebraic curves, but they cannot guarantee correctness and are prone to failure on ill-conditioned inputs. They use convex bounding polyhedra during intersection isolation, but the outputs are axis-parallel enclosures. They typically operate on polynomials given in the Bernstein-Bézier basis and involve a non-robust numerical subdivision phase followed by a robust (no false positive) certification phase on candidate intersections [9]. They can isolate the vertices and edges of an arrangement, but an axis-parallel enclosure requires $\Omega(1/\epsilon)$ cells for ϵ-separated curves (ϵ distant under the Hausdorff metric). Other work focuses on improving the efficiency rather than reliability of the subdivision phase through the use of low degree approximations [3], blending schemes for quick elimination of regions containing no roots [2], and deflation techniques [13].

Wang, Chiang, and Yap [14] formalize resolution-exact subdivision methods for motion planning, but this work is also limited to axis-parallel enclosures.

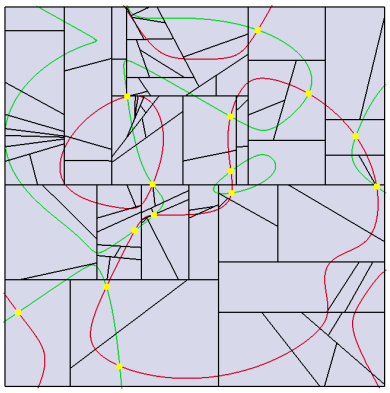


Figure 1: Encasement of f (red) with respect to g (green) and their intersections (yellow) in B (bounding square).

1.2 Encasement-Based Intersection Construction

We present an algebraic curve intersection algorithm based on *convex encasement*. The algorithm combines subdivision methods with exact computational geometry to achieve both efficiency and guaranteed accuracy.

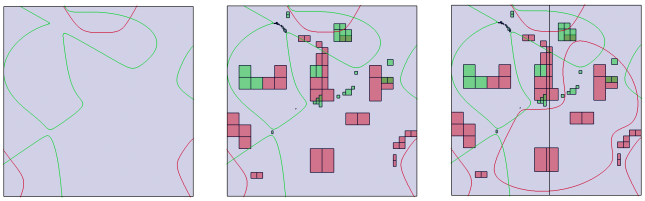
Algebraic curves f and g are generic (in general position) if they are nonsingular (i.e. no solution to $f(x,y) = f_x(x,y) = f_y(x,y) = 0$) and have no tangent intersections. A *convex encasement* of f with respect to g in a convex polygonal region B is a partition of B into convex polygonal cells such that 1) no cell contains a loop of f or g or more than one segment of f, 2) if a cell intersects both f and g, it contains a single intersection point of f and g (Fig. 1).

An encasement isolates the components of f and the intersections of f and g. The arrangement in B of a set of curves F can be reduced to the encasement in B of each pair f, g from F (Sec. 7.1).

1.2.1 Intersection Algorithm Summary

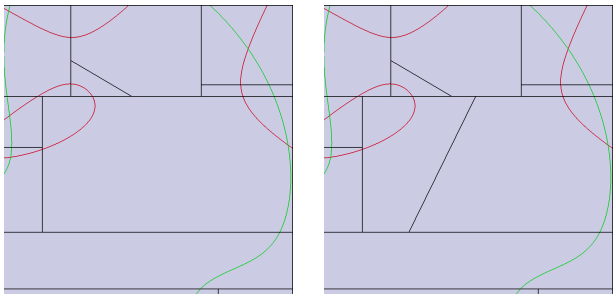
The curve intersection algorithm (Sec. 7) takes two bivariate polynomials as inputs, perturbs the coefficients by $\delta = 2^{-26} \approx 10^{-8}$, and constructs an encasement of the corresponding generic algebraic curves f and g by recursive subdivision of B by straight lines. If a cell C violates the definition of encasement, for example by containing a loop of f, the algorithm splits C by a line L. The following summarizes the selection of L with details in the indicated sections.

Loop splitting (Sec. 2) Construct a *critical set* S for f in B: S does not intersect f and contains all local extrema of $f(x,y)$. Since a loop of f must surround an extremum, it must surround a connected component of S. If a cell C contains a connected component of S, L is selected to intersect it and hence splits any loop surrounding it (Fig. 2). Likewise g.



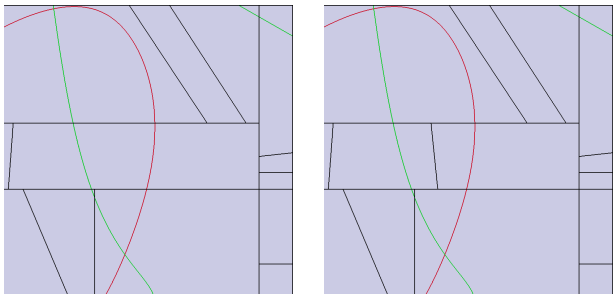
(a) (b) (c)

Figure 2: Curves f (red) and g (green) with undetected loop not shown (a). Critical regions (b) (colors match curves). Splitting lower middle f-region with vertical line reveals and splits missing loop (c).



(a) (b)

Figure 3: Cell with two segments (red) of f and one segment (green) of g (a). Segments of f separated by splitting line (b).



(a) (b)

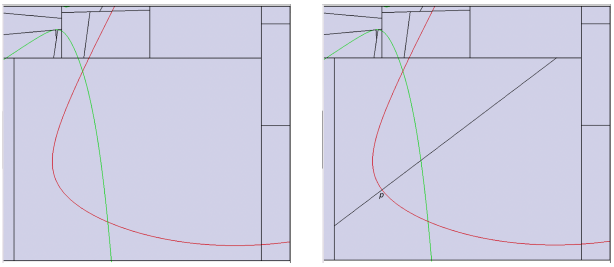
Figure 4: Cell (center) with non-intersecting segments of f and g (a). Separated by splitting line (b).

Self-separation (Sec. 3) If C contains more than one segment of f, L is selected to separate one segment from another (Fig. 3).

Curve separation (Sec. 4) If f has a single segment ab in C and no intersections with g, L separates f from g (Fig. 4).

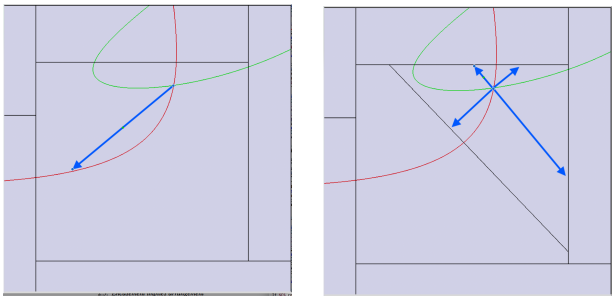
Intersection separation (Sec. 4) If ab intersects g an even number of times inside C, L splits C between two of the intersections (Fig. 5).

Intersection isolation and encasement (Secs. 5 and 6) If ab intersects g an odd number of times in C, we construct an axis-parallel rectangle $R \subset C$ containing a



(a) (b)

Figure 5: Cell with two intersections (a). Separated by splitting line through $p \in f$ in direction of $\nabla f(p)$ at a local minimum of $g(p)$ on f (red) (b).



(a) (b)

Figure 6: Segment g (red) blocks the intersection's "view" of boundary along angle bisector (a). Angle bisectors reach boundary of smaller cell proving intersection is unique (b).

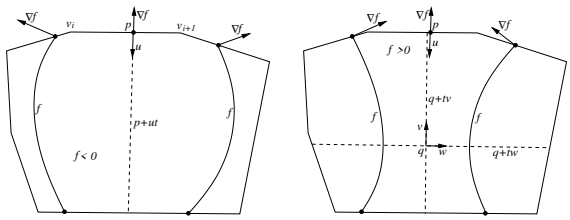
single intersection. This is standard zero isolation using a 2D interval: R is not a cell. Split C by up to four lines to encase that intersection in a cell that excludes all other intersections (Fig. 6).

Self-separation and curve separation might not be possible using a single split if the two segments are close and curved. In that case, multiple splits are required.

1.2.2 Contribution

Encasement based curve intersection improves on prior work in several ways. We ensure correctness, with accuracy δ, for all inputs by perturbing polynomials to remove singular points and tangent intersections then employing adaptive precision interval arithmetic. Replacing boxes with convex polygons reduces the space complexity for ϵ-separated curves to $\Omega(1/\sqrt{\epsilon})$. The number of splits, other than for self-separation or curve separation, is in $O(d^2)$, for d the maximum degree of f and g. We introduce a stronger criterion for showing that a cell contains a single intersection.

On the CPU, encasement generates all curve intersections, to accuracy δ, 10 to 30 times faster than CGAL for degrees 8 to 18, and it handles degrees up to 20 that CGAL cannot handle. The GPU speeds up the calculation by a factor of 3 to 4.



(a) (b)

Figure 7: p minimizes $f(p)$ on v_iv_{i+1}, $f(p) < 0$, and direction u decreases $f(x, y)$. Split by $p + ut$ (a). p maximizes $f(x, y)$ and $f(p) > 0$ but direction u decreases $f(x, y)$. If this happens for every edge, C must contain a saddle point q. Split by $q + tv$ and $q + tw$, where v and w are the eigenvectors of the Hessian of $f(x, y)$ at q (b).

2 Critical sets

A region S is a *critical set* of a curve f with respect to B if it does not intersect f and it contains all local extrema of $f(x, y)$ in B. To construct a critical set, let R be the bounding rectangle of B. If we can show that $f(x, y)$ is nonzero in R, return R. If we can show that one of the partial derivatives $f_x(x, y)$ or $f_y(x, y)$ is nonzero in R (hence R does not contain an extremum), return $\emptyset$. Otherwise, bisect R across its longer dimension, recurse on the two halves, and return the the union of the results. Since $f(x, y)$ is nonsingular, the algorithm terminates.

We test if a polynomial is nonzero on a rectangle with a generalization of Descartes' rule of signs. If $R = [a, b] \times [c, d]$, the rational function $g(x, y) = f(1/(x + 1/(b - a)) + a, 1/(y + 1/(d - c)) + c)$, takes on the same set of values on $[0, \infty] \times [0, \infty]$ as $f(x, y)$ on R, and $g(x, y)$ is nonzero on $[0, \infty] \times [0, \infty]$ (hence $f(x, y)$ on R) if all the coefficients of $x^m y^n g$ have the same sign, where m and n are the degrees of $f(x, y)$ in x and y.

3 Self-separation

We can find the intersections of a curve f with the boundary of a cell C by substituting the parametric form $v_i + t(v_{i+1} - v_i)$ of each edge v_iv_{i+1} into $f(x, y)$ and solving for the zeros of the univariate in $t \in [0, 1]$ [11]. Since f has no loops in C after loop splitting, the number of segments of f inside C is half the number of intersections with the boundary. If there are more than two intersections, we split C in a manner that partially or completely separates at least one pair of segments.

For each clockwise oriented edge v_iv_{i+1} of the boundary, solve for all p such that $(v_i - v_{i+1}) \cdot \nabla f(p) = 0$

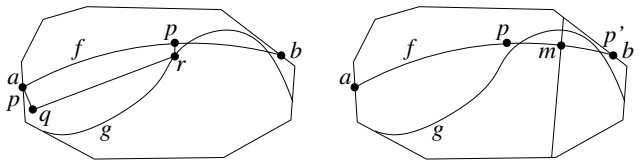


Figure 8: Chain from a (initial p) to q to r to p separates ap from g (a). $g(x, y)$ is decreasing at p and p' $(= b)$ towards b and a, so we split at minimum m of $g(x, y)$ on f, separating two intersections with g (b).

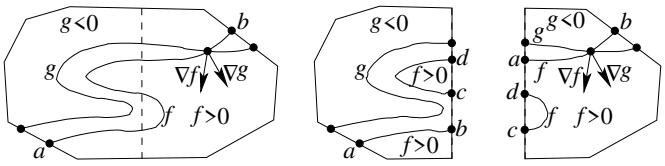


Figure 9: a is a positive tail of f but b is a negative head (a), so $i = -1 = \text{sign}(\nabla f(p) \times \nabla g(p))$ (a). a and d are positive tails and b and d are positive heads $i = 0$ (b). a and c are positive tails, b is negative head, and d is positive head so $i = -1$ (c).

(Fig. 7(a)). The vector $u = s\nabla f(p)$ with $s = \text{sign}((v_i - v_{i+1}) \times \nabla f(p))$ points inward. If $\text{sign}(f(p)) = s$, $|f(p + tu)|$, $t > 0$, increases at $t = 0$. We split by the line $p + tu$. If there is more than one such p, we choose the one between the closest pair of boundary intersections. If no such p exists on any edge, we claim that $\nabla f(p)$ makes at least one full counterclockwise turn as p traverses the boundary clockwise. This claim is a specialization of the generalized Poincaré-Hopf index theorem [8]. We isolate an intersection q of f_x ($f_x(x, y) = 0$) and f_y in a rectangle with the same property (Sec. 5), which implies that q is a saddle point (Fig. 7(b)). Let v and w be principle directions of f at q. We split by the lines $q + tv$ and $q + tw$.

4 Curve or intersection separation

If a cell C contains a single segment ab of f and $\text{sign}(g(a)) = \text{sign}(g(b))$, f crosses g an even number of times inside C. If there is a local minimum m of $g(x, y)$ on f, we expect that it separates two intersections, so we split at m. Some minima may not separate pairs of intersections, but there are at most $O(d^2)$ minima for d the maximum total degree of $f(x, y)$ and $g(x, y)$. Otherwise, we try to certify zero intersections by constructing a splitting line that separates f from g. The details of curve/intersection separation are complicated. We provide a summary here. Details are in a forthcoming full paper. We discuss separating f from g in terms of constructing a polygonal chain, but actually we split along the lines of the segments in the chain.

Suppose we are at a point $p \in f$, initially $p = a$. We have separated ap from g. Specifically, ap does not intersect a segment of g with both endpoints to the right of ab. (The left has to be handled similarly.) The sign of $\nabla f(p) \times \nabla g(p)$ tells us that $g(x, y)$ is increasing at p in the direction of b. We move away from f in the direction of $\nabla f(p)$ to q halfway to g, meaning $g(q) = g(p)/2$. Next we move in a direction perpendicular to $\nabla f(q)$, "parallel" to f. If we hit f first, that is the new position of p. If we hit g or the boundary of C at r, we drop back to f in the direction opposite of its gradient to the new p on f, with ap separated from g (Fig. 8(a)). Since the parallel move is off f and parallel to it, it goes far before hitting f. Since $g(x, y)$ is increasing in the direction of b, g is getting farther from f in that direction so the parallel move goes far before hitting g.

If $g(x, y)$ is decreasing on f at p in the direction of b, we try to work from the opposite direction, starting with $p' = b$. If $g(x, y)$ is decreasing at both p and p' in the direction of b and a, $g(x, y)$ has a local minimum on f between p and p'. We isolate the minimum m, which is an intersection between f and $\nabla f(x, y) \times \nabla g(x, y) = 0$ (Sec. 5). Then we split f by a line through m in the direction $\nabla f(m)$ (Fig. 8(b)). If there is a pair of intersections with g between p and p' and only one minimum, this will put the two intersections in different cells. If not, m becomes a new starting point for separations because $g(x, y)$ is increasing on f in both directions away from m.

5 Intersection isolation

If $\text{sign}(g(a)) = -\text{sign}(g(b))$, f intersects g an odd number of times inside C, and we isolate one of these intersections to a rectangle $R \subset C$. Let P be the precision of the arithmetic: initially double-float ($P = 53$). Isolation uses two operations: subdivide(D) subdivides a convex region D containing an intersection by the bisector of its longer dimension and returns the half D' containing an intersection (Sec. 5); and Newton(R) iterates 2D Interval Newton's method [12] on a rectangle R until it stops shrinking.

While $0 \in \nabla f(\text{bbox}(D)) \times \nabla g(\text{bbox}(D))$ or $\text{Newton}(\text{bbox}(D)) = \text{bbox}(D)$, $D \leftarrow \text{subdivide}(D)$. Return $\text{Newton}(\text{bbox}(D))$. Isolation does not alter C: the subdivisions are temporary. The output R can be made smaller by doubling the P used to create it and returning Newton(R). We speed up the method by running ordinary Newton's method on each cell centroid. If it converges to a point inside the cell, we expand it to a rectangle based on its condition.

Subdivision might result in one or both halves containing more than one segment of f. We can tell which half contains the intersection by examining the inter-

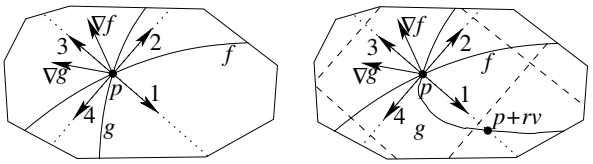


Figure 10: Each angle bisector of the gradient vectors can "see" the boundary (a). Split the cell 90% of the way to the nearest intersection (b) or boundary. Boundary splits are dropped if not needed.

sections of f. A *tail* is a point $a \in f \cap \partial C$ such that $f > 0$ in a neighborhood counterclockwise from a. If a is on edge $v_i v_{i+1}$ of the boundary, the condition is equivalent to $(v_{i+1} - v_i) \cdot \nabla f(a) < 0$. A *head* has the opposite sign. We say $a \in f \cap \partial C$ is *positive* if $g(a) > 0$, otherwise *negative*. A cell contains an intersection if the *intersection number* i of positive heads minus positive tails is nonzero (Fig. 9(a)). This number also equals the winding number of $(f(p), g(p))$ around the origin as p travels around the boundary: $(f(p), g(p))$ sweeps counterclockwise through the first quadrant for each positive head and clockwise for each positive tail. For at least one $p \in f \cap g \cap C$, $\text{sign}(\nabla f(p) \times \nabla g(p)) = \text{sign}(i)$. Subdivision calculates the intersection number for each half and selects the one whose sign is the same as the original cell (Fig. 9(c)).

6 Intersection encasement

The output R of intersection isolation is an adaptive-precision 2D interval representation of an intersection point p of f and g inside a cell C. However, C can contain an even number of additional intersections. Intersection encasement uses up to four splits to isolate p within a cell that excludes all other intersections.

Let v_1, v_2, v_3, v_4 be vectors that bisect the angle between $\pm\nabla f(p)$ and $\pm\nabla g(p)$. If g also intersects C in a single segment and if the four rays $p + tv_i$, $t > 0$, reach the boundary of C without intersecting f or g, the four curve segments connecting p to the boundary of C via f or g are isolated, and f and g have no other intersection in C (Fig. 10(a)).

Otherwise, for each $1 \le i \le 4$, compute $t = r_i > 0$ the minimum value at which $p + tv_i$ intersects f, g, or the boundary of C, and split C by the line perpendicular to v_i through the point $p + 0.9 r_i v_i$ (Fig. 10(b)). While f or g intersects the boundary of the cell containing p more than twice, halve each r_i and split again.

7 Encasement based intersection algorithm

The encasement based algebraic curve intersection algorithm takes two bivariate polynomials, $f(x, y)$ and $g(x, y)$, and a desired accuracy δ as input and perturbs their coefficients by η uniform in $[-\delta, \delta]$. When generating a splitting line, it rounds its coefficients to double-float and perturbs them. However, if perturbation puts the line on the wrong side of a vertex, it expresses each coefficient as the sum of two double-floats and perturbs the smaller one, and so forth as necessary. It uses interval arithmetic, increasing precision [6] as necessary to correctly determine signs of predicate expressions.

Separation of curves might require multiple splits, but the separation algorithms are correct for linear curves, and each split shrinks the. Since the curves are generic, their deviation from linear also shrinks, ensuring termination [10].

7.1 Encasement implies arrangement

Using encasement of pairs of curves, we can construct an arrangement of n curves inside B. Given curves $F = \{f_1, f_2, \ldots, f_n\}$, calculate intersections of all pairs. For each $f \in F$, calculate its intersections with the partial derivatives f_x and f_y. Starting with B, add intersections of f with other curves sequentially. First add intersections with f_x and f_y. If a cell contains two intersections, split it with a horizontal or vertical line. After adding the intersections with f_x and f_y, apply self-separation of f. Each cell now contains an x or y-monotonic segment of f and hence the cell can be split with a vertical or horizontal line without creating a cell with more than one segment. Add the remaining intersections of f with other curves, splitting vertically/horizontally as appropriate. The result is the *intersection encasement* $\mathcal{I}(f)$ of f in B with respect to F.

An arrangement segment is a segment of f connecting two cell boundary intersections, in a cell $C \in \mathcal{I}(f)$ not containing an intersection, or a segment ap connecting a boundary intersection a to a curve intersection $p \in C$ with g. To trace the boundary of an arrangement cell, we need to take a "left turn" at p to the segment pc or pd of g in its encasement. The choice is determined by the the sign $\text{sign}(\nabla f(p) \times \nabla g(p))$, a byproduct of isolating the intersection (Sec. 5). Hence the arrangement cells can be traced using only information stored in the intersection encasements.

7.2 GPU speedup

The GPU version subdivides B (or its bounding box if it is not a rectangle), into rectangular cells and assigns the task of showing f or g has no zeros on a cell C to a thread. Cells which fail this test are subdivided. This process stops when the number of failing cells stabilizes. CPU based encasement is run on each resulting cell. Details in full paper.

8 Results

The first set of experiments uses random curves of degree d from 3 to 20. We use $B = [-1, 1] \times [-1, 1]$. To generate a curve, we select $d(d+1)/2 - 1$ points at random in B and interpolate through them. For each degree d, we generate a set F_d of 16 curves. The test is to generate all the intersections of every pair of curves in F. We compare the CGAL curve arrangement library with the CPU and CPU/GPU versions of encasement. For CGAL, we monotonize each curve once, compute the arrangement of every pair, and then calculate each vertex in double-float. For CPU encasement, we generate the critical regions for each curve once, calculate the encasement for each pair of curves, and increase the precision P until the interval contains at most one double-float point. For the GPU algorithm, we use an initial subdivision small enough to ensure that 90% of subcells are eliminated. The CPU results use an Intel Core i5-3570K over-clocked at 4.2GHz and 8GB RAM. The GPU results in addition use an Nvidia GTX 780 with 4GB DRAM. Results are in Table 8.

For $d > 10$, the CPU version of encasement is about 30 times faster than CGAL. CGAL times out for degrees greater than 18. For degrees up to 13, using the GPU speeds up encasement by a factor of 3. At degree 20, there is no benefit.

The three right columns of Table 8 help to analyze the number of splits required for encasement. Isolating i intersections requires at least i splits. The number of splits is almost proportional, rising slowly from $5i$ up to $7.74i$ for $d = 3$ to $d = 22$.

For the second experiment, we tested the robustness of encasement and the cost of encasing near degenerate cases. We generated pairs of curves with a tangent intersection, which is perturbed to a near-tangency. Table 8 shows the effect of the tangent intersection. Since i ordinary intersections require about $5i$ to $7i$ faces to encase, it appears that a tangency requires about 50 to 60 faces to encase. Since the perturbation is 2^{-26}, this is proportional to the number of bits of accuracy, which is still a very reasonable number.

d	CGAL	CPU	GPU	I	S	S/I
3	0.05	0.01	0.14	5	25	5.0
4	0.16	0.03	0.15	11	49	4.4
5	0.33	0.07	0.17	15	65	4.3
6	0.67	0.16	0.18	16	99	6.1
7	1.56	0.38	0.23	23	179	7.7
8	8.29	0.74	0.30	30	200	6.6
9	17.06	1.56	0.41	35	267	7.6
10	32.65	1.99	0.48	47	291	6.1
11	54.62	2.68	0.89	57	432	7.5
12	119.53	3.28	1.09	59	440	7.4
13	161.72	5.01	1.66	74	542	7.3
14	178.71	8.76	2.40	76	509	6.6
15	367.97	9.35	3.72	95	727	7.6
16	418.51	13.22	5.87	100	743	7.4
17	597.84	19.76	9.00	114	951	8.3
18	881.81	28.89	15.72	135	1062	7.8
19	∞	33.09	17.81	130	1010	7.7
20	∞	43.28	38.86	151	1168	7.7

Table 1: Degree d, CGAL, CPU encasement, and GPU/CPU encasement running times in seconds, number of intersections I, number of cell/line splits in the resulting encasement S, and ratio of S/I.

9 Conclusion

Although it uses perturbation, encasement is an exact algorithm, hence correct. The perturbation adds a controllable backwards error. The choice $\delta = 2^{-26} \approx 10^{-8}$ randomizes half the bits of the input, which makes it generic with high probability. For most applications, a 10^{-8} error is an acceptable price for a 10 to 30 times improvement in running time. The GPU is consumer grade, and so it has an acceptable price for an additional factor of 3 in running time.

We were hoping for more speed up from using a GPU, but the current version uses a quadratic approximation to $f(x, y)$, at a cost of d^2, instead of expanding $f(1/(x + 1/(b - a)) + a, 1/(y + 1/(d - c)) + c)$ (Sec. 2), which has d^3 complexity. Also, it is limited to axis-parallel subdivision.

Another goal of this research is 3D surface intersections and arrangement. We believe subdivision by non-axis-parallel planes will be similarly beneficial.

Acknowledgments

Masterjohn and Milenkovic are supported by NSF grant CCF-1526335. Sacks is supported by NSF grant CCF-1524455.

d	time	I	S	S/I
3	0.02	3	67	22.3
4	0.03	4	70	17.5
5	0.05	4	61	15.3
6	0.09	1	51	51.0
7	0.16	2	54	27.0
8	0.31	3	84	28.0
9	0.48	2	73	36.5
10	0.86	6	82	13.7
11	1.47	7	118	16.9
12	1.75	8	152	19.0
13	2.23	2	81	40.5
14	2.84	8	161	20.1
15	4.64	6	127	21.2
16	5.26	6	133	22.2
17	6.85	4	143	35.8
18	9.36	6	127	21.2
19	11.7	10	93	9.3
20	16.2	9	128	14.2

Table 2: Degree d, encasement running time for tangentially intersecting curves.

References

[1] M. Barton, G. Elber, and I. Hanniel. Topologically guaranteed univariate solutions of underconstrained polynomial systems via no-loop and single component tests. *Computer-Aided Design*, 43(8):10351044, 2011.

[2] Michael Bartoň. Solving polynomial systems using no-root elimination blending schemes. *Computer-Aided Design*, 43(12):1870–1878, 2011.

[3] Michael Bartoň and Bert Jüttler. Computing roots of polynomials by quadratic clipping. *Computer Aided Geometric Design*, 24(3):125–141, 2007.

[4] Eric Berberich, Arno Eigenwillig, Michael Hemmer, Susan Hert, Lutz Kettner, Kurt Mehlhorn, Joachim Reichel, Susanne Schmitt, Elmar Schömer, and Nicola Wolpert. Exacus: Efficient and exact algorithms for curves and surfaces. In *European Symposium on Algorithms*, pages 155–166. Springer, 2005.

[5] Efi Fogel, Dan Halperin, and Ron Wein. *CGAL Arrangements and Their Applications: A Step-by-Step Guide*. Springer, 2012.

[6] Laurent Fousse, Guillaume Hanrot, Vincent Lefèvre, Patrick Pélissier, and Paul Zimmermann. MPFR: A multiple precision binary floating point library with correct rounding. *ACM Transactions on Mathematical Software*, 33:13, 2007.

[7] I. Hanniel and G. Elber. Subdivision termination criteria in subdivision multivariate solvers using dual hyperplanes representations. *Computer Aided Design*, 39:36978, 2007.

[8] Benoit Jubin. A generalized Poincare-Hopf index theorem. `http://arxiv.org/abs/0903.0697`, 2009.

[9] Bert Jüttler and Brian Moore. A quadratic clipping step with superquadratic convergence for bivariate polynomial systems. *Mathematics in Computer Science*, 5(2):223–235, 2011.

[10] Joseph Masterjohn. Encasement: A robust method for finding intersections of semi-algebraic curves. *Open Access Theses*, 699, 2017. `https://scholarlyrepository.miami.edu/oa_theses/699`.

[11] Kurt Mehlhorn and Michael Sagraloff. A deterministic algorithm for isolating the real roots of a real polynomial. *Journal of Symbolic Computation*, 46:70–90, 2011.

[12] Ramon E. Moore. *Methods and Applications of Interval Analysis*. SIAM Studies in Applied Mathematics. SIAM, Philadelphia, 1979.

[13] Bernard Mourrain and Jean Pascal Pavone. Subdivision methods for solving polynomial equations. *Journal of Symbolic Computation*, 44(3):292–306, 2009.

[14] Cong Wang, Yi-Jen Chiang, and Chee Yap. On soft predicates in subdivision motion planning. *Computational Geometry: Theory and Applications*, 48(8):589–605, 2015.

[15] Ron Wein, Eric Berberich, Efi Fogel, Dan Halperin, Michael Hemmer, Oren Salzman, and Baruch Zukerman. 2D arrangements. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.11 edition, 2017.

Geometric Fingerprint Recognition via Oriented Point-Set Pattern Matching

David Eppstein * Michael T. Goodrich* Jordan Jorgensen* Manuel R. Torres †

Abstract

Motivated by the problem of fingerprint matching, we present geometric approximation algorithms for matching a *pattern* point set against a *background* point set, where the points have angular orientations in addition to their positions.

1 Introduction

Fingerprint recognition typically involves a three-step process: (1) digitizing fingerprint images, (2) identifying *minutiae*, which are points where ridges begin, end, split, or join, and (3) matching corresponding minutiae points between the two images. An important consideration is that the minutiae are not pure geometric points: besides having geometric positions, defined by (x, y) coordinates in the respective images, each minutiae point also has an *orientation* (the direction of the associated ridges), and these orientations should be taken into consideration in the comparison, e.g., see [13, 9, 16, 19, 10, 11, 17, 15, 12] and Figure 1.

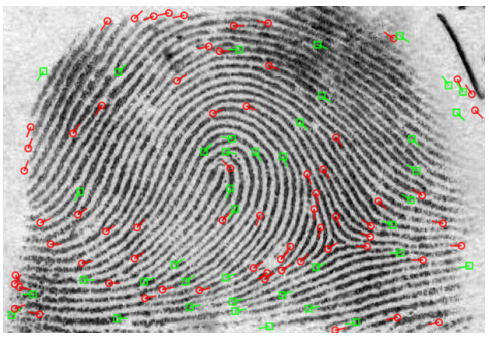


Figure 1: Screenshot of the display of fingerprint minutiae in NIST's Fingerprint Minutiae Viewer (FpMV).

In this paper, we consider computational geometry problems inspired by this fingerprint matching problem. The problems we consider are all instances of point-set pattern matching problems, where we are given a "pattern" set, P, of m points in $\mathbb{R}^2$ and a "background" set, B, of n points in $\mathbb{R}^2$, and we are asked to find a transformation of P that best aligns the points of P with a subset of the points in B, e.g., see [3, 4, 5, 6, 7].

*University of California, Irvine
†University of Illinois

A natural choice of a distance measure to use in this case, between a transformed copy, P', of the pattern, P, against the background, B, is the *directed Hausdorff distance*, defined as $h(P', B) = \max_{p \in P'} \min_{q \in B} \rho(p, q)$, where ρ is an underlying distance metric for points, such as the Euclidean metric. In other words, the problem is to find a transformation of P that minimizes the farthest any point in P is from its nearest neighbor in B. Rather than only considering the positions of the points in P and B, however, in this paper we consider instances in which each point in P and B also has an associated *orientation* defined by an angle, as in the fingerprint matching application.

It is important in such *oriented point-set pattern matching* problems to use an underlying distance that combines information about both the locations and the orientations of the points, and to use this distance in finding a good transformation. Our goal is to design efficient algorithms that can find a transformation that is a good match between P and B taking both positions and orientations into consideration.

Previous Work. In the domain of fingerprint matching, past work tends to focus on matching fingerprints heuristically or as pixelated images, taking into consideration both the positions and orientation of the minutiae or other features, e.g., see [13, 9, 16, 19, 10, 11, 17, 15, 12]. We are not aware of past work on studying fingerprint matching as a computational geometry problem, however.

Geometric pattern matching for point sets without orientations, on the other hand, has been well studied from a computational geometry viewpoint, e.g., see [1, 4, 6, 18]. For such unoriented point sets, existing algorithms can find an optimal solution minimizing Hausdorff distance, but they generally have high polynomial running times. Several existing algorithms give approximate solutions to geometric pattern matching problems [3, 5, 7, 8], but we are not aware of previous approximation algorithms for oriented point-set pattern matching. Goodrich *et al.* [7] present approximation algorithms for geometric pattern matching in multiple spaces under different types of motion, achieving approximation ratios ranging from 2 to $8 + \epsilon$, for constant $\epsilon > 0$. Cho and Mount [5] show how to achieve improved approximation ratios for such matching problems, at the expense of making the analysis more complicated.

Other algorithms give approximation ratios of $1+\epsilon$, allowing the user to define the degree of certainty they want. Indyk *et al.* [8] give a $(1+\epsilon)$-approximation algorithm whose running time is defined in terms of both the number of points in the set as well as Δ, which is defined as the the distance between the farthest and the closest pair of points. Cardoze and Schulman [3] offer a randomized $(1+\epsilon)$-approximation algorithm for $\mathbb{R}^d$ whose running time is also defined in terms of Δ. These algorithms are fast when Δ is relatively small, which is true on average for many application areas, but these algorithms are much less efficient in domains where Δ is likely to be arbitrarily large.

Our Results. In this paper, we present a family of simple algorithms for approximate oriented point-set pattern matching problems, that is, computational geometry problems motivated by fingerprint matching. Each of our algorithms uses as a subroutine a *base algorithm* that selects certain points of the pattern, P, and "pins" them into certain positions with respect to the background, B. This choice determines a transformed copy P' of the whole point set P. We then compute the directed Hausdorff distance for this transform by querying the nearest neighbor in B for each point of P'. To find nearest neighbors for a suitably-defined metric on oriented points that combines straight-line distance with rotation amounts, we adapt balanced box decomposition (BBD) trees [2] to oriented point sets, which may be of independent interest. The general idea of this adaptation is to insert two copies of each point such that, for any query point, if we find its nearest neighbor using the L_1/L_2-norm, we will either find the nearest neighbor based on μ_1/μ_2 or we will find one of its copies. The full details of this approach are described in Appendix B. The output of the base algorithm is the transformed copy P' that minimizes this distance. We refer to our base algorithms as *pin-and-query* methods.

These base algorithms are all simple and effective, but their approximation factors are larger than 2, whereas we seek $(1+\epsilon)$-approximation schemes for any constant $\epsilon > 0$. To achieve such results, our approximation schemes call the base algorithm twice. The first call determines an approximate scale of the solution. Next, our schemes apply a *grid-refinement* strategy that expands the set of background points by convolving it with a fine grid at that scale, in order to provide more candidate motions. Finally, they call the base algorithm a second time on the expanded input. This allows us to leverage the speed and simplicity of the base algorithms, gaining greater accuracy while losing only a constant factor in our running times.

The resulting approximation algorithms run in the same asymptotic time bound as the base algorithm (with some dependence on ϵ in the constants) and achieve approximations that are a $(1+\epsilon)$ factor close to optimal, for any constant $\epsilon > 0$. For instance, one of our approximation schemes, designed in this way, guarantees a worst case running time of $O(n^2 m \log n)$ for rigid motions defined by translations and rotations. Thus, our approach results in polynomial-time approximation schemes (PTASs), where their running times depend only on combinatorial parameters. Specifically, we give the runtimes and approximations ratios for our algorithms in Table 1.

Algorithm	**Running Time**	**Approx. Ratio**
T	$O(nm \log n)$	$1+\epsilon$
TR	$O(n^2 m \log n)$	$1+\epsilon$
TRS	$O(n^2 m \log n)$	$1+\epsilon$

Table 1: Running times and approximation ratios for our approximation algorithms.

The primary challenge in the design of our algorithms is to come up with methods that achieve an approximation factor of $1+\epsilon$, for any small constant $\epsilon > 0$, without resulting in a running time that is dependent on a geometric parameter like Δ. The main idea that we use to overcome this challenge is for our base algorithms in some cases to use two different pinning schemes, one for large diameters and one for small diameters, We show that one of these pinning schemes always finds a good match, so choosing the best transformation found by either of them allows us to avoid a dependence on geometric parameters in our running times. As mentioned above, all of our base algorithms are simple, as are our $(1+\epsilon)$-approximation algorithms. Moreover, proving each of our algorithms achieves a good approximation ratio is also simple, involving no more than "high school" geometry. Still, for the sake of our presentation, we postpone some proofs and simple cases to appendices.

2 Formal Problem Definition

Let us formally define the *oriented point-set pattern matching* problem. We define an *oriented point set* in $\mathbb{R}^2$ to be a finite subset of the set O of all oriented points, defined as

$$O = \big\{(x, y, a) \mid x, y, a \in \mathbb{R}, a \in [0, 2\pi)\big\}.$$

We consider three sets of transformations on oriented point sets, corresponding to the usual translations, rotations, and scalings on $\mathbb{R}^2$. In particular, we define the set of *translations*, $\mathcal{T}$, as the set of functions $T_v : O \to O$ of the form

$$T_v(x, y, a) = (x + v_x, y + v_y, a),$$

where $v = (v_x, v_y) \in \mathbb{R}^2$ is referred to as the *translation vector*.

Let $R_{p,\theta}$ be a rotation in $\mathbb{R}^2$ where p and θ are the center and angle of rotation, respectively. We extend the action of $R_{p,\theta}$ from unoriented points to oriented points by defining

$$R_{p,\theta}(x, y, a) = \big(R_{p,\theta}(x, y), (a + \theta) \bmod 2\pi\big),$$

and we let $\mathcal{R}$ denote the set of rotation transformations from O to O defined in this way.

Finally, we define the set of *scaling* operations on an oriented point set. Each such operation $S_{p,s}$ is determined by a point $p = (x_p, y_p, a_p)$ at the center of the scaling and by a scale factor, s. If a point q is Euclidean distance d away from p before scaling, the distance between q and p should become sd after scaling. In particular, this determines $S_{p,s} : O \to O$ to be the function

$$S_{p,s}(x, y, a) = \big(x_p + s(x - x_p), y_p + s(y - y_p), a\big).$$

We let $\mathcal{S}$ denote the set of scaling functions defined in this way.

As in the unoriented point-set pattern matching problems, we use a directed Hausdorff distance to measure how well a transformed patten set of points, P, matches a background set of points, B. That is, we use

$$h(P, B) = \max_{p \in P} \min_{q \in B} \mu(p, q),$$

where $\mu(p, q)$ is a distance metric for oriented points in $\mathbb{R}^2$. Our approach works for various types of metrics, μ, for pairs of points, but, for the sake of concreteness, we focus on two specific distance measures for elements of O, which are based on the L_1-norm and L_2-norm, respectively. In particular, for $(x_1, y_1, a_1), (x_2, y_2, a_2) \in O$, let

$$\mu_1((x_1, y_1, a_1), (x_2, y_2, a_2)) = |x_1 - x_2| + |y_1 - y_2| + \min(|a_1 - a_2|, 2\pi - |a_1 - a_2|),$$

and let

$$\mu_2((x_1, y_1, a_1), (x_2, y_2, a_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + \min(|a_1 - a_2|, 2\pi - |a_1 - a_2|)^2}.$$

Intuitively, one can interpret these distance metrics to be analogous to the L_1-norm and L_2-norm in a cylindrical 3-dimensional space where the third dimension wraps back around to 0 at 2π. Thus, for $i \in \{1, 2\}$, and $B, P \subseteq O$, we use the following directed Hausdorff distance:

$$h_i(P, B) = \max_{p \in P} \min_{b \in B} \mu_i(p, b).$$

Therefore, for some subset $\mathcal{E}$ of $\mathcal{T} \cup \mathcal{R} \cup \mathcal{S}$, the *oriented point-set pattern matching* problem is to find a composition E of one or more functions in $\mathcal{E}$ that minimizes $h_i(E(P), B)$.

3 Translations and Rotations

In this section, we present our base algorithm and approximation algorithm for approximately solving the oriented point-set pattern matching problem where we allow translations and rotations. Given two subsets of O, P and B, with $|P| = m$ and $|B| = n$, our goal here is to minimize $h_i(E(P), B)$ where E is a composition of functions in $\mathcal{T} \cup \mathcal{R}$. In the case of translations and rotations, we actually give two sets of algorithms—one set that works for point sets with large diameter and one that works for point sets with small diameter. Deciding which of these to use is based on a simple calculation (which we postpone to the analysis below), which amounts to a normalization decision to determine how much influence orientations have on matches versus coordinates.

Base Algorithm Under Translation and Rotation with Large Diameter. In this subsection, we present an algorithm for solving the approximate oriented point-set pattern matching problem where we allow translations and rotations. This algorithm provides a good approximation ratio when the diameter of our pattern set is large. Given two subsets P and B of O, with $|P| = m$ and $|B| = n$, we wish to minimize $h_i(E(P), B)$ over all compositions E of one or more functions in $\mathcal{T} \cup \mathcal{R}$. Our algorithm is as follows (see Figure 2).

Algorithm BaseTranslateRotateLarge(P, B):

Find p and q in P having the maximum value of $\|(x_p, y_p) - (x_q, y_q)\|_2$.
for every pair of points $b, b' \in B$ **do**
  Pin step: Apply the translation, $T_v \in \mathcal{T}$, that takes p to b, and apply the rotation, $R_{p,\theta}$, that makes p, b', and q collinear.
  Let P' denote the transformed pattern set, P.
  for every $q \in P'$ **do**
    Query step: Find a nearest-neighbor of q in B using the μ_i metric, and update a candidate Hausdorff distance accordingly.
  end for
  return the smallest candidate Hausdorff distance found as the smallest distance, $h_i(R_{p,\theta}(T_v(P)), B)$.
end for

The points p and q can be found in $O(m \log m)$ time [14]. The pin step iterates over $O(n^2)$ translations and rotations, respectively, and, for each one of these transformations, we perform m BBD queries, each of which takes $O(\log n)$ time. Therefore, our total running time is $O(n^2 m \log n)$. Our analysis for this algorithm's approximation factor uses the following simple lemma.

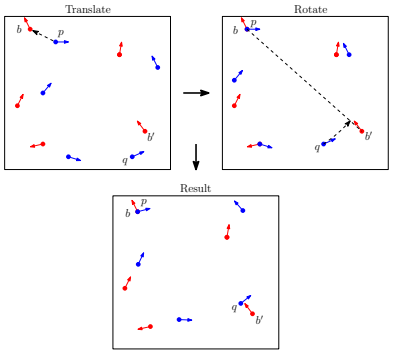


Figure 2: Illustration of the translation and rotation steps of the base approximation algorithm for translation and rotation in O when diameter is large.

Lemma 1 *Let P be a finite subset of O. Consider the rotation $R_{c,\theta}$ in $\mathcal{R}$. Let $q = (x_q, y_q, a_q)$ be the element in P such that $\|(x_q, y_q) - (x_c, y_c)\|_2 = D$ is maximized. For any $p = (x_p, y_p, a_p) \in P$, denote $R_{c,\theta}(x_p, y_p, a_p)$ as $p' = (x_{p'}, y_{p'}, a_{p'})$. Let $i \in \{1, 2\}$. Then for all $p \in P$, $\mu_i(p, p') \leq \|(x_q, y_q) - (x_{q'}, y_{q'})\|_i + \pi\|(x_q, y_q) - (x_{q'}, y_{q'})\|_2/(2D)$.*

Theorem 2 *Let h_{opt} be $h_i(E(P), B)$ where E is the composition of functions in $\mathcal{T} \cup \mathcal{R}$ that attains the minimum of h_i, for $i \in \{1, 2\}$. The algorithm above runs in time $O(n^2 m \log n)$ and produces an approximation to h_{opt} that is at most $(A_1 + \epsilon)h_{opt}$ for h_1 and at most $(A_2 + \epsilon)h_{opt}$ for h_2, where $\epsilon >$ is a fixed constant, $A_1 = 6 + \sqrt{2}\pi/D$, and $A_2 = 2 + \sqrt{2}(2 + \pi/D)$.*

Grid Refinement. In this subsection, we describe our grid refinement process, which allows us to use a base algorithm to obtain an approximation ratio of $1 + \epsilon$. To achieve this result, we take advantage of an important property of the fact that we are approximating a Hausdorff distance by a pin-and-query algorithm. Our base algorithm approximates h_{opt} by pinning a reference pattern point, p, to a background point, b. Reasoning backwards, if we have a pattern in an optimal position, where every pattern point, p, is at distance $d \leq h_{\text{opt}}$ from its associated nearest neighbor in the background, then one of the transformations tested by the base pin-and-query algorithm moves each pattern point by a distance of at most $(A_i - 1)d$ away from this optimal location when it performs its pinning operation.

Suppose we could define a constant-sized "cloud" of points with respect to each background point, such that one of these points is guaranteed to be very close to the optimal pinning location, much closer than the distance d from the above argument. Then, if we use these cloud points to define the transformations checked by the base algorithm, one of these transformations will move each point from its optimal position by a much smaller distance.

To aid us in defining such a cloud of points, consider the set of points $G(p, l, k) \subset \mathbb{R}^2$ (where $p = (x_p, y_p)$ is some point in $\mathbb{R}^2$, l is some positive real value, and k is some positive integer) defined by the following formula:

$$G(p, l, k) = \big\{q \in \mathbb{R}^2 \mid q = (x_p + il, y_p + jl), i, j \in \mathbb{Z}, -k \leq i, j \leq k\big\}.$$

Then $G(p, l, k)$ is a grid of $(2k + 1)^2$ points within a square of side length $2kl$ centered at p, where the coordinates of each point are offset from the coordinates of p by a multiple of l. An example is shown in Figure 3.

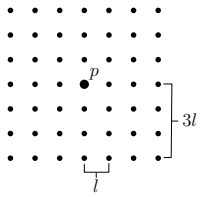


Figure 3: An example of $G(p, l, 3)$.

Now consider any point q whose Euclidean distance is no more than kl from p. This small distance forces point q to lie within the square convex hull of $G(p, l, k)$. This implies that there is a point of $G(p, l, k)$ that is even closer to q:

Lemma 3 *Let $i \in \{1, 2\}$. Given two points $p, q \in \mathbb{R}^2$, if $\|p - q\|_i \leq kl$, then $\|q - s\|_1 \leq l$ and $\|q - s\|_2 \leq l/\sqrt{2}$, where s is q's closest neighbor in $G(p, l, k)$.*

A $(1+\epsilon)$-Approximation Algorithm Under Translation and Rotation with Large Diameter. Here, achieve a $(1 + \epsilon)$-approximation ratio when we allow translations and rotations. Again, given two subsets of O, P and B, with $|P| = m$ and $|B| = n$, our goal is to minimize $h_i(E(P), B)$ over all compositions E of one or more functions in $\mathcal{T} \cup \mathcal{R}$. We perform the following steps.

1. Run algorithm, BaseTranslateRotateLarge(P, B), from Section 3 to obtain an approximation $h_{apr} \leq A \cdot h_{\text{opt}}$, where $A = A_1 + \epsilon$ or $A = A_2 + \epsilon$, for a constant $\epsilon > 0$.

2. For every $b \in B$, generate the grid of points $G_b = G(b, \frac{h_{apr}\epsilon}{A^2 - A}, \lceil \frac{A^2 - A}{\epsilon} \rceil)$ for h_1 or the grid $G_b = G(b, \frac{\sqrt{2}h_{apr}\epsilon}{A^2 - A}, \lceil \frac{A^2 - A}{\sqrt{2}\epsilon} \rceil)$ for h_2. Let B' denote the resulting point set, which is of size $O(A^4 n)$, i.e., $|B'|$ is $O(n)$ when A is a constant.

3. Run algorithm, BaseTranslateRotateLarge(P, B'), except use the original set, B, for nearest-neighbor queries in the query step.

It is easy to see that this simple algorithm runs in $O(A^8n^2m\log n)$, which is $O(n^2m\log n)$ when A is a constant (i.e., when the points in P have a large enough diameter).

Theorem 4 *Let h_{opt} be $h_i(E(P), B)$ where E is the composition of functions in $\mathcal{T} \cup \mathcal{R}$ that attains the minimum of h_i. The algorithm above runs in time $O(A^8n^2m\log n)$ and produces an approximation to h_{opt} that is at most $(1+\epsilon)h_{opt}$ for both h_1 and h_2.*

Base Algorithm Under Translation and Rotation with Small Diameter. In this subsection, we present an alternative algorithm for solving the approximate oriented point-set pattern matching problem where we allow translations and rotations. Compared to the algorithm given in Section 3, this algorithm instead provides a good approximation ratio when the diameter of our pattern set is small. Once again, given two subsets of O, P and B, with $|P| = m$ and $|B| = n$, we wish to minimize $h_i(E(P), B)$ over all compositions E of one or more functions in $\mathcal{T} \cup \mathcal{R}$. We perform the following algorithm (see Figure 4).

Algorithm BaseTranslateRotateSmall(P, B):

Choose some $p \in P$ arbitrarily.
for every points $b \in B$ **do**
  Pin step: Apply the translation, $T_v \in \mathcal{T}$, that takes p to b, and then apply the rotation, $R_{p,\theta}$, that makes p and b have the same orientation.
  Let P' denote the transformed pattern set, P.
  for every $q \in P'$ **do**
    Query step: Find a nearest-neighbor of q in B using the μ_i metric, and update a candidate Hausdorff distance accordingly.
  end for
  return the smallest candidate Hausdorff distance found as the smallest distance, $h_i(R_{p,\theta}(T_v(P)), B)$.
end for

Theorem 5 *Let h_{opt} be $h_i(E(P), B)$ where E is the composition of functions in $\mathcal{T} \cup \mathcal{R}$ that attains the minimum of h_i. The algorithm above runs in time $O(nm\log n)$ and produces an approximation to h_{opt} that is at most $(A_i+\epsilon)h_{opt}$ for h_i, where $i = \{1, 2\}$, $\epsilon > 0$ is a fixed constant, $A_1 = 2+\sqrt{2}D$, and $A_2 = 2+D$.*

A $(1+\epsilon)$-Approximation Algorithm Under Translation and Rotation with Small Diameter. In this subsection, we utilize the algorithm from Section 3 to achieve a $(1+\epsilon)$-approximation ratio when we allow translations and rotations. Again, given two subsets of

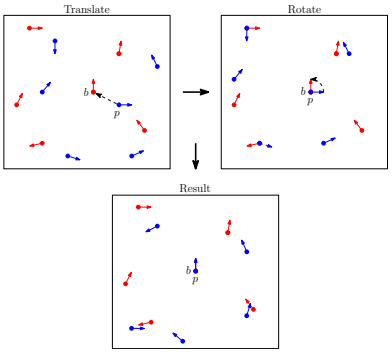


Figure 4: Illustration of the translation and rotation steps of the base approximation algorithm for translation and rotation in O when diameter is small.

O, P and B, with $|P| = m$ and $|B| = n$, our goal is to minimize $h_i(E(P), B)$ over all compositions E of one or more functions in $\mathcal{T} \cup \mathcal{R}$. We begin by describing another type of grid refinement we use in this case.

In particular, let us consider a set of points $C(p,k) \subset O$ where $p = (x_p, y_p, a_p)$ is some point in O and k is some positive integer. We define the set in the following way (see Figure 5):

$$C(p,k) = \{q \in O| \\ q = (x_p, y_p, a + 2\pi i/k \mod 2\pi), i \in \mathbb{Z}, 1 \le i \le k\}.$$

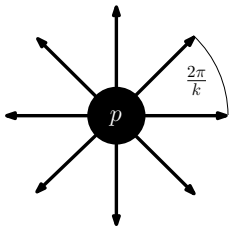


Figure 5: An example of $C(p, 8)$.

From this definition, we can see that $C(p,k)$ is a set of points that share the same position as p but have different orientations that are equally spaced out, with each point's orientation being an angle of $\frac{2\pi}{k}$ away from the previous point. Therefore, it is easy to see that, for any point $q \in O$, there is a point in $C(p,k)$ whose orientation is at most an angle of $\frac{\pi}{k}$ away from the orientation of q. Given this definition, our algorithm is as follows.

1. Run algorithm, BaseTranslateRotateSmall(P, B), from Section 3, to obtain $h_{apr} \le A \cdot h_{\text{opt}}$.
2. For every $b \in B$, generate the point set

$$G_b = G\left(b, \frac{h_{apr}\epsilon}{2(A^2-A)}, \left\lceil \frac{2(A^2-A)}{\epsilon} \right\rceil\right)$$

for h_1 or

$$G_b = G\left(b, \frac{h_{apr}\epsilon}{A^2 - A}, \left\lceil \frac{A^2 - A}{\epsilon} \right\rceil\right)$$

for h_2. Let B' denote the resulting set of points, i.e., $B' = \bigcup_{b \in B} G_b$.

3. For every $b' \in B'$, generate the point set

$$C_{b'} = C\left(b', \frac{2(A^2 - A)}{\pi h_{apr}\epsilon}\right)$$

for h_1 or

$$C_{b'} = C\left(b', \frac{\sqrt{2}(A^2 - A)}{\pi h_{apr}\epsilon}\right)$$

for h_2. Let B'' denote the resulting set of points.

4. Run algorithm, BaseTranslateRotateSmall(P, B''), but continue to use the points in B for nearest-neighbor queries.

Intuitively, this algorithm uses the base algorithm to give us an indication of what the optimal solution might be. We then use this approximation to generate a larger set of points from which to derive transformations to test, but this time we also generate a number of different orientations for those points as well. We then use this point set in the base algorithm when deciding which transformations to iterate over, while still using B to compute nearest neighbors.

The first step of this algorithm runs in time $O(nm \log n)$, as we showed. The second step takes time proportional to the number of points which have to be generated, which is determined by n, our choice of the constant ϵ, and the approximation ratio, A, of our base algorithm. The time needed to complete the second step is $O(A^4 n)$. The third step generates even more points based on points generated in step two, which increases the size of B'' to be $O(A^6 n)$. The last step runs in time $O(A^6 nm \log n)$, which is also the running time for the full algorithm.

Theorem 6 *Let h_{opt} be $h_i(E(P), B)$ where E is the composition of functions in $\mathcal{T} \cup \mathcal{R}$ that attains the minimum of h_i. The algorithm above runs in time $O(A^6 nm \log n)$ and produces an approximation to h_{opt} that is at most $(1 + \epsilon)h_{opt}$ for both h_1 and h_2.*

Combining the Algorithms for Large and Small Diameters. For the two cases above, we provided two base algorithms that each had a corresponding $(1 + \epsilon)$-approximation algorithm. As mentioned above, we classified the two by whether the algorithm achieved a good approximation when the diameter of the pattern set was large or small. This is because the large diameter base algorithm has an approximation ratio with terms that are inversely proportional to the diameter, and the small diameter base algorithm has an approximation ratio with terms that are directly proportional to the diameter.

Both of the resulting $(1+\epsilon)$-approximation algorithms have running times which are affected by the approximation ratio of their base algorithm, meaning their run times are dependent upon the diameter of the pattern set. We can easily see, however, that the approximation ratios of the large and small diameter base algorithms intersect at some fixed constant diameter, D^*. For h_1, if we compare the expressions $6 + \sqrt{2}\pi/D$ and $2 + \sqrt{2}D$, we find that the two expressions are equal at $D^* = \sqrt{2} + \sqrt{2 + \pi} \approx 3.68$. For h_2, we compare $2 + \sqrt{2}(2 + \pi/D)$ and $2 + D$ to find that they are equal at $D^* = \sqrt{2} + \sqrt{2 + \sqrt{2}\pi} \approx 3.95$. For diameters larger than D^*, the approximation ratio of the large diameter algorithm is smaller than at D^*, and for diameters smaller than D^*, the approximation ratio of the small diameter algorithm is smaller than at D^*. Thus, if we choose to use the small diameter algorithms when the diameter is less than D^* and we use the large diameter algorithms when the diameter is greater or equal to D^*, we ensure that the approximation ratio is no more than the constant value that depends on the constant D^*. Thus, based on the diameter of the pattern set, we can decide *a priori* if we should use our algorithms for large diameters or small diameters and just go with that set of algorithms. This implies that we are guaranteed that our approximation factor, A, in our base algorithm is always bounded above by a constant; hence, our running time for the translation-and-rotation case is $O(n^2 m \log n)$.

4 Conclusion

We present distance metrics that can be used to measure the similarity between two point sets with orientations and we also provided fast algorithms that guarantee close approximations of an optimal transformation. In the appendices, we provide additional algorithms for other types of transformations and we also provide results of experiments.

Acknowledgments

This work was supported in by the NSF under grants 1526631, 1618301, and 1616248, and by DARPA under agreement no. AFRL FA8750-15-2-0092. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

[1] H. Alt and L. J. Guibas. Discrete geometric shapes: Matching, interpolation, and approximation. *Handbook of computational geometry*, 1:121–153, 1999.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

[3] D. E. Cardoze and L. J. Schulman. Pattern matching for spatial point sets. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 156–165. IEEE, 1998.

[4] L. P. Chew, M. T. Goodrich, D. P. Huttenlocher, K. Kedem, J. M. Kleinberg, and D. Kravets. Geometric pattern matching under euclidean motion. *Computational Geometry*, 7(1):113–124, 1997.

[5] M. Cho and D. M. Mount. Improved approximation bounds for planar point pattern matching. *Algorithmica*, 50(2):175–207, 2008.

[6] M. Gavrilov, P. Indyk, R. Motwani, and S. Venkatasubramanian. Geometric pattern matching: A performance study. In *Proceedings of the fifteenth annual symposium on Computational geometry*, pages 79–85. ACM, 1999.

[7] M. T. Goodrich, J. S. Mitchell, and M. W. Orletsky. Approximate geometric pattern matching under rigid motions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4):371–379, 1999.

[8] P. Indyk, R. Motwani, and S. Venkatasubramanian. Geometric matching under noise: Combinatorial bounds and algorithms. In *SODA*, pages 457–465, 1999.

[9] A. K. Jain, L. Hong, S. Pankanti, and R. Bolle. An identity-authentication system using fingerprints. *Proceedings of the IEEE*, 85(9):1365–1388, 1997.

[10] T.-Y. Jea and V. Govindaraju. A minutia-based partial fingerprint recognition system. *Pattern Recognition*, 38(10):1672–1684, 2005.

[11] X. Jiang and W.-Y. Yau. Fingerprint minutiae matching based on the local and global structures. In *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, volume 2, pages 1038–1041, 2000.

[12] J. V. Kulkarni, B. D. Patil, and R. S. Holambe. Orientation feature for fingerprint matching. *Pattern Recognition*, 39(8):1551–1554, 2006.

[13] D. Maltoni, D. Maio, A. Jain, and S. Prabhakar. *Handbook of Fingerprint Recognition*. Springer Science & Business Media, 2009.

[14] F. P. Preparata and M. I. Shamos. Computational geometry: an introduction. *Springer-Verlag, New York, NY*, 1985.

[15] J. Qi, S. Yang, and Y. Wang. Fingerprint matching combining the global orientation field with minutia. *Pattern Recognition Letters*, 26(15):2424–2430, 2005.

[16] N. Ratha and R. Bolle. *Automatic Fingerprint Recognition Systems*. Springer Science & Business Media, 2007.

[17] M. Tico and P. Kuosmanen. Fingerprint matching using an orientation-based minutia descriptor. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(8):1009–1014, 2003.

[18] R. C. Veltkamp. Shape matching: similarity measures and algorithms. In *Shape Modeling and Applications, SMI 2001 International Conference on.*, pages 188–197. IEEE, 2001.

[19] H. Xu, R. N. J. Veldhuis, T. A. M. Kevenaar, and T. A. H. M. Akkermans. A fast minutiae-based fingerprint recognition system. *IEEE Systems Journal*, 3(4):418–427, Dec 2009.

A Postponed Proofs

In this appendix, we present proofs that were postponed from the body of our paper.

Lemma 3. *Let $i \in \{1,2\}$. Given two points $p, q \in \mathbb{R}^2$, if $\|p-q\|_i \leq kl$, then $\|q-s\|_1 \leq l$ and $\|q-s\|_2 \leq l/\sqrt{2}$, where s is q's closest neighbor in $G(p,l,k)$.*

Proof. Because $\|p-q\|_i \leq kl$, we know that q exists within the square of side length $2kl$ which encompasses $G(p,l,k)$ (which we will refer to as G for the remainder of this proof). This square can be divided into $(2k)^2$ non-overlapping squares of side length l. It is easy to see that the vertices of these squares are all points in G and that q exists within (or on the edge of) at least one of these squares. The point inside of a square that maximizes the distance to the square's closest vertex is the exact center of the square. If the side length is l, simple geometry shows us that at this point, the distance to any vertex is l with respect to the L_1-norm and $l/\sqrt{2}$ with respect to the L_2-norm. Thus, because q exists within a square of side length l whose vertices are points in G, the furthest that q can be from its nearest neighbor in G is l for the L_1-norm and $l/\sqrt{2}$ for the L_2-norm. □

Lemma 1. *Let P be a finite subset of O. Consider the rotation $R_{c,\theta}$ in $\mathcal{R}$. Let $q = (x_q, y_q, a_q)$ be the element in P such that $\|(x_q,y_q)-(x_c,y_c)\|_2 = D$ is maximized. For any $p = (x_p, y_p, a_p) \in P$, denote $R_{c,\theta}(x_p, y_p, a_p)$ as $p' = (x_{p'}, y_{p'}, a_{p'})$. Let $i \in \{1,2\}$. Then for all $p \in P$, $\mu_i(p,p') \leq \|(x_q,y_q)-(x_{q'},y_{q'})\|_i + \pi\|(x_q,y_q)-(x_{q'},y_{q'})\|_2/(2D)$.*

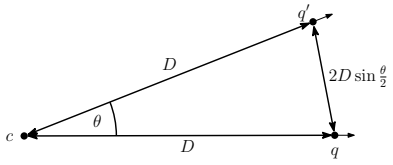


Figure 6: The rotation of q to q' about c

Proof. After applying the rotation $R_{c,\theta}$, we know q has moved at least as far than any other point because it is the farthest from the center of rotation. Without loss of generality, $0 \leq \theta \leq \pi$. Then it is easily verifiable that $\theta/\pi \leq \sin(\theta/2)$. As $2D\sin(\theta/2)$ is the Euclidean distance q moves under $R_{c,\theta}$, it follows that

$$\frac{2D\theta}{\pi} \leq 2D\sin(\theta/2) = \|(x_q,y_q)-(x_{q'},y_{q'})\|_2.$$

This scenario is illustrated in Figure 6. Thus, $\theta \leq (\pi\|(x_q,y_q)-(x_{q'},y_{q'})\|_2)/(2D)$, which implies that $R_{c,\theta}$ moves the position of q by at most $\|(x_q,y_q)-(x_{q'},y_{q'})\|_i$ and changes the orientation of q by at most $\pi\|(x_q,y_q)-(x_{q'},y_{q'})\|_2/(2D)$. Therefore, because q moves farther than any other point in P, any point $p \in P$ has moved a distance of at most $\|(x_q,y_q)-(x_{q'},y_{q'})\|_i + \pi\|(x_q,y_q)-(x_{q'},y_{q'})\|_2/(2D)$ with respect to the distance function μ_i. □

Theorem 2. *Let h_{opt} be $h_i(E(P),B)$ where E is the composition of functions in $\mathcal{T} \cup \mathcal{R}$ that attains the minimum of h_i, for $i \in \{1,2\}$. The algorithm above runs in time $O(n^2 m \log n)$ and produces an approximation to h_{opt} that is at most $(A_1+\epsilon)h_{opt}$ for h_1 and at most $(A_2+\epsilon)h_{opt}$ for h_2, where $\epsilon > 0$ is a fixed constant, $A_1 = 6+\sqrt{2}\pi/D$, and $A_2 = 2+\sqrt{2}(2+\pi/D)$.*

Proof. The additional ϵ terms come entirely from using approximate nearest neighbor queries (defining BBD trees so they return $(1+\epsilon/A_i)$-approximate nearest neighbors, for $i \in \{1,2\}$). So it is sufficient for us to prove approximation bounds that are $A_i \cdot h_{\text{opt}}$.

The first step is argued similarly to that of the proof of Theorem 8. Let E be the composition of functions in $\mathcal{T} \cup \mathcal{R}$ that attains the minimum of $h(E(P),B)$ and let P' be $E(P)$. Then for all p in P', there exists b in B such that $\mu_i(p,b) \leq h_{\text{opt}}$. Let $p', q' \in B$ be the closest background points to optimal positions of p and q respectively, where p and q are the diametric points we choose in the first step of the algorithm. Thus,

$$\|(x_p,y_p)-(x_{p'},y_{p'})\|_i \leq \mu_i(p,p') \leq h_{\text{opt}}.$$

Apply the translation T_v on P' so that p coincides with p', which is equivalent to moving every point $\|(x_p,y_p)-(x_{p'},y_{p'})\|_i$ with respect to position. Lemma 7, then, implies that all points have moved at most h_{opt}.

Next, apply the rotation $R_{p,\theta}$ to P' that makes p, q, and q' co-linear. With respect to position, q moves at most a Euclidean distance of $2D\sin(\theta/2)$ away from q' where D is the Euclidean distance between p and q. As all points were already at most $2h_{\text{opt}}$ away from their original background point in B, this implies that $2D\sin(\theta/2) \leq 2\sqrt{2}h_{\text{opt}}$. Thus, $\|(x_q,y_q)-(x_{q'},y_{q'})\|_2$ is at most $2\sqrt{2}h_{\text{opt}}$. Then by Lemma 1, as q is the furthest point from p, the rotation moves all points at most $2\sqrt{2}h_{\text{opt}} + \sqrt{2}\pi h_{\text{opt}}/D$ with respect to h_2 and at most $4h_{\text{opt}} + \sqrt{2}\pi h_{\text{opt}}/D$ for h_1.

Since each point in the pattern set started out at most h_{opt} away from a point in the background set, we combine this with the translation and rotation movements to find that every point ends up at most $(6+\sqrt{2}\pi/D)h_{\text{opt}}$ away from a background point for h_1 and at most $(2+\sqrt{2}(2+\pi/D))h_{\text{opt}}$ away from a background point for h_2. As our algorithm checks this combination of T_v and $R_{p,\theta}$, our algorithm guarantees at least this solution. Note that we assume p' and q' are not the same point. However if this is the case, then

we know that $D \leq 2h_{\text{opt}}$ thus when we translate p to p' every point is within $(\sqrt{5}+2\pi/D)h_{\text{opt}}$ of p', which is a better approximation than the case where $p' \neq q'$ under our assumption that D is large. □

Theorem 4. *Let h_{opt} be $h_i(E(P), B)$ where E is the composition of functions in $\mathcal{T} \cup \mathcal{R}$ that attains the minimum of h_i. The algorithm above runs in time $O(A^8 n^2 m \log n)$ and produces an approximation to h_{opt} that is at most $(1+\epsilon)h_{opt}$ for both h_1 and h_2.*

Proof. Let E be the composition of functions in $\mathcal{T} \cup \mathcal{R}$ that attains the minimum of $h(E(P), B)$. Let P' be $E(P)$. Then every point $q \in P'$ is at most h_{opt} from the closest background point in B. By running the base algorithm, we find $h_{apr} \leq A \cdot h_{\text{opt}}$, where A is the approximation ratio of the base algorithm. Now consider the point $b' \in B$ which is the closest background to some pattern point $p \in P$. The square which encompasses $G_{b'}$ has a side length of $2h_{apr}$. This guarantees that p, which is at most h_{opt} away from b', lies within this square. As we saw from Lemma 3, this means that p is at most $\frac{\epsilon h_{apr}}{A^2-A}$ away from its nearest neighbor in $G_{b'}$. Thus, if a transformation defined by the nearest points in B would move our pattern points at most $(A-1)h_{\text{opt}}$ from their optimal position, then using the nearest points in $G_{b'}$ to define our transformation will move our points at most $(A-1)\frac{\epsilon h_{apr}}{A^2-A} = \frac{\epsilon h_{apr}}{A} \leq \epsilon h_{\text{opt}}$. Thus, the modified algorithm gives a solution that is at most $(1+\epsilon)h_{\text{opt}}$. □

Theorem 5. *Let h_{opt} be $h_i(E(P), B)$ where E is the composition of functions in $\mathcal{T} \cup \mathcal{R}$ that attains the minimum of h_i. The algorithm above runs in time $O(nm \log n)$ and produces an approximation to h_{opt} that is at most $(A_i+\epsilon)h_{opt}$ for h_i, where $i = \{1, 2\}$, $\epsilon > 0$ is a fixed constant, $A_1 = 2+\sqrt{2}D$, and $A_2 = 2+D$.*

Proof. The additional ϵ terms come entirely from using approximate nearest neighbor queries, so it is sufficient to prove approximations which do not include the ϵ term using exact nearest neighbor queries (defining the BBD tree so that it returns points that are $(1+\epsilon/A_i)$-approximate nearest neighbors). Particularly, we will prove a bound of $(2+\sqrt{2}D)h_{\text{opt}}$ for h_1 and a bound of $(2+D)h_{\text{opt}}$ for h_2.

Let E be the composition of functions in $\mathcal{T} \cup \mathcal{R}$ that attains the minimum of $h(E(P), B)$. Let P' be $E(P)$. Then every point $p \in P'$ is at most h_{opt} from the closest background point in B. That is, for all p in P', there exists b in B such that $\mu_i(p, b) \leq h_{\text{opt}}$. Let $p' \in B$ be the closest background point to the optimal position of p where p is the point we chose in the first step of the algorithm. Thus,

$$\mu_i(p, p') \leq h_{\text{opt}}.$$

Apply the translation T_v and rotation $R_{p,\theta}$ on P' so that p coincides with p' and both points have the same orientation. It is easy to see that p has moved from its optimal position by exactly $\mu_i(p, p') \leq h_{\text{opt}}$. Using Lemma 7 and the fact that a rotation on P causes the orientation of each point in P to change by the same amount, we find that every point $q \in P$ has moved at most $\mu_i(p, p') + d$ from its original position, where d is the change in the position of q caused by the rotation.

We know that the angle rotated, θ, must be less than h_{opt} and, without loss of generality, we assume $0 \leq \theta \leq \pi$. Therefore it is easily verifiable that $\sin(\theta/2) \leq \theta/2$. If D is the diameter of P, then regardless of our choice of p, each point in P is displaced at most $2D\sin(\theta/2)$ by the rotation. Thus each point is displaced at most $D\theta \leq Dh_{\text{opt}}$.

Since each point in the pattern set started out at most h_{opt} away from a point in the background set, we combine this with the translation and rotation movements to find that every point ends up at most $(2+\sqrt{2}D)h_{\text{opt}}$ away from a background point for h_1 and at most $(2+D)h_{\text{opt}}$ away from a background point for h_2. As our algorithm checks this combination of T_v and $R_{p,\theta}$, our algorithm guarantees at least this solution. □

Theorem 6. *Let h_{opt} be $h_i(E(P), B)$ where E is the composition of functions in $\mathcal{T} \cup \mathcal{R}$ that attains the minimum of h_i. The algorithm above runs in time $O(A^6 nm \log n)$ and produces an approximation to h_{opt} that is at most $(1+\epsilon)h_{opt}$ for both h_1 and h_2.*

Proof. Let E be the composition of functions in $\mathcal{T} \cup \mathcal{R}$ that attains the minimum of $h_i(E(P), B)$. Let P' be $E(P)$. Then every point $q \in P'$ is at most h_{opt} from the closest background point in B. By running the base algorithm, we find $h_{apr} \leq A \cdot h_{\text{opt}}$ where A is the approximation ratio of the base algorithm. Now consider the point $b' \in B$ which is the closest background to some pattern point $p \in P$. The square which encompasses $G_{b'}$ has a side length of $2h_{apr}$. This guarantees that p, which is at most h_{opt} away from b', lies within this square. As we saw from Lemma 3, this means that p is at most $\frac{\epsilon h_{apr}}{2(A^2-A)}$ away from its nearest neighbor g in $G_{b'}$ with respect to the $L1$-norm, and at most $\frac{\epsilon h_{apr}}{\sqrt{2}(A^2-A)}$ with respect to the $L2$-norm. For this point, g, there are a number of points in C_g which are at the same position but with different orientation. For some point c in C_g, the orientation of point p is within an angle of at most $\frac{h_{apr}\epsilon}{2(A^2-A)}$ for h_1 and at most $\frac{h_{apr}\epsilon}{\sqrt{2}(A^2-A)}$ for h_2. If we combine together the maximum difference in position between p and c, and the maximum difference in orientation between p and c, then we see that for both μ_1 and μ_2, the distance between p and c is at most $\frac{h_{apr}\epsilon}{A^2-A}$. Thus, if a transformation defined by the nearest point in B would

move our pattern points at most $(A-1)h_{\text{opt}}$ from their optimal position, then using the nearest point in C_g to define our transformation will move our points at most $(A-1)\frac{\epsilon h_{apr}}{A^2-A} = \frac{\epsilon h_{apr}}{A} \leq \epsilon h_{\text{opt}}$. Thus, the modified algorithm gives a solution that is at most $(1+\epsilon)h_{\text{opt}}$. □

B Translations Only

In this section, we present our base algorithm and approximation algorithm for approximately solving the oriented point-set pattern matching problem where we allow only translations. In this way, we present the basic template and data structures that we will also use for the more interesting case of translations and rotations ($\mathcal{T} \cup \mathcal{R}$).

Our methods for handling translations, rotations, and scaling is an adaptation of our methods for $\mathcal{T} \cup \mathcal{R}$; hence, we give our methods for $\mathcal{T} \cup \mathcal{R} \cup \mathcal{S}$ in an appendix.

Given two subsets of O, P and B, with $|P| = m$ and $|B| = n$, our goal here is to minimize $h_i(E(P), B)$ where E is a transformation function in $\mathcal{T}$.

Base Algorithm Under Translation Only. Our base pin-and-query algorithm is as follows.

Algorithm BaseTranslate(P, B):

Choose some $p \in P$ arbitrarily.
for every $b \in B$ **do**
  Pin step: Apply the translation, $T_v \in \mathcal{T}$, that takes p to b.
  for every $q \in T_v(P)$ **do**
    Query step: Find a nearest-neighbor of q in B using the μ_i metric, and update a candidate Hausdorff distance for T_v accordingly.
  end for
  return the smallest candidate Hausdorff distance found as the smallest distance, $h_i(T_v(P), B)$.
end for

This algorithm uses a similar approach to an algorithm of Goodrich *et al.* [7], but it is, of course, different in how it computes nearest neighbors, since we must use an oriented distance metric rather than unoriented distance metric. One additional difference is that rather than find an exact nearest neighbor, as described above, we instead find an *approximate* nearest neighbor of each point, q, since we are ultimately designing an approximation algorithm anyway. This allows us to achieve a faster running time.

In particular, in the query step of the algorithm, for any point $q \in T_v(P)$, we find a neighbor, $b \in B$, whose distance to q is at most a $(1+\epsilon)$-factor more than the distance from q to its true nearest neighbor. To achieve this result, we adapt the balanced box-decomposition (BBD) tree of Arya *et al.* [2] to oriented point sets. Specifically, we insert into the BBD tree the following set of $3n$ points in $\mathbb{R}^3$:

$$\big\{b, b', b'' \mid b \in B, \quad b' = (x_p, y_b, a_b + 2\pi), \quad b'' = (x_b, y_b, a_b - 2\pi)\big\}.$$

This takes $O(n \log n)$ preprocessing and it allows the BBD tree to respond to nearest neighbor queries with an approximation factor of $(1+\epsilon)$ while using the L_1-norm or L_2-norm as the distance metric, since the BBD is effective as an approximate nearest-neighbor data structure for these metrics. Indeed, this is the main reason why we are using these norms as our concrete examples of μ_i metrics. Each query takes $O(\log n)$ time, so computing a candidate Hausdorff distance for a given transformation takes $O(m \log n)$ time. Therefore, since we perform the pin step over n translations, the algorithm overall takes time $O(nm \log n)$. To analyze the correctness of this algorithm, we start with a simple observation that if we translate a point using a vector whose L_i-norm is d, then the distance between the translated point and its old position is d.

Lemma 7 *Let (x, y, a) be an element of O. Consider a transformation T_v in $\mathcal{T}$ where v is a translation vector. Let $T_v(x, y, a) = (x', y', a)$. If the L_i-norm of v is $\|v\|_i = d$, then $\mu_i\big((x, y, a), (x', y', a)\big) = d$, where $i \in \{1, 2\}$.*

Proof. First consider the case where $i = 1$. By definition of μ_1 and T_v,

$$\begin{aligned}
&\mu_1\big((x, y, a), (x', y', a)\big) \\
&= |x - x'| + |y - y'| + \min(|a - a|, 2\pi - |a - a|) \\
&= |v_x| + |v_y| \\
&= d.
\end{aligned}$$

Now consider the case where $i = 2$:

$$\begin{aligned}
&\mu_2\big((x, y, a), (x', y', a)\big) \\
&= \sqrt{(x - x')^2 + (y - y')^2 + \min(a - a, 2\pi - |a - a|)^2} \\
&= \sqrt{v_x^2 + v_y^2} \\
&= d.
\end{aligned}$$

Thus, for either case, the lemma holds. □

Theorem 8 *Let h_{opt} be $h_i(E(P), B)$ where E is the translation in $\mathcal{T}$ that attains the minimum of h_i. The algorithm above runs in time $O(nm \log n)$ and produces an approximation to h_{opt} that is at most $(2+\epsilon)h_{opt}$, for either h_1 and h_2, for any fixed constant $\epsilon > 0$.*

Proof. The ϵ term comes from the approximate nearest neighbor queries using the BBD tree, and expanding B to a set of size $3n$ by making a copy of each point in B to have an angle that is 2π greater and less than its original value. So it is sufficient to prove a 2-approximation using exact nearest neighbor queries (while building the BBD tree to return $(1 + \epsilon/2)$-approximate nearest neighbors). We prove this claim by a type of "backwards" analysis. Let E be a translation in $\mathcal{T}$ that attains the minimum of $h_i(E(P), B)$, and let $P' = E(P)$. Then every point $q \in P'$ is at most h_{opt} from its closest background point in B. That is, for all q in P', there exists b in B such that $\mu_i(q, b) \leq h_{\text{opt}}$. Let $b' \in B$ be the closest background point to the optimal position of p, where p is the point we choose in the first step of the algorithm. Thus,

$$\|(x_p, y_p) - (x_{b'}, y_{b'})\|_i \leq \mu_i(p, b') \leq h_{\text{opt}}.$$

Apply the translation T_v on P' so that p coincides with b', which is equivalent to moving every point's position by $\|(x_p, y_p)-(x_{b'}, y_{b'})\|_i$. Hence, by Lemma 7, all points have moved at most h_{opt}.

As all points in the pattern started at most h_{opt} away from a point in the background set and the translation T_v moves all points at most h_{opt}, all points in $T_v(P')$ are at most $2h_{\text{opt}}$ from a point in the background set B. Since our algorithm checks T_v as one of the translations in the second step of the algorithm, it will find a translation that is at least as good as T_v. Therefore, our algorithm guarantees an approximation of at most $2h_{\text{opt}}$, for either h_1 and h_2. □

A $(1 + \epsilon)$-Approximation Algorithm Under Translations Only. In this subsection, we utilize the algorithm from Appendix B to achieve a $(1 + \epsilon)$-approximation when we only allow translations. Suppose, then, that we are given two subsets of O, P and B, with $|P| = m$ and $|B| = n$, and our goal is to minimize $h_i(E(P), B)$ over translations E in $\mathcal{T}$. Our algorithm is as follows:

1. Run the base algorithm, BaseTranslate(P, B), from Appendix B, to obtain an approximation, $h_{apr} \leq A \cdot h_{\text{opt}}$.
2. For every $b \in B$, generate the point set
$$G_b = G\left(b, \frac{\epsilon\, h_{apr}}{A^2 - A}, \left\lceil \frac{A^2 - A}{\epsilon} \right\rceil\right)$$
for h_1 or
$$G_b = G\left(b, \frac{\epsilon\sqrt{2} h_{apr}}{A^2 - A}, \left\lceil \frac{A^2 - A}{\epsilon\sqrt{2}} \right\rceil\right)$$
for h_2. Let B' denote this expanded set of background points, i.e., $B' = \bigcup_{b \in B} G_b$, and note that if A is a constant, then $|B'|$ is $O(n)$.
3. Return the result from calling BaseTranslate(P, B'), but restricting the query step to finding nearest neighbors in B rather than in B'.

Intuitively, this algorithm uses the base algorithm to give us a first approximation for the optimal solution. We then use this approximation to generate a larger set of points from which to derive transformations to test. We then use this point set again in the base algorithm when deciding which transformations to iterate over, while still using B to compute nearest neighbors. The first step of this algorithm runs in time $O(nm \log n)$, as we showed. The second step takes time proportional to the number of points which have to be generated, which is determined by n, our choice of the constant ϵ, and the approximation ratio of our base algorithm A, which we proved is the constant $2+\epsilon$. The time needed to complete the second step is $O(n)$. In the last step, we essentially call the base algorithm again on sets of size m and $O(n)$, respectively; hence, this step requires $O(nm \log n)$ time.

Theorem 9 *Let h_{opt} be $h_i(E(P), B)$ where E is the translation in $\mathcal{T}$ that attains the minimum of h_i, for $i \in \{1, 2\}$. The algorithm above runs in time $O(nm \log n)$ and produces an approximation to h_{opt} that is at most $(1 + \epsilon)h_{opt}$, for either h_1 and h_2.*

Proof. Let E be the translation in $\mathcal{T}$ that attains the minimum of $h_i(E(P), B)$. Let P' be $E(P)$. Then every point $q \in P'$ is at most h_{opt} from the closest background point in B. By running the base algorithm the first time, we find $h_{apr} \leq A \cdot h_{\text{opt}}$, where A is the approximation ratio of the base algorithm. Now consider the point, $b' \in B$, that is the closest background to some pattern point $p \in P$. The square which encompasses $G_{b'}$ has a side length of $2h_{apr}$. This guarantees that p, which is at most h_{opt} away from b', lies within this square. As we saw from Lemma 3, this means that p is at most $\frac{\epsilon h_{apr}}{A^2 - A}$ away from its nearest neighbor in $G_{b'}$. Thus, if a transformation defined by the nearest point in B would move our pattern points at most $(A-1)h_{\text{opt}}$ from their optimal position, then using the nearest point in $G_{b'}$ to define our transformation will move our points at most $(A - 1)\frac{\epsilon h_{apr}}{A^2 - A} = \frac{\epsilon h_{apr}}{A} \leq \epsilon h_{\text{opt}}$. Therefore, our algorithm gives a solution that is at most $(1 + \epsilon)h_{\text{opt}}$ from optimal. □

C Translation, Rotation, and Scaling

In this appendix, we show how to adapt our algorithm for translations and rotations so that it works for translations, rotations, and scaling. The running times are the same as for the translation-and-rotation cases.

Base Algorithm Under Translation, Rotation and Scaling with Large Diameter. In this section we present an algorithm for solving the approximate oriented point-set pattern matching problem where we allow translations, rotations and scaling. This algorithm is an extension of the algorithm from Section 3 and similarly provides a good approximation ratio when the diameter of our pattern set is large. Given two subsets P and B of O, with $|P| = m$ and $|B| = n$, we wish to minimize $h_i(E(P), B)$ over all compositions E of one or more functions in $\mathcal{T} \cup \mathcal{R} \cup \mathcal{S}$. We perform the following algorithm:

Algorithm BaseTranslateRotateScaleLarge(P, B):

Find p and q in P having the maximum value of $\|(x_p, y_p) - (x_q, y_q)\|_2$.
for every pair of points $b, b' \in B$ **do**
  Pin step: Apply the translation, $T_v \in \mathcal{T}$, that takes p to b, and apply the rotation, $R_{p,\theta}$, that makes p, b', and q collinear. Then apply the scaling, $S_{p,s}$, that makes q and b' share the same position.
  Let P' denote the transformed pattern set, P.
  for every $q \in P'$ **do**
    Query step: Find a nearest-neighbor of q in B using the μ_i metric, and update a candidate Hausdorff distance accordingly.
  end for
  return the smallest candidate Hausdorff distance found as the smallest Hausdorff distance, $h_i(S_{p,s}(R_{p,\theta}(T_v(P))), B)$.
end for

This algorithm extends the algorithm presented in Section 3 so that after translating and rotating, we also scale the point set such that, after scaling, p and b have the same x and y coordinates, and q and b' have the same x and y coordinates. As with the algorithm presented in Section 3, this algorithm runs in $O(n^2 m \log n)$ time.

Theorem 10 *Let h_{opt} be $h_i(E(P), B)$ where E is the composition of functions in $\mathcal{T} \cup \mathcal{R} \cup \mathcal{S}$ that attains the minimum of h_i. The algorithm above runs in time $O(n^2 m \log n)$ and produces an approximation to h_{opt} that is at most $(6 + \sqrt{2}(2 + \pi/D) + \epsilon)h_{opt}$ for h_1 and at most $(4 + \sqrt{2}(2 + \pi/D) + \epsilon)h_{opt}$ for h_2.*

Proof. The additional ϵ terms come entirely from using approximate nearest neighbor queries, so it is sufficient to prove approximations which do not include the ϵ term using exact nearest neighbor queries. Particularly, we will prove a bound of $(6 + \sqrt{2}(2 + \pi/D))h_{\text{opt}}$ for h_1 and a bound of $(4 + \sqrt{2}(2 + \pi/D))h_{\text{opt}}$ for h_2.

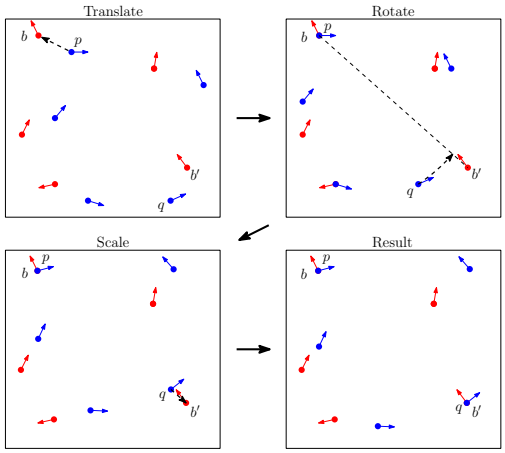


Figure 7: Illustration of the translation, rotation and scaling steps of the base approximation algorithm for translation, rotation and scaling in O when diameter is large.

Let E be the composition of functions in $\mathcal{T} \cup \mathcal{R} \cup \mathcal{S}$ that attains the minimum of $h_i(E(P), B)$. Let P' be $E(P)$. Because this algorithm is only an extension of the algorithm in Section 3 we can follow the same logic as the proof of Theorem 2 to see that after the translation and rotation steps, each point $p \in P'$ is at most Ah_{opt} away from a background point $b \in B$ where $A = 6 + \sqrt{2}\pi/D$ for h_1 and $A = 2 + \sqrt{2}(2 + \pi/D)$ for h_2. Now we need only look at how much scaling increases the distance our points have moved.

If $p, q \in P'$ are our diametric points after translation and rotation, and $p', q' \in B$ are the closest background points to the optimal position of p and q respectively, then let us define the point q_t as the position of q after translation, but prior to the rotation step. Now it is important to see that the points q, q' and q_t are three vertices of an isosceles trapezoid where the line segment $q_t q'$ is a diagonal of the trapezoid and the line segment qq_t is a base of the trapezoid. This situation is depicted in Figure 8. The length of the line segment qq' is equal to the distance that q will move when we scale P' so that q and q' share the same position. Because qq' is a leg of the trapezoid, the length of that leg can be no more than the length of the diagonal $q_t q'$. In the proof of Theorem 2, we showed that q_t is at most $2h_{\text{opt}}$ away from q' so this implies that the distance q moves from scaling is at most $2h_{\text{opt}}$.

Point q is the farthest point away from the point p that is the center for scaling. Thus, no point moved farther as a result of the scaling than q did, with respect to μ_2. For μ_1 it is possible that, if q moved a distance d, another point could have moved up to a distance $\sqrt{2}d$. Thus, we find that after scaling, any point in P' is at most $(A + 2\sqrt{2})h_{\text{opt}}$ and $(A + 2)h_{\text{opt}}$ from its nearest background point for μ_1 and μ_2 respectively. Because

this is a transformation that the algorithm checks, we are guaranteed at least this solution. Note that we assume p' and q' are not the same point. However if this is the case, then we know that $D \leq 2h_{\text{opt}}$ thus when we translate p to p' and scale q down to p' every point is within $(2\pi/D)h_{\text{opt}}$ of p', which is a better approximation than the case where $p' \neq q'$ under our assumption that D is large. □

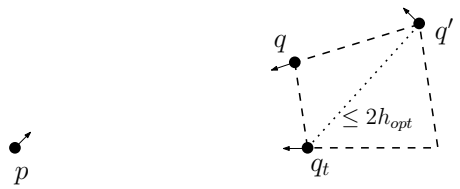


Figure 8: Illustration of the points q, q', and q_t forming three of the corners of an isosceles trapezoid, as described in the proof of Theorem 10

A $(1+\epsilon)$-Approximation Algorithm Under Translation, Rotation and Scaling with Large Diameter. In this subsection, we utilize the algorithm from Appendix C to achieve a $(1+\epsilon)$-approximation ratio when we allow translations, rotations, and scaling. Again, given two subsets of O, P and B, with $|P| = m$ and $|B| = n$, our goal is to minimize $h_i(E(P), B)$ over all compositions E of one or more functions in $\mathcal{T} \cup \mathcal{R} \cup \mathcal{S}$. We perform the following steps.

1. Run BaseTranslateRotateScaleLarge(P, B), from Appendix C, to obtain an approximation $h_{apr} \leq A \cdot h_{\text{opt}}$.
2. For every $b \in B$, generate the point set $G_b = G(b, \frac{h_{apr}\epsilon}{A^2-A}, \lceil \frac{A^2-A}{\epsilon} \rceil)$ for h_1 or $G_b = G(b, \frac{\sqrt{2}h_{apr}\epsilon}{A^2-A}, \lceil \frac{A^2-A}{\sqrt{2}\epsilon} \rceil)$ for h_2. Let B' denote the resulting set.
3. Run BaseTranslateRotateScaleLarge(P, B'), from Appendix C, but use the set B for the nearest-neighbor queries.

This algorithm uses the base algorithm to give us an indication of what the optimal solution might be. We then use this approximation to generate a larger set of points from which to derive transformations to test. We next use this point set in the base algorithm when deciding which transformations to iterate over, while still using B to compute nearest neighbors. The running time is $O(A^8 n^2 m \log n)$, which is $O(n^2 m \log n)$ for constant A.

Theorem 11 *Let h_{opt} be $h_i(E(P), B)$ where E is the composition of functions in $\mathcal{T} \cup \mathcal{R} \cup \mathcal{S}$ that attains the minimum of h_i. The algorithm above runs in time $O(A^8 n^2 m \log n)$ and produces an approximation to h_{opt} that is at most $(1+\epsilon)h_{opt}$ for both h_1 and h_2.*

Proof. Let E be the composition of functions in $\mathcal{T} \cup \mathcal{R} \cup \mathcal{S}$ that attains the minimum of $h_i(E(P), B)$. Let P' be $E(P)$. Then every point $q \in P'$ is at most h_{opt} from the closest background point in B. By running the base algorithm, we find $h_{apr} \leq Ah_{\text{opt}}$ where A is the approximation ratio of the base algorithm. Now consider the point $b' \in B$ which is the closest background to some pattern point $p \in P$. The square which encompasses $G_{b'}$ has a side length of $2h_{apr}$. This guarantees that p, which is at most h_{opt} away from b', lies within this square. As we saw from Lemma 3, this means that p is at most $\frac{\epsilon h_{apr}}{A^2-A}$ away from its nearest neighbor in $G_{b'}$. Thus, if a transformation defined by the nearest points in B would move our pattern points at most $(A-1)h_{\text{opt}}$ from their optimal position, then using the nearest points in $G_{b'}$ to define our transformation will move our points at most

$$(A-1)\frac{\epsilon h_{apr}}{A^2-A} = \frac{\epsilon h_{apr}}{A} \leq \epsilon h_{\text{opt}}.$$

Thus, the modified algorithm gives a solution that is at most $(1+\epsilon)h_{\text{opt}}$. □

Base Algorithm Under Translation, Rotation and Scaling with Small Diameter. In this subsection, we present an alternative algorithm for solving the approximate oriented point-set pattern matching problem where we allow translations, rotations and scaling. This algorithm is an extension of the algorithm from Section 3 and similarly provides a good approximation ratio when the diameter of our pattern set is small. Once again, given two subsets of O, P and B, with $|P| = m$ and $|B| = n$, we wish to minimize $h_i(E(P), B)$ over all compositions E of one or more functions in $\mathcal{T} \cup \mathcal{R}$. We perform the following algorithm:

Algorithm BaseTranslateRotateSmall(P, B):

Find p and q in P having the maximum value of $\|(x_p, y_p) - (x_q, y_q)\|_2$.
for every point $b \in B$ **do**
  1st *Pin:* Apply the translation, $T_v \in \mathcal{T}$, that takes p to b, and then apply the rotation, $R_{p,\theta}$, that makes p, b have the same orientation.
  Let P' denote the transformed pattern set, P.
  for each point p in P' and each $b' \in B$ **do**
    2nd *pin:* Apply the scaling, $S_{p,s}$, so that $\|(x_p, y_p) - (x_q, y_q)\|_2 = \|(x_b, y_b) - (x_{b'}, y_{b'})\|_2$
    Let P'' denote the transformed pattern set.
    for every $q \in P''$ **do**

Query step: Find a nearest-neighbor of q in B using the μ_i metric, and update a candidate Hausdorff distance accordingly.
end for
end for
return the smallest candidate Hausdorff distance found as the smallest Hausdorff distance, $h_i(S_{p,s}(R_{p,\theta}(T_v(P))), B)$.
end for

This algorithm extends the algorithm from Section 3 by scaling the point set for so that p, q, and b' form the vertices of an isosceles triangle. This requires a factor of n more transformations to be computed. Thus, the running time of this algorithm is $O(n^2 m \log n)$.

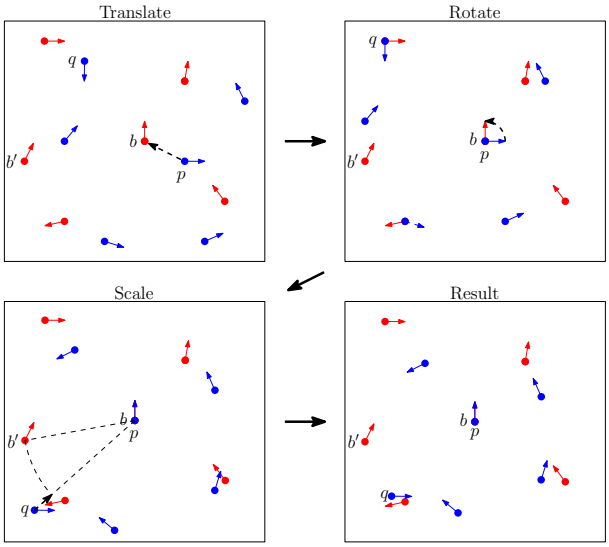


Figure 9: Illustration of the translation, rotation and scaling steps of the base approximation algorithm for translation, rotation and scaling in O when diameter is small.

Theorem 12 *Let h_{opt} be $h_i(E(P), B)$ where E is the composition of functions in $\mathcal{T} \cup \mathcal{R} \cup \mathcal{S}$ that attains the minimum of h_i. The algorithm above runs in time $O(n^2 m \log n)$ and produces an approximation to h_{opt} that is at most $((2+2\sqrt{2})(1+D)+\epsilon)h_{opt}$ for h_1 and at most $(4+2D+\epsilon)h_{opt}$ for h_2.*

Proof. The additional ϵ terms come entirely from using approximate nearest neighbor queries, so it is sufficient to prove approximations which do not include the ϵ term using exact nearest neighbor queries. Particularly, we will prove a bound of $(6+\sqrt{2}(2+\pi/D))h_{\text{opt}}$ for h_1 and a bound of $(4+\sqrt{2}(2+\pi/D))h_{\text{opt}}$ for h_2.

Let E be the composition of functions in $\mathcal{T} \cup \mathcal{R} \cup \mathcal{S}$ that attains the minimum of $h_i(E(P), B)$. Let P' be $E(P)$. Because this algorithm is only an extension of the algorithm in Section 3 we can follow the same logic as the proof of Theorem 5 to see that after the translation and rotation steps, each point $p \in P'$ is at most Ah_{opt} away from a background point $b \in B$ where $A = 2+\sqrt{2}D$ for h_1 and $A = 2+D$ for h_2. Now we need only look at how much scaling increases the distance our points have moved.

If $p, q \in P'$ are our diametric points after translation and rotation, and $p', q' \in B$ are the closest background points to the optimal position of p and q respectively, then let us define the point q_s as the position of q after scaling. The points q, q' and q_s are three vertices of an isosceles trapezoid where the line segment qq' is a diagonal of the trapezoid and the line segment $q_s q'$ is a base of the trapezoid. The length of the line segment qq_s is equal to the distance that q will move when we scale P'. Because qq_s is a leg of the trapezoid, the length of that leg can be no more than the length of the diagonal qq'. In the proof of Theorem 5, we showed that q is at most Ah_{opt} away from q' so this implies that the distance q moves from scaling is at most Ah_{opt}.

Point q is the farthest point away from the point p which is the center of our scaling. Thus, no point moves farther as a result of the scaling than q does, with respect to μ_2. For μ_1 it is possible that, if q moved a distance d, another point could have moved up to a distance $\sqrt{2}d$. Thus we find that after scaling, any point in P' is at most $(1+\sqrt{2})Ah_{\text{opt}}$ and $2Ah_{\text{opt}}$ from its nearest background point for μ_1 and μ_2 respectively. Because this is a transformation that the algorithm checks, we are guaranteed at least this solution. □

A $(1+\epsilon)$-Approximation Algorithm Under Translation, Rotation and Scaling with Small Diameter. In this subsection, we utilize the algorithm from Appendix C to achieve a $(1+\epsilon)$-approximation ratio when we allow translations, rotations, and scalings. Again, given two subsets of O, P and B, with $|P| = m$ and $|B| = n$, our goal is to minimize $h_i(E(P), B)$ over all compositions E of one or more functions in $\mathcal{T} \cup \mathcal{R} \cup \mathcal{S}$. We perform the following steps.

1. Run BaseTranslateRotateScaleSmall(P, B), from Appendix C to obtain an approximation $h_{apr} \leq A \cdot h_{\text{opt}}$.

2. For every $b \in B$, generate the point set $G_b = G(b, \frac{h_{apr}\epsilon}{2(A^2-A)}, \lceil \frac{2(A^2-A)}{\epsilon} \rceil)$ for h_1 or $G_b = G(b, \frac{h_{apr}\epsilon}{A^2-A}, \lceil \frac{A^2-A}{\epsilon} \rceil)$ for h_2. Let $B' = \bigcup_{b \in B} G_b$ denote the resulting set of points.

3. For every $b' \in B'$, generate the point set $C_{b'} = C(b', \frac{2(A^2-A)}{\pi h_{apr}\epsilon})$ for h_1 or $C_{b'} = C(b', \frac{\sqrt{2}(A^2-A)}{\pi h_{apr}\epsilon})$ for h_2. Let B'' denote the resulting set of points.

4. Run BaseTranslateRotateScaleSmall(P, B''), but use the points in B for nearest-neighbor queries.

This algorithm uses the base algorithm to give us an indication of what the optimal solution might be. We use this approximation to generate a larger set of points from which to derive transformations to test, but this time we also generate a number of different orientations for those points as well. We then use this point set in the base algorithm when deciding which transformations to iterate over, while still using B to compute nearest neighbors. The running time of this algorithm is $O(A^{12}n^2 m \log n)$.

Theorem 13 *Let h_{opt} be $h_i(E(P), B)$ where E is the composition of functions in $\mathcal{T} \cup \mathcal{R}$ that attains the minimum of h_i. The algorithm above runs in time $O(A^{12}n^2 m \log n)$ and produces an approximation to h_{opt} that is at most $(1+\epsilon)h_{opt}$ for both h_1 and h_2.*

Proof. Let E be the composition of functions in $\mathcal{T} \cup \mathcal{R} \cup \mathcal{S}$ that attains the minimum of $h_i(E(P), B)$. Let P' be $E(P)$. Then every point $q \in P'$ is at most h_{opt} from the closest background point in B. By running the base algorithm, we find $h_{apr} \leq Ah_{\text{opt}}$ where A is the approximation ratio of the base algorithm. Now consider the point $b' \in B$ which is the closest background to some pattern point $p \in P$. The square which encompasses $G_{b'}$ has a side length of $2h_{apr}$. This guarantees that p, which is at most h_{opt} away from b', lies within this square. As we saw from Lemma 3, this means that p is at most $\frac{\epsilon h_{apr}}{2(A^2-A)}$ away from its nearest neighbor g in $G_{b'}$ with respect to the $L1$-norm, and at most $\frac{\epsilon h_{apr}}{\sqrt{2}(A^2-A)}$ with respect to the $L2$-norm. For this point g, there are a number of points in C_g which are at the same position but with different orientation. For some point c in C_g, the orientation of point p is within an angle of at most $\frac{h_{apr}\epsilon}{2(A^2-A)}$ for h_1 and at most $\frac{h_{apr}\epsilon}{\sqrt{2}(A^2-A)}$ for h_2. If we combine together the maximum difference in position between p and c, and the maximum difference in orientation between p and c, then we see that for both μ_1 and μ_2, the distance between p and c is at most $\frac{h_{apr}\epsilon}{A^2-A}$. As we explain at the beginning of this section, if a transformation defined by the nearest points in B would move our pattern points at most $(A-1)h_{\text{opt}}$ from their optimal position, then using the nearest points in C_g to define our transformation will move our points at most $(A-1)\frac{\epsilon h_{apr}}{A^2-A} = \frac{\epsilon h_{apr}}{A} \leq \epsilon h_{\text{opt}}$. Thus the modified algorithm gives a solution that is at most $(1+\epsilon)h_{\text{opt}}$. □

As with our methods for translation and rotation, we can compute in advance whether we should run our algorithm for large diameter point sets or our algorithm for small diameter point sets. For h_1, we compare the expressions $6+\sqrt{2}(2+\pi/D)$ and $(2+2\sqrt{2})(1+D)$, and we find that the two expressions are equal at $D^* \approx 1.46$. For h_2, we compare $4+\sqrt{2}(2+\pi/D)$ and $4+2D$ to find that they are equal at $D^* \approx 2.36$. Using D^* as the deciding value allows us to then find a transformation in $\mathcal{T} \cup \mathcal{R} \cup \mathcal{S}$ that achieves a $(1+\epsilon)$-approximation, for any constant $\epsilon > 0$, in $O(n^2 m \log n)$ time.

D Experiments

In reporting the results of our experiements, we use the following labels for the algorithms:

- *GR*: the non-oriented translation and rotation algorithm from Goodrich *et al.* [7],
- LD_{h_1/h_2}: the base version of the large diameter algorithm using either the h_1 or h_2 distance metric,
- SD_{h_1/h_2}: the base version of the small diameter algorithm using either the h_1 or h_2 distance metric.

These algorithms were implemented in C++ (g++ version 4.8.5) and run on a Quad-core Intel Xeon 3.0GHz CPU E5450 with 32GB of RAM on 64-bit CentOS Linux 6.6.

Accuracy Comparison. We tested the ability of each algorithm to identify the orginal point set after it had been slightly perturbed. From set of randomly generated oriented background point sets, one was selected and a random subset of the points in the set were shifted and rotated by a small amount. Each algorithm was used to match this modified pattern against each of the background point sets and it was considered a success if the background set from which the pattern was derived had the smallest distance (as determined by each algorithm's distance metric). Figure 10 shows the results of this experiment under two variables: the number of background sets from which the algorithms could choose, and the size of the background sets. Each data point is the percentage of successes across 1000 different pattern sets.

In every case, the oriented algorithms are more successful at identifying the origin of the pattern than GR. LD was also more successful for each distance metric than SD.

Performance Comparison. We also compared the performance of the LD and SD algorithms against GR as we increased the pattern size and the background size. The most significant impact of increasing the background size is that the number of nearest neighbor queries increase, and thus the performance in this case is dictated by quality of the nearest neighbor data structure used. Therefore in Figure 11 we use the number of nearest neighbor queries as the basis for comparing performance. As the FD and GR algorithms only differ in how the nearest neighbor is calculated,

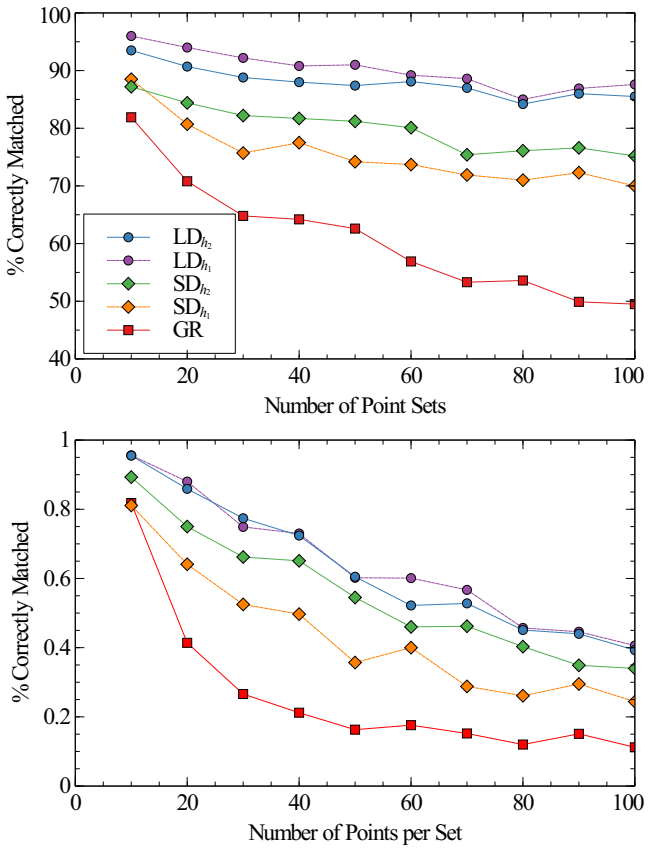


Figure 10: Results of Accuracy Comparison

they both perform the same number of queries while the SD algorithm performs significantly fewer nearest neighbor queries.

For pattern size, we compared running time and the results are shown in Figure 12. In this case, LD is slower than GR, while SD is signifcantly faster than either of the others.

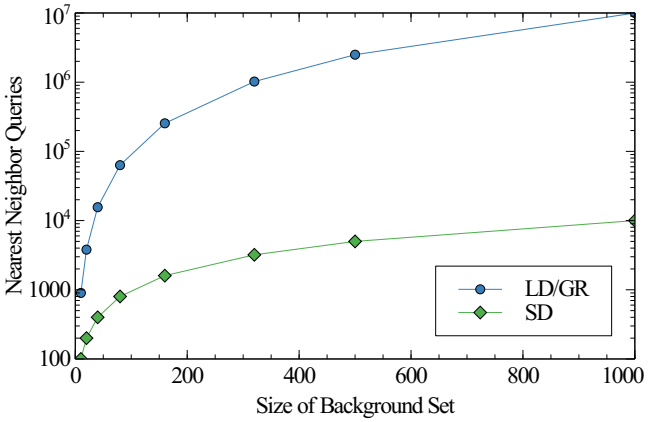


Figure 11: Comparison of nearest neighbor queries as function of background size

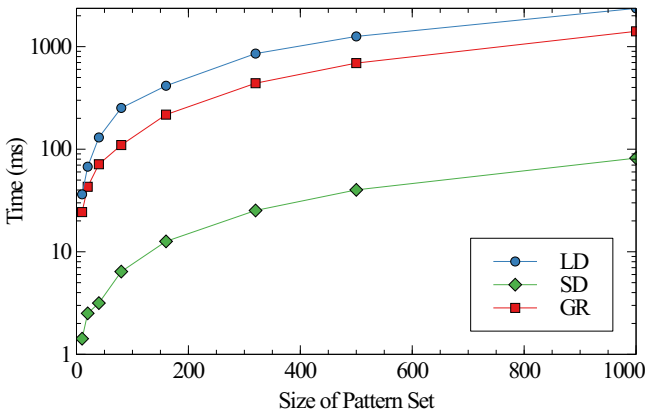


Figure 12: Comparison of running time as a function of pattern size

The Computational Complexity of Finding Hamiltonian Cycles in Grid Graphs of Semiregular Tessellations

Kaiying Hou* Jayson Lynch†

Abstract

Finding Hamitonian cycles in square grid graphs is a well studied and important question. Recent work has extended these results to triangular and hexagonal grids, as well as further restricted versions such as solid or thin grids [7, 8, 4]. In this paper, we examine a class of more complex grids, as well as investigate the problem with restricted types of paths. We investigate the hardness of Hamiltonian cycle problem in grid graphs of semiregular tessellations. We give NP-hardness reductions for finding Hamiltonian paths in grid graphs based on all eight of the semiregular tessellations. Next, we investigate variations on the problem of finding Hamiltonian Paths in grid graphs when the path is forced to turn at every vertex. Related problems were considered in[6]. We show deciding if 3D square grid graphs admit a Hamiltonian cycle is NP-complete, even if the height of the grid is restricted to 2 vertices. We give a polynomial time algorithm for deciding if a solid square grid graph admits a Hamiltonian cycle which visits vertices at most twice and turns at every vertex.

1 Introduction

The Hamiltonian cycle problem (HCP) in grid graphs has been well studied and has led to application in numerous NP-hardness proofs for problems such as the milling problem [2], Pac-Man [10], finding optimal solutions to a Rubik's Cube [3], and routing in wireless mesh networks [11]. The problem has been of interest to computer scientists for many years and recently a number of variations on the problem have been investigated. Itai, Papadimitriou, and Szwarcfiter proved that the HCP in square grid is NP-complete by reducing from the HCP in planar max-degree-3 bipartite graphs [7]. More recently, the HCPs in triangular and hexagonal grid were shown to be NP-complete[8]. This paper also introduced several new constrains on grid graphs, such as being thin or polygonal. Several of those open problems were solved by Demaine and Rudoy [4] by reducing from 6-Regular Tree-Residue Vertex Breaking problem (TRVB) [5]. These papers also show results on grid graphs with restrictions such as thin, polygonal, and solid. With all the interest in the computational complexity of the HCP in grid graphs, it is reasonable to ask whether we can generalize or adapt these results to different types of grids. In addition, we investigate the notion of angle-restricted tours, studied in [6], in the context of grid graphs. We give both algorithms and hardness proofs for finding Hamiltonian paths with this 'always-turning' constraint.

Although the hardness of the HCP in semiregular grids seems like an abstract question, it has many possible applications. Grids are natural structures that things may be formatted into. For example, the layout of buildings or modular structures used in space may form a network that follow the patterns of semiregular, or more general, grids. If certain locations in such networks need to be visited for maintenance, and one wants an optimal route, then this is well-modeled by the Hamiltonian path problem. Our reductions both give insight into what sorts of regular structures will be difficult to find optimal paths for, as well as ways of potentially transferring other efficient algorithms to these new problems. Finally, the results and techniques in this paper may be useful in proving hardness of other problems by reducing from HCPs in semiregular grids.

The always-turning Hamiltonian path problems also has some relation to more concrete questions. First, one can see the always-turning constraint as path planning in a world with reflections at fixed angles and locations. One may be routing optics to various locations on an optics table. Reflections of 45 degrees in a grid-based world are also a common element in puzzles and games. In addition, there has been study of problems which try to minimize the number of turns taken in a covering tour [1]. In many ways this can be seen as the opposite, modeling a case where turning is significantly easier than continuing straight.

A full version of this paper is available on the arXiv[1].

Results In Section 3 we extend the class of grid graphs studied to those based on semiregular tessellations. There are a total of eight semiregular tessellations [12], which are shown in Figure 1. For all eight semiregular tessellations we show the corresponding Hamiltonian cycle problem in the induced grid graph is NP-complete.

*Phillips Academy Andover, khou@andover.edu

†MIT Computer Science and Artificial Intelligence Laboratory, jaysonl@mit.edu

[1] arXiv:1805.03192

We show hardness by reducing from three NP-complete problems: HCP in planar max-degree-3 bipartite graphs [7], HCP in hexagonal grids [8], and TRVB [4].

In Section 4 we examine the question of Hamiltonian paths which turn at every vertex. We show this problem is hard in 3D square grid graphs. We also show it is easy in triangular grids with only 60 degree turns but hard in triangular grids when 120 degree turns are allowed. Finally, we examine a problem in square grids where a path must visit every vertex at least once, must turn at every vertex, and cannot reuse edges. We give a linear time algorithm for solving this double turning problem in solid square grid graphs.

2 Definitions

A *tessellation* is a tiling of a plane with polygons without overlapping. A *semiregular tessellation* is a tessellation which is formed by two or more kinds regular polygons of side length 1 and in which the corners of polygons are identically arranged. Figure 1 depicts part of each of the eight semiregular tessellations.

An infinite lattice of a semiregular tessellation is a lattice formed by taking the vertices of the regular polygons in the tessellation as the points of the lattice. A graph G is *induced* by the point set S if the vertices of G are the points in S and its edges connect vertices that are distance 1 apart. A *grid graph* of a semiregular tessellation, or a *semiregular grid*, is a graph induced by a subset of the infinite lattice formed by that tessellation. Call the infinite graph induced by the full lattice a *full grid*.

A *pixel* is the simple cycle bounding a face in a grid graph that contains the same bounding edges and vertices as the corresponding face in the full grid. Thus a pixel can be thought of as a cycle in a graph which bounds precisely one tile in the original tessellation. We may use pixel interchangeably to refer to the bounding cycle, the face bound by the cycle, or the set of vertices around that face. A *solid grid graph* is one in which every bounded face is a pixel.

A *Hamiltonian cycle* is a cycle that passes through each vertex of a graph exactly once. The *Hamiltonian cycle problem*, sometimes abbreviated as HCP, asks that given a graph, whether or not that graph admits a Hamiltonian cycle. The HCP in a semiregular tessellation asks, given a grid graph of that tessellation, whether it admits a Hamiltonian cycle.

3 Finding Hamiltonian Paths in Semi-Regular Tessellations is NP-Complete

This section shows the NP-completeness of HCPs in all eight semiregular tessellations. There are three NP-complete problems that we reduce from: the HCP in hexagonal grid, the HCP in planar max-degree-3 bipartite graphs, and the TRVB problem. We give some representative reductions and leave the rest of the constructions to Appendix A.

Theorem 1 *The HCP in grid graphs of the 3.4.6.4 tessellation is NP-complete.*

Proof. We will reduce from the HCP in hexagonal grids. Given a hexagonal grid graph G', we will construct a grid graph G of the 3.4.6.4. tessellation in this way: for every edge in G' we add the edge gadget shown in Figure 2 to G and for every vertex in G' we add the vertex gadgets shown in Figure 3 to G. Since the 3.4.6.4 tesselation has scaled versions of the translational symmetries of the hexagonal grid, picking an embedding for our construction is straightforward. An example can be seen in Figure 4. Since the hexagonal grid G' is bipartite, we can design different vertex gadgets for each end of the edges. We call the two classes of vertices even and odd vertices.

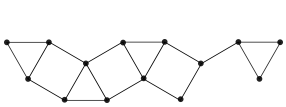


Figure 2: Edge gadget

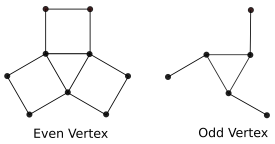


Figure 3: Vertex gadgets

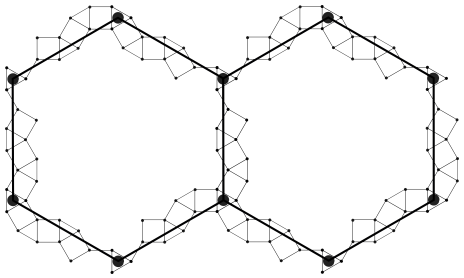


Figure 4: A simulated graph

Now, we will show that the original graph G' has a Hamiltonian cycle C' if and only if the simulated graph G has a Hamiltonian cycle C. If the G' has a Hamiltonian cycle C', for any taken edge in it, we go through the corresponded edge gadget in G with the cross path in Figure 5; for any untaken edge, we go through the corresponded edge gadget with the return path. Because the simulated vertices in G are triangles (K_3), there is always a path to take the simulated vertex by entering from one point and leaving at the other. Therefore, if there is a Hamiltonian cycle C' in the original graph G', then there is a Hamiltonian cycle

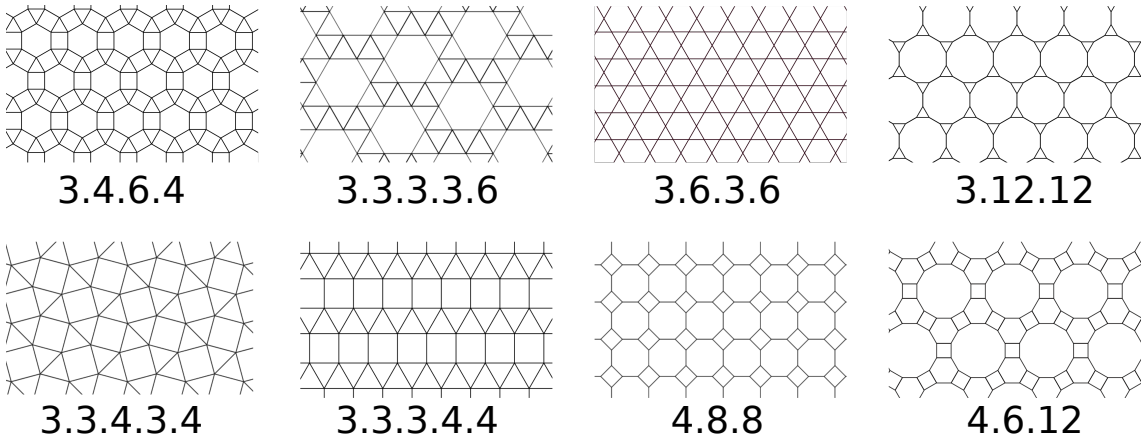


Figure 1: The eight semiregular tessellations. It is common to refer to them by the size of the faces while walking around a vertex.

C in the simulated graph G.

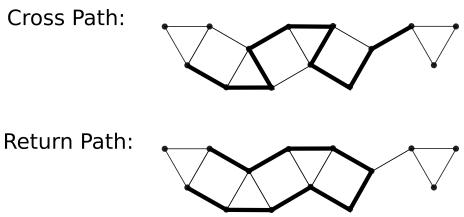


Figure 5: Two kinds of paths in the 3.4.6.4 edge gadget

The essential difference between the cross path and the return path is that a cross path starts and finishes at different ends of an edge while the return path starts and finishes in the same end. Note that the return and cross paths are the only two paths which go through the edge gadget and visit all of its vertices. The odd vertex gadget is connected to the edge gadget through a single edge connection which prevents the return path from entering the odd vertex gadget. If a Hamiltonian cycle C exists in the simulated graph G, each odd vertex gadget in G must be connected to two cross paths and the even vertex gadgets can either be connected to two cross paths or two cross paths and a return path. Then, we can find a cycle C' in the original graph G' by making each edge gadget with a cross path in C a taken edge in C'. Thus, if there is a cycle C in the simulated graph G, there is a cycle C' in the original graph G'. This way, we showed the original graph G' has a Hamiltonian cycle C' if and only if the simulated graph G has a Hamiltonian cycle C.

□

Theorem 2 *The HCP in grid graphs of the 3.3.3.3.6 tessellation is NP-complete.*

Proof. Similar to the 3.4.6.4 tessellation in Theorem 1, see Appendix A for details. □

Theorem 3 *The HCP in grid graphs of the 3.6.3.6 tessellation is NP-complete.*

Proof. Similar to the 3.4.6.4 tessellation in Theorem 1, see Appendix A for details. □

Theorem 4 *The HCP in grid graphs of the 3.12.12 tessellation is NP-complete.*

Proof. See Appendix A for the proof. □

Theorem 5 *The HCP in grid graphs of the 3.3.4.3.4 tessellation is NP-complete.*

Proof. We will reduce from HCP in planar max-degree-3 bipartite graphs. First observe that this tessellation can be viewed as a square grid with some extra diagonals. We directly use the gadgets of the square grid proof in the 1982 paper for constructing G [7]. The edge, even vertex and odd vertex gadgets are shown below. Note that these gadgets are identical to the square grid gadgets except they have some extra edges. In creating the simulated graph G based on a planar max-degree-3 bipartite graph G', we go through the same process as that in the square grid reduction: first create a parity-preserving embedding of the max-degree-3 bipartite graph; then replace the edges and vertices of the embedding with respective gadgets [7].

There are only two kinds of traversals for the edge gadget: cross paths and a return paths. Although there is more than one kind of cross path due to the extra edges, they have the essential characteristic of starting from one end of the gadgets and finishing at the other end (unlike the return path that begins and finishes at the same end). Another difference from the square grid reduction is that the odd vertex

Figure 6: Edge gadget for 3.3.4.3.4

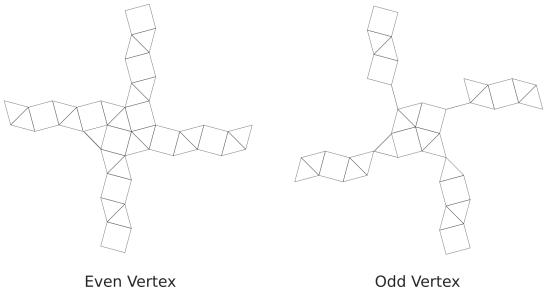


Figure 7: Vertex gadgets for 3.3.4.3.4

gadgets connect to the bottom edge gadget through a single point rather than a single edge as the other edge gadgets. This single point connection also prevents a return path from entering the odd vertex gadget. We call the single edge and single point connections that the path only enters and exits odd vertices once. Since the graph is bipartite, this forces the other two edges to be return paths, ensuring our simulated path can only enter and exit each vertex once.

□

Theorem 6 *The HCP in grid graphs of the 3.3.3.4.4 tessellation is NP-complete.*

Proof. See Appendix A. □

We now show the HCPs in the 4.8.8 tessellation and the 4.6.12 tessellation are NP-complete by reducing from the Tree-Residue Vertex Breaking (TRVB) Problem studied in [5]. Here, *breaking* a degree-n vertex means turning the vertex into n degree-1 vertices that are at the ends of the n edges. The TRVB problem asks that given a planar multigraph M and with some of its vertices marked breakable, is it possible to break some of the breakable vertices so that the resulting graph is a single connected tree. N-Regular Breakable Planar TRVB problem asks that given a planar multigraph with all the vertices degree-n and breakable, is it possible to produce a tree from breaking some vertices. The HCPs in these section reduce from 4-Regular Breakable Planar TRVB problem and 6-Regular Breakable Planar TRVB problem, both of which are NP-complete [5]. The reduction works in this fashion: for any graph M, we will construct a grid graph G of the tessellation so that G has a Hamiltonian cycle if and only if M is breakable.

Theorem 7 *The HCP in grid graphs of the 4.8.8 tessellation is NP-complete.*

Proof. We reduce from the 4-Regular Breakable Planar TRVB problem. When constructing a grid graph G of the 4.8.8 tessellation based on M, we first make a square grid embedding of M, using a method such as the one described in [9]. Then, for each vertex of M, we use the vertex gadget in Figure 9. For the edges in the embedding, we use the edge gadget formed by the boundary vertices of a three-octagon wide strip, as shown in Figure 8. Notice that the edge gadget can shift and turn easily. Due to this flexibility, we can form a graph G based on the embedding using the gadgets.

Now, we will show the constructed graph G has a

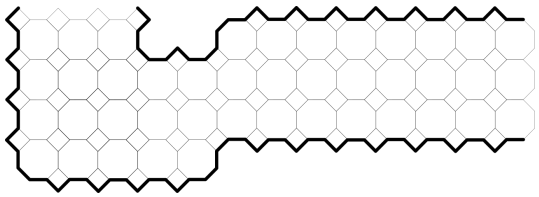


Figure 8: Edge gadget with a turn for 4.8.8. Only edges in bold are present in the gadget.

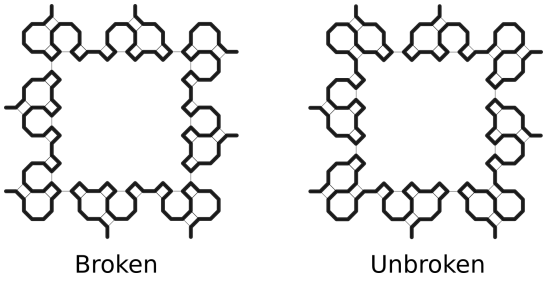


Figure 9: Solutions for the 4.8.8 vertex gadget. The valid paths are shown in bold.

Hamiltonian cycle if and only if M is breakable. Noticed that if G has a cycle C, both sides of the edge gadgets must be in C and the freedom is only in how to traverse the vertex gadgets. Figure 9 shows two solution to the vertex gadget. The four edge gadgets connect to the vertex on its four sides. Each edge gadgets has two separate paths of vertices that go into the vertex gadget. We call each of these a strand. Note that there are eight single connection edges in the vertex gadgets, each of which is in between a pair of adjacent series. If a cycle exists and a path comes in from a strand, the path must enter one of the two adjacent single edge connections and then connect with the path coming in from another strand. Thus, for a vertex gadget, there are only two kinds of solution: one that has two strands of the same edge connected or one that has two strands of two adjacent edges connected. The first kind is illustrated by the solution on the left, which correspond to a broken vertex in M while the second kind is illustrated by the solution on the right side, which correspond to a unbro-

ken vertex in M. To show that G has a Hamiltonian cycle if and only if M is breakable, we apply the reasoning used in the 2017 paper [4]. If M is breakable, then for every broken vertex in M, we traverse through the corresponding vertex gadget using the broken solution; for every unbroken vertex, we traverse through the corresponding vertex gadget using the unbroken solution. Note that after this procedure, the graph produced by breaking M is the same as the region inside the edge gadgets in G. If the graph produced by breaking M is indeed a tree, which is connected and acyclic, then the region inside the edges must also be connected and hole-free, which shows that there is a Hamiltonian cycle. If there is a Hamiltonian cycle in G, the region inside must by connected and hole-free, which then show that the graph M can be broken down to a tree. □

Theorem 8 *The HCP in grid graphs of the 4.6.12 tessellation is NP-complete.*

Proof. We prove that the HCP in 4.6.12 Tessellation is NP-complete by reducing from the 6-Regular Breakable Planar TRVB problem. When constructing a grid graph G in 4.8.8 tessellation based on the multigraph M, we first embed the multigraph in the triangular grid. Then, we use the vertex gadget shown in Figure 11 for every vertex in M and the edge gadget shown in Figure 10 for the edges in M. The edge gadget only includes the boundary vertices of the shape depicted in Figure 10. Because the turning demonstrated in 10 can have turning of 60 and 120 degrees, we can construct the induced subgraph G based on the triangular grid embedding.

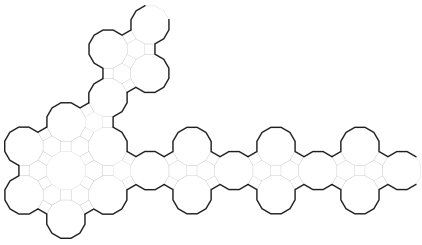


Figure 10: Edge gadget with a turn for 4.6.12

Now, we will show why the constructed graph G has a Hamiltonian cycle if and only if M is breakable. The traversals of the edge gadgets of 4.6.12 tessellation are already set and the only freedom is in how to traverse the vertex gadgets. The six edge gadgets connect to the vertex gadget on the six sides and each edge gadget consists of two strands of vertices. As mentioned in the 4.8.8 tessellation, because of the single edge connections between each pair of adjacent strands, there are only two kinds of traversals for a vertex gadget: the one that has two strands of the same edge connected or the one

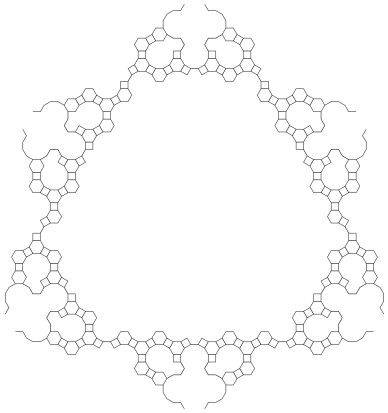


Figure 11: Vertex gadget for 4.6.12

that has two strands of two adjacent edges connected. The first kind is illustrated by the solution in Figure 20, which corresponds to a broken vertex in M. The second kind is illustrated by the solution in Figure 21, which corresponds to an unbroken vertex in M. Just as the argument in 4.8.8 tessellation proof states, the region inside the edge gadgets represents the produced graph after breaking M. □

4 Hamiltonian Cycles with Turns

In this section we explore whether grid graphs contain Hamiltonian cycles which turn at every vertex. In [6], Fekete and Woeginger give a near linear algorithm for finding Hamiltonian paths among a set of points in the plane when the path must turn by 90° at every vertex. We show that this angle-restricted tour problem becomes NP-complete when generalized to 3D, even when we restrict to 3D square grid graphs whose height is only two vertices. We also characterize the complexity of finding always-turning Hamiltonian cycles in triangular grids. See Appendix C and D for these results.

We also investigate a version where each vertex can be visited at most twice, as long as no edges in the cycle overlap. We give linear time algorithms for finding always-turning cycles, as well as double visiting cycles in solid square grid graphs. This question initially came to our attention as special cases of finding Hamiltonian cycles in the 4.8.8 tessellation. There are clear reductions between various problems in this section and restricted versions of that problem in which all vertices around a square pixel are included if any one is included. Although this did not lead to our eventual hardness proof we found the problem to be interesting and well motivated on its own. One can look at this problem as mirroring a problem laid out in a grid where movement is reflected by barriers at 45 degree angles. One can also think of this as counterpart to the discrete milling problem, where our cost function makes turns much less

expensive than straight paths.

Theorem 9 *There is a polynomial time algorithm for determining whether a square grid graph admits a Hamiltonian Path which turns at every vertex.*

Proof. See Appendix B for the proof. □

Theorem 10 *The Always-Turning Hamiltonian Path problem in 3D square grid graphs is NP-complete even if the height of the grid is restricted to be* 2 *vertices.*

Proof. The proof is by reduction from from planar max-degree-3 graphs and follows the structure of the original square gird proof. See Appendix C for details. □

4.1 Double Turning in Solid Graphs

Initially inspired by the HCP in the 4.8.8 tessellation grid we consider the following problem. We define the Double Turning Hamiltonian Cycle Problem to be the following: Given a square grid graph, does there exist a cycle in that graph which visits every vertex at least once, never traverses an edge more than once, and turns at every vertex? In particular, this allows the path to visit degree-4 vertices twice, taking a different turn each time. If we consider the square and octagon tessellation in which we have every vertex around a square pixel if any vertex is present around that pixel, then one can see these are equivalent problems.

This section will begin by observing some useful properties of the Hamiltonian path. Then we will connect those to properties of the graph to show that these graphs have a property we call a checkering. Next, we demonstrate that spanning trees of the checkering correspond to Hamiltonian cycles in our graph. Finally, we argue that all of these properties can be checked in polynomial time.

Lemma 11 *If a solid graph admits a double turning Hamiltonian cycle, it also admits such a cycle where all degree-4 vertices are visited twice.*

Proof. If a degree-4 vertex has only one visit then two adjacent edges must be in the path and two adjacent edges must not be in the path. Let us consider some properties of the empty edges. Since each edge must have a partner in the vertex, then each 'path' of empty edges must either connect to degree-3 vertices or be in a cycle. Degree-3 vertices only occur on the boundary since this grid is solid. Thus if we have a path from one degree-3 vertex to another, then the path has gone from one boundary to another and has thus separated two parts of our graph unless those boundary edges were adjacent. This means the empty edges must form a cycle. If this cycle contains any vertices on its interior, then those vertices are disconnected and the cycle cannot be part of a valid solution. Finally a vertex cannot be visited by an empty path more than once, otherwise it is never visited in the actual path. The only cycles in a grid which obey these constraints are single pixels. If there is a pixel without edges, then we pick one vertex arbitrarily to extend the cycle into the pixel. □

With this lemma we can now restrict our examination to the case where all degree-4 vertices are visited twice.

First, we notice that the graph must have even parity on all external boundaries. Given this parity constraint and that the graph is solid, we know that if the graph contains a Hamiltonian cycle then it is composed of some number of full pixels, possibly connected at the corners[2]. We now wish to consider an alternate view of this grid graph. Call the *checkerboard* of this graph the set of alternating pixels in the graph starting with the upper left. We call the other pixels the odd checkering.

Now we will imagine connecting the pixels in the checkerboard and show that the existence of a Hamiltonian cycle depends on its properties. Consider the degree-4 vertices, all of which are visited twice by our prior lemma. There are two configurations of paths, each one connects two diagonally adjacent pixels and separates the other two. We can now think of every degree-4 vertex of our graph as either connecting two adjacent checkered pixels or two adjacent odd checkered pixels. We call this connection a checkering graph.

Lemma 12 *The Double Turning Hamiltonian Cycle Problem in solid grid graphs admits a Hamiltonian cycle if and only if it has a valid checkering and it admits a checkering graph which is a single connected tree.*

Proof. First, we will prove that we can construct a Hamiltonian cycle from a spanning tree of a checkering of the graph. To do so, we will simply visit each of the vertices in an Euler tour order around the spanning tree. Each vertex in the original graph corresponds to a potential edge location in the checkering. We use this term loosely as there may not be vertices in the checkering to connect to. Around each pixel we give the vertices a clockwise ordering. From a vertex we check if that vertex corresponds to an edge in the checkering spanning tree. If not we move clockwise around our current pixel. If it is an edge, we instead consider the pixel we connect to to be our current pixel and move clockwise around that one. We know that every vertex is adjacent to exactly one or two pixels in the checkering and accordingly is visited either once or twice. This process creates a path which never crosses the spanning tree and is free to continue around the entire spanning tree, thus

[2]These corner connections are local cuts and what prevent this graph from being categorized as polygonal.

resulting in a single cycle as desired. An example can be seen in the Appendix.

Now we will argue that if no spanning tree exists then no Hamiltonian cycle exists. If there are two disconnected components of the checkering then this means either there are disconnected pixels in which case either the graph itself is disconnected, the graph is not checkerable, or along all connecting vertices their checkerboard edges were assigned to the odd checkering. The graph must obviously be connected and by the prior parity argument it must be checkerable for it to admit a double turning Hamiltonian cycle. This leaves the case where we have assigned edges in our checkering graph such that it is disconnected. To do so means we would have a path through the odd checkering which separates the two parts of our checkering graph. In the same way that a Hamiltonian path cannot cross edges in the checkerboard graph, it also cannot cross edges in the odd checkering. Thus we have a vertex cut with no paths passing through it, meaning we either have more than one cycle or miss some vertices in our path. □

Now we merely need to show that the checkering and its spanning tree can be found in linear time.

Theorem 13 *The Double Turning Hamiltonian Cycle Problem in solid grid graphs can be solved in linear time.*

Proof. By Lemma 12 we see that deciding if the graph is checkerable and finding a spanning tree of the checkering suffices. See Appendix E for analysis. □

5 Conclusion

In this paper, we have shown that the HCPs in all of the eight semiregular tessellations are NP-complete and shown new upper and lower bounds on finding Hamiltonian paths which always turn in various grids. These generalizations we investigated lead to a large variety of open questions. Most of the restrictions from [8] also apply to the semi-regular tessellation graphs and it would be interesting to know whether solid or super-thin versions of these graphs also admit polynomial time algorithms. We also leave open the questions of the complexity of double turning paths in square grid graphs. In addition, the dual graphs of the tessellation graphs are an obvious next target because of their regular structure and connection to discrete motion planning. One could also look at other general classes of tessellation graphs allowing more general shapes, including higher dimensional structures. We are also rather curious whether anything can be shown about finding Hamiltonian paths in aperiodic tessellation graphs.

There are also other interesting extensions of the always turning paths. The polynomial time proofs only hold for grids in the plane, however the arguments seem like they might lead to algorithms for grids on surfaces of bounded genus. It would be interesting to explore the question on square grids on a torus or other topologically distinct surfaces. In addition, the algorithm for finding double turning Hamiltonian cycles in solid square grids looks related to the number of spanning trees of certain types of graphs, as well as the potential removal of squares of edges. It would be interesting to know if it is computationally tractable to count the number of distinct double turning Hamiltonian cycles and whether it bears nice relation to other combinatorial problems. Finally, this notion of restricted turn paths can be applied to other grids or graphs with appropriate geometry.

Acknowledgments. We want to thank Professor Erik Demaine for useful discussion and feedback on this research.

References

[1] Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Sándor P. Fekete, Joseph S. B. Mitchell, and Saurabh Sethia. Optimal covering tours with turn costs. *SIAM Journal on Computing*, 35(3):531–566, 2005. URL: `https://doi.org/10.1137/S0097539703434267`, `arXiv:https://doi.org/10.1137/S0097539703434267`, `doi:10.1137/S0097539703434267`.

[2] Esther M Arkin, Sándor P Fekete, and Joseph SB Mitchell. The lawnmower problem. In *CCCG*, pages 461–466, 1993.

[3] Erik D Demaine, Sarah Eisenstat, and Mikhail Rudoy. Solving the rubik's cube optimally is np-complete. *arXiv preprint arXiv:1706.06708*, 2017.

[4] Erik D. Demaine and Mikhail Rudoy. Hamiltonicity is hard in thin or polygonal grid graphs, but easy in thin polygonal grid graphs. *CoRR*, abs/1706.10046, 2017. URL: `http://arxiv.org/abs/1706.10046`.

[5] Erik D. Demaine and Mikhail Rudoy. Tree-residue vertex-breaking: a new tool for proving hardness. *CoRR*, abs/1706.07900, 2017. URL: `http://arxiv.org/abs/1706.07900`.

[6] Sándor P Fekete and Gerhard J Woeginger. Angle-restricted tours in the plane. *Computational Geometry*, 8(4):195–218, 1997.

[7] Alon Itai, Christos H. Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982. URL: `http://dx.doi.org/10.1137/0211056`, `arXiv:http://dx.doi.org/10.1137/0211056`, `doi:10.1137/0211056`.

[8] Esther M. Arkin, Sándor P. Fekete, Kamrul Islam, Henk Meijer, Joseph Mitchell, Yurai Núñez Rodríguez, Valentin Polishchuk, David Rappaport, and Henry Xiao. Not being (super)thin or solid is hard: A study of grid hamiltonicity. 42:582–605, 08 2009.

[9] Markus Schäffter. Drawing graphs on rectangular grids. *Discrete Applied Mathematics*, 63(1):75–89, 1995. URL: http://www.sciencedirect.com/science/article/pii/0166218X9400020E, doi:http://dx.doi.org/10.1016/0166-218X(94)00020-E.

[10] Giovanni Viglietta. Gaming is a hard job, but someone has to do it! *Theory of Computing Systems*, 54(4):595–621, 2014.

[11] S. Waharte, A. Golynski, and R. Boutaba. On the complexity of routing in wireless multihop network. In *2012 8th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 431–436, Aug 2012. doi:10.1109/IWCMC.2012.6314243.

[12] Robert Williams. *The geometrical foundation of natural structure*. Dover New York, 1979.

A Appendix: Semiregular Tessilation NP-Completeness Proofs

Theorem 14 *The HCP in the grid graphs of the 3.3.3.3.6 tessellation is NP-complete.*

Proof. Similar to the 3.4.6.4 tessellation in Section 3, the NP-completeness of the HCP tessellation can also be proven by reducing from the HCP in hexagonal grid. We use the gadgets shown in Figure 12 to simulate the vertices and edges of the hexagonal grid. Now, we can construct a simulated graph G for any hexagonal grid G'. For example, the graph formed by two hexagons can be simulated by the grid in Figure 13.

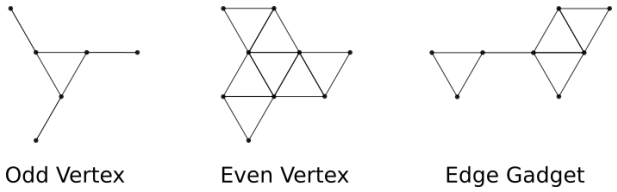


Figure 12: Gadgets for the 3.3.3.3.4 reduction

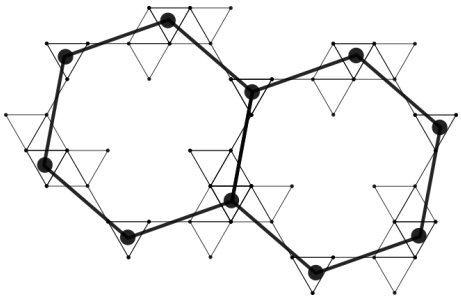


Figure 13: An example of a simulated graph in the 3.3.3.3.4 tessellation

Similar to the gadgets used in Section 3, there are two kinds of traversals for the edge gadget: a cross path that goes from one end to the other end and the return path that begins and finishes on the same end. The following reasoning on why G has a Hamiltonian cycle if and only G' has a Hamiltonian cycle is identical to that of the previous section. If a hexagonal grid G' has a Hamiltonian cycle, we can create a Hamiltonian cycle in G by going through the edge gadgets of the taken edges with cross paths and and the edge gadgets of the untaken edges with return paths. If there is a Hamiltonian cycle in G, each vertex gadget of G must be connected to exactly two cross paths, indicating that there exists a Hamiltonian cycle in G. The reduction is then complete.

□

Theorem 15 *The HCP in the grid graphs of the 3.6.3.6 tessellation is NP-complete.*

Proof. We prove that the HCP in this tessellation is NP-complete by reducing from HCP in hexagonal grid. Using the following vertex gadgets and edge gadget, shown in Figures 14 and Figure 15, for any hexagonal grid G' we can construct a simulated graph G in the tessellation.

Each edge gadget has two kinds of traversals: return paths

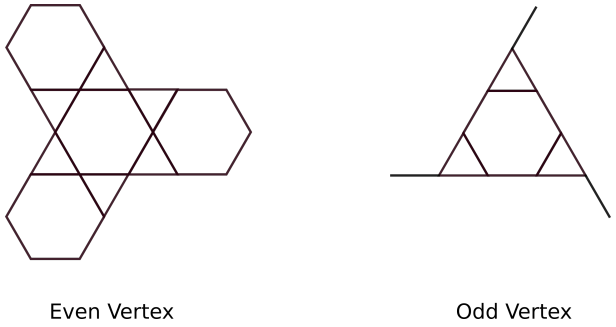


Figure 14: Vertex gadgets for 6.3.6.3

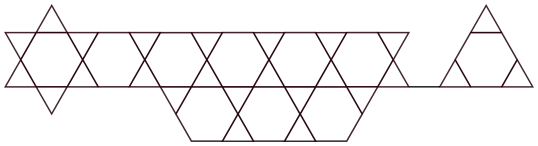


Figure 15: Edge gadgets for 6.3.6.3

and cross paths. Return paths begin and end on the same end of the edge while cross paths start and finish on different ends. With some inspection, it is clear that return paths and cross paths are the only two kinds of traversals allowed in the edge gadget. Figure 16 shows a possible return path. Different from those of previous tessellations, the edge gadget here has two kinds of cross paths as shown in Figure 17. Although the two kinds of cross paths start out the same from the odd vertex gadget on the right, they finish in the even vertex on the left differently. The way a cross path connects to an even vertex gadgets dictates which direction it can go next. The upper cross path must turns clockwise when going through the even vertex, allowing it to connect to an upper cross path while the lower one must turn counter-clockwise, allowing it to connect to a lower cross path. By choosing the correct kind of cross paths, any pair of the three edges of the even vertex gadget can be taken by compatible cross paths. By inspection, we can easily see that odd vertex gadget can connect to any pair of the three edges in two cross paths as well.

Now, we will show that the simulated graph G has a Hamil-

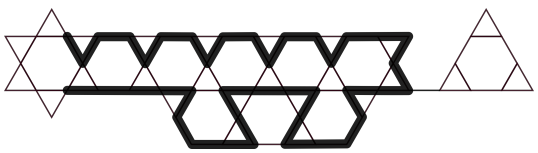


Figure 16: Return path for 6.3.6.3 edge gadget

tonian cycle if and only if the original graph G' has a cycle.

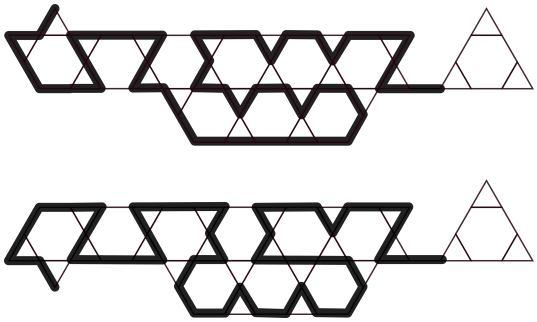


Figure 17: Two Kinds of Cross Paths for 6.3.6.3 edge gadget

If the original hexagonal grid G' has a cycle C', then we go through the edges gadgets representing taken edges in C' with a cross path and those representing untaken edges with a return path. Note that we need to use the correct kind of cross paths so that the choice matches the turning at the vertex. If so, then there is a also a Hamiltonian cycle in G. If the simulated graph G has a Hamiltonian cycle, each vertex gadget must be connected to exactly two cross paths, which indicate that there is a Hamiltonian cycle in G'.

□

Theorem 16 *The HCP in the grid graphs of the 3.12.12 tessellation is NP-complete.*

Proof. This tessellation is composed of dodecagons and triangles. For a hexagonal grid G', we construct a simulated graph G in the tessellation by using the triangles as vertex of G' and the edges in between triangles as the edges of G'. If a Hamiltonian cycle exists in G, each triangle must be connected to two paths that form a 120° angle. Then, there must also be a Hamiltonian cycle in the hexagonal grid G'. If there is a Hamiltonian path in the hexagonal grid G', then there exist one in G.

□

Theorem 17 *The HCP in the grid graphs of the 3.3.3.4.4 tessellation is NP-complete.*

Proof. Similar to the 3.3.4.3.4 tessellation, this tessellation can also be considered as a square grid with extra diagonals. Because its resemblance to square grid, we again use the square grid gadgets. However, if we use the same reduction as in [7], an extra diagonal may disable a pin connection, being an extra edge that connects the odd vertex gadget with the edge gadget. Then, a return path can enter into the odd vertices through this extra edge, causing the former pin connection to no longer function. The connection to the upper edge gadget in an odd vertex gadget shown in Figure 18 is an example of a disabled pin connection. Thus, we will need to modify the reduction.

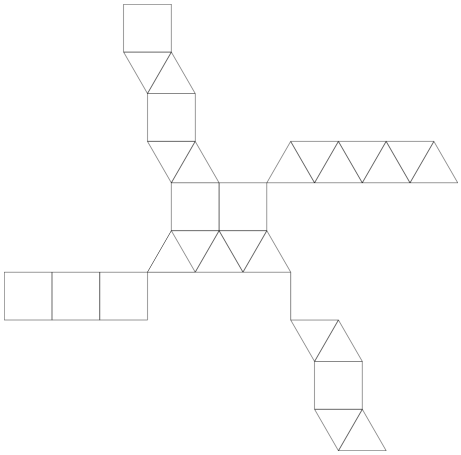


Figure 18: An odd vertex gadget

Although one pin connection may be disabled in a odd vertex gadget, there remains three other functioning pin connections. Because the reduction only requires max degree-3 vertices, there are still ways to make the reduction work. We construct the simulated grid G in the following way. Given a parity preserving square grid embedding of the original max degree-3 bipartite graph G' as mentioned in the 1982 paper [7], we enlarge the embedding by a factor of 3 so that any single segment is at least three segments long and the parities of the vertices are preserved. We then adjust the embedding by replacing every disabled pin connection with a functioning pin connection. Figure 19 shows that if the upper connection is disabled, we use the left or right connection to replace it (the upper row represents embedding before adjustment while the lower row represents embedding after adjustment). Based on the adjusted embedding, we can then construct a simulated graph G using the square grid gadgets. Since the pin connections are all functioning in G, the reduction works.

□

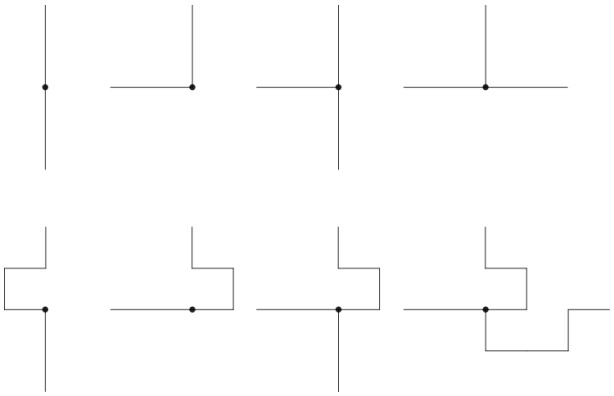


Figure 19: Embedding adjustment

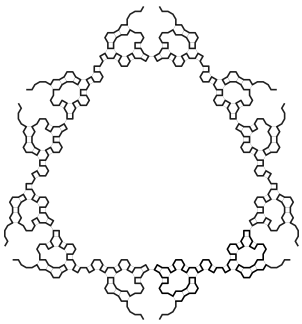


Figure 20: A broken 4.6.12 vertex

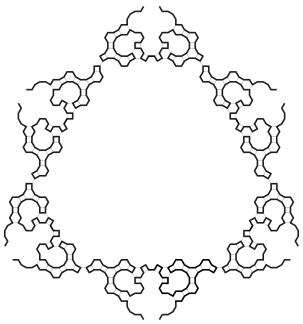


Figure 21: An unbroken 4.6.12 vertex

B Appendix: Turning in Square Grids is Easy

We start with a proof for max-degree-3 square grid graphs because of it's simplicity, and then extend those arguments to handle all square grid graphs.

Theorem 18 *There is a polynomial time algorithm for determining whether a max-degree-3 square grid graph admits a Hamiltonian Path which turns at every vertex.*

Proof. First we notice that any degree-1 vertices, and any degree-2 vertices without a turn make it impossible to have a Hamiltonian cycle. Next, all other degree-2 vertices must have both edges in the cycle. Finally, degree-3 vertices form T intersections. If the path must turn then the middle edge of the T must be included in the cycle.

The only choice remaining is which of the two straight edges in each T intersection is included in the cycle. If one of these two edges lies next to a forced edge, then we know which of the two edges must be included and further that the opposite edge cannot be in the cycle. Now we consider a line of T intersections as in Figure 22. At some point this line must terminate, implying the edge directly opposite the last in the chain does not exist. This implies the last edge must be in the path, which forces the second to last edge to not be in the path and so forth. Thus all T intersections are also forced leading to the following algorithm.

1. If the graph contains any degree-1 vertices, return false.
2. If the graph contains any degree-2 vertices without a turn, return false.
3. Find all degree-2 vertices and mark all edges as being in the path. When you mark an edge as being in a path, follow it to the next vertex and mark the opposite edge as not in the graph. Repeat the process of alternating marking edges as in or not in the cycle until an already marked edge is reached. If the marks disagree, return false, otherwise continue.
4. Find all degree-3 vertices and marks all middle edges as being in the path.When you mark an edge as being in a path, follow it to the next vertex and mark the opposite edge as not in the graph. Repeat the process of alternating marking edges as in or not in the cycle until an already marked edge is reached. If the marks disagree, return false, otherwise continue.

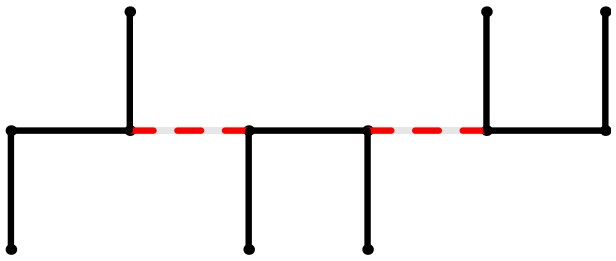


Figure 22: An example row of connected vertices. Forced edges are in bold and forbidden edges are in dashed red.

5. At this point we have marked all edges. Pick a start and check if the edges marked as being in the cycle in fact form a Hamiltonian cycle. If so, return true, otherwise return false.

□

Now we present the main theorem.

Theorem 19 *There is a polynomial time algorithm for determining whether a square grid graph admits a Hamiltonian Path which turns at every vertex.*

Proof. A Hamiltonian Path which turns at every vertex in a grid graph imposes the following constraint: for every vertex except the start and end, precisely one vertical and one horizontal edge must be in the path. This quickly leads to the conclusion that degree-1 and degree-2 vertices either make a Hamiltonian path impossible or forces the edges to be in the path. First, guess the first and last vertex and edge in the path and remove the unused edges next to those vertices from the graph. Now consider a row or column in the graph. In that row we have some number of groups of contiguous vertices which we will consider one at a time. Take the farthest left vertex in this group, it cannot have a left edge. This forces the right edge to be in the Hamiltonian path. Now look at the next vertex. We've already determined it's left edge is in the Hamiltonian path, forcing its right edge to not be in the path. Continuing the next vertex must have its right edge in the path and so on. If there are an even number of vertices, this is consistent, if there are an odd number of vertices then we'll reach a contradiction, declaring a non-existent edge to be in the path. We repeat this process for every group in every horizontal row, and then similarly for every group in every column (starting with the top vertex of each group rather than the left one). We have now assigned every single edge to be either in the path or not. We simply walk the graph to ensure the path is Hamiltonian (aka checking connectivity of our assigned edges) and return the result. □

C Appendix: Turning in 3D Square Grids is Hard

Theorem 20 *The Always-Turning Hamiltonian Path problem in 3D square grid graphs is NP-complete even if the height of the grid is restricted to be* 2 *vertices.*

Proof. We closely follow the proof that deciding if square grid graphs admit a Hamiltonian path is NP-complete. We reduce from deciding whether planar max-degree-3 graphs admit a Hamiltonian path. We also construct edge gadgets and even and odd vertex gadgets. In this subsection, all figures are two vertices high with paths on the bottom layer represented by black lines and paths on the top layer represented by dotted blue lines.

Edge gadgets are sequences of $2 \times 2 \times 2$ cubes. They admit a forward path, representing an edge taken in the graph, shown in Figure 23. They also admit a return path, shown in Figure 24, representing an edge not taken in the graph. The edges can be turned, as shown in Figures 25 and 26.

Vertex gadgets are represented by $4 \times 8 \times 2$ rectangles. Edges attach across the marked edges e_1 to e_4. Figures 27, 28, and 29 show three different paths through a vertex which will connect any three of the four target edges. Unlike the original grid proof, it is critical that the problem we are reducing from is max-degree-3. Even vertex gadgets attach to the edge gadgets by a $1 \times 2 \times 1$ pair of vertices. If a path enters this pair of vertices, it is then forced to take the edge connecting them. Thus in an even vertex the path can only pass from the vertex to each edge a single time. Since it is max-degree-3, this means precisely two of the adjacent edge gadgets are taken and one is not. Since every odd vertex is connected to an even vertex, this means the odd vertex gadgets must also have precisely two of their adjacent edge gadgets have a taken path. Thus there is only a Hamiltonian cycle if the original graph admits a Hamiltonian cycle.

□

D Appendix: Turning in Triangular Grid Graphs

We now examine the question of Hamiltonian Paths in triangular grid graphs which must turn at every vertex. First, notice that there are now two types of turns: 60° and 120°. It is simple to show the turning Hamiltonian Path problem in triangular grids with 60° is easy and with 120° is NP-complete. For the 120° turns, first notice that we can remove alternating vertices from a triangular grid to leave a hexagonal grid. Further, all remaining edges are 120° turns from each other. Thus we have a very simple reduction from finding Hamiltonian Paths in Hexagonal grid graphs to finding Hamiltonain Paths which always turn 120° in Triangular Grids.

For the case of 60° turns, the answer becomes simple. With out loss of generality, pick two adjacent edges to be in the Hamiltonain Path, such as the two in Figure 30. Now consider the next edge (options show as dotted edges), either it turns to the right and completes the triangle (red edge) or it turns left (blue edge). If it turns left then we once again have two edges of a triangle in our path, only allowing one legal option as seen in Figure 31. Thus, the only Hamiltonian paths which always turn 60 are subsets of a straight zig-zag. Similarly, the only allowed Hamiltonian cycle is a triangle. We've now characterized turning Hamiltonian paths in triangular grid with both polynomial time algorithms and NP-completeness depending on what turns are allowed.

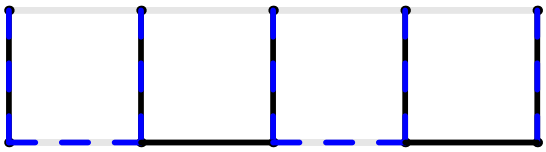


Figure 23: An edge taken in the simulated graph. The path here starts on one side and ends on the other

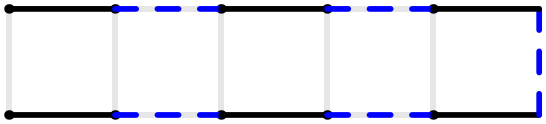


Figure 24: An edge not taken in the simulated graph. The path starts and ends both on the left side.

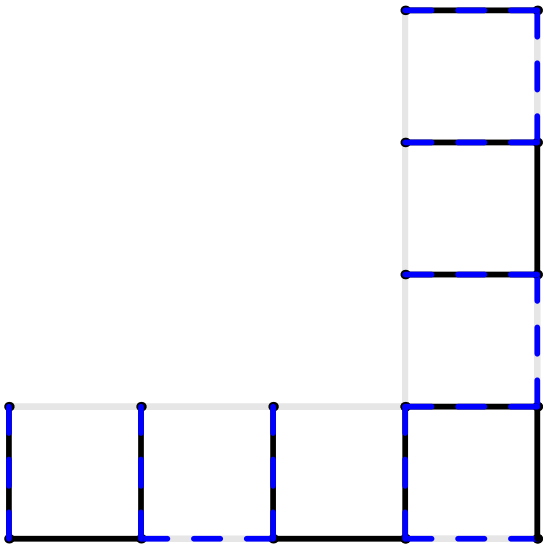


Figure 25: A taken edge being turned.

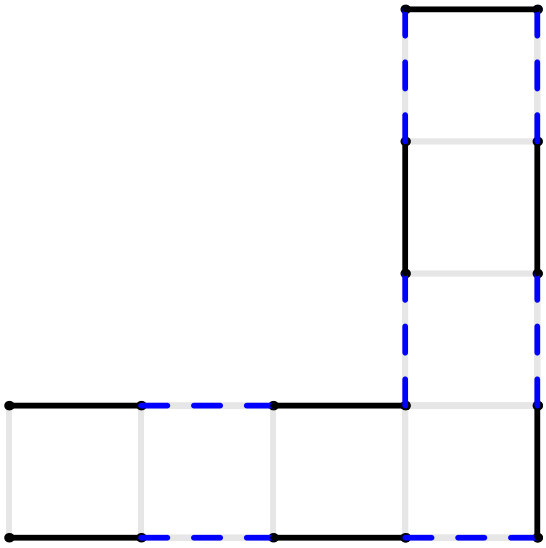


Figure 26: A non-taken edge being turned.

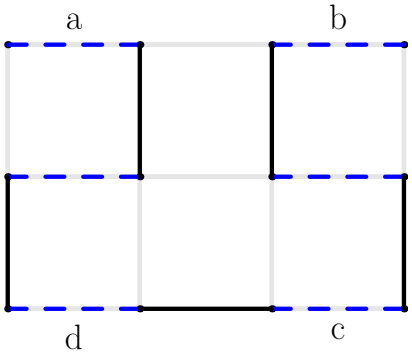


Figure 27: Enters a, crosses c, leaves b.

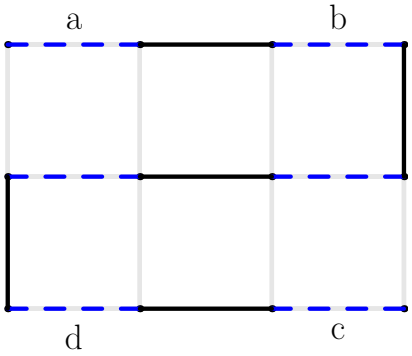


Figure 28: Enters a, crosses b, leaves c

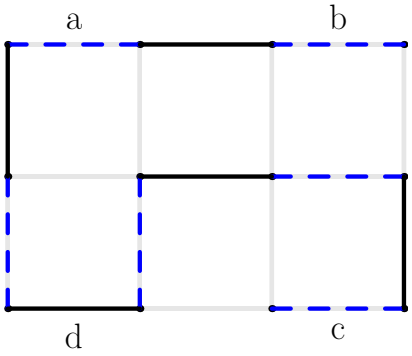


Figure 29: Enters b, crosses a, leaves c.

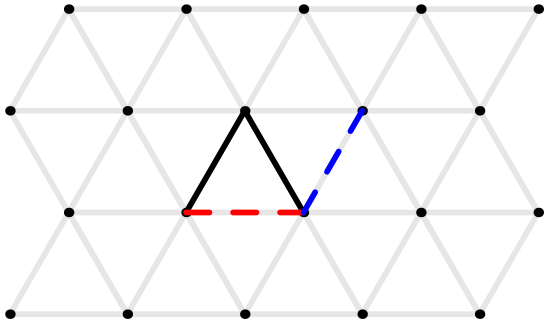


Figure 30: Turns in a triangular grid with forced 60° turns.

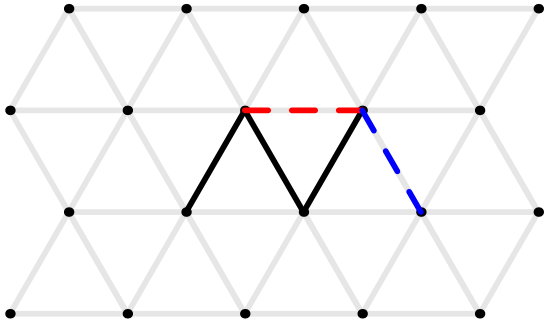


Figure 31: After another step the edges are still forced.

E Appendix: Double Turning in Solid Square Grids

Theorem 21 *The Double Turning Hamiltonian Cycle Problem in solid grid graphs can be solved in linear time.*

Proof. By Lemma 12 we see that deciding if the graph is checkerable and finding a spanning tree of the checkering suffices. We assume we are given the graph embedding. First, we pick the left-most, top-most vertex in the graph and check for the 4-cycle defining the only pixel it is a part of. We construct a vertex in our checkerboard graph for this vertex and associate all four of the vertices around the pixel with it. Now we will perform a breath-first search over the checkered pixels in our graph. From each vertex of our current pixel, look at their exterior edges. If there are two, check whether it has been considered before and if not put that in a queue as a candidate checkerable pixel. If there is only one edge, we maintain a separate list of such degree-3 external edges. First, we check if that edge is in our list, if so we remove the edge and if not we add the edge to the list. Next, pop a candidate pixel off of the queue, verify that all four vertices are there, making it a valid pixel, and recursively check for its neighbors as before. If we ever discover a candidate pixel which is missing a vertex, then one of our properties is violated and we respond that there is no Hamiltonian cycle. Otherwise, we will have constructed a partial checkering and the bfs ordering will give us a spanning tree. Now, we need to take all edges from degree-3 vertices which need to be verified and check that both ends of the edge are vertices which belong to pixels in our checkering. We do the latter simply by checking whether our external degree-3 list

is empty because every such edge will be added and then removed the two times it touches a valid pixel. If this is true, return that there exists a Hamiltonian cycle, and if not return false. Verifying each pixel and constructing a new vertex in our checkering takes constant time. Running our bfs touches each pixel, and thus each vertex in our graph a constant number of times. Each extra edge is touched twice. Thus the whole algorithm can be constructed to run in linear time. □

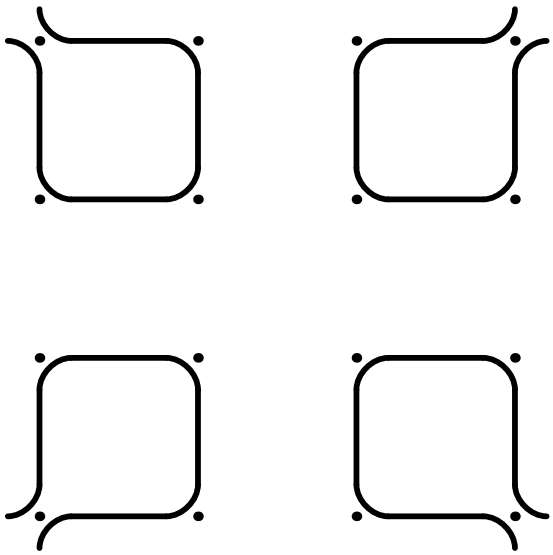


Figure 32: A pixel with all four edges not included in the path.

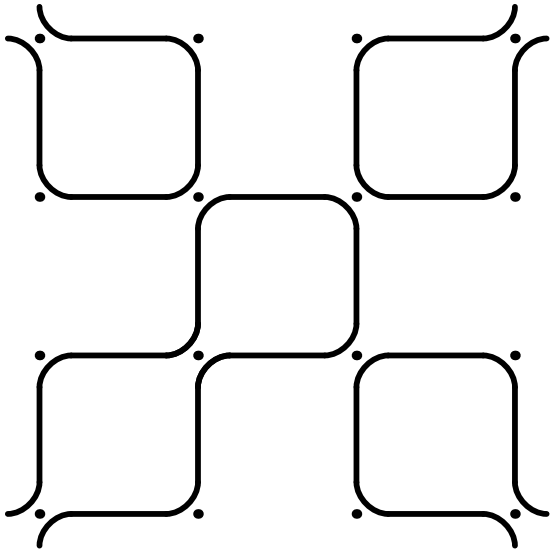


Figure 33: We can augment the path to visit each vertex twice..

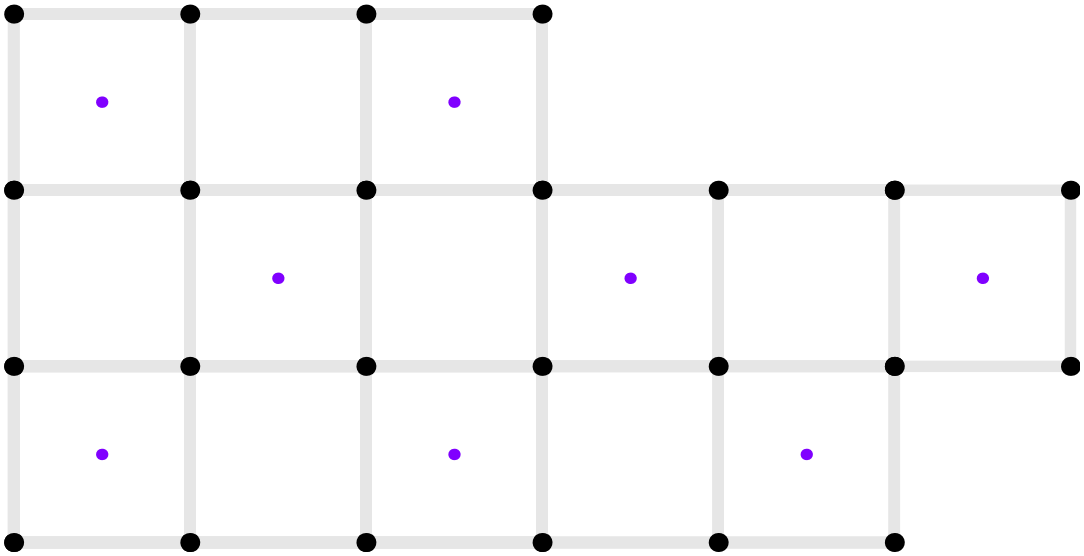


Figure 34: An example solid grid graph with its checkering in purple.

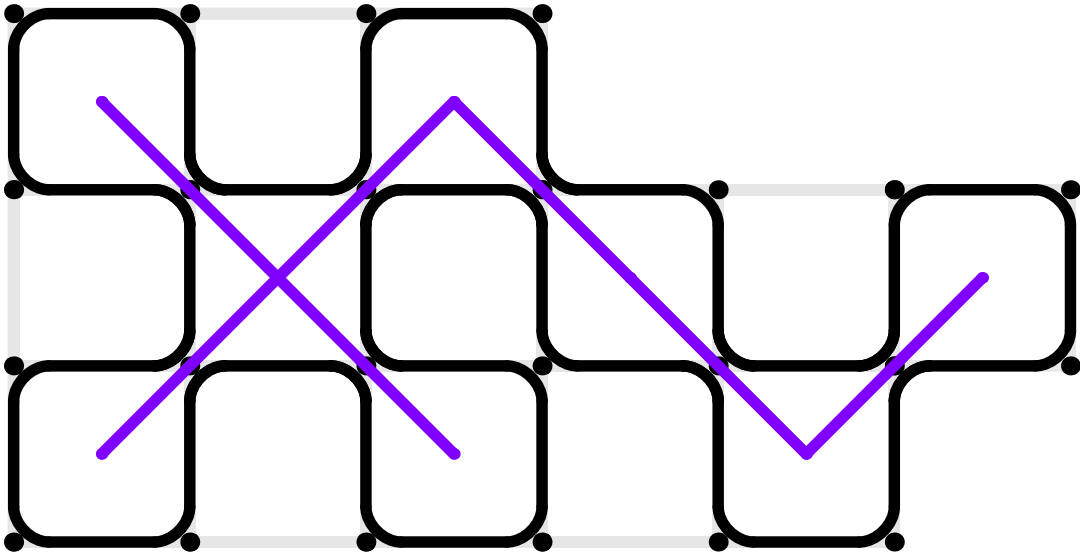


Figure 35: The solution to the example with the spanning tree of its checkering in purple.

Improved Bounds for the Traveling Salesman Problem with Neighborhoods on Uniform Disks

Ioana Oriana Bercea*

Abstract

Given a set of n disks of radius R in the Euclidean plane, the Traveling Salesman Problem With Neighborhoods (TSPN) on uniform disks asks for the shortest tour that visits all of the disks. The problem is a generalization of the classical Traveling Salesman Problem(TSP) on points and has been widely studied in the literature. For the case of disjoint uniform disks of radius R, Dumitrescu and Mitchell [14] give a PTAS and also show that the optimal TSP tour on the centers of the disks is a 3.547-approximation to the TSPN version. The core of the latter analysis is based on bounding the detour that the optimal TSPN tour has to make in order to visit the centers of each disk and shows that it is at most $2Rn$ in the worst case. Häme, Hyytiä and Hakula [21] asked whether this bound is tight when R is small and conjectured that it is at most $\sqrt{3}Rn$.

We further investigate this question and derive structural properties of the optimal TSPN tour to describe the cases in which the bound is smaller than $2Rn$. Specifically, we show that if the optimal TSPN tour is not a straight line, at least one of the following is guaranteed to be true: the bound is smaller than $1.999Rn$ or the TSP on the centers is a 2-approximation. The latter bound of 2 is the best that we can get in general. Our framework is based on using the optimality of the TSPN tour to identify local structures for which the detour is large and then using their geometry to derive better lower bounds on the length of the TSPN tour. This leads to an improved approximation factor of 3.53 for disjoint uniform disks and 6.728 for the general case. We further show that the Häme, Hyytiä and Hakula conjecture is true for the case of three disks and discuss the method used to obtain it.

1 Introduction

We study the Traveling Salesman Problem with Neighborhoods (TSPN) when each neighborhood is a disk of fixed radius R. The problem is a generalization of the classical Euclidean Traveling Salesman Problem (TSP), when each point to be visited is replaced with a region (interchangeably, a neighborhood) and the objective is to compute a tour of minimum length that visits at least one point from each of these regions. While it is known that Euclidean TSP admits a Polynomial Time Approximation Scheme (PTAS) due to the celebrated results of Arora [3] and Mitchell [25], Euclidean TSPN has been shown in fact to be APX-hard [29, 13] even for line segments of comparable length [17]. The geometric version of TSPN was first studied by Arkin and Hassin [2] who gave constant factor approximations for a variety of cases. Since then, there has been a wide ranging study of TSPN for different types of regions. In the case of connected regions, there is a series of $O(\log n)$ approximations [24, 17, 20]. Better approximations are known for cases that consider various restrictions on the regions such as comparable sizes (i.e. diameter), fatness (ratio between the smallest circumscribing radius and largest inscribing radius, or how well can a disk approximate the region) and pairwise disjointness or limited intersection [14, 13, 17, 27, 8, 15, 6, 26, 9]. We refer the reader to [19] for a comprehensive survey of the results.

We study the disk version which models the situation in which each customer is willing to travel a distance R to meet the salesperson. This is considered an important special case of the general TSPN [15] and is especially relevant since it has found applicability in other areas such as path planning algorithms for coverage with a circular field of view [1, 18] and most recently for data collection in wireless sensor networks [33, 23, 12]. Various heuristics [33, 7, 10, 22] and variations [4, 23, 30] have been considered, all of which have as their basis the TSPN on uniform disks problem.

Dumitrescu and Mitchell [14] were the first to specifically address the case of uniform disks in 2001. They showed a PTAS for disjoint unit disks and simpler constant factor approximations for the disjoint and overlapping cases. The specific factor of 3.547 for disjoint disks is relative to using a routine for TSP on points (i.e. the actual constant depends on the subroutine used). The PTAS and the 3.547-approximation are the best known factors for the disjoint case. Later, Dumitrescu and Tóth [15] improved the constant factor in [14] for the overlapping case from 7.62 to 6.75 and extended it to unit balls in $\mathbb{R}^d$, giving a $O(7.73^d)$-approximation. When the balls are disjoint, Elbassioni et al. [17] showed a $O(2^d/\sqrt{d})$-approximation. Most recently, Dumitrescu and Tóth [16] gave a randomized constant factor for (po-

*Department of Computer Science, University of Maryland, College Park, ioana@cs.umd.edu

tentially overlapping) disks of arbitrary radii (the actual constant is not mentioned and seems large). As noted by the authors in [15], while the complexity of the disk case is well understood generally, the question of obtaining practical and better constant factor approximations remains of high interest.

In this paper, we aim for an improved constant factor algorithm for the case of uniform radius disks and note that the algorithm proposed by Dumitrescu and Mitchell [14] for the disjoint case outputs an approximate TSP tour on the *centers* of each disk. The core of their analysis is a bound that compares the length of the optimal TSP on the centers of each disk($|TSP^*|$) with the length of the optimal TSPN on the disks ($|TSPN^*|$) and says that

$$|TSP^*| \leq |TSPN^*| + 2Rn, \tag{1}$$

where $n \geq 2$ is the number of disks in the instance. In addition, the authors use a packing argument to lower bound the length of the optimal TSPN tour in terms of R and n and get that $\frac{\pi}{4}Rn - \pi R \leq |TSPN^*|$. Overall, this gives a 3.547- approximation and in addition, the authors show that the algorithm cannot give better than a factor 2 approximation. While other methods for choosing representative points can be employed [16, 2, 14], this approach is appealing both in its elegance and because it does not depend on R. Moreover, other existing constant factors approximations for TSPN often hide large constants [13, 17, 16] that are incurred as a consequence of using general bounds on the length of the optimal tour that do not directly exploit the structure of the regions or of the optimal TSPN tour (bounding rectangle argument and Combination Lemma in [2]). In order to improve on them, the challenge then becomes to develop bounds that exploit the difference in behavior between a TSP tour (on points) and the TSPN tour on the regions and furthermore, avoid using general purpose techniques that add on to the overall approximation factor.

In this context, one way to improve the approximation factor for disjoint disks is to better understand the relationship between the optimal TSP tour on the centers and the optimal TSPN on the disks. Specifically, is the $2Rn$ term in (1) tight or can it be improved by using specific structural properties of the optimal TSPN? A similar question was asked in 2011 by Häme, Hyytiä and Hakula [21] for the case when R is very small (and hence, TSP^* and $TSPN^*$ respect the same order and the disks are pairwise disjoint). They conjectured that the true detour term should be $\sqrt{3}nR$ and constructed arbitrarily large instances of disjoint disks that converge to this case. We refer to this as the **Häme, Hyytiä and Hakula conjecture.** Subsequent experiments by Müller [28], however, suggest that this might be true only for tours up to five disks and higher otherwise. No further progress has been made towards the conjecture since then.

Contributions. We make the first progress on the conjecture and develop a twofold method that either improves the bound in (1) or shows that the TSP on the centers is a good approximation for the TSPN on the disks. Formally, we get that

Theorem 1 *For any $n \geq 4$ disjoint disks of radius R at least one of the following is true:*

- *$TSPN^*$ is supported by a straight line,*
- *$|TSP^*| \leq |TSPN^*| + 1.999Rn$,*
- *$|TSP^*| \leq 2 \cdot |TSPN^*|$.*

Our framework also gives an overall 3.53-approximation for the case of uniform disjoint disks and a 6.728-approximation for the overlapping case.

While the improvement in the overall approximation factor is small, our framework strives to provide new insight into the problem that can be explored further. Specifically, the 2-approximation (optimal with respect to the method of computing a TSP on the centers [14]) comes from the case in which the TSPN tour takes a lot of sharp turns. Furthermore, it is based on a lower bound that does not rely on packing arguments. To the best of our knowledge, this is the first such bound specifically for TSPN out of all arguments for general fat regions [27]. As such, it might be of independent interest and it could, for example, lead to improved approximation factors for balls in $\mathbb{R}^d$ that do not depend exponentially on the dimension. Moreover, the fatness of the disks is used in showing that short sharp turns lead to a disk being visited multiple times and can conceivably be used to show similar properties for other fat regions.

We start by fixing an order σ and comparing the TSPN tour that visits the disks in that order to the TSP tour that visits their respective centers in the same order. The $2Rn$ term in (1) comes from considering the points at which the TSPN touches the boundary of each disk and charging each such vertex with a $2R$ detour for going to its respective center and coming back. In this view, the $2Rn$ term cannot be improved since the charge on each vertex will always be $2R$. Instead, we reinterpret the bound as charging the *edges* of the TSPN tour instead of its vertices and notice that the charge for each edge can now be anywhere between $-2R$ and $2R$, depending on how close the tour is to (locally) visiting pairs of disks optimally. In this context, we define a "bad" edge to be one that incurs a large charge (i.e. $> (2-\epsilon)R$ for some $\epsilon > 0$). We show that such bad edges lead to the TSPN tour exhibiting sharp turns (i.e. with

small interior angle). When the edges of the sharp turn are long, we use that to derive a better lower bound on the overall TSPN tour length. On the other hand, when one of them is short, we show that the tour must then visit a disk twice (i.e. visit it once, then touch another disk and return back to it). The crux of the argument is in understanding how these short sharp turns that visit a disk multiple times influence the global detour term.

When a tour visits a disk more than once, two scenarios follow naturally from the classical TSP case of just visiting points: either the order σ is not optimal or the tour must follow a straight line. Surprisingly, we show that a third alternative scenario is also possible, whose local structure we call a β-triad. The main technical contribution of the paper is in describing structural properties of such β-triads and showing that they actually have a low *average* detour. Specifically, we construct an additional order σ' and use an averaging argument to show that β-triads have low detour when compared to the TSP tours that visit the centers in the order σ and σ'. This then allows us to conclude that they have a low detour with respect to the optimal TSP on the centers.

Along the way, we also show that the Häme, Hyytiä and Hakula conjecture is true for $n = 3$ and use it to bound the average detour of β-triads. We include a discussion of the method used to derive it, involving Fermat-Weber points, which might be useful for the case of $n \geq 4$. We also discuss how our approach can be used within the framework of Dumitrescu and Tóth [15] to yield improved approximation factors for the overlapping disks case.

Preliminaries. We consider $n \geq 3$ disjoint disks of radius R in the Euclidean plane. We denote an optimal TSP tour on the centers of the disks as TSP^*. Similarly, $TSPN^*$ will denote an optimal TSPN tour on the disks. Our results will be with respect to a fixed TSPN tour (which we call simply $TSPN$) described by a sequence of ordered points $P_1, P_2, \ldots, P_n$ on the boundary of the disks such that the tour is a polygonal cycle with edges (P_i, P_{i+1}). Furthermore, we have that for each of the input disks, there exists some $i \in [1, n]$ such that point P_i is on the boundary of the disk.

Notice that the points P_i induce a natural order σ on the disks with centers $O_1, O_2, \ldots O_n$, i.e. σ corresponds to the identity permutation on $P_1, \ldots, P_n$. For the majority of our theorems, we will assume that TSP always refers to a tour *on the centers* and in the order σ on the disks. When we need to make a difference, we will further use $TSP(\sigma')$ to be the tour which visits the centers in the order given by the permutation σ'. Given two such permutations σ and σ', we say that $\sigma \cap \sigma'$ refers to the maximal set of points on which σ and σ' agree. In this context, $TSPN(\sigma \cap \sigma')$ refers to the collection of paths we get from visiting the points P_i according to $\sigma \cap \sigma'$. Similarly, $TSP(\sigma \cap \sigma')$ corresponds to the collection of paths that we get from visiting the points O_i according to $\sigma \cap \sigma'$.

Finally, we denote the length of a tour $\mathcal{T}$ as $|\mathcal{T}|$. When $\mathcal{T}$ is a collection of paths, we have that $|\mathcal{T}|$ represents the total length of each of the paths. When A and B are points, we have that $|AB|$ denotes the length of the segment AB. We therefore have that $|TSPN| = \sum_{i=1}^{n} |P_iP_{i+1}|$ and $|TSP| = \sum_{i=1}^{n} |O_iO_{i+1}|$, where $P_{n+1} = P_1$ and $O_{n+1} = O_1$.

2 β-Triads and a Structural Theorem

Before we formally define what a "bad" edge is, we will describe how to interpret the $2Rn$ *detour bound* from [14] as charging edges instead of vertices. We fix an order σ and consider the points P_i and O_i as previously defined. The argument in [14] then says that we must have:

$$\textstyle\sum_i |O_iO_{i+1}| \leq \sum_i |P_iP_{i+1}| + 2Rn.$$

In this context, the term $\sum_i P_iP_{i+1} + 2Rn$ is the length of a tour that follows the TSPN tour and additionally, at each point P_i, takes a detour of $2R$ to visit the center O_i and come back. Choosing σ to be the optimal order in which $TSPN^*$ visits the disks gives us (1). In this view, the detour term $2Rn$ is obtained by charging $2R$ to each point P_i of the TSPN tour. Instead, we can also think of it as coming from charging each *edge* P_iP_{i+1} of the tour with a *local detour* of $2R$ in the following sense:

$$|O_iO_{i+1}| \leq |P_iP_{i+1}| + 2R.$$

This new perspective is quite natural since it captures the observation that the shortest edge which visits the disks centered at O_i and O_{i+1} has length exactly $|O_iO_{i+1}| - 2R$ and hence the $TSPN$ tour has to pay at least that for each pair of consecutive disks it visits. In this sense, we decompose the global detour term of $2Rn$ into n local detour terms $|O_iO_{i+1}| - |P_iP_{i+1}|$ that essentially quantify how efficient the TSPN on the disks is locally.

In this context, saying a TSPN edge has a high local detour is equivalent to saying that it is close to being locally optimal or shortest possible: when the edge is exactly of length $|O_iO_{i+1}| - 2R$, its local detour is $2R$ (the maximum). If, on the other hand, we know that the edge is bounded away from $|O_iO_{i+1}| - 2R$, i.e. $|P_iP_{i+1}| > |O_iO_{i+1}| - 2R + \epsilon R$, for some $\epsilon > 0$, this translates into a local detour of at most $(2 - \epsilon)R$. Intuitively, such an edge is "good" for us because it allows us to lower the overall detour term. In contrast, "bad" P_iP_{i+1} edges are the ones for which the local detour term is large and consequently, their length is closer to

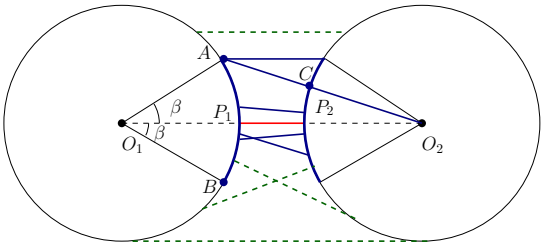


Figure 1: Bad edges are guaranteed to have both endpoints in the blue arcs. Furthermore, if $|P_1P_2| \leq |AC|$, then P_1P_2 is bad. In contrast, the dashed edges are guaranteed to be good edges.

$|O_iO_{i+1}| - 2R$. Our technique is motivated by trying to describe the behavior of such bad edges.

Formally, we consider a fixed angle parameter $\beta \in [0, \pi/12]$ that we instantiate later when we derive the overall bounds. We define the function:

$$f(O_1O_2, \beta) = \sqrt{|O_1O_2|^2 + R^2 - 2R|O_1O_2| \cos\beta},$$

which is $|O_1O_2| - R$ when $\beta = 0$ and $|O_1O_2| + R$ when $\beta = \pi$. Intuitively, the quantity $f(O_1O_2, \beta) - R$ will control how close we are to $|O_1O_2| - 2R$. We then say that the edge P_1P_2 is **bad** if $|P_1P_2| \leq f(O_1O_2, \beta) - R$ and **good** otherwise (we abstract away the dependency on β for simplicity). Bad edges are close to $|O_1O_2| - 2R$ and will incur a large local detour. In contrast, using straightforward algebra, one can show that a good edge P_1P_2 is guaranteed to have a small detour: $|O_1O_2| \leq |P_1P_2| + (1 + \cos\beta)R$.

Consecutive bad edges. The idea behind defining bad edges in terms of $f(O_1O_2, \beta) - R$ is that it allows us to restrict the location of P_1 and P_2 on the boundary of their respective disks as seen in Figure 1. Specifically, there are exactly two points A and B on the boundary of the first disk with the property that the shortest distance from A or B to the boundary of the second disk is exactly $f(O_1O_2, \beta) - R$. Not coincidentally, they form an angle of β with $O_1O_2 : \angle AO_1O_2 = \angle BO_1O_2 = \beta$. In general, P_1 (and in a similar fashion P_2) is guaranteed to lie in the short arc between A and B whenever P_1P_2 is upper bounded by $f(O_1O_2, \beta) - R$ (Lemma 5).

When a second bad edge P_2P_3 is considered, we can conclude that the angle $O_1O_2O_3$ has to be at most 2β and hence the TSP on the centers must make a sharp turn after it visits O_2 (Corollary 6). If that happens and the disks are close to each other, we have that one of the edges of the TSP must actually intersect a disk twice. Specifically, if $|O_1O_2| \leq R/\sin(2\beta)$, then the support line for O_2O_3 must pass through the disk centered at O_1. Theorem 7 shows that if this happens, then the corresponding TSPN edge P_2P_3 must also cross this disk (Appendix A).

The fact that the disk centered at O_1 is crossed by both P_1P_2 and P_2P_3 suggests that the TSPN might not be optimal because it could be shortcut. Our structural theorem identifies when that is the case and isolates the remainder as having a specialized local structure which we call a β-triad. Formally, we say that a specific TSPN subpath $P_n - P_1 - P_2 - P_3$ is a **β-triad** if it satisfies all of the following properties (Figure 2):

- P_1P_2 and P_2P_3 are bad edges and $O_1O_2 \leq R/\sin(2\beta)$,
- P_1, P_2, P_3 are not collinear but P_n, P_1, P_2 are collinear with P_1 between P_n and P_2.

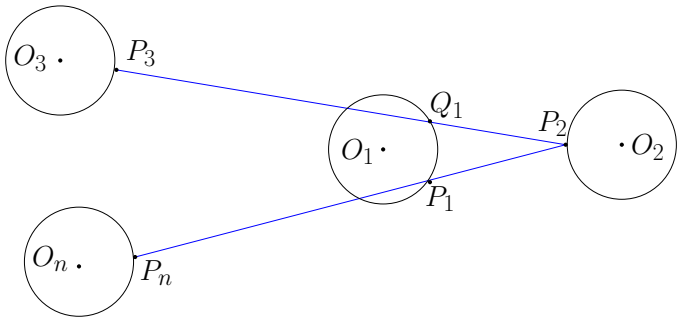


Figure 2: The path $P_n - P_1 - P_2 - P_3$ forms a β-triad.

We state the structural theorem here and refer the reader to Appendix A for a complete argument. The case in which the TSPN tour follows a straight line that stabs all the disks is discussed separately in Appendix C and is of separate interest.

Theorem 2 *For $n \geq 4$, if P_1P_2 and P_2P_3 are bad edges and $O_1O_2 \leq R/\sin(2\beta)$ then at least one of the following is true: the TSPN tour is not optimal, the TSPN tour is supported by a straight line or the path $P_n - P_1 - P_2 - P_3$ forms a β-triad.*

β-triads. The case of β-triads is interesting because it arises naturally as a consequence of dealing with regions instead of points. Given any optimal tour that exhibits internal angles $\leq \pi/6$, we can always add an extra disk at each sharp turn that will maintain optimality, pairwise disjointness and be intersected twice by this tour, giving rise to a β-triad. It is therefore important that we understand their behaviour.

Because of the fact that P_1P_2 and P_2P_3 are bad edges, the β-triad is likely to have a high detour with respect to $TSP(\sigma)$. Nevertheless, we show that there exists an alternate ordering σ' such that the average detour of the three edges in the β-triad with respect to $(|TSP(\sigma)| + |TSP(\sigma')|)/2$ is $3\sqrt{3}R$. The order σ' takes advantage of the fact that the disk centered at O_1 is crossed twice and inverts the order in which it is visited without changing the cost of the underlying TSPN

tour. The $3\sqrt{3}R$ bound comes from proving the Häme, Hyytiä and Hakula conjecture for $n = 3$ (Appendix F). In order to be able to construct σ' consistently across multiple β-triads, we also show that β-triads are isolated events and specifically that they are edge-disjoint (Lemma 8 in Appendix B).

Theorem 3 *If the TSPN in the order σ has k β-triads that together cover a set of edges of total length L_T, then we can construct another order σ' that agrees with σ on everything except the order inside the β-triads such that:*

$$\frac{|TSP(\sigma)| + |TSP(\sigma')|}{2} \leq |TSP(\sigma \cap \sigma')| + L_T + 3\sqrt{3}Rk.$$

Proof. We discuss the case for $k = 1$ and show how to modify the argument for $k > 1$. Suppose that $P_n - P_1 - P_2 - P_3$ form a β-triad. We know that $TSP(\sigma)$ visits the centers of each disk in the order $O_1, O_2, O_3, \ldots O_n$. We consider an additional order σ' such that $TSP(\sigma')$ visits the centers in the order $O_2, O_1, O_3, \ldots O_n$. Notice that both σ ad σ' agree on the order $O_3, O_4, \ldots, O_n$ and that they differ in the fact that σ visits O_1 after O_n and before O_2 and σ' visits O_1 after O_2 and before O_3. Therefore, for $T' = TSP(\sigma \cap \sigma')$, we have that $|T'| = |O_3O_4| + \ldots + |O_{n-1}O_n|$, $|TSP(\sigma)| = |T'| + |O_nO_1| + |O_1O_2| + |O_2O_3|$ and $|TSP(\sigma')| = |T'| + |O_nO_2| + |O_2O_1| + |O_1O_3|$.

On the other hand, the length of the TSPN with respect to the orders σ and σ' stays the same. The local cost of visiting $P_n - P_1 - P_2 - P_3$ is $L_T = |P_nP_1| + |P_1P_2| + |P_2P_3| = |P_nP_2| + |P_2P_3|$, since P_n, P_1 and P_2 are collinear and P_1 is between P_n and P_2. We also know that P_2P_3 intersects the disk centered at O_1 at some point Q_1 that is different from P_1 (Theorem 7). In other words, the TSPN that visits the points $P_n - P_1 - P_2 - P_3$ can be reimagined as visiting the points $P_n - P_2 - Q_1 - P_3$ and therefore respecting the order σ'. The local cost of crossing these edges is the same as before: $|P_nP_2| + |P_2Q_1| + |Q_1P_3| = |P_nP_2| + |P_2P_3| = L_T$.

We now apply Theorem 10 (the $3\sqrt{3}R$ bound for $n = 3$) on the TSP tour $O_n - O_1 - O_2$ with the TSPN tour $P_n - P_1 - P_2$ and get that:

$$\begin{aligned}|O_nO_1| + |O_1O_2| + |O_nO_2| &\leq |P_nP_1| + |P_1P_2| + |P_nP_2| + \\ &\quad + 3\sqrt{3}R \\ &\leq 2|P_2P_n| + 3\sqrt{3}R.\end{aligned}$$

On the other hand, if we consider the tour $O_1 - O_2 - O_3$ with the TSPN tour $P_2 - Q_1 - P_3$, we get that:

$$\begin{aligned}|O_1O_2| + |O_2O_3| + |O_1O_3| &\leq |Q_1P_2| + |P_2P_3| + |P_3Q_1| + \\ &\quad + 3\sqrt{3}R \\ &\leq 2|P_2P_3| + 3\sqrt{3}R.\end{aligned}$$

Combining the two inequalities and rearranging some terms gives us that:

$$\begin{aligned}|TSP(\sigma)| + |TSP(\sigma')| =& 2|T'| + |O_nO_1| + |O_1O_2| + |O_2O_3| + \\ &+ |O_nO_2| + |O_2O_1| + |O_1O_3| \\ =& 2|T'| + |O_nO_1| + |O_1O_2| + |O_nO_2| + \\ &+ |O_1O_2| + |O_1O_3| + |O_2O_3| \\ \leq& 2|T'| + 2|P_2P_n| + 2|P_2P_3| + 6\sqrt{3}R.\end{aligned}$$

Since $L_T = |P_2P_n| + |P_2P_3|$, we get our conclusion.

When $k > 1$, we construct the order σ' by switching the order in which we visit the centers in each β-triad in the same way as before. Since all the β-triads are edge disjoint (Lemma 8), we can construct σ' without any conflicts because any reordering that happens in one β-triad will not affect another β-triad. □

3 Improved Bounds on the TSPN Tour

Our main strategy will be a careful balancing of good and bad edges, in which the detour of good edges will be upper bounded by $(1+\cos\beta)R$ and that of bad edges by $2R$. While the bad edges will have the highest detour possible, we will use the fact that they must also be large in order to lower bound the TSPN tour more efficiently than Lemma 4 from [14] and [15], which we quote here for completeness.

Lemma 4 *[14, 15] For n disjoint disks of radius R, we have that any TSPN tour $\mathcal{T}$ on them satisfies:*

$$\tfrac{\pi}{4}Rn - \pi R \leq |\mathcal{T}|.$$

We will now show the proof of Theorem 1.

Proof. Assume the $TSPN^*$ is not a straight line. We start by singling out the β-triads and considering the two orderings σ and σ' from Theorem 3. If there are k_1 β-triads $\mathcal{T}_1, \ldots, \mathcal{T}_{k_1}$ spanning edges of total length L_T, we get that:

$$\begin{aligned}|TSP^*| &\leq \frac{|TSP(\sigma)| + |TSP(\sigma')|}{2} \\ &\leq |TSP(\sigma \cap \sigma')| + L_T + 3\sqrt{3}R \cdot k_1.\end{aligned}$$

Observe that $TSPN(\sigma \cap \sigma')$ is a collection of disjoint paths. From all of these paths, we further extract each from these a total of k_2 subpaths $\mathcal{G}_1, \ldots, \mathcal{G}_{k_2}$ consisting of good edges. Notice that the remaining subpaths left in $\sigma \cap \sigma'$ consist of bad edges which do not form a β-triad. Suppose we obtain l such remaining subpaths paths $\mathcal{B}_1, \ldots, \mathcal{B}_l$. In other words, we have decomposed the TSPN into three categories of subpaths:

- k_1 β-triads $\mathcal{T}_1, \ldots, \mathcal{T}_{k_1}$,
- k_2 paths $\mathcal{G}_1, \ldots, \mathcal{G}_{k_2}$ that cover the remaining good edges, and

- l paths $\mathcal{B}_1, \ldots, \mathcal{B}_l$ that consist only of bad edges which do not form β-triads.

We are now ready to evaluate the detour that each of these paths takes. For each $i \in [1, k_2]$ let ψ_i the natural order on the disks associated with $\mathcal{G}_i$ and let n_i be the number of edges in $\mathcal{G}_i$. We have that:

$$|TSP(\psi_i)| \leq |TSPN(\psi_i)| + (1 + \cos\beta)R \cdot n_i.$$

When it comes to the paths $\mathcal{B}_j$, with $j \in [1, l]$, let σ_j be their natural associated orders and let m_j be the number of edges it contains. We have that $|TSP(\sigma_j)| \leq |TSPN(\sigma_j)| + 2R \cdot m_j$.

Let $N = \sum_{i=1}^{k_2} n_i$ be the total number of edges in $\mathcal{G}_1, \ldots, \mathcal{G}_{k_2}$ and $M = \sum_{j=1}^{l} m_j$ the total number of edges in $\mathcal{B}_1, \ldots, \mathcal{B}_l$. By construction, we decomposed $TSPN(\sigma \cap \sigma')$ into these two groups of edge disjoint paths and we therefore get that:

$$\begin{aligned}
|TSP(\sigma \cap \sigma')| &= \sum_{i=1}^{k_2} |TSP(\psi_i)| + \sum_{j=1}^{l} |TSP(\sigma_j)| \\
&\leq \sum_{i=1}^{k_2} \Big(|TSPN(\psi_i)| + (1 + \cos\beta)R \cdot n_i \Big) \\
&\quad + \sum_{j=1}^{l} \Big(|TSPN(\sigma_j)| + 2R \cdot m_j \Big) \\
&\leq |TSPN(\sigma \cap \sigma')| + (1 + \cos\beta)RN + 2RM.
\end{aligned}$$

Including the β-triads back into our bound, we get that:

$$\begin{aligned}
|TSP^*| &\leq \frac{|TSP(\sigma)| + |TSP(\sigma')|}{2} \\
&\leq |TSP(\sigma \cap \sigma')| + L_T + 3\sqrt{3}R \cdot k_1 \\
&\leq |TSPN(\sigma \cap \sigma')| + L_T + 3\sqrt{3}R \cdot k_1 + \\
&\quad + (1 + \cos\beta)RN + 2RM \\
&\leq |TSPN| + 3\sqrt{3}R \cdot k_1 + (1 + \cos\beta)RN + 2RM.
\end{aligned}$$

In other words, we've expressed the total detour of the $TSPN$ according to edges that participate in β-triads, edges in $\mathcal{G}_1, \ldots, \mathcal{G}_{k_2}$ and edges in $\mathcal{B}_1, \ldots, \mathcal{B}_l$. By construction, none of these paths share edges and so $3k_1 + N + M = n$. Let $K = 3k_1 + N$ be the total number of edges either in a β-triad or in $\mathcal{G}_1, \ldots, \mathcal{G}_{k_2}$ and since $\sqrt{3} \leq 1 + \cos\beta$, we have that:

$$|TSP^*| \leq |TSPN| + (1 + \cos\beta)R \cdot K + 2R \cdot (n - K).$$

Case 1: when $K \geq \frac{n}{2}$. In this situation, we have that:

$$|TSP^*| \leq |TSPN^*| + \frac{3 + \cos\beta}{2} \cdot R \cdot n.$$

The average detour per edge $\frac{3+\cos\beta}{2}$ is better than the $2R$ bound, but it is constrained by the choice of $\beta \in [0, \pi/12]$, which means that the best we could hope for is an average detour of $\frac{1}{2}(3+\cos\frac{\pi}{12})R < 1.983R$. We note that the average detour in the Häme, Hyytiä and Hakula conjecture is $\sqrt{3}R \approx 1.732R$. Using Lemma 4 gives us that

$$\begin{aligned}
|TSP^*| \leq \Big(1 + \frac{2}{\pi} \cdot (3 + \cos\beta)\Big) \cdot |TSPN^*| + \\
+ 2 \cdot (3 + \cos\beta)R \qquad .
\end{aligned}$$

For large n, the $1 + \frac{2}{\pi} \cdot (3 + \cos\beta)$ term will dominate our approximation factor and is at most 3.525 when $\beta = \pi/12$.

Case 2: when $K < \frac{n}{2}$. In this situation, even the overall detour might be large, we will show that in fact, in this case, TSP^* is a 2-approximation and therefore, the best that it can be in general. We know that each path $\mathcal{B}_j$ consists of bad edges which do not form any β-triads. In other words, if P_1P_2 is an edge in it, then we know that $|O_1O2| > R/\sin(2\beta)$ which in turn means that $|P_1P_2| > (1/\sin(2\beta) - 2) \cdot R$. Overall we have that:

$$\begin{aligned}
|TSPN| &\geq \sum_{j=1}^{l} |\mathcal{B}_j| \geq \Big(\frac{1}{\sin(2\beta)} - 2\Big)R \cdot (n - K) \\
&\geq \Big(\frac{1}{2\sin(2\beta)} - 1\Big)R \cdot n.
\end{aligned}$$

Since the total detour could be at most $2R$ per edge, we get that:

$$|TSP^*| \leq \Big(1 + \frac{2}{\frac{1}{2\sin(2\beta)} - 1}\Big) \cdot |TSPN|.$$

When $\beta = \frac{1}{2}\arcsin\frac{1}{6}$, the detour from Case 1 becomes $\frac{3+\cos\beta}{2} \approx 1.998$ and the approximation factor from Case 2 becomes exactly 2. We note that the machinery described can be used to obtain more nuanced results. In particular, lower choices for β will drive the approximation factor in Case 2 even lower than 2, at the expense of a higher detour bound for Case 1.

□

In Appendix D, we perform a similar analysis for a more general threshold of $\frac{N}{\alpha}$ for some $\alpha > 1$ (rather than $\frac{N}{2}$). We give there the specific values of α and β that give us a factor 2.53-approximation. The case for overlapping disks follows from the same analysis when we use the algorithm of Dumitrescu and Tóth [15]. We include the details for it in Appendix E.

Acknowledgements. The author would like to thank Prof. Samir Khuller for suggesting the problem and for inspiring conversations on the topic, as well as the reviewers for helpful comments on the paper.

References

[1] E. M. Arkin, S. P. Fekete, and J. S. Mitchell. Approximation algorithms for lawn mowing and milling. *Computational Geometry*, 17(1-2):25–50, 2000.

[2] E. M. Arkin and R. Hassin. Approximation algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics*, 55(3):197–218, 1994.

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

[4] D. Bhadauria, O. Tekdas, and V. Isler. Robotic data mules for collecting data over sparse sensor fields. *Journal of Field Robotics*, 28(3):388–404, 2011.

[5] B. Bhattacharya, J. Czyzowicz, P. Egyed, G. Toussaint, I. Stojmenovic, and J. Urrutia. Computing shortest transversals of sets. *International Journal of Computational Geometry & Applications*, 2(04):417–435, 1992.

[6] H. L. Bodlaender, C. Feremans, A. Grigoriev, E. Penninkx, R. Sitters, and T. Wolle. On the minimum corridor connection problem and other generalized geometric problems. *Computational Geometry*, 42(9):939–951, 2009.

[7] F. Carrabs, C. Cerrone, R. Cerulli, and M. Gaudioso. A novel discretization scheme for the close enough traveling salesman problem. *Computers & Operations Research*, 78:163–171, 2017.

[8] T.-H. H. Chan and K. Elbassioni. A QPTAS for TSP with fat weakly disjoint neighborhoods in doubling metrics. *Discrete & Computational Geometry*, 46(4):704–723, 2011.

[9] T.-H. H. Chan and S. H.-C. Jiang. Reducing curse of dimensionality: Improved PTAS for TSP (with neighborhoods) in doubling metrics. *ACM Transactions on Algorithms (TALG)*, 14(1):9, 2018.

[10] W.-L. Chang, D. Zeng, R.-C. Chen, and S. Guo. An artificial bee colony algorithm for data collection path planning in sparse wireless sensor networks. *International Journal of Machine Learning and Cybernetics*, 6(3):375–383, 2015.

[11] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[12] G. Citovsky, J. Gao, J. S. Mitchell, and J. Zeng. Exact and approximation algorithms for data mule scheduling in a sensor network. In *International Symposium on Algorithms and Experiments for Sensor Systems, Wireless Networks and Distributed Robotics*, pages 57–70. Springer, 2015.

[13] M. de Berg, J. Gudmundsson, M. J. Katz, C. Levcopoulos, M. H. Overmars, and A. F. van der Stappen. Tsp with neighborhoods of varying size. *Journal of Algorithms*, 57(1):22–36, 2005.

[14] A. Dumitrescu and J. S. Mitchell. Approximation algorithms for TSP with neighborhoods in the plane. *Journal of Algorithms*, 48(1):135–159, 2003.

[15] A. Dumitrescu and C. D. Tóth. The traveling salesman problem for lines, balls, and planes. *ACM Transactions on Algorithms (TALG)*, 12(3):43, 2016.

[16] A. Dumitrescu and C. D. Tóth. Constant-factor approximation for TSP with disks. In *A Journey Through Discrete Mathematics*, pages 375–390. Springer, 2017.

[17] K. Elbassioni, A. V. Fishkin, and R. Sitters. Approximation algorithms for the Euclidean traveling salesman problem with discrete and continuous neighborhoods. *International Journal of Computational Geometry & Applications*, 19(02):173–193, 2009.

[18] E. Galceran and M. Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous systems*, 61(12):1258–1276, 2013.

[19] J. E. Goodman, J. O'Rourke, and C. D. Tóth, editors. *Handbook of Discrete and Computational Geometry, Third Edition*. CRC Press, 2017.

[20] J. Gudmundsson and C. Levcopoulos. A fast approximation algorithm for TSP with neighborhoods. *Nord. J. Comput.*, 6(4):469, 1999.

[21] L. Häme, E. Hyytiä, and H. Hakula. The Traveling Salesman Problem with Differential Neighborhoods. In *European Workshop on Computational Geometry (EuroCG)*, Morschach, Switzerland, Mar. 2011.

[22] J.-S. Liu, S.-Y. Wu, and K.-M. Chiu. Path planning of a data mule in wireless sensor network using an improved implementation of clustering-based genetic algorithm. In *Computational Intelligence in Control and Automation (CICA), 2013 IEEE Symposium on*, pages 30–37. IEEE, 2013.

[23] M. Ma, Y. Yang, and M. Zhao. Tour planning for mobile data-gathering mechanisms in wireless sensor networks. *IEEE Transactions on Vehicular Technology*, 62(4):1472–1483, 2013.

[24] C. S. Mata and J. S. Mitchell. Approximation algorithms for geometric tour and network design problems. In *Proceedings of the eleventh annual symposium on Computational geometry*, pages 360–369. ACM, 1995.

[25] J. S. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal on computing*, 28(4):1298–1309, 1999.

[26] J. S. Mitchell. A PTAS for TSP with neighborhoods among fat regions in the plane. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 11–18. Society for Industrial and Applied Mathematics, 2007.

[27] J. S. Mitchell. A constant-factor approximation algorithm for TSP with pairwise-disjoint connected neighborhoods in the plane. In *Proceedings of the twenty-sixth annual symposium on Computational geometry*, pages 183–191. ACM, 2010.

[28] C. L. Müller. Finding maximizing Euclidean TSP tours for the Häme-Hyytiä-Hakula conjecture. Technical report, Technical Report CGL-TR-13, ETHZ, 2011.

[29] S. Safra and O. Schwartz. On the complexity of approximating TSP with neighborhoods and related problems. *Computational Complexity*, 14(4):281–307, 2006.

[30] O. Tekdas, D. Bhadauria, and V. Isler. Efficient data collection from wireless nodes under the two-ring communication model. *The International Journal of Robotics Research*, 31(6):774–784, 2012.

[31] A. Weber. *Ueber den standort der industrien*, volume 2. 1909.

[32] G. O. Wesolowsky. The Weber problem: History and perspectives. *Computers & Operations Research*, 1993.

[33] B. Yuan, M. Orlowska, and S. Sadiq. On the optimal robot routing problem in wireless sensor networks. *IEEE Transactions on Knowledge and Data Engineering*, 19(9):1252–1261, 2007.

Appendix

A Proof of Theorem 2 : Introducing β-triads

We begin by getting some intuition as to where on the boundary must the TSPN hit the disks in order to visit them within a short distance (depending on the parameter β). Specifically, we show that if a segment P_1P_2 is bad, then the possible locations for P_1 and P_2 are limited to a small interval on the boundary.

Lemma 5 *If P_1P_2 is bad, then the angles $\angle O_1O_2P_2$ and $\angle O_2O_1P_1$ are $\leq \beta$.*

Proof. Let $\gamma = \angle O_1O_2P_2 \leq \pi$ and notice that $O_1P_2 = f(O_1O_2, \gamma)$. Consider the point Q where O_1P_2 intersects the first disk and note that the shortest distance from P_2 to the first disk is exactly $P_2Q = f(O_1O_2, \gamma) - R$. We therefore get that $P_1P_2 \geq P_2Q$. Now notice that, if $\gamma > \beta$, then $f(O_1O_2, \gamma) > f(O_1O_2, \beta)$ and so $P_2Q > f(O_1O_2, \beta) - R$, which would lead to a contradiction. The same argument can be applied for P_2 and we get our conclusion. □

We now consider the scenario in which there is a second bad edge P_2P_3 and further explore the local structure of the associated TSP on the centers. Specifically, let O_3 be the center of the disk visited next at P_3 and assume that the edge P_2P_3 is also bad. Notice that the angle $\angle O_1O_2O_3$ formed by the TSP is either $\angle O_1O_2P_2 + \angle P_2O_2O_3$ or $|\angle O_1O_2P_2 - \angle P_2O_2O_3|$. Regardless, we have that $\angle O_1O_2O_3 \leq \angle O_1O_2P_2 + \angle P_2O_2O_3$ and get the following corollary:

Corollary 6 *If both P_1P_2 and P_2P_3 are bad edges, then the angle $\angle O_1O_2O_3$ is $\leq 2\beta$.*

We now have that if P_1P_2 and P_2P_3 are bad edges and O_1O_2 is small, then the TSP edge O_2O_3 edge must itersect the disk centered at O_1. In general, it is not true that if O_2O_3 intersects the first disk, we immediately get that the associated TSPN edge P_2P_3 must also intersect it. In our case, however, we have that the slope of P_2P_3 is very close to the one of O_2O_3 due to the fact that it is a bad edge. We use this information to show that if O_2O_3 does not intersect the first disk, then P_2P_3 cannot be a bad edge.

Theorem 7 *If P_1P_2 and P_2P_3 are bad edges and $O_1O_2 \leq R/\sin(2\beta)$, then the segment P_2P_3 intersects the disk centered at O_1.*

Proof. We consider the case in which $\angle O_1O_2O_3 = \angle O_1O_2P_2 + \angle P_2O_2O_3$ and note that all the other cases are similar. We denote the two lines originating at O_2 that are tangent to the first circle as ℓ_1 and ℓ_2 such that the line O_2O_3 is in between ℓ_1 and O_2O_1. Note that this is possible because the angle that ℓ_1 forms with O_2O_1 is at least 2β (since $O_1O_2 \leq R/\sin(2\beta)$) but the angle that O_2O_3 forms with O_2O_1 is at most 2β (Corollary 6).

Our strategy will be to first show that the segment P_2P_3 is contained in the wedge defined by ℓ_1 and ℓ_2 (Figure 3). Notice that, since the wedge defines a convex space, it is enough to show that P_2 and P_3 are contained in it.

We first show that the point P_2 has to be in the wedge. Let S_1 and S_2 be the points in which the segment O_1O_2 intersects the first and second disk. Similarly, let T_2 and T_3 be the points in which O_2O_3 intersects the second and third disk. We then have that P_2 is between T_2 and S_2.

Now we only need to show that P_3 is in between ℓ_1 and ℓ_2. We will do that by arguing that any choice of P_3 outside of the wedge will contradict the fact that P_2P_3 is a bad edge.

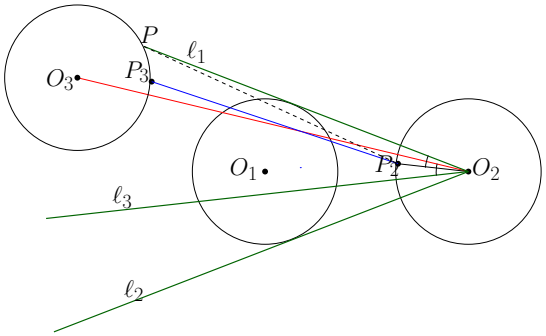


Figure 3: When O_2O_3 crosses the disk centered at O_1, we must also have that the segment P_2P_3 also crosses it. We show this by arguing that P_2P_3 is contained between the two lines ℓ_1 and ℓ_3 and that P_2 and P_3 are on separate sides of the first disk.

Let $\alpha_1 = \angle O_1O_2P_2$ and $\alpha_2 = \angle P_2O_2O_3$, and so $\alpha_1, \alpha_2 \leq \beta$ (Lemma 5). First notice that if neither ℓ_1 nor ℓ_2 intersect the third disk, then we are done because we have that the entire boundary is contained in the convex space (since O_3 is already in between ℓ_1 and ℓ_2). Assume then that ℓ_1 intersects the third disk at a point P above the line O_2O_3, since $\angle O_3O_2O_1 \leq 2\beta \leq \angle PO_2O_1$. Moreover, since $\angle PO_2O_1 \geq 2\beta$, we have that $\angle PO_2P_2 \geq 2\beta - \alpha_1 \geq \beta$ and so $PP_2 \geq f(PO_2, \beta)$ (because $|P_2O_2| = R$). Since $|PO_2| \geq |T_3O_2|$ and $|T_3O_2| \geq R$, this implies that $|PP_2| \geq f(T_3O_2, \beta)$. Using the fact that $f(x, \beta) \geq x - R\cos\beta$ for any x and $\beta \neq 0$, one can verify that:

$$\begin{aligned} f(T_3O_2, \beta) &= f(O_2O_3 - R, \beta) \\ &= \sqrt{(|O_2O_3| - R)^2 + R^2 - 2R(|O_2O_3| - R)\cos\beta} \\ &> \sqrt{|O_2O_3|^2 + R^2 - 2R|O_2O_3|\cos\beta} - R \\ &> f(O_2O_3, \beta) - R. \end{aligned}$$

This means that P cannot be a possible position for P_3 because then P_2P_3 would be too big. Moreover, any point Q "above" P (i.e. such that $\angle O_3O_2Q > \angle O_3O_2P$) would also not work as a possible position for the same reason. In other words, P_3 has to be underneath the line $PO_2 = \ell_1$.

In order to prove that P_3 is also above the line ℓ_2, we will consider an additional line ℓ_3 originating at O_2 that makes an angle of β with O_2P_2 and is underneath it. This new line makes an angle of $\beta + \alpha_2$ with O_2O_3 and since ℓ_2 makes an angle of $\geq 2\beta + \alpha_1 + \alpha_2$ with O_2O_3, we get that ℓ_3 is in between O_2O_3 and ℓ_2. In other words, if we show that P_3 is above ℓ_3, then we also get that P_3 is above ℓ_2. If ℓ_3 does not intersect the third disk, then we are done as before, so assume that it intersects it at a point Q on the boundary. Similarly as before, we have that $P_2Q = f(O_2Q, \beta) \geq f(O_2T_3, \beta) > f(O_2O_3, \beta) - R$. This in turn implies that P_3 has to be above Q, otherwise P_2P_3 would be too big. Therefore P_3 must be above the line ℓ_3.

At this point, we have that the segment P_2P_3 is contained in the wedge defined by ℓ_1 and ℓ_2. We know that the first disk is tangent on both sides to ℓ_1 and ℓ_2 but this does not directly imply that P_2P_3 must actually intersect it. In order to have that, we must also ensure that P_2 and P_3 lie on different sides of the first disk. We argue this by showing that O_3 itself must be on the other side of the first disk as O_2. Since the disks do not intersect, this implies that P_3 is on a different side from P_2. In order to show this, notice that we can assume, without loss of generality, that $O_1O_2 \leq O_2O_3$. Let T be the point on O_2O_3 such that $O_1T \perp O_2O_3$. Since $\angle O_1O_2O_3 \leq 2\beta$ and $O_1O_2 \leq R/\sin(2\beta)$, this means that T is contained in the first disk. Suppose that O_3 is on the segment O_2T (effectively in between O_1 and O_2). Then $O_2O3 < O_2T$ but, since $O_2T = O_1O_2\cos(\angle O_1O_2T) \leq O_1O_2$, this would lead to a contradiction. We therefore get that O_2 and O_3 are on different sides of the first disk and that the same is true for P_2 and P_3. This shows that the segment P_2P_3 must intersect the first disk. □

We are now ready to prove **Theorem 2**: *For $n \geq 4$, if P_1P_2 and P_2P_3 are bad edges and $O_1O_2 \leq R/\sin(2\beta)$ then at least one of the following is true:*

- *the TSPN tour is not optimal,*
- *the TSPN tour is supported by a straight line or*
- *the path $P_n - P_1 - P_2 - P_3$ forms a β-triad.*

Proof. We distinguish between the case in which P_2P_3 intersects the first disk at P_1 and otherwise. In the first case, we will show that either the TSPN is not optimal or all the disks are stabbed by it. The second case is more involved and reduces to describing what the local structure of the TSPN must be such that it does not necessarily fall in the previous two cases.

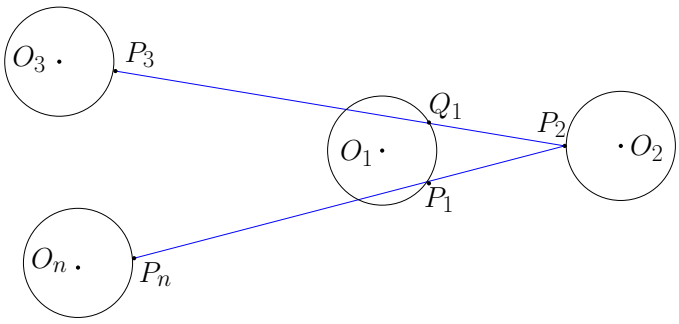


Figure 4: The path $P_n - P_1 - P_2 - P_3$ forms a β-triad.

Case 1: P_1, P_2, P_3 are collinear. Then consider the point P_n that connects to P_1. The cost that the TSPN pays for visiting the four disks is $|P_nP_1| + |P_1P_2| + |P_2P_3|$ but by triangle inequality, we know that $|P_nP_2| \leq |P_nP_1| + |P_1P_2|$, so the TSPN would visit P_2 directly and pass through P_1 on its way to P_3. If the inequality is strict, then this

directly implies that the TSPN is not optimal. When we have equality, however, this implies that P_n, P_1 and P_2 are now also collinear and furthermore, that P_1 lies between P_2 and P_n. In other words, we have that on the line from P_2 to P_n, we have both P_3 and P_n to the left of P_1. Now look at how point P_4 connects to P_3 and notice that the portion of TSPN for the five disks is now $|P_4P_3|+|P_3P_2|+|P_1P_2|+|P_1P_n|$ and again, we can ask the question of why wouldn't the TSPN go straight to P_2 instead and visit P_3 along the line P_2P_3. Specifically, we have $|P_4P_2| \leq |P_4P_3| + |P_3P_2|$ with the TSPN not being optimal whenever this inequality is strict. We therefore consider the case in which $|P_4P_2| = |P_4P_3| + |P_3P_2|$ and get that now P_4 has to also be collinear with the other points and furthermore, P_3 has to be between P_4 and P_2. Continuing this process, we get that all the TSPN points would have to be collinear and in the order $P_2, P_1, P_3, P_4, \ldots, P_{n-1}$ with P_n potentially being anywhere past P_1. In this case, we have that the TSPN is a supported by a straight line that stabs all of the disks.

□

Case 2: P_1, P_2, P_3 are not collinear. Let the line P_1P_2 intersect the first disk for the first time at Q_1. By the argument from before, we know that if $|P_nP_2| < |P_nP_1| + |P_1P_2|$, then the TSPN cannot be optimal since another tour could go from P_n straight to visiting P_2 and then visit P_1 on the way to P_3, at a lesser cost. When P_n, P_1 and P_2 are collinear, in that order, we say that $P_n - P_1 - P_2 - P_3$ form a β-triad.

B Proof of Lemma 8 : β-triads are Edge Disjoint

As a reminder, we defined a bad triad to be a subpath of the tour $P_n - P_1 - P_2 - P_3$ that visits the disks centered at O_n, O_1, O_2 and O_3 and has all the following properties:

- P_1P_2 is a bad edge, i.e. $|P_1P_2| \leq f(O_1, O_2, \beta) - R$,
- P_2P_3 is a bad edge, i.e. $|P_2P_3| \leq f(O_2, O_3, \beta) - R$,
- O_1O_2 is short, i.e. $|O_1O_2| \leq R/\sin(2\beta)$,
- P_1, P_2, P_3 are not collinear and
- P_n, P_1, P_2 are collinear with P_1 between P_n and P_2.

Theorem 2 says that if $|TSPN^*|$ is not a straight line, then the triad has a local detour of at most $3\sqrt{3}R$. Lemma 8 further states that all the bad triads are also edge disjoint. In order to prove that, we go back to the proof of Theorem 2. Note that we distinguished between the case in which P_1, P_2 and P_3 are collinear (Case 1) and when they are not(Case 2). The first case leads to the TSPN being a straight line, which is ruled out by our assumptions. In the second case, the optimality of TSN^* implies that P_n, P_1 and P_2 are also collinear, with P_1 between P_2 and P_n.

Lemma 8 *All the β-triads in a given TSPN tour are edge disjoint.*

Proof. Assume there is another bad triad that shares edges with $P_n - P_1 - P_2 - P_3$. We distinguish four cases, based on the type of edges they have in common.

Case 1: $P_{n-1} - P_n - P_1 - P_2$ is a bad triad. This case cannot happen since P_n, P_1, P_2 are collinear.

Case 2: $P_{n-2} - P_{n-1} - P_n - P_1$ is a bad triad. Then, by definition, we must have that P_nP_1 is also a bad edge. We will show, however, than this cannot be. For this, we will use an additional lemma:

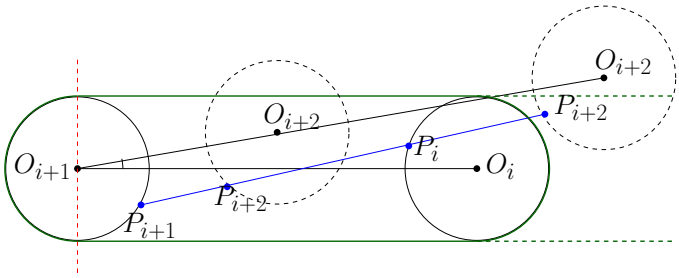


Figure 5: A potential TSPN path is drawn in blue. Because the angle $\angle O_iO_{i+1}O_{i+2} \leq \pi/2$, we have that O_i and O_{i+1} are on the same side of the hyperplane described by the red line. That, in turns, give two options for the disk centered at O_{i+2} to intersect the (extended) convex hulls of the other two diskss, drawn in green. In each case, the points are visited in the wrong order.

Lemma 9 *If P_iP_{i+2} is a straight line that passes through point P_{i+1} such that P_{i+1} is between P_i and P_{i+2}, then it cannot be that both P_iP_{i+1} and $P_{i+2}P_{i+1}$ are bad edges.*

Proof. Assume that both P_iP_{i+1} and $P_{i+2}P_{i+1}$ are bad edges. Then Corollary 6 implies that the angle $\angle O_iO_{i+1}O_{i+2} \leq 2\beta$. Now consider the convex hull of the two disks centered at O_i and O_{i+1} (Figure 5). If the disk centered at O_{i+2} intersects the convex hull, then P_{i+2} must be contained in that convex hull, otherwise the line $P_i - P_{i+1} - P_{i+2}$ would not exist. But in that case, the points would be visited out of order. Specifically, P_{i+2} would be between P_i and P_{i+1}.

Now assume that the disk centered at O_{i+2} does not intersect the convex hull. Since the angle $\angle O_iO_{i+1}O_{i+2} \leq 2\beta \leq \pi/6$, this implies that O_{i+2} is in the same halfspace as O_i with respect to the line perpendicular to O_iO_{i+1} passing through O_{i+1}. We extend the convex hull infinitely in that halfspace by allowing the tangent lines to be infinite on that side. By the same argument as before, we know that the disk centered at O_{i+2} must intersect this extended region. But then we would get again that the points are out of order: P_i would be between P_{i+1} and P_{i+2}. □

When $P_i = P_n, P_{i+1} = P_1$ and $P_{i+2} = P_2$, Lemma 9 tells us that it cannot be that P_1 is between P_n and P_2 and both edges P_1P_n and P_1P_2 are bad. Therefore we are done with this case.

Case 3: $P_1 - P_2 - P_3 - P_4$ is a bad triad. This case is similar to Case 1 and cannot happen, since P_1, P_2 and P_3 cannot be collinear.

Case 4: $P_2 - P_3 - P_4 - P_5$ is a bad triad. This case is similar to Case 2 because we have that P_2, P_3 and P_4 are collinear with P_3 between P_2 and P_4 and both P_2P_3 and P_3P_4 being bad edges.

□

C The TSPN that follows a straight line

Here we focus on the second possibility in Theorem 2 in which the optimal TSPN is supported by a straight line that stabs all the disks. We show that in this case, we can return in polynomial time a solution that is within an additive factor of $4R$ from the optimal $TSPN^*$. We note that when the TSPN might not be a line but the disks themselves admit a line transversal, a $\sqrt{2}$-approximation follows from the work of Dumitrescu and Mitchell [14]. We explain the result for completeness.

We start by identifying the centers that are the farthest apart and considering the direction orthogonal to the line going through them. This direction induces parallel segments of length $2R$ in each of the disks (that each go through the centers). It is easy to check that any line transversal through the disks is a line transversal through the segments except for the first and last disk in the associated geometric permutation (for those two disks, the TSPN will stop at the boundary of the disk and never cross the entire circle). Conversely, any line transversal through the segments will automatically also stab the disks. Now compute a shortest line segment that stabs all of these segments in time $O(n \log n)$ using the algorithm of Bhattacharya et al. [5]. We note that this is optimal up to an additive factor of $4R$ that comes from the fact that the optimal $TSPN^*$ might have to travel $4R$ to hit the first and the last two segments in the geometric permutation.

In general, when we know that the disks admit a line transversal, we can output a solution that is a $\sqrt{2}$-approximation [14]. This follows indirectly from an algorithm used for connected regions of the same diameter, when there is a line that stabs all of the diameters. Given the parallel segments of length $2R$ that we constructed earlier, we know that they can also be stabbed by a line. Now consider the smallest perimeter axis-aligned rectangle that intersects all of the segments, of width w and height h. This will be the solution that we return. Arkin and Hassin [2] argued that any tour which touches all four sides of the rectangle must have length at least $2\sqrt{h^2+w^2}$. Since $h+w \leq \sqrt{2}\cdot\sqrt{h^2+w^2}$, we get that the rectangle is a $\sqrt{2}$-approximation.

D Using different thresholds

In this section, we give a more general analysis of the general approximation factor with a parameter $\alpha > 1$ that we will set later in the proof. We include here only the aspects that change. Depending on whether $K \leq \frac{n}{\alpha}$ or not, we will employ different lower bounds on $|TSPN|$, in a similar fashion as before.

Case 1: when $K \geq \frac{n}{\alpha}$. In this situation, we have that:

$$|TSP^*| \leq |TSPN^*| + \frac{1+\cos\beta+2(\alpha-1)}{\alpha} \cdot R \cdot n.$$

Using Lemma 4 gives us that

$$\begin{aligned}|TSP^*| &\leq \Big(1+\frac{4}{\pi}\cdot\frac{1+\cos\beta+2(\alpha-1)}{\alpha}\Big)\cdot|TSPN^*|+\\ &\quad+4\cdot\frac{1+\cos\beta+2(\alpha-1)}{\alpha}R\\ &\leq \Big(1+\frac{4}{\pi}\cdot\frac{1+\cos\beta+2(\alpha-1)}{\alpha}\Big)\cdot|TSPN^*|+8R\\ &\leq \Big(1+\frac{8}{\pi}-\frac{4}{\pi}\cdot\frac{1-\cos\beta}{\alpha}\Big)\cdot|TSPN^*|+8R.\end{aligned}$$

Case 2: when $K < \frac{n}{\alpha}$. Similarly as before, we get that:

$$\begin{aligned}|TSPN| &\geq \sum_{j=1}^{l}|\mathcal{B}_j|\\ &\geq \Big(\frac{1}{\sin(2\beta)}-2\Big)R\cdot(n-K)\\ &\geq \frac{\alpha-1}{\alpha}\cdot\Big(\frac{1}{\sin(2\beta)}-2\Big)R\cdot n.\end{aligned}$$

Since the total detour could be at most $2R$ per edge, we get that:

$$|TSP^*| \leq \Big(1+\frac{\alpha}{\alpha-1}\cdot\frac{2}{\frac{1}{\sin(2\beta)}-2}\Big)\cdot|TSPN|.$$

Setting $\alpha = 1 + 2/(c/\sin(2\beta) - 2c - 2)$ for $c = 2.53$ and $\beta = 0.1831$ gives us that both of these cases lead to a 2.53-approximation.

E Algorithm for overlapping disks

We discuss how the analysis from the disjoint case carries over to the case of overlapping disks. As we mentioned before, the best known approximation for this case is by Dumitrescu and Tóth [15]. In general, approaches for this case take advantage of known analyses for the disjoint case and adapt them in a smart way to the overlapping case. We begin by roughly describing the technique of Dumitrescu and Tóth [15] and the show how the analysis changes when we use our framework.

Specifically, Dumitrescu and Tóth [15] start by computing a monotone maximal set of disjoint disks $\mathcal{I}$ by greedily selecting the leftmost disk and deleting all of the other input disks that intersect it. Let k be the size of the set we end up wth. They then compute an approximate TSP tour on the centers of the disks in $\mathcal{I}$, either using the available schemes [3, 25] or Christofides [11]. We call this tour $T_{\mathcal{I}}$. They then augment this tour in such a way that we visit all the input disks, not just the ones in $\mathcal{I}$. Before we discuss the augmentation part, we first define some notation and mention some bounds that follow naturally.

Let the optimal TSP tour on the centers in $\mathcal{I}$ be $TSP^*_{\mathcal{I}}$. The eventual tour $T_{\mathcal{I}}$ that we compute will be an a-approximation to $TSPN^*_{\mathcal{I}}$ so we have that:

$$|T_{\mathcal{I}}| \leq a \cdot |TSP^*_{\mathcal{I}}|. \tag{2}$$

On the other hand, we know that this set of disks also has an associated optimal TSPN tour, which we call $TSPN^*_I$.

Finally, we denote the optimal TSPN tour on all the disks by $TSPN^*$. We know that the tour on $\mathcal{I}$ is a lower bound:

$$|TSPN^*_{\mathcal{I}}| \leq |TSPN^*|. \tag{3}$$

The size of our final solution will be compared to $|TSPN^*|$ and to that end, we use lower bounds on $|TSPN^*_{\mathcal{I}}|$ in conjunction with (3) to get lower bounds on $|TSPN^*|$. This is the part where our new framework will come in, because $|TSPN^*_{\mathcal{I}}|$ is a tour on disjoint disks by definition.

The next step is to augment $T_{\mathcal{I}}$ with detours of length $O(R)$ along the disks in $\mathcal{I}$ such that it touches every other disk not in $\mathcal{I}$. The total length of the solution would then become $|\mathcal{T}_{\mathcal{I}}| + O(1) \cdot |\mathcal{I}| \cdot R$. Specifically, Dumitrescu and Tóth [15] consider short curves around each disk in $\mathcal{I}$ that are guaranteed to cross any of the disks to its right that intersect it. Because the maximal set was chosen from left to right, that covers all the disks that could possibly intersect it. We refer the reader to [15] for the detailed construction. The authors show that the length of the resulting tour T is within $O(1) \cdot |\mathcal{I}| \cdot R$ of $|T_{\mathcal{I}}|$:

$$|T| \leq |T_{\mathcal{I}}| + (A \cdot k + B) \cdot R, \tag{4}$$

where $A = 2 \cdot (\frac{\pi}{6} + \sqrt{3} - 1)$ and $B = 4 - \sqrt{3}$.

Combining 2 and 4, we upper bound the length of the solution $|T|$ in terms of $|TSP^*_{\mathcal{I}}|$ as such:

$$\begin{aligned} |T| &\leq |T_{\mathcal{I}}| + (A \cdot k + B) \cdot R \\ &\leq a \cdot |TSP^*_{\mathcal{I}}| + (A \cdot k + B) \cdot R \end{aligned}$$

In order to complete the analysis, we would need to bound $|TSP^*_{\mathcal{I}}|$ in terms of $|TSPN^*|$ and we do that through $|TSPN^*_{\mathcal{I}}|$. The analysis from Dumitrescu and Tóth [15] uses the bounds from Dumitrescu and Mitchell [14] for the case of disjoint disks. Specifically, they apply Lemma 4 to get that:

$$kR \leq \frac{4}{\pi} \cdot |TSPN^*_{\mathcal{I}}| + 4R.$$

This, together with the bound $|TSP^*_{\mathcal{I}}| \leq |TSPN^*_{\mathcal{I}}| + 2Rk$ and (3) yields:

$$\begin{aligned} |T| &\leq a \cdot |TSP^*_{\mathcal{I}}| + (Ak + B) \cdot R \\ &\leq a \cdot (|TSPN^*_{\mathcal{I}}| + 2Rk) + (Ak + B) \cdot R \\ &\leq a \cdot |TSPN^*_{\mathcal{I}}| + (2a + A) \cdot kR + BR \\ &\leq a \cdot |TSPN^*_{\mathcal{I}}| + (2a + A) \cdot \Big(\frac{4}{\pi}|TSPN^*_{\mathcal{I}}| + 4R\Big) + BR \\ &\leq \Big(a + (2a + A)\frac{4}{\pi}\Big) \cdot |TSPN^*_{\mathcal{I}}| + (8a + 4A + B)R \\ &\leq \Big((1 + \frac{8}{\pi})a + \frac{4A}{\pi}\Big) \cdot |TSPN^*_{\mathcal{I}}| + (8a + 4A + B)R \\ &\leq \Big((1 + \frac{8}{\pi})a + \frac{4A}{\pi}\Big) \cdot |TSPN^*| + (8a + 4A + B)R \end{aligned}$$

Plugging in the values for A and B gives an overall approximation term of:

$$(1 + \frac{8}{\pi})a + \frac{4A}{\pi} \leq \Big(\frac{7}{3} + \frac{8\sqrt{3}}{\pi}\Big) \cdot (1 + \epsilon) \leq 6.75 \cdot (1 + \epsilon).$$

Our framework changes the last stage in which we compare $|TSP^*_{\mathcal{I}}|$ with $|TSPN^*_{\mathcal{I}}|$. We do a similar analysis as in the disjoint case, except for the tour on $\mathcal{I}$. We get that **Case 1** would therefore correspond to getting that:

$$|TSP^*_{\mathcal{I}}| \leq |TSPN^*_{\mathcal{I}}| + X \cdot R \cdot k,$$

where $X = 2 - \dfrac{1 - \cos\beta}{\alpha}$ (instead of $2R$). We can then replace it in the analysis and get:

$$\begin{aligned} |T| &\leq a \cdot (|TSPN^*_{\mathcal{I}}| + XRk) + (Ak + B) \cdot R \\ &\leq \Big(a + (Xa + A)\frac{4}{\pi}\Big) \cdot |TSPN^*_{\mathcal{I}}| + (8a + 4A + B)R \\ &\leq \Big((1 + \frac{4X}{\pi})a + \frac{4A}{\pi}\Big) \cdot |TSPN^*| + (4Xa + 4A + B)R \end{aligned}$$

In **Case** 2, we have that the overall detour is $2Rk$, but there is a different lower bound on $|TSPN^*_{\mathcal{I}}|$:

$$|TSPN^*_{\rangle}| \geq Y \cdot Rk,$$

where $Y = \frac{\alpha - 1}{\alpha} \cdot \Big(1/(2\sin(2\beta)) - 1\Big)$. Using the fact that $Rk \leq 1/Y \cdot |TSPN^*_{\mathcal{I}}|$, the analysis then becomes:

$$\begin{aligned} |T| &\leq \alpha \cdot |TSPN^*_{\mathcal{I}}| + (2\alpha + A) \cdot kR + BR \\ &\leq \alpha \cdot |TSPN^*_{\mathcal{I}}| + \frac{2\alpha + A}{Y} \cdot |TSPN^*_{\mathcal{I}}| + BR \\ &\leq \Big(\alpha + \frac{2\alpha + A}{Y}\Big) \cdot |TSPN^*_{\mathcal{I}}| + BR. \end{aligned}$$

If we set α and β like in the previous section, we get that both of the approximation factors are upper bounded by 6.728.

F The Fermat-Weber Point Approach for $n = 3$

In this section, we prove that the Häme, Hyytiä and Hakula conjecture is true for $n = 3$ and discuss a different way of looking at the TSPN tour that we believe might be of independent interest. We start with the observation that the shortest tour on the centers is equivalent to the shortest tour on translates of those centers, as long as all those centers are translated according to the same vector. In other words, if we fix a direction and translate each center along that direction until it reaches its boundary, the shortest tour on the newly obtained points will be exactly the same as the shortest tour on the centers themselves.

Formally, let B_i be the point we obtain by translating the center O_i along a fixed vector of length R. Then the TSP on the points $B_1, B_2, \ldots, B_n$ (in that order) has the same length as the TSP tour on $O_1, O_2, \ldots O_n$ (Figure 6). One advantage of visiting the first set of points (instead of the center points) is that it might be more similar geometrically to what the TSPN actually does. In terms of the following analysis, we would get that:

$$|TSP^*| \leq |TSPN^*| + 2\sum_{i=1}^{n} |P_iB_i|.$$

In this context, a natural question arises about the choice for the points B_i that minimizes the term $\sum_{i=1}^{n} |P_iB_i|$. In order to see what this best choice would be, we transform this input instance into another one by essentially superimposing all the disks on top of each other (Figure 7). Specifically, our

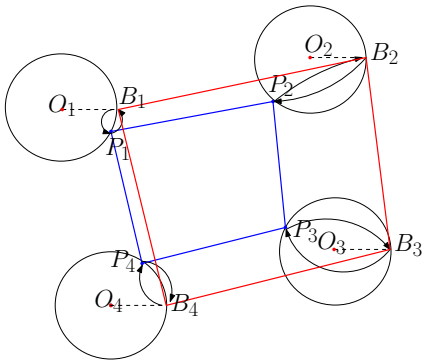


Figure 6: The translated view, when the tour visits the same point on the boundary of each disk.

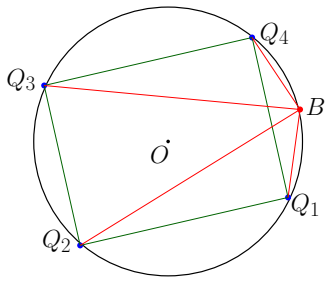


Figure 7: In red, the unified view when we translate each O_i to the same point B on the boundary. In green, the detour when we pick a different B for each pair of points O_i and O_{i+1}.

new instance will consist of one disk of radius R centered at a point O such that the points B_i map to a single point B (corresponding to O translated by the same fixed vector). We then map each point P_i of the TSPN to a corresponding point Q_i on the boundary of this disk such the vector OQ_i is a translate of the vector O_iP_i. We then get that:

$$\sum_{i=1}^{n}|P_iB_i| = \sum_{i=1}^{n}|Q_iB|,$$

and so the best choice for B is the one that minimizes the sum $\sum_{i=1}^{n}|Q_iB|$, otherwise know as the *Fermat-Weber point* or 1-*median* of the points $Q_1, Q_2, \ldots, Q_n$ [32, 31]. We note, however, that while the average distance to the Fermat-Weber point will never be greater than $2R$, there are instances in which this is tight. Consider, for example, the points Q_i to be the vertices of a convex $2n$-gon and notice by triangle inequality that the center of the disk is exactly their Fermat-Weber point (any other point will incur distances greater than the sum of the diagonals).

We can therefore say that when the points B_i are evenly spaced on the boundary of the disk the Fermat-Weber point is exactly the center and so we gain no improvement by moving the centers O_i towards the points B_i. It turns out, however, that the location of the points on the boundary is not as restrictive as the order in which the TSPN visits them. To see that, consider a different transformation in which we only move the centers O_i and O_{i+1} along a fixed vector. In other words, we choose a new vector for each pair of consecutive centers and only compare $|P_iP_{i+1}|$ locally against the newly obtained segment. This does not give us an overall valid tour on the centers, but it allows us to tailor the choice of B for each two points P_i and P_{i+1}. Specifically, we would get that:

$$|O_iO_{i+1}| \leq |P_iP_i + 1| + |Q_iB| + |Q_{i+1}B|.$$

In this case, we know that any point on the segment Q_iQ_{i+1} minimizes the distances in question and so we get that:

$$|O_iO_{i+1}| \leq |P_iP_i + 1| + |Q_iQ_{i+1}| \text{ and}$$
$$|TSP^*| \leq |TSPN^*| + \sum_{i=1}^{n}|Q_iQ_{i+1}|.$$

In other words, the largest detour obtained in this way is when the TSPN visits the points P_i in the order of the Maximum TSP on the associated points Q_i (Figure 7). The case in which all the points are evenly distributed along the boundary no longer becomes that restrictive. We can still construct, however, instances for which the Max TSP is exactly $2Rn$ and that is when the points visited are exactly diametrically opposite each other. Nevertheless, we are able to show that for $n = 3$, the detour is bounded by $3\sqrt{3}R$. Let A, B, C be any three points on the boundary of a circle of disk R centered at O. We then have that that $|AB| + |AC| + |BC| \leq 3\sqrt{3}R$ and the Häme, Hyytiä and Hakula conjecture for $n = 3$ follows:

Theorem 10 *For $n = 3$, we have that any tour which visits the disks in an order σ satisfies the bound*

$$|TSP(\sigma)| \leq |TSPN(\sigma)| + 3\sqrt{3}R.$$

Width and Bounding Box of Imprecise Points

Vahideh Keikha* Maarten Löffler† Ali Mohades* Zahed Rahmati*

Abstract

In this paper we study the following problem: we are given a set $L = \{l_1, ..., l_n\}$ of parallel line segments, and we wish to find a set $P = \{p_1, ..., p_n\}$, where $p_i \in l_i$ such that we maximize/minimize the width of P or the area of the bounding box of P among all possible choices for P. We design an $O(n^2\epsilon^{-4.5})$ approximation algorithm for computing the largest width. We also show that the smallest width and the smallest bounding box can be computed in $O(n^2)$ time. We then proceed to present an $O(n^6)$ time dynamic programming algorithm for computing the largest-area bounding box. We also present an FPTAS for this problem which runs in $O(n^2\epsilon^{-5})$ time.

1 Introduction

Shape fitting is a fundamental problem in computational geometry, computer vision, clustering, data mining and many other areas, which asks the following question: suppose we are given a set P of points in the plane, find a shape that best fits P under some fitting criterion. In computational geometry, many problems fit into the class of shape fitting, e.g., computing the bounding box, the width, the smallest enclosing circle, etc. However, in the real-world, the input is subject to be *imprecise*. Then the question is finding tight bounds on the size of the objective shape.

Imprecise data. Let $P = \{p_1, ..., p_n\}$ be a set of fixed points in the plane. In many applications, each element of P is subject to be computed with some errors, such that we do not know e.g., the exact coordinates of each p_i, or even the existence of p_i. In this situation we call P a set of *imprecise/uncertain points*.

Many studies have focused on solving geometric problems in the presence of imprecise input. Depending on the information we have about the input, different models of imprecision are introduced. Here we briefly mention related models: the *Epsilon-geometry* model, the *Region-based* model, the *Locational* and the *Existential* model, where in these models, it is assumed that there exists a set P_i of points instead of each p_i, but the exact location of p_i in P_i is unknown. See e.g., [3, 5].

*Department of Mathematic and Computer Science, Amirkabir University, [va.keikha,mohades,zrahmati]@aut.ac.ir

†Department of Information and Computing Sciences, Utrecht University, Utrecht, The Netherlands, m.loffler@uu.nl

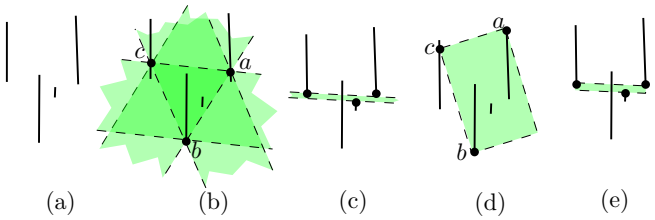


Figure 1: An example. (a) A given set of parallel line segments. (b) The Largest possible width determined by 3 pairs simultaneously. If we move any of a, b or c among their line segments, we reduce at least one of the computed width. (c) The smallest possible width. (d) The largest possible area bounding box. (e) The smallest possible bounding box.

In this paper, we study our problems in the Region-based model. Let R be a set of imprecise points. An *instance* of R is a set P of points selected from distinct regions of R. Then each instance P of R will have different convex hull, width, bounding box, etc. Löffler and van Kreveld introduced a framework for computing some tight lower and upper bounds on the size of such measures, where they modeled the uncertainty of the input by line segments, squares or disks [3].

Contribution. In this paper, we study the following problems: given a set $L = \{l_1, ..., l_n\}$ of parallel line segments, choose a set $P = \{p_1, ..., p_n\}$ of points, where $p_i \in l_i$, such that the size of width or the area of the bounding box of P is as small/large as possible among all possible choices for P (see Figure 1(b-e)). These problems can be interpreted as finding the optimal facilities in the form of a box or a strip which intersects each line-segment-customer.

Preliminaries. Löffler and van Kreveld firstly studied the problem of computing the largest/smallest axis-aligned bounding box of a set of imprecise points modeled as a set of disks or squares in the plane, where their algorithms varied from $O(n \log n)$ to $O(n^2)$ [2]. In the same paper, they proved that computing the largest possible width of a set of imprecise points modeled as a set of arbitrary line segments is NP-hard. The same problem for parallel line segments, squares or disks was posed as open question.

The *axis-aligned bounding box* of a set P of fixed points in the plane is the minimum area bounding box containing P, subject to the constraint that the edges

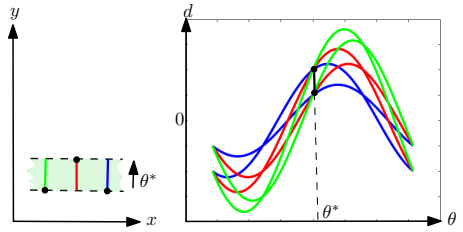


Figure 2: An example. The $\mathcal{A}(\theta, d)$ diagrams of the endpoints of three line segments and two determined widths in direction θ^*. Both the smallest and largest possible width occur in direction θ^*. When we select the lower endpoints of the blue and green line segments, the location of the point on the red line segment determines the width: the lower endpoint of the red segment realizes the smallest possible width, while the upper endpoint realizes the maximum possible width.

of the bounding box are parallel to the $x-y$ coordinate axes. The smallest *oriented bounding box* of P is the minimum area rectangle containing P. From now on, we simply call it *bounding box*. The *width* of a set of points is the narrowest strip containing P. These problems are extensively studied and efficient algorithms are known for them. Once the convex hull of P is known, all these problems can be solved in linear time based on the rotating calipers method [6]. While the convex hull of P is unknown there is an $\Omega(n \log n)$ lower bound for both problems of computing the bounding box and width of P in 2-D.

Results. Let $L = \{l_1, ..., l_n\}$ be a set of parallel line segments. We obtain the following results.

- We show that the largest possible width of L can be approximated within a factor $(1-2\epsilon)$ in $O(n^2\epsilon^{-4.5})$ time (Section 2.1).[1]
- We show that the smallest bounding box of L can be computed in $O(n^2)$ time (Section 3.1).
- We present a more involved $O(n^6)$ time dynamic programming algorithm for computing the largest bounding box of L. We also present an FPTAS for this problem which runs in $O(n^2\epsilon^{-5})$ time (Section 3.2).

We also note that all missing proofs are in Appendix A.1.

2 Width

We start with the *width* problem. Two problems can be considered: finding an instance P on L, so that P maximizes/minimizes the width of P. The minimum width of a set of imprecise points modeled as line segments (or any other convex regions), can be computed in $O(n \log n)$ time [4], in which the problem is so-called *strip transversal*, and the authors studied the problem of computing the thinnest strip that intersects a given set of convex objects. The maximum width problem looks more difficult, because we should find an instance so that our instance maximizes the width of the resulting point set. Since width can be determined by multiple triples of points, and each point can take part in different triples, it looks difficult to find the optimal position of the points (see Figure 1(b)).

Let $L = \{l_1, ..., l_n\}$ be a set of parallel line segments in the plane. Let l_i^- and l_i^+, respectively, denote the lower and upper endpoints of l_i. Let E denote the set of all the endpoints of segments in L. For a point p, let l_p denote the segment that includes p. For each point $p = (x, y)$, we define a $\mathcal{A}(\theta, d)$ diagram to be the plot of the function $d_\theta = x \sin\theta + y \cos\theta$ (1). It is the (signed) distance of p to a line through the origin perpendicular to the ray with angle θ.

$$p_\theta = \begin{bmatrix} x_\theta \\ y_\theta \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x\cos\theta - y\sin\theta \\ x\sin\theta + y\cos\theta \end{bmatrix}$$

In Figure 2, the $\mathcal{A}(\theta, d)$ diagram of the endpoints of a set of segments is depicted, the region between same color diagrams denotes the $\mathcal{A}(\theta, d)$ diagrams of the remaining points of the segment. From now on, we use $T(P)$ to address the set of $\mathcal{A}(\theta, d)$ diagrams of a set P of points. An example is depicted in Figure 2, where each $T(\{l_i^+, l_i^-\})$ for $i = 1, ..., n$ is assigned a unique color. Note that the $T(\{l_i^+, l_i^-\})$ of a segment l_i intersects any other $T(\{l_j^+, l_j^-\})$ for $j \neq i, j = 1, ..., n$ in a constant number of intersections. Thus there are a quadratic number of intersection points. We call I the set of intersection points, where each element of I is an intersection point between two diagrams with distinct colors.

Notice the smallest possible width of L equals the vertical shortest distance between a point $p \in I$ and a point f_p on another diagram with distinct color, so that at least one color from each diagram is intersected by vertical line segment $\overline{pf_p}$. Since we want to minimize the length of segment $|\overline{pf_p}|$ among all directions θ, it will have one endpoint on an intersection point. Notice that the instances determined in this way will introduce a larger width among all other directions in $\mathcal{A}(\theta, d)$ diagrams of L. Thus this gives a valid width, and further the smallest possible width. But computing the smallest width by this method will cost $O(n^2)$ time.

2.1 Largest width

Let s be a set of $\mathcal{A}(\theta, d)$ diagrams of distinct colors. If s includes exactly one instance of each color, we call s a *complete set*. For a complete set s of diagrams, we define

[1] Our method solves the smallest width problem in $O(n^2)$ time, however, there exists an $O(n \log n)$ time algorithm for this problem [4].

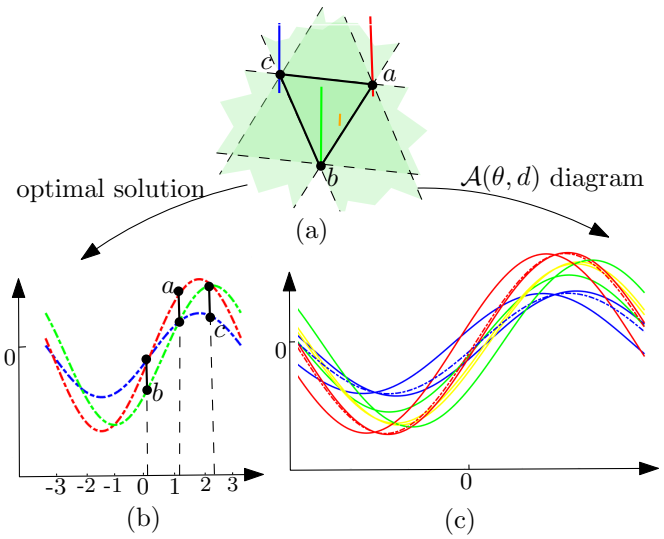


Figure 3: (a) An example. The maximal width is determined by a, b and c. (b) The optimal solution. (c) The $\mathcal{A}(\theta, d)$ diagrams of the endpoints is denoted by solid curves, and the optimal solution is shown in dashed-dotted. The green dashed-dotted is not visible because it is the same as a solid green curve. For better visibility this figure is shown wider.

w_s as the shortest vertical distance between the topmost and bottommost diagrams in s among all values of θ. Obviously w_s determines the width of instances in s.

Observation 1 *Let P be a set of n points from distinct line segments, then $w_{T(P)}$ determines the width of P.*

Proof. The correctness comes from the fact that $w_{T(P)}$ is computed among all values of θ. □

Let P^* denote the set of points which maximizes the width. In the $\mathcal{A}(\theta, d)$ diagram of L, the solution to the largest width problem is equivalent to a complete set $T(P^*)$ of $\mathcal{A}(\theta, d)$ diagrams, so that the value of $w_{T(P^*)}$ is as large as possible among all possible choices of s. In other words, for any other complete set s, $w_{T(P^*)} \geq w_s$. See Figure 3 as an example. As can be seen in Figure 3, the problem probably has algebraic issues. We design an approximation algorithm to solve it. We start stating our results with some observations.

Lemma 1 *Let L be a given set of parallel line segments in the plane, and let E denote the set of all endpoints of L. There exists a solution to the largest width problem so that one of the elements of E is involved in the optimal solution.*

Proof. It is easy to observe that we can translate the set of points realizing the largest width in up or down direction until one of the points determining the strip of width (indeed, the point which has the smallest distance to one of the endpoints of its segment) reaches to the closest endpoint of its segment, or the width misses covering another segment l, which this happens at an endpoint of l again, then the new set realizes the same width, but one element of E is involved in the optimal solution. □

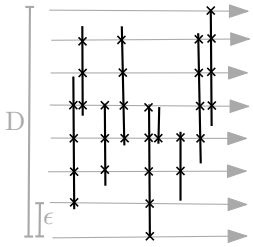


Figure 4: The crosses denote the points approximating the line segments.

Lemma 2 *There exists an ϵ-kernel of size $O(\sqrt{\epsilon^{-1}})$ for the maximum width problem.*

Proof. Agarwal *et al.* [1] proved that for any point set in d-dimensional space, there is an ϵ-kernel of size $O(1/\epsilon^{d-1/2})$ and it is also worst case optimal. Let $opt(L)$ denote the optimal solution to the maximum width problem of L. Then there exists an ϵ-kernel Q_{opt} for $opt(L)$, that is $opt(L) < (1+\epsilon)widt(Q_{opt})$. Let $S(Q_{opt})$ denote the set of segments which share a point on Q_{opt}. Then obviously $opt(S(Q_{opt})) \leq opt(L)$. Also we have $width(Q_{opt}) \leq opt(S(Q_{opt}))$. Then $opt(S(Q_{opt})) \leq opt(L) \leq (1+\epsilon)opt(S(Q_{opt}))$. □

Although we do not know what is our ϵ-kernel, we still can use its size to design a more efficient algorithm.

Let D denote the vertical distance between the highest and smallest y-coordinates of any two endpoints of segments of L, as illustrated in Figure 4. Then we emanate a set ρ of horizontal parallel rays, where the vertical distance between any two consecutive rays is ϵ. For any $l_i \in L$, the intersection points $\rho \cap l_i$ approximate l_i. We will compute the $\mathcal{A}(\theta, d)$ diagrams of $V = \rho \cap L$. Obviously $V \in O(n\epsilon^{-1})$. We postpone the discussions of why this gives us the desired $(1-\epsilon)$ factor, and first discuss the solution on the approximated points.[2]

From Lemma 2 and considering any triple of points which are potentially involved in the maximum width, a naive approach solves the problem in $O(n^{\epsilon^{-1/2}}(\epsilon^{-1/2}\epsilon^{-1})^3)$ time.

Corollary 3 *There exists a PTAS for the maximum width problem which runs in $O(n^{\epsilon^{-1/2}}\epsilon^{-4.5})$ time.*

Observation 2 *If for each $l_i \in L$, $T(\{l_i^+, l_i^-\})$ is always entirely located between two other $\mathcal{A}(\theta, d)$ diagrams in $T(E)$ among all values of θ (shown in yellow in Figure 3(c)), then l_i does not have a role in the constitution of the optimal solution.*

[2]We have supposed $D = 1$, since we are considering the relative error.

The above observation does not necessarily reduce the complexity of the algorithm, but still can reduce the total running time.

2.2 Dynamic programming algorithm

As a consequence of Observation 1 and since we look for the shortest vertical distance between a set of diagrams, the shortest vertical distance will at least use one intersection point of two diagrams with distinct colors. Suppose we have fixed one endpoint b of a segment, and we have computed the $\mathcal{A}(\theta, d)$ diagram of b, $T(\{b\})$ (for simplicity we denote it by $T(b)$). Now the question is how to find a complete set s of diagrams, so that $T(b) \in s$ and other elements of s maximize their vertical distances (w_s) from $T(b)$.

First notice that the number of points in V is in $O(n\epsilon^{-1})$. By considering any triple of points in V which are potentially involved in the optimal solution, obviously the problem can be solved in $O(n^4\epsilon^{-3})$. In the extra $O(n)$ we check whether the strip of triple includes one instance from each segment or not. We will design a DP algorithm which runs in $O(n^2\epsilon^{-4.5})$ time.

For a fix endpoint b, let $w(b)$ denote the length of the shortest vertical segment which is intersected by all the transformations of $\sqrt{\epsilon^{-1}} - 1$ other instances (in the ϵ-kernel) among all directions θ. Let w^* denote the maximum possible width of L. We should maximize the value $w(b)$ for each b. Obviously $w(b)$ in a direction θ can be defined by

$$w(b) = Max_{\forall p_i, d_\theta(T(p_i)) \leq d_\theta(T(b))} \, |d_\theta(T(b)) - d_\theta(T(p_i))| +$$

$$Max_{\forall p_j, d_\theta(T(p_j)) \geq d_\theta(T(b))} \, |d_\theta(T(b)) - d_\theta(T(p_j))|,$$

with $i \neq j$, in which $T(p_i)$ and $T(p_j)$ has the smallest and largest vertical distances from $T(b)$ in direction θ.

$$W(b) = Max\,\big[w(b)\big],\ w^* = Max_{b \in E}\, W(b),$$

where $W(b)$ is the maximum over all possible ϵ-kernels on b. There only exist $O(\sqrt{\epsilon^{-1}})$ candidates for each of p_i and p_j, since they belong to an ϵ-kernel of this size. Also we only need to consider the directions θ which is determined by the intersection points of the $\mathcal{A}(\theta, d)$ diagrams of elements in the ϵ-kernel, since the minimum value of $w(b)$ that needs to be maximized happens there. Thus there exist $O((\epsilon^{-1/2}\epsilon^{-1})^2)$ directions θ in total. Also there exist $O(n\epsilon^{-1/2})$ different ϵ-kernels (of size $O(\epsilon^{-1/2})$) to be defined on b. Consequently the dynamic program runs in $(n^2\epsilon^{-4.5})$ time.

Theorem 4 *Let L be a given set of parallel line segments in the plane. The largest possible width of L can be approximated within factor $(1 - 2\epsilon)$ in $O(n^2\epsilon^{-4.5})$ time.*

3 Bounding box

Similarly, for computing the bounding box of L two problems can be considered, neither of these has been studied yet: the smallest area bounding box and the largest area bounding box. Let B^* denote the optimal solution to any of these problems.

3.1 Smallest bounding box

Now we extend our approach for the minimum width problem to design an algorithm for the smallest-area bounding box.

Lemma 5 *Let L be a set of parallel line segments, and let E be set of the endpoints of segments in L. There exists an optimal solution B to the smallest bounding box of L, where each edge of B passes through at least one point of E, and these points belong to distinct segments.*

Proof. Suppose the lemma is false. Then the minimal bounding box B still has an edge e which is determined by a point p_i somewhere on the middle of l_i. Consider a line ℓ through p_i and parallel to e. If we sweep ℓ toward the opposite side of e on B, it will intersect l_i, until it leaves it at an endpoint p'_i (or B misses covering another segment l_j, which happens at an endpoint p_j). Then p'_i (or p_j) can be substituted for p_i to give us a smaller area bounding box. Contradiction. □

Now we have discretized the problem on the endpoints. For any set of fixed points, the smallest bounding box can be determined by five points. Consequently, there exists a naive $O(n^6)$ time algorithm for the smallest bounding box problem, where in the extra $O(n)$ time we should check whether an instance of any segment is included in the solution.[3]

Lemma 6 *Let L be a set of n parallel line segments. Only the directions determined by the intersection points of the elements of $T(E)$ in the $\mathcal{A}(\theta, d)$ diagram of L can be candidates to determine the direction of two parallel edges of B^*.*

Proof. Only the intersection points of $T(E)$ in the $\mathcal{A}(\theta, d)$ diagram of L denote the directions in which a minimum width may exist. Suppose the lemma is false. Then there exists a solution B to the minimum bounding box problem, so that none of the two directions determined by edges of B, are determined by a direction in which a minimal width happens (in an intersection point). Then we find the closest direction $\theta_{l'}$ (to the directions of any of two edges of B) which is a candidate for the smallest width (which happens at an intersection point) (see Figure 5). Let θ_l denote the direction of two parallel edges of B which is closer to $\theta_{l'}$.

[3]Notice that a rotating caliper technique does not look applicable here, since we do not exactly know the convex hull.

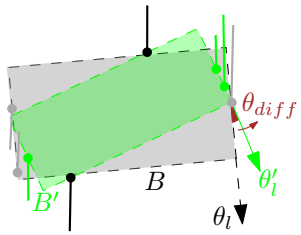


Figure 5: The smallest-area bounding box will be constructed in a direction in which a minimal width exists; if not we still can reduce its area.

Also first suppose the segments determined by direction $\theta_{l'}$ are distinct from the segments involved in the other edges of B'. We define $\theta_{diff} = |\theta_{l'} - \theta_l|$. We substitute the determined width in direction $\theta_{l'}$ for the one in direction θ_l. We then rotate the two other edges of B with θ_{diff} through the previously determined points. Obviously the achieved box B' has a smaller area than B. Contradiction.

Now suppose the determined segments by the width in direction $\theta_{l'}$ are not distinct from the segments involved in the other edges of B'. Then the shared point would be located at a corner of the new box B', while the area of the B' is smaller than B. Contradiction. □

In each intersection point $p \in I$ in direction θ, there may exist a strip with minimal size which includes at least one point from each line segment. The other sides of the potential optimal solution can be determined in direction $\theta + \pi/2$. Since $|I| \in O(n^2)$, the minimum area determined box among the intersection points realizes B^*, and the algorithm works in $O(n^2)$ time.

Theorem 7 *Let L be a set of n parallel line segments. The optimal solution to the smallest bounding box problem of L can be computed in $O(n^2)$ time.*

3.2 Largest bounding box

This problem looks difficult. Even a brute-force algorithm is not straightforward, since we cannot simply expand the edges of a possible box (by using the endpoints of the segments), since collinearity of the points may reduce the size of the optimal box. See Figure 6(a). Let θ^* denote the direction of the largest bounding box. Also notice that at least six points are involved in the optimal solution, since the determined widths in both direction θ^* and $\theta^* + \pi/2$ have the smallest size among all possible directions in which a width can be determined, if not we still can reduce its size, and it is not a valid bounding box. Also as can be seen in Figure 1(d), the largest area bounding box does not necessarily use the orientation of the maximum width.

Lemma 8 *Let L be a set of n parallel line segments, and let E be the set of the endpoints of segments in L. There exists a solution to the largest-area bounding box that uses two points (from distinct segments) of E on its two opposite sides, so that each edge includes one of them.*

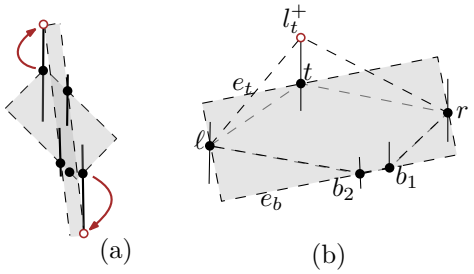


Figure 6: (a) Expanding the edges of a box B to the endpoints of the segments determining the edges of B does not necessarily increase the area of B. (b) The largest-area bounding box at least uses two elements of E on its two parallel sides.

From the $\mathcal{A}(\theta, d)$ diagram of L, the number of different configurations of having two distinct endpoints on two opposite sides of a rectangle is bounded by $O(n^2)$, since in the intersection points the vertical order of two $\mathcal{A}(\theta, d)$ diagrams changes, and there only exists a quadratic number of intersection points. Let t and b denote such endpoints. Also an instance of any other line segment is included in the determined strip by these two endpoints. Notice that four subproblems need to be considered, since we do not know which of the upper or lower endpoints of l_t and l_b are the right ones. With the same argument we had in Lemma 2, there exists an ϵ-kernel of size $O(\epsilon^{-1/2})$ for the largest bounding box. By approximating the set of line segments with a set of parallel rays with ϵ difference between consecutive rays, as discusses in Section 2.1, for any pair of endpoints we need to find a triple of other points to construct a bounding box, where there are $O(\epsilon^{-1/2})$ candidates for each point of triple and $O(\epsilon^{-3})$ possible directions for the optimal box. Thus a DP similar to the one presented in Section 2.2 can solve the problem in $O(n^2\epsilon^{-5})$ time. In the following we try to solve it exactly.

Corollary 9 *Let L be a set of n parallel line segments. There exists an FPTAS for the largest bounding box of L that runs in $O(n^2\epsilon^{-5})$ time.*

Algorithm. Let L be a given set of n parallel line segments. Recall that from $\mathcal{A}(\theta, d)$ diagrams of L we can compute all possible directions which there is a strip $S = d(L, \theta)$, such that S includes at least one instance from each element of L in direction θ. From Lemma 8 we know two distinct segments determine two opposite sides of B^*. As said before, from the $\mathcal{A}(\theta, d)$ diagram of L we understand at most $O(n^2)$ candidates can determine two opposite sides of B^*, since there are $O(n^2)$ intersection points and thus the number of configurations in which all other diagrams are resides between two different diagrams is bounded by $O(n^2)$. Then we should find a valid width with these two points. Let θ denote a direction of such valid width. Notice that there may exist $O(n)$ possible directions for θ. We will consider computing a valid solution from a specific θ, and of course we will repeat it for the remaining possible directions. Then a bounding box B in direction θ can be defined by $B = d(L, \theta)$. $d(L, \theta + \frac{\pi}{2})$ (2), where B is the smallest box which bounds S in $\theta + \frac{\pi}{2}$ direction, so that B is a rectangle. With a bit abusing of the

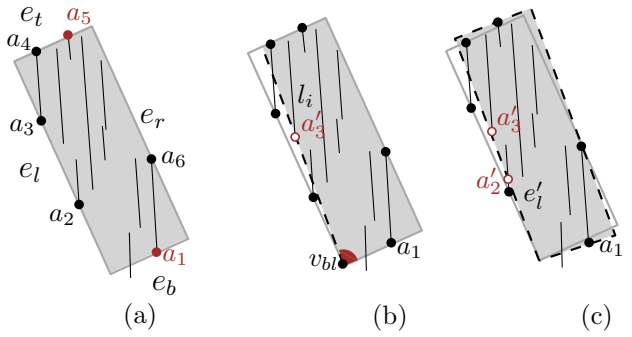


Figure 7: (a) A non-valid bounding box, where it is determined by a_1 and a_5. (b) The maximum inner angle at v_{bl} is determined at a'_3. (c) The corrected edge e_l is denoted by e'_l. $a_1a'_2a'_3a_5a_6$ denote the computed valid box after one step of the DP, which is not completely valid yet. Notice that the hidden set of e_r need to be updated for the next step.

notation, let S and B also denote size of the width and area of the bounding box, respectively.

The above definition of a bounding box B does not necessarily give us a valid box, since some segments may share more than one vertex on the boundary of B, or B may not be the smallest possible box with these instances, so that we still can reduce its area. But correcting this should be done in such away that the removed area from B is minimized. This procedure is called *correcting* B. Also we should do such correction for all possible sub-problems in which a non-valid bounding box is determined by any pair of endpoints. Finally, the largest-area bounding box among all determines the largest-area bounding box of L. Obviously correcting a bounding box B (first computed in direction θ) may also change the direction of B. In the following we show that we still can compute the exact possible rotation of B.

When looking for the largest smallest possible box B, we consider all sets of six points which may define a bounding box in Equation(2), where two of them already determine a valid width and also the first direction of box B, but not necessarily a valid bounding box in that direction. And the other four points are computed accordingly. In other words, for two fixed points, we first determine a valid width through them in a direction θ, and then we compute a valid width in direction $\theta+\pi/2$. Notice that the computed box is a superset for the optimal solution with these instances. Finally in the DP algorithm we try to make the biggest valid box which is determined by these instances. See Figure 7(a) for an example. Let A denote a set consisting of six such points on the boundary of box B. Let e_b, e_l, e_t and e_r denote the edges of B in clockwise direction, and let a_i for $i = 1, ..., 6$ denote such a set A. Also let a_1 be located on edge e_b, a_2, a_3 be located on edge e_l, a_4, a_5 be located on edge e_t, etc. Also let v_{bl} denote the common endpoint of edges e_b and e_l, as can be seen in Figure 7. W.l.o.g suppose B is defined on two endpoints a_1 and a_5.

For each non-distinct element of A, e.g., a_3, a distinct line segment l_i (which is already intersected by B) will share a vertex a'_3 on e_l, so that a'_3 can be substituted for a_3 to give us a smaller (but a bit more valid) bounding box B'. We call a'_3 the *hidden* line segment by e_l. Let $H(A)$ denote all the line segments hidden by the elements of A. In the worst case, correction of B needs to find five hidden vertices by the elements of A, but such substitution should be done in such a way that removed area from B is as small as possible, and the resulting valid B has the largest possible area. (Notice that since $H(A)$ has constant complexity, computing the best configuration can be done in constant time.) Also a hidden line segment l_i might simultaneously be hidden by the points on two edges of B, e.g., in Figure 7, l_i is hidden by both e_l and e_t. We will check both cases in different sub-problems, of which there are constantly many. First suppose l_i should share a vertex on e_l. To do so, we should select a'_3 such that the inner angle $a_1v_{bl}a'_3$ has the maximum possible value among all possible choices for a'_3. Further the hidden elements $H(A)$ determine the exact value of the possible rotation of B.

Let Θ denote all possible directions in which a valid possible box B is determined by two points a_1 and a_5, as discussed before, and let $\alpha(A)$ denote the maximum angle of rotation for correcting elements of A. Let A' denote set A, where one distinct element is substituted for one non-distinct element of A. Then we can write our DP as follows:

$$d(L, \theta) = width\ of\ L\ in\ direction\ of\ \theta$$

$$b(L, \theta) = d(L, \theta).d(L, \theta + \frac{\pi}{2}), b(L, \Theta) = Min_{\,\forall\theta\in\Theta}\ b(L, \theta)$$

Then we have:

$$V(A) = Max\ (Min\ b(A \cup A', [\alpha(A), \alpha(A')]), V(A'))$$

where $V(A)$ denotes the largest bounding box on a possible set A in direction θ. Then the maximum value among all possible $V(A)$ denotes the optimal solution.

The correctness of the algorithm comes from the fact that we consider all possible pairs that can define a bounding box, and then we find the largest possible bounding box on this pair by our dynamic program. Finally, the largest-area corrected box determines the optimal solution. Notice that we may need to rework on a corrected set A', since several elements may need to be corrected. We will correct them clockwise. They only increase a constant number of sub-problems, which at most equals 4×5. Notice that there are constant possible directions in Θ to make a valid box, and it is needed to consider $O(n^2)$ pairs, and for each pair we need to consider $O(n)$ directions, and for each direction θ we should find the width in direction $\theta + \pi/2$. Then we compute the hidden elements of the edges in $O(n)$ time, and we repeat it for any pairs between any two intersection points in $\mathcal{A}(\theta, d)$ of L. Thus the algorithm runs in $O(n^6)$ time and space.

Theorem 10 *Let L be a given set of parallel line segments. The largest bounding box of L can be computed in $O(n^6)$ time and space.*

4 Concluding remarks and open questions

In this paper we present several algorithms for computing an instance P on a set of line segments, so that P maximizes/minimizes the width or the area of the bounding box of P. Solving maximum width problem on a set of squares remained open. We wish to extend our presented algorithms to solve these problems on a set of squares.

References

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

[2] M. Löffler and M. van Kreveld. Largest bounding box, smallest diameter, and related problems on imprecise points. *Computational Geometry*, 43(4):419 – 433, 2010.

[3] M. Löffler. *Data imprecision in computational geometry.* PhD thesis, Utrecht Univesity, 2009.

[4] J.-M. Robert and G. Toussaint. Computational geometry and facility location. In *Proc. International Conference on Operations Research and Management Science*, pages B1–B19, 1990.

[5] D. Salesin, J. Stolfi, and L. Guibas. Epsilon geometry: Building robust algorithms from imprecise computations. In *Proc. 5th Annual Symposium on Computational Geometry*, pages 208–217, 1989.

[6] G. T. Toussaint. Solving geometric problems with the rotating calipers. In *IEEE Melecon*, volume 83, page A10, 1983.

A Appendix

A.1 Omitted proofs

Theorem 4 *Let L be a given set of parallel line segments in the plane. The largest possible width of L can be approximated within factor $(1-2\epsilon)$ in $O(n^2\epsilon^{-4.5})$ time.*

Proof. The only remaining unproved part is the $(1-2\epsilon)$ ratio of approximation. Let $T(P_{app})$ denote a complete set of diagrams which maximizes size of width. Let θ_{app} denote the direction in which $T(P_{app})$ gives the optimal solution. Then on the $\mathcal{A}(\theta, d)$ diagrams, using Equation (1) we have $|w^*| \leq w_{T(P_{app})} + 2\epsilon \cos\theta_{app}$, since $w_{T(P_{app})}$ has the largest value among other complete sets. Obviously $|w^*| \geq \sin\theta_{app}$. But then if $\theta_{app} \geq \pi/4$, $\sin\theta_{app} \geq cos\theta_{app}$ and then, $|w^*|(1-2\epsilon) \leq w_{T(P_{app})}$. In the case where $\theta_{app} < \pi/4$ we obviously have the same width in direction $\theta_{app} + \pi$. The lemma follows. □

Lemma 8 *Let L be a set of n parallel line segments, and let E be the set of the endpoints of segments in L. There exists a solution to the largest-area bounding box that uses two points (from distinct segments) of E on its two opposite sides, so that each edge includes one of them.*

Proof. Like in the case of Lemma 1, there always exists an optimal solution B so that at least one element of E is involved on determining some edge of B. Let e_b denote such edge. In the following, we discuss the existence of another element of E on the opposite side of e_b. Let b_1 denote the endpoint which has determined the edge e_b. W.l.o.g suppose b_1 is located on the bottom side of B.

Suppose the lemma is false. Then there exists a maximal bounding box B for L which is passing through some points ℓ, t, r, b_1 and b_2, so that B has maximal area and B only uses one element of E. Let e_t and e_b denote the top and bottom edges of B. And let t and b_1, b_2, respectively, denote the points that e_t and e_b are passing through them, as illustrated in Figure 6(b). Consider a line l through t and parallel to e_t. If we sweep l away from e_t, it will intersect the segment l_t, so that it leaves it at an endpoint l_t^+. Then obviously the pentagon $\ell l_t^+ r b_1 b_2$ includes the pentagon $\ell t r b_1 b_2$. Since with a fixed length, the new bounding box should now include some new point which previously where located outside the bounding box, it must be expanded from the width. But then the changes of the area of four boxes should be considered, if we substitute l_t^+ for t, the box with an edge through b_1, b_2 will increase its size, and the same argument holds for the boxes with an edge through b_1, r and b_2, l. Thus the collinearity of l_t^+ with existing vertices cannot make a smaller area box. All together, we do not reduce the size of any other possible bounding box of L, and thus any bounding box which is passing through $\ell l_t^+ r b_1$ and b_2 will have a larger area than B. Thus B could not be the largest area bounding box. Contradiction. □

Open Problems from CCCG 2017

Joseph O'Rourke*

The following is a description of the problems presented on July 26th, 2017 at the open-problem session of the *29th Canad. Conf. Computational Geometry* held at Carleton University, in Ottawa.

Near-Delaunay Triangulations

Joseph O'Rourke
Smith College
jorourke@smith.edu

Let T be a triangulation of a finite point set in the plane. Say that a triangulation is *near-Delaunay* if the opposite angles α and β of each pair of triangles that share an edge sum to at most $\pi + \epsilon$, for $\epsilon > 0$. Note that, if $\epsilon = 0$, then T is Delaunay; see Figure 1. Near-Delaunay triangulations can be constructed by an edge-flipping algorithm.

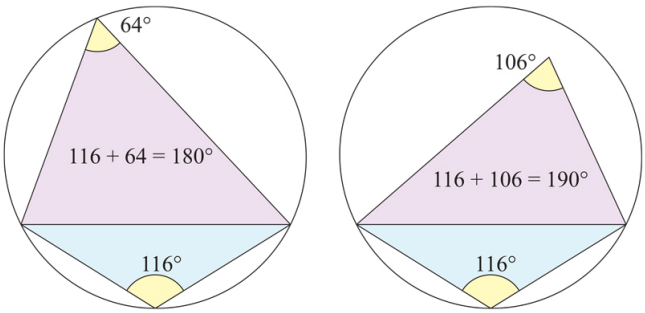


Figure 1: Left: Delaunay triangles. Right: Near-Delaunay triangles.

Have these triangulations been defined previously? Do they have any nice properties?

Update. Scott Mitchell suggested "measuring the signed distance between circumcenters of triangles sharing an edge; for Delaunay triangulations this is simply the dual edge length and non-negative, but for non-DT the circumcenters can be in the wrong order and hence have a negative distance between them. So one could look at the ratio of the dual edge signed-length to the primal edge length (for 2D triangulations) as a continuous measure of how close it is to non-Delaunay." This concept has appeared in the literature on Hodge-optimized triangulations, e.g., [MM+11].

References

[MM+11] P. Mullen, P. Memari, F. de Goes, and M. Desbrun. HOT: Hodge-optimized triangulations. *ACM Transactions on Graphics (TOG)*, 30(4):103, 2011.

Counting Closed Billiard Paths

Joseph O'Rourke
Smith College
jorourke@smith.edu

Let a collection of rectangles, all axis-aligned, all enclosed in one rectangle, have a total of n edges. A simple, closed billiard path is a path that is (a) closed, (b) non-self-intersecting, and so forms a simple polygon, (c) never touches a rectangle corner, and (d) all reflections are mirror reflections. Label all rectangle edges, and define the *signature* of a billiard path by the labels of the edges from which it reflects, reducing repeated edge reflections $(ab)^k$ to ab. Thus in Figure 2, the path $12373(56)^24$

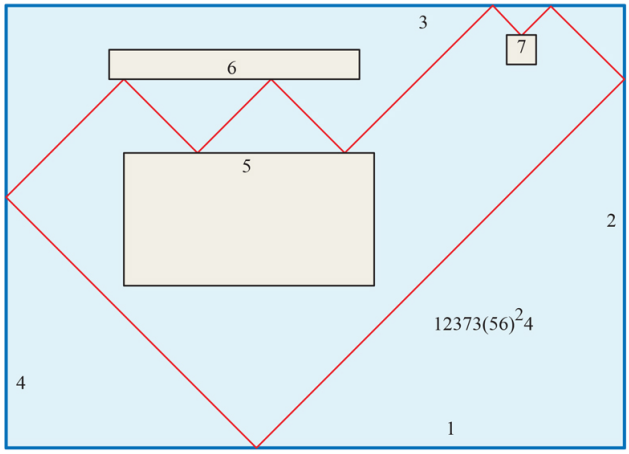


Figure 2: A billiard path of signature length 8.

has signature 12373564, reducing $(56)^2$ to 56.

For simple, closed billiard paths, for any arrangement of rectangles of a total of n edges:

1. What is the maximum length of such a signature?

*Department of Computer Science, Smith College, Northampton, MA 01063, USA. jorourke@smith.edu

2. What is the largest number of distinct signatures achievable for one fixed reflection angle (45° in the figure).
3. What is the largest number of distinct signatures achievable for paths at arbitrary reflection angles?

Tubes in Space
Joseph O'Rourke
Smith College
jorourke@smith.edu

Let S be a unit-radius sphere in $\mathbb{R}^3$. Place n lines intersecting S to minimize the maximum distance between any two points in S, where distance is measured as follows. Distance off the lines is Euclidean distance, but the distance between any two points on one line is zero. The lines are like very fast transportation tubes. See Figure 3. One line L ($n = 1$)

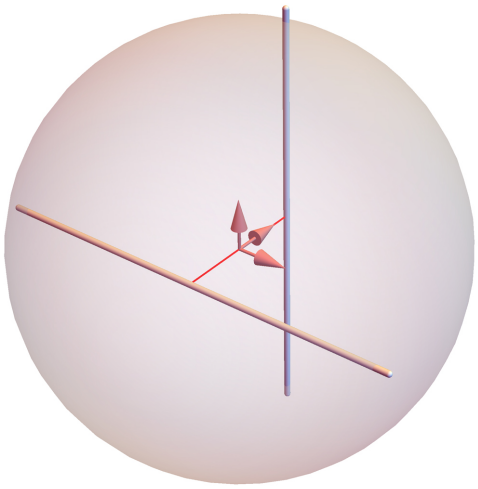


Figure 3: $n = 2$ skew transport tubes.

is useless for pairs of points on antipoldal on the equator formed by the plane perpendicular to L: their distance remains 2. It was observed at the presentation that two lines are also useless: antipodal points on a great circle orthogonal to the two lines are still 2 apart. Three x, y, z-axis lines seem best, apparently reducing the maximum distance to $2\sqrt{\frac{2}{3}} \approx 1.63$. In two dimensions, it seems that equi-angular lines through the center of a circle is the optimal arrangement. All these are conjectures.

Variations:

- Within a unit cube rather than a sphere.
- Assign off-tube speeds 1, and in-tube speeds $s > 1$.
- The same questions in $\mathbb{R}^d$.

General-position subconfigurations
David Eppstein
University of California, Irvine
eppstein@uci.edu

For the purposes of this problem, a set of points is in *general position* if no line contains three or more of its points. This problem's first two parts concern the d-dimensional point set $\{-1, 0, 1\}^d$ (a grid of size three in each dimension), shown in Figure 4 in projection to the plane.

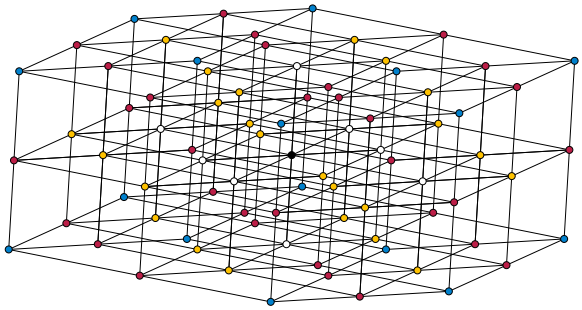


Figure 4: $\{-1, 0, 1\}^d$ for $d = 4$, from [E17].

(a) These points can be partitioned into $d + 1$ subsets, each in general position, by grouping points according to how many coordinates are zero. (The figure's colors show this partition.) The Hales–Jewett theorem [HJ63, S88] implies that any general-position partition has $\omega(1)$ subsets. (This holds even for the weaker condition that no three points form a monotonic line, one in which the three points can be ordered so that all coordinates are increasing or constant.) Can these points be partitioned into fewer than $d+1$ general-position subsets?

(b) The largest subset in this partition (for which the number of zero coordinates of each point is approximately $d/3$) has size $\Theta(3^d/\sqrt{d})$. The density version of the Hales–Jewett theorem implies that all general-position subsets have size $o(3^d)$ [FK89, F91]. Is there a general-position subset with size $\omega(3^d/\sqrt{d})$?

(c) How well can the largest general-position subset and the partition into the fewest general-position subsets be approximated? Is it achievable in polynomial time for arbitrary planar point sets? Both problems are NP-complete and APX-hard, and can be approximated within a factor of $O(\sqrt{n})$ by a simple greedy algorithm that adds each point (in arbitrary order) to the first subset in which it is in general position [E17]. Results of Füredi, Payne and Wood, relating

general-position subsets to lines with many points [F91, PW13], suggest that it may be possible to shave a logarithmic factor from this approximation ratio. But is $O(n^{1/2-\epsilon})$ possible, for some $\epsilon > 0$? (It is safe to consider only two dimensions, because points in higher dimensions can be projected to the plane without changing collinearity.)

References

[E17] D. Eppstein. *Forbidden Configurations in Discrete Geometry*. Manuscript.

[FK89] H. Furstenberg and Y. Katznelson. A density version of the Hales–Jewett theorem for $k = 3$. *Discrete Math.* 75:227–241, 1989.

[F91] Z. Füredi. Maximal independent subsets in Steiner systems and in planar sets. *SIAM J. DIscrete Math.* 4:196–199, 1991.

[HJ63] A.W. Hales and R.I. Jewett. Regularity and positional games. *Proc. Amer. Math. Soc.* 106:222–229, 1963.

[PW13] M.S. Payne and D.R. Wood. On the general position subset selection problem. *SIAM J. Discrete Math.* 27:1727–1733, 2013.

[S88] S. Shelah. Primitive recursive bounds for van der Waerden numbers. *J. Amer. Math. Soc.* 1:683–697, 1988.

Constructing separators for Geometric Graphs

Stefan Langerman
Université Libre de Bruxelles (ULB)
stefan.langerman@ulb.ac.be

Given a planar graph $G = (V, E)$, $|V| = n$, the *Planar Separator* theorem of Lipton and Tarjan [LT79] states that there always exists a set of $O(\sqrt{n})$ vertices in V whose removal partitions the graph G into disjoint connected subgraphs, each of size at most $2n/3$. Such a separator can be constructed in $O(n)$ time when the graph is provided.

There are many situations however when a graph is defined implicitly, by a collection of points or of geometric objects, such as for example, the Delaunay triangulation of a set of n points, the edge structure of the convex hull of n points in $\mathbb{R}^3$, or the intersection graph of a collection of disks in the plane where no point is covered by more than two disks. The explicit construction of, e.g., a Delaunay triangulation for n points in the plane requires $O(n \log n)$ time, however it might be possible to construct a separator without having to construct the graph explicitly.

Question 1: Given a set S of n points in $\mathbb{R}^2$, is it possible to find a separator of the Delaunay triangulation of P in time $O(n)$?

Question 2: Given a set S of n points in $\mathbb{R}^3$, is it possible to find a separator of the convex hull of P in time $O(n)$?

For some geometric graphs, e.g., the disk-intersection graph mentioned above, a separator can be found in $O(n)$ time [MTTV97].

References

[LT79] R.J. Lipton and R.E. Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189, 1979.

[MTTV97] G.L. Miller, S.-H. Teng, W. Thurston, and S.A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *J. ACM*, 44(1):1–29, January 1997.

[SW] W.D. Smith and N. Wormald. Geometric separator theorems and applications. https://www.math.uwaterloo.ca/~nwormald/papers/geomsep.ps.gz.

Optimizing Sum of Products

Bereg et. al. (posed by Lily Li)
Simon Fraser University
xyl9@sfu.ca

Given sequences $A = \langle a_0, a_1, ..., a_{n-1} \rangle$ and $B = \langle b_0, b_1, ..., b_{n-1} \rangle$ of real numbers, find a permutation π of A which maximizes

$$\sum_{i=0}^{n-1} a_{\pi(i)} a_{\pi(i+1)} b_i$$

where the indices are taken modulo n. If $b_i = 1$ for all i, then the solution is any cyclic shift of the sequence $\langle a'_0, a'_2, ..., a'_3, a'_1 \rangle$ where $a'_0 \geq a'_1 \geq a'_2 \geq a'_3 \geq \cdots$.

This problem is a modified version of an open problem presented in [BD$^+$16]. The paper showed that a variant of the problem allowing the permutation of both A and B can be solved optimally in $O(n \log n)$ time. It is not known if the posed problem is NP−Hard.

References

[BD$^+$16] S. Bereg, J.M. Díaz-Báñez, D. Flores-Peñaloza, S. Langerman, P. Pérez-Lantero, and J. Urrutia. Optimizing some constructions with bars: New geometric knapsack problems. *J. Combinatorial Optimization*, 31(3):1160–1173, 2016.

Compatible Triangulations of Labeled Point Sets
Debajyoti Mondal and Anna Lubiw
University of Waterloo, Canada
dmondal@uwaterloo.ca, alubiw@uwaterloo.ca

Let P_1, P_2 be a pair of point sets, each containing n points that are labeled from 1 to n. A pair of triangulations T_1 and T_2 of P_1 and P_2 are called *compatible triangulations* or *joint triangulations* if for every face, the clockwise cyclic order of vertices on the boundary is the same, e.g., see Fig. 5(a).

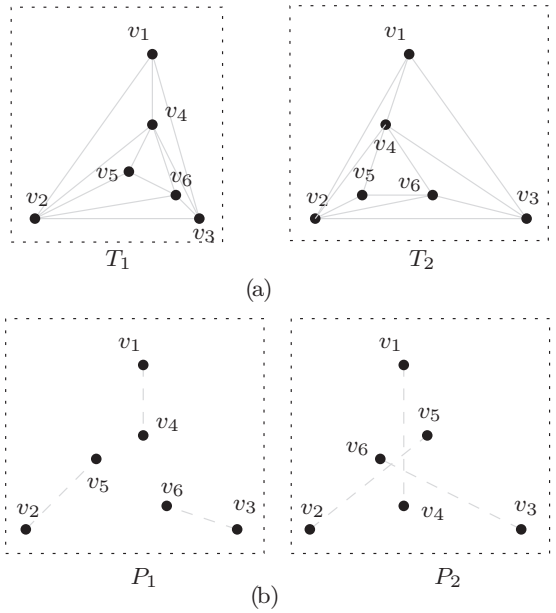


Figure 5: (a) A pair of point sets and their compatible triangulations. (b) A pair of point sets that do not admit compatible triangulations. Any triangulation of P_1 would contain the edges $(v_1, v_4), (v_2, v_5), (v_3, v_6)$, and they intersect in P_2.

Not all pairs of point sets admit compatible triangulations, even when they have the same number of points on the convex-hulls, e.g., see Fig 5(b). Saalfeld in 1987 [S87] proved that, using Steiner points, any pair of point sets with rectangular convex-hull can be triangulated compatibly. In fact, $O(n^2)$ Steiner points suffice for every point set [BSW97], and $\Omega(n^2)$ Steiner points are sometimes necessary [PSS96]. If we are allowed to choose the labels, then such compatible triangulations are conjectured to exist without Steiner points (when P_1 and P_2 have the same number of points on the convex hull) [AAHK03].

In this context we ask the following question: Does there exist an algorithm that, given a pair of labeled point sets, can decide in polynomial time whether they admit compatible triangulations without Steiner points?

The analogous question for compatible triangulations of polygons is solvable in polynomial time [ASS93]. The more general question of minimizing the number of Steiner points required for compatible triangulations of polygonal regions is NP-hard [LM17].

References

[AAHK03] O. Aichholzer, F. Aurenhammer, F. Hurtado, and H. Krasser. Towards compatible triangulations. *Theoretical Comput. Sc.*, Elsevier, 296(1):3–13,2003.

[ASS93] B. Aronov, R. Seidel, and D.L. Souvaine. On Compatible Triangulations of Simple Polygons. *Comput. Geom.*, 3:27–35, 1993.

[BSW97] M. Babikov, D.L. Souvaine, and R. Wenger. Constructing piecewise linear homeomorphisms of polygons with holes. *Proc. 9th Canad. Conf. Comput. Geom. (CCCG)*, 1997.

[LM17] A. Lubiw and D. Mondal. On Compatible Triangulations with a Minimum Number of Steiner Points. *Proc. 29th Canad. Conf. Comput. Geom. (CCCG)*, 101–106, 2017.

[PSS96] J. Pach, F. Shahrokhi, and M. Szegedy. Applications of the crossing number. *Algorithmica*, Springer, 16(1):111-117, 1996.

[S87] A. Saalfeld. Joint Triangulations and Triangulation Maps. *Proc. 3rd Symp. Comput. Geom. (SoCG)*, ACM, 195–204, 1987.

Binary trees in the $\{/, \backslash, \overline{\ }\}$-grid
Therese Biedl
University of Waterloo
biedl@uwaterloo.ca

The $\{/, \backslash, \overline{\ }\}$-grid consists of the points with integer coordinates, and all horizontal or diagonal lines through such points. Given a binary tree T, we want an embedding of T in the $\{/, \backslash, \overline{\ }\}$-grid, i.e., vertices are mapped to distinct grid-points, and edges are mapped to straight-line segments along the grid in such a way that no two edges cross. The *width* of such a drawing is the maximal x-coordinate (presuming that the minimal x-coordinate is 1). The main question is:

> How much width (relative to the number of vertices n) is sufficient to embed any binary tree in the $\{/, \backslash, \overline{\ }\}$-grid?

The question could also be asked for variations where we want an *upward* drawing (i.e., the tree is rooted and the y-coordinate of the parent of a

node v is no smaller than the y-coordinate of v) and/or an *order-preserving* drawing (i.e., the order of edges around each node is fixed and must be respected in the drawing).

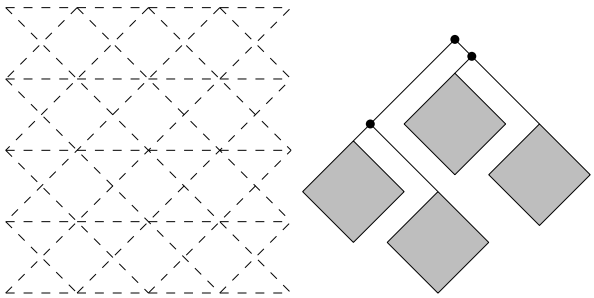


Figure 6: The $\{/, \backslash, \overline{\ }\}$-grid, and embedding a complete binary tree in it.

It is known that for the complete binary tree we need width $\Omega(\sqrt{n/\log n})$ [B17]. In the same paper, it was also argued that width $O(\sqrt{n})$ can be achieved for the complete binary tree, by taking an orthogonal construction due to Creszenci et al. [CDP92] and rotating it by 45°. Can we achieve width $O(\sqrt{n})$ for all binary trees?

References

[B17] T. Biedl. Upward Order-Preserving 8-Grid-Drawings of Binary Trees *Proc. 29th Canad. Conf. Comput. Geom. (CCCG)*, 232–237, 2017.

[CDP92] P. Crescenzi, G. Di Battista, and A. Piperno. A note on optimal area algorithms for upward drawings of binary trees. *Comput. Geom.*, 2(4):187–200, 1992.

Gabriel Circle Range Counting

Rasoul Shahsavarifar
University New Brunswick
Ra.Shahsavari@unb.ca

Suppose that $S = \{x_1, x_2, ..., x_n\} \subseteq \mathbb{R}^2$ and $q \in \mathbb{R}^2$. Can we answer the following question with $O(n)$ storage, $O(n \log n)$ expected preprocessing time, and sub-$O(n^{1/2+\epsilon})$ (optimally $O(\log n)$) query time?
Question: How many x_i fall inside the Gabriel circle $GC(q, x_k)$ for some $1 \leq k \leq n$? See Figure 7.

The Gabriel circle $GC(a, b)$ is the circle with diameter ab. The characteristic of having a common point q among all Gabriel circles may help to answer the question. A query time $O(n^{1/2+\epsilon})$ is achieved in [AMM13] for general circle range counting.

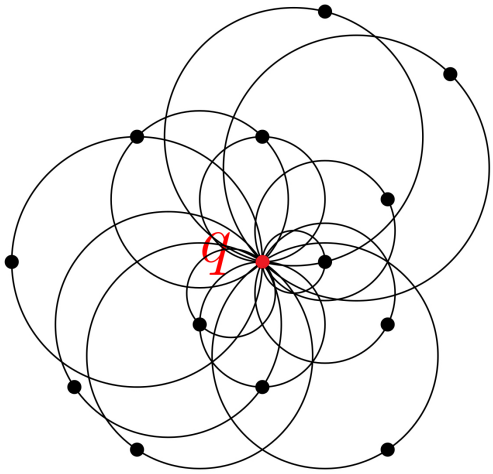


Figure 7: Point q is shared among all Gabriel circles $GC(q, x_k)$.

References

[AMM13] P.K. Agarwal, J. Matousek, and M. Sharir. On range searching with semi-algebraic sets. II. *SIAM J. Computing*, V.42, No. 6, 2039–2062, 2013.

Hamiltonian Order-k Delaunay Graphs

Prosenjit Bose
Carleton University
jit@scs.carleton.ca

Given a set P of n points in the plane, a pair of points $x, y \in P$ has order k if there exists a disk with x and y on its boundary containing at most k points of P. The edges of a standard Delaunay triangulation have order 0. A graph whose edges consist of every pair of points with order at most k will be referred to as the order-k Delaunay graph. The order-k Gabriel graph, which is a subgraph of the order-k Delaunay graph, is the graph whose edges consist of every pair of points whose Gabriel disk has contains at most k points.

Dillencourt [Dil87] showed that there exist point sets whose Delaunay triangulation is not Hamiltonian. Dillencourt [Dil90] also showed that the Delaunay triangulation is 1-tough, which when n is even implies that it contains a perfect matching. Abellanas et al. [ABG+09] showed that the order-15 Gabriel graph is Hamiltonian. Subsequently, Kaiser et al. [KSC15] showed that the order-10 Gabriel graph is Hamiltonian.

Conjecture: The order-1 Delaunay graph is Hamiltonian.

References

[ABG+09] M. Abellanas, P. Bose, J. García-López, F. Hurtado, C.M. Nicolás, and P. Ramos. On structural and graph theoretic properties of higher order Delaunay graphs. *Int. J. Comput. Geom. Appl.*, 19(6):595–615, 2009.

[Dil87] M.B. Dillencourt. A non-Hamiltonian, nondegenerate Delaunay triangulation. *Inf. Process. Lett.*, 25(3):149–151, 1987.

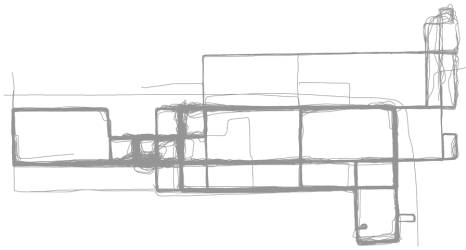
[Dil90] M.B. Dillencourt. Toughness and Delaunay triangulations. *Discrete & Comput. Geom.*, 5:575–601, 1990.

[KSC15] T. Kaiser, M. Saumell, and N. Van Cleemput. 10-Gabriel graphs are Hamiltonian. *Inf. Process. Lett.*, 115(11):877–881, 2015.

On Map Construction, Map Comparison, and Trajectory Clustering

Carola Wenk*

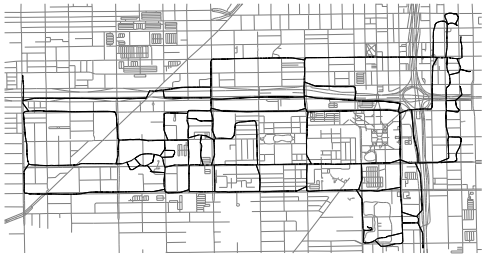
Geo-referenced trajectory data is collected in a wide range of applications, such as for a variety of location-based services on street maps, hiking trail logging, and the study of social behavior in animals. There has been a recent surge of algorithms for aggregating trajectory data, in particular by constructing road maps, e.g., [1–3, 8–36]. Road map construction is a type of geometric reconstruction problem in which the task is to extract the underlying geometric graph structure described by a set of movement-constrained trajectories, or in other words reconstruct a geometric domain that has been sampled with continuous curves that are subject to noise. See Figures 1 and 2 for an example trajectory data set and different constructed road maps.



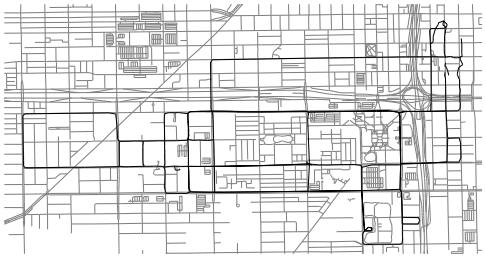
(a) Trajectories

Figure 1: Chicago dataset of shuttle bus trajectories [9, 10]. The data and figure are available on `mapconstruction.org`.

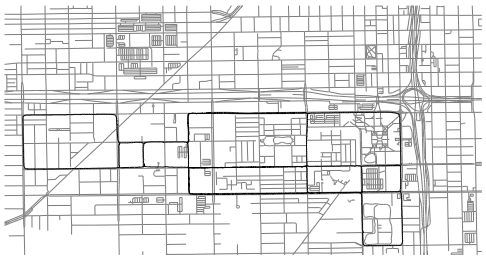
A related problem is *map comparison*, which can be used to assess the quality of map construction algorithms or two compare road maps from different sources. Different approaches for map comparison include signatures and distance measures, e.g., [4, 5, 9]. Comparisons of map construction algorithms, including experimental quality assessments, can be found in [6, 7, 9].



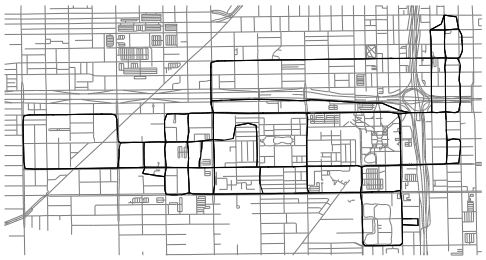
(a) Reconstruction by Ahmed et al. [8]



(b) Reconstruction by Biagioni et al. [9]



(c) Reconstruction by Davies et al. [17]



(d) Reconstruction by Karagiorgou et al. [27]

Figure 2: Constructed maps (in black) overlayed on ground-truth map (in gray) for the Chicago dataset of shuttle bus trajectories [9, 10]. The data and figures are available on `mapconstruction.org`.

*Department of Computer Science, Tulane University, `cwenk@tulane.edu`. CW acknowledges the generous support of the National Science Foundation under grants CCF-1618469 and CCF-1637576.

References

[1] M. Aanjaneya, F. Chazal, D. Chen, M. Glisse, L. J. Guibas, and D. Morozov. Metric graph reconstruction from noisy data. In *Proc. 27th ACM Symp. on Comp. Geometry*, pages 37–46, 2011.

[2] G. Agamennoni, J. I. Nieto, and E. M. Nebot. Robust inference of principal road paths for intelligent transportation systems. *IEEE Trans. on Intelligent Transportation Systems*, 12(1):298–308, Mar. 2011.

[3] M. Ahmed, B. T. Fasy, M. Gibson, and C. Wenk. Choosing thresholds for density-based map construction algorithms. In *Proceedings of the 23rd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2015.

[4] M. Ahmed, B. T. Fasy, K. S. Hickmann, and C. Wenk. A path-based distance for street map comparison. *ACM Trans. Spatial Algorithms Syst.*, 1(1):3:1–3:28, 2015.

[5] M. Ahmed, B. T. Fasy, and C. Wenk. Local persistent homology based distance between maps. In *Proceedings of the 22Nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 43–52, 2014.

[6] M. Ahmed, S. Karagiorgou, D. Pfoser, and C. Wenk. A comparison and evaluation of map construction algorithms using vehicle tracking data. *GeoInformatica*, 19(3):601–632, 2015.

[7] M. Ahmed, S. Karagiorgou, D. Pfoser, and C. Wenk. *Map Construction Algorithms*. Springer, 2015.

[8] M. Ahmed and C. Wenk. Constructing street networks from GPS trajectories. In *Proc. 20th Ann. European Symp. on Algorithms*, pages 60–71, 2012.

[9] J. Biagioni and J. Eriksson. Inferring road maps from global positioning system traces: Survey and comparative evaluation. *Transportation Research Record: Journal of the Transportation Research Board*, 2291:61–71, 2012.

[10] J. Biagioni and J. Eriksson. Map inference in the face of noise and disparity. In *Proc. 20th ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems*, pages 79–88, 2012.

[11] R. Bruntrup, S. Edelkamp, S. Jabbar, and B. Scholz. Incremental map generation with GPS traces. In *Proc. IEEE Intelligent Transportation Systems*, pages 574 – 579, 2005.

[12] K. Buchin, M. Buchin, D. Duran, B. Fasy, R. Jacobs, V. Sacristán, R. Silveira, F. Staals, and C. Wenk. Clustering trajectories for constructing maps. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, page 10 pages, 2017.

[13] L. Cao and J. Krumm. From GPS traces to a routable road map. In *Proc. 17th ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems*, pages 3–12, 2009.

[14] F. Chazal and J. Sun. Gromov-Hausdorff approximation of metric spaces with linear structure. *CoRR*, abs/1305.1172, 2013.

[15] C. Chen and Y. Cheng. Roads digital map generation with multi-track GPS data. In *Proc. Workshops on Education Technology and Training, and on Geoscience and Remote Sensing*, pages 508–511. IEEE, 2008.

[16] D. Chen, L. J. Guibas, J. E. Hershberger, and J. Sun. Road network reconstruction for organizing paths. In *Proc. 21st ACM-SIAM Symp. on Discrete Algorithms*, pages 1309–1320, 2010.

[17] J. J. Davies, A. R. Beresford, and A. Hopper. Scalable, distributed, real-time map generation. *IEEE Pervasive Computing*, 5(4):47–54, Oct. 2006.

[18] T. K. Dey, J. Wang, and Y. Wang. Improved road network reconstruction using discrete morse theory. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 58:1–58:4, 2017.

[19] T. K. Dey, J. Wang, and Y. Wang. Graph reconstruction by discrete morse theory. In *Symposium on Computational Geometry*, pages 31:1–31:15, 2018.

[20] D. Duran, V. Sacristán, and R. Silveira. Map construction algorithms: an evaluation through hiking data. In *Proc. 5th Int. Workshop on Mobile Geographic Information Systems*, pages 74–83, 2016.

[21] S. Edelkamp and S. Schrödl. Route planning and map inference with global positioning traces. In *Computer Science in Perspective*, pages 128–151. Springer, 2003.

[22] B. T. Fasy, R. Komendarczyk, S. Majhi, and C. Wenk. Topological and geometric reconstruction of geodesic subspaces of the euclidean space. Manuscript, 2018.

[23] A. Fathi and J. Krumm. Detecting road intersections from GPS traces. In *Proc. 6th Int. Conf. on Geographic Information Science*, pages 56–69, 2010.

[24] X. Ge, I. Safa, M. Belkin, and Y. Wang. Data skeletonization via Reeb graphs. In *Proc. 25th Ann. Conf. on Neural Information Processing Systems*, pages 837–845, 2011.

[25] T. Guo, K. Iwamura, and M. Koga. Towards high accuracy road maps generation from massive GPS traces data. In *Proc. IEEE Int. Geoscience and Remote Sensing Symp.*, pages 667–670, 2007.

[26] S. Jang, T. Kim, and E. Lee. Map generation system with lightweight GPS trace data. In *Proc. 12th Int. Conf. on Advanced Communication Technology*, pages 1489–1493, 2010.

[27] S. Karagiorgou and D. Pfoser. On vehicle tracking data-based road network generation. In *Proc. 20th ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems*, pages 89–98, 2012.

[28] S. Karagiorgou, D. Pfoser, and D. Skoutas. Segmentation-based road network construction. In *Proc. 21st ACM SIGSPATIAL Int. Conf. on Advances in Geographic Information Systems*, pages 450–453, 2013.

[29] F. Lecci, A. Rinaldo, and L. A. Wasserman. Statistical analysis of metric graph reconstruction. *Journal of Machine Learning Research*, 15(1):3425–3446, 2014.

[30] X. Liu, J. Biagioni, J. Eriksson, Y. Wang, G. Forman, and Y. Zhu. Mining large-scale, sparse GPS traces for map inference: comparison of approaches. In *Proc. 18th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 669–677, 2012.

[31] S. Schroedl, K. Wagstaff, S. Rogers, P. Langley, and C. Wilson. Mining GPS traces for map refinement. *Data Mining and Knowledge Discovery*, 9:59–87, 2004.

[32] W. Shi, S. Shen, and Y. Liu. Automatic generation of road network map from massive GPS vehicle trajectories. In *Proc. 12th Int. IEEE Conf. on Intelligent Transportation Systems*, pages 48–53, 2009.

[33] A. Steiner and A. Leonhardt. Map generation algorithm using low frequency vehicle position data. In *Proc. 90th Ann. Meeting of the Transportation Research Board*, pages 1–17, January 2011.

[34] S. Wang, Y. Wang, and Y. Li. Efficient map reconstruction and augmentation via topological methods. In *Proceedings of the 23rd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 2015.

[35] S. Worrall and E. Nebot. Automated process for generating digitised maps through GPS data compression. In *Proc. Australasian Conf. on Robotics and Automation*, 2007.

[36] L. Zhang, F. Thiemann, and M. Sester. Integration of GPS traces with road map. In *Proc. 3rd ACM SIGSPATIAL Int. Workshop on Computational Transportation Science*, pages 17–22, 2010.

On the Coverage of Points in the Plane by Disks Centered at a Line

Logan Pedersen* Haitao Wang†

Abstract

Given a set P of n points and a line L in the plane, we consider the problem of computing a set S of disks centered at L such that their union covers all points of P. The cost of a disk is defined as a function $f(r) = r^\alpha$, where $\alpha \geq 1$ is a constant and r is the radius of the disk. The objective is to minimize the total sum of the cost of all disks of S. Previously [Alt et al., SoCG 2006], the problem was solved in $O(n^4 \log n)$ time in any fixed L_p metric (and in $O(n^2 \log n)$ time if $\alpha = 1$). In this paper, we present a new algorithm that runs in $O(n^2)$ time for any $\alpha \geq 1$ in any fixed L_p metric. In addition, we also give algorithms for two variations of the 1D problem where all points of P are in L: (1) there is an upper bound k on $|S|$, and (2) the disk centers must be chosen from another given set of potential locations on L.

1 Introduction

In this paper, we consider the following disk coverage problem. Given a set P of n points and a line L in the plane, we want to find a set S of disks centered at L such that each point of P is covered by at least one disk and the total sum of the cost of all disks of S is minimized. Here, the *cost* of a disk of S is defined to be $f(r) = r^\alpha$ for a given constant $\alpha \geq 1$, where r is the radius of the disk. Note that if $\alpha = 1$ (resp., $\alpha = 2$), we are minimizing the total sum of the radii (resp., areas) of all disks. The problem is motivated by power consumption models in wireless network design, where α is often larger than or equal to 2 [3, 13, 21]. We consider the general metric L_p for any $p \geq 1$, where a point q is said to be covered by a disk centered at a point c with radius r if the L_p distance between q and c is at most r. We refer to the problem as the *disk coverage* problem.

Previously, Alt et al. [3] gave an algorithm that solves the problem in $O(n^4 \log n)$ time in any L_p metric and for any $\alpha \geq 1$. Better algorithms for some special cases of the problem were also presented in [3]. If $\alpha = 1$, then they solved the problem in $O(n^2 \log n)$ time in any L_p metric. In the L_∞ metric, they gave an $O(n^3 \log n)$ time algorithm for any $\alpha \geq 1$.

*Department of Computer Science, Utah State University, Logan, UT 84341, USA, logan.pedersen@aggiemail.usu.edu

†Department of Computer Science, Utah State University, Logan, UT 84341, USA, haitao.wang@usu.edu

In this paper, we propose a new algorithm of $O(n^2)$ time for any L_p metric and any $\alpha \geq 1$, which improves the $O(n^4 \log n)$ time algorithm in [3] by a more than quadratic factor. Our algorithm first reduces the problem to finding a shortest path in a directly acyclic graph (DAG) G, with $n + 1$ vertices and $\Theta(n^2)$ edges. One difficulty is to compute the weights of the edges of G. We propose an algorithm that can compute each edge weight in $O(1)$ amortized time. Consequently, a shortest path in G can be found in $O(n^2)$ time by a textbook dynamic programming algorithm [12].

In addition, we consider the one-dimensional version of the problem where the points of P are all on L (in contrast, we may consider the above more general problem as a "1.5D" problem). Note that if there are no constraints on the disks, then one could obtain an optimal solution by placing a disk with zero radius at each point of P. Thus, we consider two variations with constraints on the disks.

In the first problem, we are allowed to place at most k disks, for a given $k \in [1, n]$. To our best knowledge, we have not seen any previous work on this problem before. We reduce the problem to computing a k-link shortest path in a DAG of $n + 1$ vertices and $O(n^2)$ edges, which can then be solved in $O(kn^2)$ time by an easy dynamic programming algorithm. Further, we show that the edge weights of the graph obey the concave Monge condition, and consequently, we can solve it in $O(nk)$ [1], $O(n\sqrt{k \log n} + n \log n)$ [2], or $n2^{O(\sqrt{\log k \log \log n})}$ time [22], after the points of P are sorted on L. We refer to this problem as the *k-interval coverage* problem because a disk in 1D becomes an interval on L.

In the second problem, in addition to P, we are given another set Q of m points on L as the potential locations for the centers of the disks (i.e., the center of each disk of S must be in Q). This problem has been studied before. Bilò et al. [6] first showed that the problem is solvable in polynomial time. Lev-Tov and Peleg [17] gave an algorithm of $O((n + m)^3)$ time for any $\alpha \geq 1$. Some progress has been made recently by Biniaz et al. [7], who proposed an $O((n + m)^2)$ time algorithm for the case $\alpha = 1$. In this paper, we solve the problem in $O(n(n + m) + m \log m)$ time for any $\alpha \geq 1$, again by reducing it to finding a shortest path in a DAG. We refer to this problem as the *discrete interval coverage* problem.

1.1 Related Work

Some faster approximation algorithms are also known for these problems. For the discrete interval coverage problem, Lev-Tov and Peleg [17] derived a linear time algorithm with approximation ratio 4 for the case $\alpha = 1$, and the ratio was reduced to 3 by Alt et al. [3] with the same running time. Alt et al. [3] also proposed a 2-approximation algorithm with $O(m + n \log m)$ time for the case $\alpha = 1$. Efficient approximation algorithms were also given by Alt et al. [3] for the 1.5D disk coverage problem in the L_∞ metric for any $\alpha \geq 1$. In addition, Alt et al. [3] considered a variation of the 1.5D disk coverage problem where we are given the slope of L but its location may be chosen freely to minimize the total cost. The problem was shown not computable by radicals when $\alpha = 1$, but FPTAS were given for $\alpha = 1$ and $\alpha > 1$ [3].

The discrete case of the 2D disk coverage problem where both P and Q are points in the Euclidean plane is shown to be NP-hard for any $\alpha > 1$ [3]. For the case $\alpha = 1$, Lev-Tov and Peleg [17] gave a PTAS, and later, Gibson et al. [14] showed that the problem is solvable in polynomial time[1]. A variant of the problem in which $P = Q$ but there is an upper bound k on the number of disks is also solved in polynomial time [14]. Other variations of the problem are considered elsewhere, e.g., [5, 6]

The traditional k-center and k-median problems are closely related to the disk coverage problem. Roughly speaking, the k-center problem is to minimize the largest radius of the disks and the k-median problem is to minimize the total sum of distances from all points to their closest disk centers. Both problems have an upper bound k on the number of disks that can be used. These problems are in general NP-hard [19], but have polynomial time solutions in some special cases, e.g., the 1D case [4, 10, 11, 15, 20], the 1.5D case [8, 16, 23], 1 or 2-center in the Euclidean plane [9, 18], etc.

Paper Outline. The rest of the paper is organized as follows. Section 2 defines some notation. In Section 3, we present our algorithm for the 1.5D disk coverage problem. The algorithms for the 1D problems are given in Section 4. Section 5 concludes the paper with remarks on possible extensions of our results to other more general cost functions $f(r)$ and possible improvements on our results.

2 Preliminaries

For ease of exposition, for all three problems studied in the paper, we make a general position assumption that no two points of P have the same x-coordinate. Without loss of generality, we assume that the line L is the x-axis. These assumptions can be easily lifted without affecting the performance of our algorithms.

In each problem, we first sort all points of P by their x-coordinates from left to right, and let the sorted list be $p_1, p_2, \ldots p_n$. For any i, j with $1 \leq i \leq j \leq n$, let $P[i, j]$ denote the sublist $p_i, p_{i+1}, \ldots, p_j$.

For any point q in the plane, let $x(q)$ and $y(q)$ denote the x- and y-coordinates of q, respectively.

For any two points q and q' in the plane, we use $d_p(q, q')$ to denote their L_p distance. We say that q is *to the left* of q' if $x(q) \leq x(q')$, and q is *to the right* of q' if $x(q) \geq x(q')$.

In any solution of each problem, if a point p_i is covered by a disk centered at c, then we call c a *server* and we say that p_i is "served" or "covered" by c.

3 The 1.5D Disk Coverage Problem

In this section, we present our $O(n^2)$ time algorithm for the 1.5D disk coverage problem. In this problem, the points of P are in the plane. Recall that L is the x-axis.

We assume that all points of P are above or on the x-axis (since otherwise if a point $p \in P$ was below the x-axis, we could replace p by its symmetrical point with the x-axis without affecting the optimal solution).

Recall that $P = \{p_1, p_2, \ldots p_n\}$, already sorted on L from left to right. We first model the problem to a shortest path problem in a directly acyclic graph (DAG) G. To this end, the following lemma is critical (the lemma is also applicable to the two 1D problems).

Lemma 1 *In any fixed L_p metric, for any $\alpha \geq 1$, there exists an optimal solution in which the points of P served by the same server are consecutive in their index order.*

Proof. Consider an optimal solution in which the lemma statement does not hold. Then, there must exist two consecutive points p_i and p_{i+1} (we call them a *conflict pair*) and two servers c_1 and c_2 with $x(c_1) < x(c_2)$ such that p_i is served by c_2 and p_{i+1} is served by c_1 in the solution (e.g., see Fig. 1). In the following, we show that we can switch the service of p_i and/or p_{i+1} so that either they are served by the same server, or we can use c_1 to serve p_i and use c_2 to serve p_{i+1}, without affecting the total cost of the solution.

Let r_1 be the radius of the disk centered at c_1, and r_2 the radius of the disk centered at c_2. Since p_{i+1} is served by c_1 and p_i is served by c_2, we have $d_p(c_1, p_{i+1}) \leq r_1$ and $d_p(c_2, p_i) \leq r_2$. Without loss of generality, we assume that $y(p_i) \geq y(p_{i+1})$. Depending on whether $x(p_{i+1}) \leq x(c_2)$, there are two cases.

If $x(p_{i+1}) \leq x(c_2)$, then since $y(p_i) \geq y(p_{i+1})$ and $x(p_i) < x(p_{i+1})$, it holds that $d_p(c_2, p_{i+1}) \leq d_p(c_2, p_i) \leq$

[1] The result is based on the assumption that the two sums of square roots of integers can be compared in polynomial time. The algorithm can be extended to L_1 and L_∞ cases without the assumption [14].

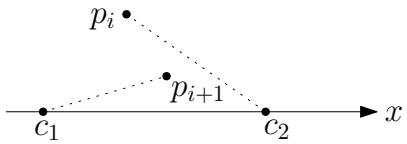


Figure 1: Illustrating the case where p_i is served by c_2 and p_{i+1} is served by c_1.

r_2. Hence, we can use c_2 to serve p_{i+1} without increasing the total cost of the solution.

If $x(p_{i+1}) > x(c_2)$, since $y(p_i) \geq y(p_{i+1})$ and $x(p_i) < x(p_{i+1})$, the two line segments $\overline{p_ic_2}$ and $\overline{p_{i+1}c_1}$ cross each other (e.g., see Fig. 2). By triangle inequality of the metric space, we have the following

$$d_p(c_1, p_i) + d_p(c_2, p_{i+1}) \leq d_p(c_2, p_i) + d_p(c_1, p_{i+1}). \quad (1)$$

If $d_p(c_2, p_{i+1}) \leq d_p(c_2, p_i)$, then since $d_p(c_2, p_i) \leq r_2$, we obtain $d_p(c_2, p_{i+1}) \leq r_2$. Thus, we can use c_2 to serve p_{i+1} without increasing the total cost of the solution. Otherwise, by Equation (1), we can derive $d_p(c_1, p_i) < d_p(c_1, p_{i+1})$, which implies that we can use c_1 to serve p_i without increasing the total cost of the solution.

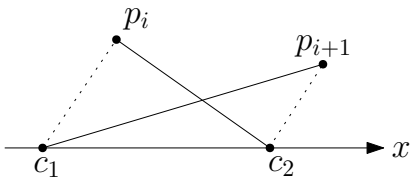


Figure 2: The two segments $\overline{p_ic_2}$ and $\overline{p_{i+1}c_1}$ cross each other. By triangle inequality, the sum of the lengths of the two solid segments is larger than or equal to the sum of the lengths of the two dotted segments.

The above shows that our switch operation "fixed" a conflict pair without increasing the total cost of the solution. If after the switch operation the new optimal solution still does not satisfy the lemma statement, then there must exist another conflict pair and we can continue applying the switch operation on them. Note that this procedure will be finite and thus eventually we will obtain an optimal solution without any conflict pairs, which implies that the optimal solution satisfies the lemma statement. □

Based on Lemma 1, we define a DAG G as follows. The vertex set consists of $n+1$ vertices $v_0, ...v_n$ so that each vertex v_i corresponds to an imaginary point between p_i and p_{i+1} (v_0 is to the left of p_1 and v_n is to the right of p_n). For all $0 \leq i < j \leq n$, we create a directed edge $e(i, j)$ from v_i to v_j, and the weight $w(i, j)$ of the edge is defined as $f(r)$, where r is the radius of the smallest disk centered at L that can cover all points of $P[i+1, j]$ (which is $\{p_{i+1}, p_{i+2}, \ldots, p_j\}$). Lemma 1 immediately leads to the following result.

Corollary 2 *A shortest path π from v_0 to v_n in G corresponds to an optimal solution to the disk coverage problem, i.e., the length of π is equal to the total cost and each edge $e(i, j)$ corresponds to a smallest disk centered at L covering all points of $P[i+1, j]$.*

Since G is a DAG and has $O(n^2)$ edges, a shortest path from v_0 to v_n can be computed in $O(n^2)$ time by a dynamic programming algorithm [12] if the weights of all graph edges are known. In the following, we show that the weights of all edges of G can be computed in $O(n^2)$ time. In particular, we have the following lemma.

Lemma 3 *For each vertex v_i, the weights of all its outgoing edges, i.e., $w(i, j)$ for all $j \in [i+1, n]$, can be computed in $O(n-i)$ time.*

Proof. To simplify the notation, we only consider the case $i = 0$. The algorithm can be generalized to any other index i in a straightforward manner. Our goal is to compute $w(0, j)$ for all $j \in [1, n]$ in $O(n)$ time.

For each $j \in [1, n]$, define c_j and r_j respectively as the center and the radius of the smallest disk centered at L that covers all points of $P[1, j]$. By definition, $w(0, j) = f(r_j)$. Below, we will give an incremental algorithm to compute c_j and r_j for all $j = 1, 2, \ldots, n$ in a total of $O(n)$ time.

Note that since $x(p_1) < x(p_2) < \cdots < x(p_n)$, we have $x(c_1) \leq x(c_2) \leq \cdots \leq x(c_n)$. This implies that when computing c_j from $j = 1$ to $j = n$, we only need to consider the locations of L from left to right.

For any two points p_i and p_j of P with $i < j$, there is a point, denoted by $q(i, j)$, on L such that for any point $c \in L$, $d_p(c, p_i) \leq d_p(c, p_j)$ if c is to the left of $q(i, j)$ and $d_p(c, p_i) \geq d_p(c, p_j)$ otherwise. We assume that given p_i and p_j, $q(i, j)$ can be computed in $O(1)$ time.

For any point p_i, we use p'_i to denote the point on L with the same x-coordinate as p_i. Clearly, for each $j \in [1, n]$, $r_j \geq d_p(p'_j, p_j)$.

As a warm-up and for better understanding the rationale of our algorithm, we first show how to process the first two points p_1 and p_2 (to compute c_j and r_j for $j = 1, 2$).

Initially, when $j = 1$, c_1 is p'_1 and $r_1 = d_p(c_1, p_1)$. Next, consider $j = 2$. We first compute the point $q(1, 2)$. The two points p'_1 (which is also c_1) and p'_2 divide L into three parts, and depending on which part contains $q(1, 2)$, there are three cases.

If $x(q(1, 2)) \leq x(c_1)$, then $d_p(c_1, p_2) \leq d_p(c_1, p_1) = r_1$. Thus, $c_2 = c_1$ and $r_2 = r_1$. Further, the point p_2 can be ignored in the future algorithm. Indeed, for any point $c \in L$ to the right of c_1, it holds that $d_p(c, p_1) \geq d_p(c, p_2)$. Since $x(c_j) \geq x(c_2)$ for all $j \geq 3$, when computing c_j for any $j \geq 3$, p_1 "dominates" p_2, and thus p_2 can be ignored.

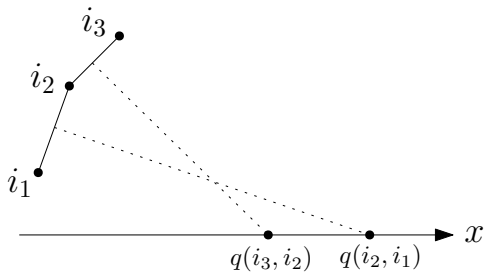


Figure 3: Illustrating a canonical list of three points in the L_2 metric. Each dotted segment is a perpendicular bisector of the segment connecting two consecutive points in the canonical list.

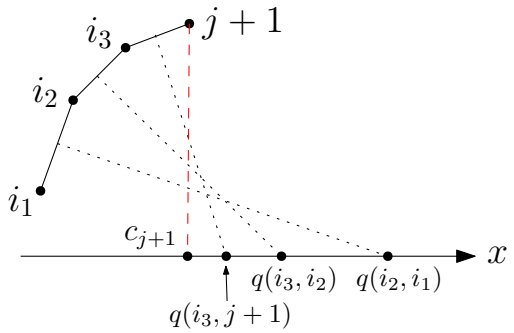


Figure 4: Update the canonical list of Fig. 3 by adding p_{j+1} to the end (i.e., setting $i_4 = j + 1$).

If $x(c_1) < x(q(1,2)) \leq x(p_2)$, then $c_2 = q(1,2)$ and $r_2 = d_p(c_2, p_1) = d_p(c_2, p_2)$. Further, as the above argument, p_2 can be ignored in the future algorithm.

If $x(q(1,2)) > x(p_2)$, then $d_p(p'_2, p_1) \leq d_p(p'_2, p_2)$. Thus, $c_2 = p'_2$ and $r_2 = d_p(p'_2, p_2)$. However, in this case, neither p_1 nor p_2 should be ignored because we do not know whether c_j is to the left or right of $q(1,2)$, e.g., for $j = 3$.

The above discusses our algorithm for processing p_1 and p_2. In the following, we describe our general algorithm. The pseudocode is given in Algorithm 1.

Suppose our algorithm has processed the points $p_1, p_2, \ldots, p_j$ (and thus c_i and r_i for all $i \in [1, j]$ have been computed) and is about to preprocess p_{j+1}. Then, our algorithm maintains a *canonical list* of h points $p_{i_1}, p_{i_2}, \ldots, p_{i_h}$ for $h \leq j$ with the following invariants (e.g., see Fig. 3).

1. $i_1 < i_2 < \cdots < i_h$.
2. $x(q(i_h, i_{h-1})) < x(q(i_{h-1}, i_{h-2})) < \cdots < x(q(i_2, i_1))$.
3. $x(p_{i_h}) \leq x(c_j) < x(q(i_h, i_{h-1}))$.
4. $r_j = d_p(c_j, p_{i_h})$.
5. To simplify the discussion, let $q(i_{h+1}, i_h) = c_j$ and $q(i_1, i_0)$ be the point on L with x-coordinate $+\infty$. For each $t \in [1, h]$, with respect to any point c between $q(i_{t+1}, i_t)$ and $q(i_t, i_{t-1})$ on L, the point p_{i_t} dominates all other points of $P[1, j]$, i.e., $d_p(c, p_{i_t}) \geq d_p(c, p_i)$ for all $i \in [1, j]$.

Initially, after p_1 is processed, our canonical list consists of a single point p_1, and all algorithm invariants hold (as an exercise, one can check that after p_2 is processed as discussed above the invariants also hold). Next, we discuss a general step of our algorithm for processing p_{j+1}.

We first compute the point $q(i_h, j+1)$ on L, and depending on its location with respect to c_j and $q(i_h, i_{h-1})$, there are three cases. Note that according to our algorithm invariants, $x(p_{i_h}) \leq x(c_j) < x(q(i_h, i_{h-1}))$.

If $x(q(i_h, j+1)) \leq x(c_j)$, then $d_p(c_j, p_{j+1}) \leq d_p(c_j, p_{i_h}) = r_j$. By definition, the disk centered at c_j with radius r_j is the smallest one covering all points of $P[1, j]$. As $d_p(c_j, p_{j+1}) \leq r_j$, the disk also covers p_{j+1} and thus is the smallest one covering all points of $P[1, j+1]$. Hence, $c_{j+1} = c_j$ and $r_{j+1} = r_j$. Further, for any point c on L to the right of c_{j+1}, $d_p(c, p_{i_h}) \geq d_p(c, p_{j+1})$, and thus p_{j+1} is dominated by p_{i_h} and can be ignored in the future algorithm. Therefore, in this case the canonical list $p_{i_1}, p_{i_2}, \ldots, p_{i_h}$ does not change, and all algorithm invariants hold since $c_{j+1} = c_j$.

If $x(c_j) < x(q(i_h, j+1)) < x(q(i_h, i_{h-1}))$ (this includes the case $h = 1$; recall that we have assumed $x(q(i_1, i_0)) = +\infty$), depending on whether $q(i_h, j+1)$ is to the left of p'_{j+1}, there are two subcases.

1. If $x(q(i_h, j+1)) \leq x(p_{j+1})$, then $c_{j+1} = q(i_h, j+1)$ and $r_{j+1} = d_p(c_{j+1}, p_{i_h})$. Indeed, by our algorithm invariants, since $x(c_j) < x(c_{j+1}) < x(q(i_h, i_{h-1}))$, c_{j+1} covers all points of $P[1, j]$ with distance r_{j+1}. On the other hand, by the definition of c_{j+1} and r_{j+1}, the disk centered at c_{j+1} with radius r_{j+1} is the smallest one covering p_{i_h} and p_{j+1}.

   Further, for any point c to the right of c_{j+1}, $d_p(c, p_{i_h}) \geq d_p(c, p_{j+1})$, and thus p_{j+1} is dominated by p_{i_h} and can be ignored in the future algorithm. Therefore, in this case the canonical list does not change, and all algorithm variants still hold since $x(c_j) < x(c_{j+1}) < x(q(i_h, i_{h-1}))$.

2. If $x(q(i_h, j+1)) > x(p_{j+1})$, then again by our algorithm invariants, $c_{j+1} = p'_{j+1}$ and $r_{j+1} = d_p(c_{j+1}, p_{j+1})$. In this case, we add p_{j+1} to the end of our canonical list by setting $i_{h+1} = j + 1$ and incrementing h by one (e.g., see Fig. 4). Due to $x(c_{j+1}) < x(q(i_h, j+1)) < x(q(i_h, i_{h-1}))$, one can verify that all our algorithm invariants hold.

If $x(q(i_h, j+1)) \geq x(q(i_h, i_{h-1}))$, then observe that p_{i_h} is dominated by p_{j+1} with respect to any location $c \in L$ to the left of $q(i_h, i_{h-1})$. Further, according to our algorithm invariants, p_{i_h} is dominated by at least one point of p_{i_t} for $t \in [1, h-1]$ with respect to any location $c \in L$ to the right of $q(i_h, i_{h-1})$. Hence, in this case we remove p_{i_h} from the canonical list, by decrementing h by one. In the following discussion, we assume h has been decremented and thus use $p_{i_{h+1}}$ to denote the removed point. We consider the location of $q(i_h, j+1)$ (again, this h has been decremented) and proceed as follows. As $x(q(i_{h+1}, j+1)) \geq x(q(i_{h+1}, i_h))$, $d_p(q(i_{h+1}, i_h), p_{j+1}) \geq d_p(q(i_{h+1}, i_h), p_{i_{h+1}}) = d_p(q(i_{h+1}, i_h), p_{i_h})$. Hence, $q(i_h, j+1)$ is to the right of $q(i_{h+1}, i_h)$. Since $x(q(i_{h+1}, i_h)) > x(c_j)$ (by algorithm invariants), we obtain $x(q(i_h, j+1)) > x(c_j)$. If $h > 1$ and $x(q(i_h, j+1)) \geq x(q(i_h, i_{h-1}))$, then we repeat the same procedure as above. Otherwise, depending on whether $x(q(i_h, j+1)) \leq x(p_{j+1})$, there are two subcases, whose processing is the same as above for the case $x(c_j) < x(q(i_h, j+1)) < x(q(i_h, i_{h-1}))$, and we omit the details.

The above describes a general step of our algorithm for processing p_{j+1}. The algorithm stops once p_n is processed. For the running time, processing p_{j+1} takes $O(1+t)$ time, where t is the number of points removed from the canonical list. Observe that each point of P will be added to the list and removed from the list at most once in the entire algorithm. Therefore, the total time of the algorithm is $O(n)$. □

Algorithm 1: Computing c_j and r_j for all $j \in [1, n]$

```
1  c_1 ← p'_1, r_1 ← d_p(p_1, p'_1), i_1 ← 1, h ← 1;
2  for j ← 1 to n − 1 do
3      compute q(i_h, j + 1);
4      if x(q(i_h, j + 1)) ≤ x(c_j) then
5          c_{j+1} ← c_j, r_{j+1} ← r_j;
6      else /* The following combines the
           second and third cases in the algorithm
           description                          */
7          while h > 1 and
             x(q(i_h, j + 1)) ≥ x(q(i_h, i_{h−1})) do
8              h ← h − 1, compute q(i_h, j + 1);
9          if x(q(i_h, j + 1)) ≤ x(p_{j+1}) then
10             c_{j+1} ← q(i_h, j + 1), r_{j+1} ← d_p(c_{j+1}, p_{i_h});
11         else
12             c_{j+1} ← p'_{j+1}, r_{j+1} ← d_p(c_{j+1}, p_{j+1}),
                 i_{h+1} ← j + 1, h ← h + 1;
```

By Lemma 3, we can compute a shortest path from v_0 to v_n in G in $O(n^2)$ time, after which an optimal solution for our original problem can be readily obtained according to Corollary 2. Note that the shortest path algorithm can be implemented in $O(n)$ space. Indeed, whenever a vertex v_i is processed, it is sufficient to know the weights of the outgoing edges of v_i by applying Lemma 3, and the weights of other edges of the graph can be ignored. Thus, we have the following theorem.

Theorem 4 *In any fixed L_p metric, for any $\alpha \geq 1$, the 1.5D disk coverage problem can be solved in $O(n^2)$ time and $O(n)$ space.*

4 The One-Dimensional Problem

In this section, we consider the two variations of the 1D problem. Note that in the 1D problem, for any two points q and q' on the x-axis, $d_p(q, q') = |x(q) - x(q')|$ in any L_p metric. Therefore, we will use $d(q, q')$ to denote the value $|x(q) - x(q')|$. We begin with the k-interval coverage problem.

4.1 The k-Interval Coverage Problem

In this problem, we are given a set P of n points on L (the x-axis), an integer $k \in [1, n]$, and $\alpha \geq 1$. The goal is to compute a set of at most k disks centered at L covering all points of P such that the total cost of all disks is minimized.

We follow the same notation as before and use $p_1, p_2, \ldots, p_n$ as the sorted list of P from left to right. Observe that Lemma 1 still holds for this problem. Thus, we build the same DAG G as before. The weights of the edges of G are also defined in the same way as before. Consequently, our problem is equivalent to computing a shortest path from v_0 to v_n in G with at most k edges (this is usually called a *k-link shortest path*). Further, we have a simple algorithm to compute the edge weights of the graph, as shown in the following lemma.

Lemma 5 *For any edge $e(i, j)$ with $0 \leq i < j \leq n$, the weight $w(i, j)$ can be computed in constant time.*

Proof. According to the definition, $w(i, j) = f(r) = r^\alpha$, where r is the radius of the smallest disk centered at L covering all points of $P[i+1, j]$. Observe that $r = |x(p_j) - x(p_{i+1})|/2$. Hence, $w(i, j)$ can be computed in constant time. □

Using Lemma 5, we can find a k-link shortest path from v_0 to v_n in G in $O(kn^2)$ time by a straightforward dynamic programming algorithm. However, we can do better due to that the edge weights of the graph obey the concave Monge condition [1, 2], which is proved in the following lemma.

Lemma 6 *The graph G has the concave Monge property, i.e., for any i and j with $0 < i+1 < j < n$, it holds that $w(i, j) + w(i+1, j+1) \leq w(i, j+1) + w(i+1, j)$.*

Proof. For any i', j' with $1 \leq i' \leq j' \leq n$, define $r(i', j') = |x(p_{j'}) - x(p_{i'})|/2$.

As discussed in the proof of Lemma 5, we have $w(i,j) = (r(i+1,j))^\alpha$, $w(i+1,j+1) = (r(i+2,j+1))^\alpha$, $w(i,j+1) = (r(i+1,j+1))^\alpha$, and $w(i+1,j) = (r(i+2,j))^\alpha$. Observe that $r(i+1,j)+r(i+2,j+1) = r(i+1,j+1)+r(i+2,j)$. Since $r(i+1,j+1) > r(i+1,j)$, $r(i+1,j+1) > r(i+2,j+1)$, and $\alpha \geq 1$, we can obtain that $(r(i+1,j))^\alpha + (r(i+2,j+1))^\alpha \leq (r(i+1,j+1))^\alpha + (r(i+2,j))^\alpha$. Therefore, $w(i,j)+w(i+1,j+1) \leq w(i,j+1) + w(i+1,j)$ holds. □

Due to the concave Monge property, we can resort to faster algorithms for computing a k-link shortest path from v_0 to v_n in G in $O(nk)$ [1], $O(n\sqrt{k \log n} + n \log n)$ [2], or $n2^{O(\sqrt{\log k \log\log n})}$ time [22]. Note that when applying these algorithms, we will not compute the graph G explicitly; rather, whenever the algorithm needs an edge weight, we use the algorithm in Lemma 5 to compute it in $O(1)$ time. Therefore, we have the following result.

Theorem 7 *For any $\alpha \geq 1$, after the points of P are sorted on L, the k-interval coverage problem can be solved in $\min\{O(nk), O(n\sqrt{k \log n} + n \log n), n2^{O(\sqrt{\log k \log\log n})}\}$ time.*

4.2 The Discrete Interval Coverage Problem

In this problem, we are given a set P of n points and another set Q of m points on L (the x-axis), as well as $\alpha \geq 1$. The goal is to compute a set of disks centered at the points of Q to cover all points of P such that the total cost of the disks is minimized.

Again, let $p_1, p_2, \ldots, p_n$ be the sorted list of P from left to right. We also sort all points of Q from left to right on L, and let $q_1, q_2, \ldots, q_m$ be the sorted list. One can verify that Lemma 1 still applies. With respect to the sorted list of P, we define the same DAG G as before. Here, the weight $w(i,j)$ of each edge $e(i,j)$ is defined as the smallest disk centered at a point in Q covering all points of $P[i+1,j]$. Hence, our problem is equivalent to finding a shortest path from v_0 to v_n in G. The following lemma gives an algorithm for computing the weights of the edges of G.

Lemma 8 *For each vertex v_i, the weights of all its outgoing edges, i.e., $w(i,j)$ for all $j \in [i+1,n]$, can be computed in $O(m+n-i)$ time.*

Proof. For any $j \in [i+1,n]$, let D be the smallest disk covering all points of $P[i+1,j]$ such that the center is a point in Q. Let $q(i+1,j)$ be the middle point of the line segment $\overline{p_{i+1}p_j}$ on L. Let c a point of Q that is closest to $q(i+1,j)$, and $r = \max\{|x(c)-x(p_{i+1})|, |x(c)-x(p_j)|\}$. Observe that c must be a center of D and r must be the radius. Therefore, to compute the weight $w(i,j)$, it is sufficient to determine the point of Q closest to $q(i+1,j)$.

We first compute the points $q(i+1,j)$ for all $j \in [i+1,n]$ in $O(n-i)$ time. Then the points of Q closest to $q(i+1,j)$'s for all $j \in [i+1,n]$ can be found in $O(m+n-i)$ time by a linear scan simultaneously on both the sorted list of Q and the list $q(i+1,i+1), q(i+1,i+2), \ldots, q(i+1,n)$, which is also sorted on L from left to right. Consequently, the weights $w(i,j)$ for all $j \in [i+1,n]$ can be computed in $O(m+n-i)$ time. □

By Lemma 8, we can compute a shortest path from v_0 to v_n in G in $O(n(m+n))$ time, after which an optimal solution for our original problem can be readily obtained. As in Section 3, with Lemma 8, the algorithm can be implemented in $O(n+m)$ space. Therefore, we have the following theorem, where the $O(m \log m)$ factor is due to the sorting of Q.

Theorem 9 *For any $\alpha \geq 1$, the discrete interval coverage problem can be solved in $O(n(m+n) + m \log m)$ time and $O(m+n)$ space.*

5 Concluding Remarks

In this paper, we present new algorithms for covering points by disks. We have been considering the cost function $f(r) = r^\alpha$ for a constant $\alpha \geq 1$. In fact, our algorithms for the 1.5D case and for the discrete 1D case also work with the same complexities for any non-decreasing function $f(r)$ as long as the following assumption holds: given any r, $f(r)$ can be computed in constant time. Our algorithm for the k-interval coverage problem, however, may not work for all non-decreasing functions, because the Monge property in Lemma 6 may not hold any more (in which case we can still use the straightforward $O(kn^2)$ time dynamic programming algorithm to solve the problem).

In addition, for the 1.5D case and the discrete 1D case, if there is an upper bound k on the number of disks that are allowed to be used, then the problem is equivalent to computing a k-link shortest path from v_0 to v_n in the DAG G, which can be done in $O(kn^2)$ time by dynamic programming after the graph G is computed.

It would be interesting to see whether the algorithms can be further improved, especially for the 1.5D problem and the discrete 1D problem. One might wonder whether the DAGs for these two problems also have Monge properties (either convex or concave). Unfortunately, we have found examples showing that the DAG for each problem does not have either convex or concave Monge property. Therefore, new techniques may be needed for further improvement.

References

[1] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilbur. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987.

[2] A. Aggarwal, B. Schieber, and T. Tokuyama. Finding a minimum weight k-link path in graphs with concave monge property and applications. *Discrete & Computational Geometry*, 12:263–280, 1994.

[3] H. Alt, E. Arkin, H. Brönnimann, J. Erickson, S. Fekete, C. Knauer, J. Lenchner, J. Mitchell, and K. Whittlesey. Minimum-cost coverage of point sets by disks. In *Proc. of the 22nd Annual Symposium on Computational Geometry*, pages 449–458, 2006.

[4] V. Auletta, D. Parente, and G. Persiano. Placing resources on a growing line. *Journal of Algorithms*, 26(1):87–100, 1998.

[5] B. Behsaz and M. Salavatipour. On minimum sum of radii and diameters clustering. *Algorithmica*, 73:143–165, 2015.

[6] V. Bilò, I. Caragiannis, C. Kaklamanis, and P. Kanellopoulos. Geometric clustering to minimize the sum of cluster sizes. In *Proc. of the 13th European Symposium on Algorithms*, pages 460–471, 2005.

[7] A. Biniaz, P. Bose, P. Carmi, A. Maheshwari, I. Munro, and M. Smid. Faster algorithms for some optimization problems on collinear points. 2018. To appear in *the 34th International Symposium on Computational Geometry*, full version at arXiv:1802.09505.

[8] P. Brass, C. Knauer, H.-S. Na, C.-S. Shin, and A. Vigneron. The aligned k-center problem. *International Journal of Computational Geometry and Applications*, 21:157–178, 2011.

[9] T. Chan. More planar two-center algorithms. *Computational Geometry: Theory and Applications*, 13(3):189–198, 1999.

[10] D. Chen, J. Li, and H. Wang. Efficient algorithms for the one-dimensional k-center problem. *Theoretical Computer Science*, 592:135–142, 2015.

[11] D. Chen and H. Wang. New algorithms for facility location problems on the real line. *Algorithmica*, 69:370–383, 2014.

[12] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 3nd edition, 2009.

[13] H. Fan, M. Li, X. Sun, P. Wan, and Y. Zhao. Barrier coverage by sensors with adjustable ranges. *ACM Transactions on Sensor Networks*, 11:14:1–14:20, 2014.

[14] M. Gibson, G. Kanade, E. Krohn, I. Pirwani, and K. Varadarajan. On clustering to minimize the sum of radii. *SIAM Journal on Computing*, 41:47–60, 2012.

[15] R. Hassin and A. Tamir. Improved complexity bounds for location problems on the real line. *Operations Research Letters*, 10:395–402, 1991.

[16] A. Karmakar, S. Das, S. Nandy, and B. Bhattacharya. Some variations on constrained minimum enclosing circle problem. *Journal of Combinatorial Optimization*, 25(2):176–190, 2013.

[17] N. Lev-Tov and D. Peleg. Polynomial time approximation schemes for base station coverage with minimum total radii. *Computer Networks*, 47:489–501, 2005.

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

[19] N. Megiddo and K. Supowit. On the complexity of some common geometric location problems. *SIAM Journal on Computing*, 13:182–196, 1984.

[20] N. Megiddo and A. Tamir. New results on the complexity of p-centre problems. *SIAM Journal on Computing*, 12(4):751–758, 1983.

[21] K. Pahlavan and A. Levesque. *Wireless Information Networks*. Wiley, New York, NY, 2nd edition, 2005.

[22] B. Schieber. Computing a minimum weight k-link path in graphs with the concave monge property. *Journal of Algorithms*, 29(2):204–222, 1998.

[23] H. Wang and J. Zhang. Line-constrained k-median, k-means, and k-center problems in the plane. *International Journal of Computational Geometry and Applications,*, 26:185–210, 2016.

A Composable Coreset for k-Center in Doubling Metrics

Sepideh Aghamolaei* Mohammad Ghodsi†

Abstract

A set of points P in a metric space and a constant integer k are given. The k-center problem finds k points as *centers* among P, such that the maximum distance of any point of P to their closest centers (r) is minimized.

Doubling metrics are metric spaces in which for any r, a ball of radius r can be covered using a constant number of balls of radius $r/2$. Fixed dimensional Euclidean spaces are doubling metrics. The lower bound on the approximation factor of k-center is 1.822 in Euclidean spaces, however, $(1+\epsilon)$-approximation algorithms with exponential dependency on $\frac{1}{\epsilon}$ and k exist.

For a given set of sets $P_1, \ldots, P_L$, a *composable coreset* independently computes subsets $C_1 \subset P_1, \ldots, C_L \subset P_L$, such that $\cup_{i=1}^{L} C_i$ contains an approximation of a measure of the set $\cup_{i=1}^{L} P_i$.

We introduce a $(1+\epsilon)$-approximation composable coreset for k-center, which in doubling metrics has size sublinear in P. This results in a $(2+\epsilon)$-approximation algorithm for k-center in MapReduce with a constant number of rounds and sublinear communications, which improves upon the previous 4-approximation algorithm. We also prove a trade-off between the size and the approximation factor of our coreset, and give a composable coreset for a related problem called dual clustering.

1 Introduction

Coresets are subsets of points that approximate a measure of the point set. A method of computing coresets on big data sets is composable coresets. Composable coresets [20] provide a framework for adapting constant factor approximation algorithms to streaming and MapReduce models. Composable coresets summarize distributed data so that the scalability is increased, while keeping the desirable approximation factor and time complexity.

There is a general algorithm for solving problems using coresets which known by different names in different settings: mergeable summaries [1] and merging in a tree-like structure [2] for streaming $(1+\epsilon)$-approximation algorithms, small space (divide and conquer) for constant factor approximations in streaming [15], and composable coresets in MapReduce [20]. A consequence of using constant factor approximations instead of $(1+\epsilon)$-approximations with the same merging method is that it can add a $O(\log n)$ factor to the approximation factor of the algorithm on an input of size n.

Composable coresets [20] require only a single round and sublinear communications in the MapReduce model, and the partitioning is done arbitrarily.

Definition 1 (Composable Coreset) *A composable coreset on a set of sets $\{S_i\}_{i=1}^{L}$ is a set of subsets $C(S_i) \subset S_i$ whose union gives an approximation solution for an objective function $f : (\cup_{i=1}^{L} S_i) \to \mathbf{R}$. Formally, a composable coreset of a minimization problem is an α-approximation if*

$$f(\cup_i S_i) \leq f(\cup_i C(S_i)) \leq \alpha . f(\cup_i S_i),$$

for a minimization problem. The maximization version is similarly defined.

A *partitioned composable coreset* is a composable coreset in which the initial sets are a partitioning, i.e. sets $\{S_i\}_{i=1}^{L}$ are disjoint. Using Gonzalez's algorithm for k-center [14], Indyk, et al. designed a composable coreset for a similar problem known as the diversity maximization problem [20]. Other variations of composable coresets are randomized composable coresets and mapping coresets. Randomized composable coresets [26] share the same divide and conquer approach as other composable coresets and differ from composable coresets only in the way they partition the data. More specifically, randomized composable coresets, randomly partitioning the input, as opposed to other composable coresets which make use of arbitrary partitioning. Mapping coresets [5] extend composable coresets by adding a mapping between coreset points and other points to their coresets and keep almost the same amount of data in all machines. Algorithms for clustering in ℓ^p norms using mapping coresets are known [5]. Further improvements of composable coresets for diversity maximization [20] include lower bounds [3] and multi-round composable coresets in metrics with bounded doubling dimension [6].

Metric k-center is a NP-hard problem for which 2-approximation algorithms that match the lower bound for the approximation factor of this problem are

*Department of Computer Engineering, Sharif University of Technology, aghamolaei@ce.sharif.edu

†Department of Computer Engineering, Sharif University of Technology, School of Computer Science, Institute for Research in Fundamental Sciences (IPM), ghodsi@sharif.edu

known [28, 14]. Among approximation algorithms for k-center is a parametric pruning algorithm, based on the minimum dominating set [28]. In this algorithm, an approximate dominating set is computed on the disk graph of the input points. The running time of the algorithm is $O(n^3)$. The greedy algorithm for k-center requires only $O(nk)$ time [14] and unlike the algorithm based on the minimum dominating set[28], uses r-nets [17]. A $(1+\epsilon)$-approximation coreset exists for k-center [4] with size exponentially dependent on $\frac{1}{\epsilon}$.

Let the optimal radius of k-center for a point set P be r. The problem of finding the smallest set of points that cover P using radius r is known as the *dual clustering problem* [7].

Metric dual clustering (of k-center) has an unbounded approximation factor [7]. In Euclidean metric, there exists a streaming $O(2^d d \log d)$-approximation algorithm for this problem [7]. Also, any α-approximation algorithm for the minimum disk/ball cover problem gives a 2-approximation coreset of size αk for k-center, so 2-approximation coresets of size $(1+\epsilon)k$ exist for this problem [23]. A greedy algorithm for dual clustering of k-center has also been used as a preprocessing step of density-based clustering (DBSCAN) [11]. Implementing DBSCAN efficiently in MapReduce is an important problem [18, 9, 13, 27, 21].

Randomized algorithms for metric k-center and k-median in MapReduce [10] exist. These algorithms take α-approximation offline algorithms and return $(4\alpha+2)$-approximation and $(10\alpha+3)$-approximation algorithms for k-center and k-median in MapReduce, respectively. The round complexity of these algorithms depends on the probability of the algorithm for finding a good approximation.

Current best results on metric k-center in MapReduce have 2 rounds and give the approximation factor 4 [24]. However, a 2-approximation algorithm exists if the cost of the optimal solution is known [19]. Experiments in [25] suggest that running Gonzalez's algorithm on a random partitioning and an arbitrary partitioning results in the same approximation factor.

Warm-Up

Increasing the size of coresets in the first step of computing composable coresets can improve the approximation factor of some problems. The approximation factor of k-median algorithm of [15] is $2c(1+2b)+2b$, where b and c are the approximation factors of k-median and weighted k-median, respectively. This algorithm computes a composable coreset, where a coreset for k-median is the set of k medians weighted by the number of points assigned to each median.

A pseudo-approximation for k-median finds $k+O(1)$ median and has approximation factor $1+\sqrt{3}+\epsilon$ [22]. Using a pseudo-approximation algorithm in place of k-median algorithms in the first step of [15], it is possible to achieve a better approximation factor for k-median using the same proof as [15]. Since any pseudo-approximation has a cost less than or equal to the optimal solution; replacing them will not increase the cost of clustering.

The approximation factor using [8] as weighted k-median coresets is 91.66, while the best k-median algorithm would give a 99.33 factor using the same algorithm ($b = 1+\sqrt{3}$). The lower bound on the approximation factor of this algorithm using the same weighted k-median algorithm but without pseudo-approximation is 63.09 ($b = 1+\frac{2}{e}$).

Contributions

We give a $(1+\epsilon)$-approximation coreset of size $(\frac{4}{\epsilon})^{1+2b}k$ for k-center in metric spaces with doubling dimension b. Using composable coresets, our algorithm generalizes to MapReduce setting, where it becomes a $(1+\epsilon)$-approximation coreset of size $(\frac{4}{\epsilon})^{1+2b}\frac{n}{m}k$, given memory m, which is sublinear in the input size n.

Conditions	Approx.	Reference
Metric k-center:		
$O(1)$-rounds	4	[24]
$O(\log^{\Delta}_{1+\epsilon})$ rounds	$2+\epsilon$	[19]
Lower bound	2	offline [28]
Doubling metrics:		
$O(1)$-rounds	$2+\epsilon$	Theorem 7
Lower bound	1.822	[12]
Dual clustering:		
General metrics	$O(\log n)$	min dominating set [28], composable coreset [20]
Doubling metrics	$O(1)$	Theorem 3

Table 1: Summary of results for k-center and dual clustering in MapReduce. Δ is the diameter of the point-set.

Using the composable coreset for dual clustering, we find a $(2+\epsilon)$-approximation composable coreset for k-center, which has a sublinear size in metric spaces with constant doubling dimension. More specifically, if an α-approximation exists for doubling metrics, our algorithm provides $(\alpha+\epsilon)$-approximation factor. It improves the previous 4-approximation algorithm [24, 25] in MapReduce. A summary of results on k-center is shown in Table 1. Note that for MapReduce model, each round can take a polynomial amount of time, however, the space available to each machine is sublinear.

Our algorithm achieves a trade-off between the approximation factor and the size of coreset (see fig. 1). The approximation factor of our algorithm and the size of the resulting composable coreset for L input sets are

$\alpha = 2 + \epsilon$ and $kL\beta$, respectively. This trade-off is the main idea of our algorithm.

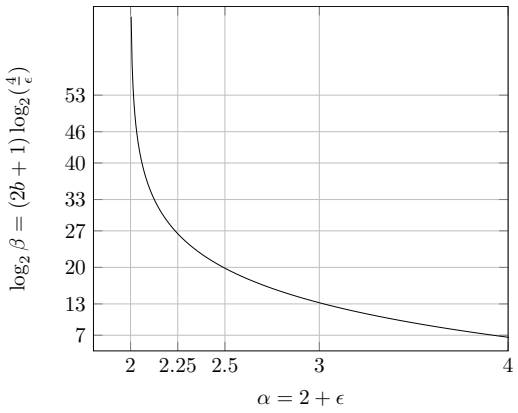


Figure 1: Space-approximation factor trade-off of our α-approx. coreset of size βkL for k-center in Euclidean plane.

Our composable coresets give single pass streaming algorithms and 1-round approximation algorithms in MapReduce with sublinear communication, since each coreset is communicated once, and the size of the coreset is constant.

2 Preliminaries

First we review some basic definitions, models and algorithms in computational geometry and MapReduce.

2.1 Definitions

Some geometric definitions and notations are reviewed here, which have been used in the rest of the paper.

Definition 2 (Metric Space) *A (possibly infinite) set of points P and a distance function $d(.,.)$ create a metric space if the following three conditions hold:*

- $\forall p, q \in P \quad d(p,q) = 0 \Leftrightarrow p = q$
- $\forall p, q \in P \quad d(p,q) = d(q,p)$
- $\forall p, q, t \in P \quad d(p,q) + d(q,t) \geq d(p,t)$, *known as triangle inequality*

Metrics with bounded doubling dimension are called *doubling metrics*. Constant dimension Euclidean spaces under ℓ^p norms and Manhattan distance are examples of doubling metrics.

Doubling constant [16] of a metric space is the number of balls of radius r that lie inside a ball of radius $2r$. The logarithm of doubling constant in base 2 is called *doubling dimension*. Many algorithms have better approximation factors in doubling metrics compared to general metric spaces. The doubling dimension of Euclidean plane is $\log_2 7$.

Definition 3 (Doubling Dimension [16]) *For any point x in a metric space and any $r \geq 0$, if the ball of radius $2r$ centered at x can be covered with at most 2^b balls of radius r, we say the doubling dimension of the metric space is b.*

k-Center is a NP-hard clustering problem with clusters in shapes of d-dimensional balls.

Definition 4 (Metric k-Center [28]) *Given a set P of points in a metric space, find a subset of k points as cluster centers C such that*

$$\forall p \in P, \min_{c \in C} d(p,c) \leq r$$

and r is minimized.

The best possible approximation factor of metric k-center is 2 [28].

Geometric intersection graphs represent intersections between a set of shapes. For a set of disks, their intersection graph is called a disk graph.

Definition 5 (Disk Graph) *For a set of points P in a metric space with distance function $d(.,.)$ and a radius r, the disk graph of P is a graph whose vertices are P, and whose edges connect points with distance at most $2r$.*

Definition 6 (Dominating Set) *Given a graph $G = (V, E)$, the smallest subset $Q \subset V$ is a minimum dominating set, if $\forall v \in V, v \in Q \vee \exists u \in Q : (v,u) \in E$.*

We define the following problem as a generalization of the dual clustering of [7] by removing the following two conditions: the radius of balls is 1, and the set of points are in $\mathbf{R}^d$.

Definition 7 (Dual Clustering) *Given a set of points P and a radius r, the dual clustering problem finds the smallest subset of points as centers $(C), C \subset P$ such that the distance from each point to its closest center is at most r.*

2.2 An Approximation Algorithm for Metric k-Center

Here, we review the parametric pruning algorithm of [28] for metric k-center.

Algorithm 1 Parametric Pruning for k-Center [28]

Input: A metric graph $G = (V, E)$, an integer k
Output: A subset $C \subset V, |C| \leq k$
  Sort E such that $e_1 \leq e_2 \leq \cdots \leq e_{|E|}$.
  $G' = (V, E') \leftarrow (V, \emptyset)$
  for $i = 1, \ldots, |E|$ **do**
    $E' \leftarrow E' \cup \{e_i\}$
    Run algorithm 2 on G'.
    if $|IS| \leq k$ **then return** IS

Using this algorithm on a metric graph G, a 2-approximation for the optimal radius r can be determined. In algorithm 1, edges are added by increasing order of their length until reaching r. Given this radius, another graph (G') is built, where edges exist between points within distance at most r of each other.

Algorithm 2 Approximate dominating set of G [28]

Input: A metric graph $G' = (V, E)$
Output: A subset $C \subset V$
  $G'^2 \leftarrow G'$
  for $\forall (u,t),(t,v) \in E$ **do**
    Add (u, v) to G'^2.
  Find a maximal independent set IS of G'^2
  return IS

Hence, by definition, a minimum dominating set of G' is an optimal k-center of G. Every cluster is a star in G' which turns into a clique in G'^2. Therefore, a maximal independent set of G'^2 chooses at most one point from each cluster. Algorithm 2 computes G'^2 and returns a maximal independent set of G'^2.

Computing a maximal independent set takes $O(|E|)$ time. The graph G'^2 in Algorithm 2 only changes in each iteration of Algorithm 1 around the newly added edge, so, updating the previous graph and IS takes $O(n)$ time. Therefore, the time complexity of Algorithm 1 is $O(|E| \cdot n) = O(n^3)$.

3 A Coreset for Dual Clustering in Doubling Metrics

In this section, we prove a better approximation offline coreset for the dual clustering problem. Our method is based on Algorithm 1 which first builds the disk graph with radius r, then covers this graph using a set of stars. We prove the maximum degree of those stars is D^2, where D is the doubling constant. The result is an approximation algorithm for dual clustering in doubling metrics.

3.1 Algorithm

We add a preprocessing step to Algorithm 1 to find a better approximation factor for k-center and dual clustering problems.

Algorithm 3 A Coreset for k-Center

Input: A set of points P, an integer k or a radius r
Output: A subset $C \subset P, |C| \leq k$
  if k is given in the input **then**
    Compute a 2-approximation solution for k-center (radius r).
  $E \leftarrow$ all pairs of points with distance at most $r/2$.
  Run algorithm 2 on $G = (P, E)$ to compute IS.
  return IS

3.2 Analysis

Unlike in general metric spaces, k-center in doubling metrics admits a space-approximation factor trade-off. More specifically, doubling or halving the radius of k-center changes the number of points in the coreset by a constant factor, since the degrees of vertices in the minimum dominating set are bounded in those metric spaces.

Lemma 1 *For each cluster C_i of Algorithm 3 with radius r', the maximum number of points $(\Delta + 1)$ from C_i that are required to cover all points inside C_i with radius $r'/2$ is at most D^2, i.e.*

$$(\Delta + 1) \leq D^2,$$

where D is the doubling constant of the metric space.

Proof. Assume a point $p \in IS$ returned by Algorithm 3. By the definition of doubling metrics, there are D balls of radius $r'/2$ centered at $b_1, \ldots, b_D$ called $B_1, \ldots, B_D$ that cover the ball of radius r' centered at p, called B.

$$\forall q \in B, \exists B_i, i = 1, \ldots, D : d(p, b_i) \leq r'/2$$

Repeating this process for each ball B_i results in a set of at most D balls $(B'_{i,1}, \ldots, B'_{i,D})$ of radius $r'/4$ centered at $b'_{i,1}, \ldots, b'_{i,D}$.

$$\forall q \in B'_{i,j}, d(b'_{i,j}, q) \leq r'/4$$

Choose a point $p_{i,j} \in P \cap B'_{i,j}$. Using triangle inequality,

$$\begin{aligned} \forall q \in B'_{i,j}, d(p_{i,j}, q) &\leq d(p_{i,j}, b'_{i,j}) + d(b'_{i,j}, q) \\ &\leq r'/4 + r'/4 = r'/2. \end{aligned}$$

We claim any minimal solution needs at most one point from each ball $B'_{i,j}$. By contradiction, assume there are two point $p_{i,j}, q'$ in the minimal solution that lie inside a ball $B'_{i,j}$. After removing q', the ball with radius $r'/2$ centered at $p_{i,j}$ still covers $B'_{i,j}$, since:

$$\begin{aligned} \forall q \in P, \exists B_i, B'_{i,j} \ni q, p_{i,j} \\ d(q, p_{i,j}) &\leq d(q, b'_{i,j}) + d(b'_{i,j}, p_{i,j}) \\ &\leq r'/4 + r'/4 = r/2'. \end{aligned}$$

Then we have found a point (q') whose removal decreases the size of the solution, which means the solution was not minimal. So the size of any minimal set of points covering B is at most D^2. □

Lemma 2 *In a metric space with doubling constant D, if a dual clustering with radius r has k points, then a dual clustering with radius $r/2$ exists which has D^2k points.*

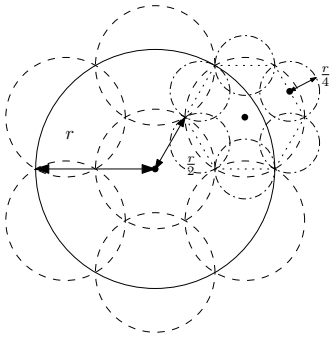


Figure 2: Applying the doubling dimension bound twice (Lemma 1).

Proof. Let p be a center in the k-center problem. Based on the proof of Lemma 1, there are Δ vertices adjacent to p that cover the points inside the ball of radius r centered at p, using balls of radius $r/2$ and a ball of radius $r/2$ centered at p. By choosing all these vertices as centers, it is possible to cover all input points P with radius $r/2$. Using the same reasoning for all clusters, it is possible to cover all points using $(\Delta+1)k$ centers. Using the bound in Lemma 1, these are D^2k centers. □

Theorem 3 *The approximation factor of Algorithm 3 is D^2 for the dual clustering.*

Proof. Since the radius of balls in Lemma 2 is at most the optimal radius for k-center, the approximation factor of dual clustering is the number of points chosen as centers divided by k, which is D^2. □

Theorem 4 *The approximation factor of the coreset for k-center in Algorithm 3 is 2^{-R} and its size is $D^{2(R+1)}k$.*

Proof. Applying Lemma 2 halves the radius and multiplies the number of points by D^2. So, applying this lemma R times gives $(D^2)^{R+1}k$ points, since it might be the case that in the first step of the algorithm the optimal radius was found, and we divided it by 2. The radius remains $\frac{r}{2^R}$ because of the case where we had found a 2-approximation. □

Theorem 5 *Algorithm 3 given $(\frac{4}{\epsilon})^{2\log_2 D}k$ as input, is a $(1+\epsilon)$-approximation coreset of size $(\frac{4}{\epsilon})^{2\log_2 D}k$ for the k-center problem.*

Proof. For $R = \lceil \log_2 \frac{2}{\epsilon} \rceil$, the proof of Theorem 4 gives $(\frac{4}{\epsilon})^{2\log_2 D}$ points and radius $r\epsilon$. Assume O is the set of k centers returned by the optimal algorithm for point-set P, and C is the set of centers returned by running the optimal algorithm on the coreset of P. For any point $p \in P$, let o be the center that covers p and c be the point that represents o in the coreset. Using triangle inequality:

$$d(p,c) \le d(p,o) + d(o,c) \le r + r\epsilon = (1+\epsilon)r$$

So, computing a k-center on this coreset gives a $(1+\epsilon)$-approximation. □

4 A Composable Core-Set for k-Center in Doubling Metrics

Our general algorithm for constructing coresets based on dual clustering has the following steps:

- Compute the cost of an approximate solution (X).
- Find a composable coreset for dual clustering with cost X.
- Compute a clustering on the coreset.

In this section, we use this general algorithm for solving k-center.

4.1 Algorithm

Knowing the exact or approximate value of r, we can find a single-round $(2+\epsilon)$-approximation for metric k-center in MapReduce. Although the algorithm achieves the aforementioned approximation factor, the size of the coreset and the communication complexity of the algorithm depend highly on the doubling dimension.

Algorithm 4 k-Center

Input: A set of sets of points $\cup_{i=1}^{L} S_i$, a k-center algorithm

Output: A set of k centers

1: Run a k-center algorithm on each S_i to find the radius r_i.
2: Run Algorithm 2 on the disk graph of each set S_i with radius $\frac{\epsilon r_i}{2}$ locally to find $C(S_i)$.
3: Send $C(S_i)$ to set 1 to find the union $\cup_i C(S_i)$.
4: Run a 2-approximation k-center algorithm on $\cup_{i=1}^{L} C(S_i)$ to find the set of centers C.
5: **return** C.

Based on the running time of Algorithm 2 and Gonzalez's algorithm, the running time of Algorithm 4 is $\sum_i [O(k \cdot |S_i|) + O(|S_i|^2)] + O(k \sum_i |C(S_i)|) = O(kn)$. Since the sum of running times of machines is of the same order as the best sequential algorithm, Algorithm 4 is a work-efficient parallel algorithm.

We review the following well-known lemma:

Lemma 6 *For a subset $S \subset P$, the optimal radius of the k-center of S is at most twice the radius of the k-center of P.*

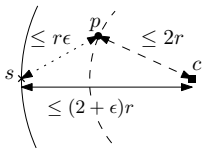


Figure 3: The dominating set on $\cup_i C(S_i)$ covers $\cup_i S_i$ with radius $(2+\epsilon)$(Theorem 7).

Proof. Consider the set of clusters O_i in the optimal k-center of P centered at $c_i, i = 1, \ldots, k$ with radius r. If $c_i \in S$, then the points of $O_i \cap S$ are covered by c_i with radius r, as before. Otherwise, select an arbitrary point in $O_i \cap S$ as the new center c_i'. Using the triangle inequality on c_i, c_i' and any point $p \in O_i \cap S$:

$$d(p, c_i') \leq d(p, c_i) + d(c_i, c_i') \leq r + r = 2r$$

Since c_i' was covered using c_i with radius r. So the set $S \cap O_i$ can be covered with radius $2r$. Note that since we choose at most one point from each set, the number of new centers is at most k. □

Theorem 7 *The approximation factor of Algorithm 4 is $2+\epsilon$ for metric k-center.*

Proof. Let r be the optimal radius of k-center for $\cup_i S_i$. Since $\cup_i C(S_i) \subset \cup_i S_i$, using Lemma 6, the radius of k-center for $\cup_i C(S_i)$ is at most $2r$. The radius of k-center inside each set S_i is at most $2r$ for the same reason. The algorithm computes a covering S_i with balls of radius $r_i\epsilon/2$. Based on the fact that offline k-center has 2-approximation algorithms and the triangle inequality, the approximation factor of the algorithm proves to be $(2+\epsilon)$-approximation (Figure 3). Let $p = \arg\min_{p \in \cup_i C(S_i)} dist(s, p)$, then

$$\begin{aligned} \forall s \in S_i \exists c \in C, d(s, c) &\leq d(s, p) + d(p, c) \leq r' + r_i\epsilon/2 \\ &\leq 2r + 2r\epsilon/2 = (2+\epsilon)r \end{aligned}$$

where r' is the radius of the offline k-center algorithm on C. □

4.2 Analysis

Lemma 8 *In a metric space with doubling constant D, the union of dual clusterings of radius r computed on sets $S_1, \ldots, S_L$ is a $(L \times D^{2\log_2 \frac{8}{\epsilon}})$-approximation for the dual clustering of radius $r(1+\epsilon)$ of their union $(\cup_{i=1}^{L} S_i)$.*

Proof. Each center in the dual clustering with radius r of $P = (\cup_{i=1}^{L} S_i)$ has at most Δ adjacent vertices covered by this center. Consider a point $p \in P$ covered by center c in a solution for P. If p and c belong to the same set S_i, assign p to c. Otherwise, pick any point that was previously covered by c as the center that covers p.

While this might increase the radius by a factor 2, it does not increase the number of centers in each set. Since the algorithm uses radius $\epsilon.r/2$, it increases the number of centers to $D^{2\log_2 \frac{8}{\epsilon}} k$ (based on Theorem 4 for $R = \frac{4r}{\epsilon r/2}$) but keeps the approximation factor of the radius to $1+\epsilon$. There are L such sets, so the size of the coreset is $L \times D^{2\log_2 \frac{8}{\epsilon}} k$. □

Theorem 9 *Algorithm 4 returns a coreset of size $O(kL)$ for k-center in metric spaces with fixed doubling dimension.*

Proof. The coreset of each set S_i has a radius r_i varying from the optimal radius ($r = r_i$) to $2\beta.r$, where β is the approximation factor of the offline algorithm for k-center. Clearly, the lower bound holds because any radius is at least as much as the optimal (minimum) radius, which means $r \leq r_i$; and Lemma 6 when applied to $S_i \subset \cup_i S_i$, yields the upper bound.

$$r \leq r_i \leq 2\beta.r \Rightarrow \frac{r\epsilon}{4\beta} \leq \frac{r_i\epsilon}{4\beta} \leq \frac{\epsilon r}{2}$$

Reaching value $r\epsilon$ requires applying Theorem 7 at most $\log_2 \frac{4\beta}{\epsilon}$ times.

The size of the resulting coreset is therefore at most

$$(4^{\log_2 D})^{\log_2 \frac{4\beta}{\epsilon}} kL = (\frac{4\beta}{\epsilon})^{2(\log_2 D)} kL.$$

Here, we use the best approximation factor for metric k-center ($\beta = 2$), which gives a coreset of size $(\frac{8}{\epsilon})^{2(\log_2 D)} kL = O(kL)$ for fixed ϵ. □

5 Conclusions

We proved a trade-off between the approximation factor and the number of centers for the k-center problem in doubling metrics. To improve the trade-off in MapReduce, local partitioning methods such as grid-based or locality sensitive hashing, or degree based partitioning of disk graph with lower radius might be effective.

Gonzalez's algorithm [14] is a version of parametric pruning algorithm [28] in which the greedy maximal independent set computation prioritizes the points with maximum distance from the currently chosen points. Our algorithm and trade-off partially answers the open question of [25] about comparing and improving these two algorithms in MapReduce.

Our composable coreset for dual clustering gives constant factor approximation for minimizing the size of DBSCAN cluster representatives if half the input radius is used, and the dominating set subroutine is replaced with the connected dominating set.

References

[1] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi. Mergeable summaries. *ACM Transactions on Database Systems (TODS)*, 38(4):26, 2013.

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

[3] S. Aghamolaei, M. Farhadi, and H. Zarrabi-Zadeh. Diversity maximization via composable coresets. In *Canadian Conference on Computational Geometry (CCCG)*, 2015.

[4] M. Bādoiu, S. Har-Peled, and P. Indyk. Approximate clustering via core-sets. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 250–257. ACM, 2002.

[5] M. Bateni, A. Bhaskara, S. Lattanzi, and V. Mirrokni. Distributed balanced clustering via mapping coresets. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2591–2599, 2014.

[6] M. Ceccarello, A. Pietracaprina, G. Pucci, and E. Upfal. Mapreduce and streaming algorithms for diversity maximization in metric spaces of bounded doubling dimension. *Proceedings of the VLDB Endowment*, 10(5):469–480, 2017.

[7] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *SIAM Journal on Computing*, 33(6):1417–1440, 2004.

[8] M. Charikar, S. Guha, E. Tardos, and D. B. Shmoys. A constant-factor approximation algorithm for the k-median problem (extended abstract). In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99, pages 1–10, New York, NY, USA, 1999. ACM.

[9] B.-R. Dai and I.-C. Lin. Efficient map/reduce-based dbscan algorithm with optimized data partition. In *2012 IEEE 5th International Conference on Cloud Computing (CLOUD)*, pages 59–66. IEEE, 2012.

[10] A. Ene, S. Im, and B. Moseley. Fast clustering using mapreduce. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD)*, pages 681–689. ACM, 2011.

[11] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD)*, volume 96, pages 226–231, 1996.

[12] T. Feder and D. Greene. Optimal algorithms for approximate clustering. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 434–444. ACM, 1988.

[13] Y. X. Fu, W. Z. Zhao, and H. F. Ma. Research on parallel dbscan algorithm design based on mapreduce. In *Advanced Materials Research*, volume 301, pages 1133–1138. Trans Tech Publ, 2011.

[14] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science (TCS)*, 38:293–306, 1985.

[15] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(3):515–528, 2003.

[16] A. Gupta, R. Krauthgamer, and J. R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *Foundations of Computer Science, 2003. Proceedings. 44th Annual IEEE Symposium on*, pages 534–543. IEEE, 2003.

[17] S. Har-Peled and M. Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM Journal on Computing*, 35(5):1148–1184, 2006.

[18] Y. He, H. Tan, W. Luo, S. Feng, and J. Fan. Mr-dbscan: a scalable mapreduce-based dbscan algorithm for heavily skewed data. *Frontiers of Computer Science*, 8(1):83–99, 2014.

[19] S. Im and B. Moseley. Brief announcement: Fast and better distributed mapreduce algorithms for k-center clustering. In *Proceedings of the 27th ACM symposium on Parallelism in Algorithms and Architectures*, pages 65–67. ACM, 2015.

[20] P. Indyk, S. Mahabadi, M. Mahdian, and V. S. Mirrokni. Composable core-sets for diversity and coverage maximization. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS)*, pages 100–108. ACM, 2014.

[21] Y. Kim, K. Shim, M.-S. Kim, and J. S. Lee. Dbcure-mr: an efficient density-based clustering algorithm for large data using mapreduce. *Information Systems*, 42:15–35, 2014.

[22] S. Li and O. Svensson. Approximating k-median via pseudo-approximation. *SIAM Journal on Computing*, 45(2):530–547, 2016.

[23] C. Liao and S. Hu. Polynomial time approximation schemes for minimum disk cover problems. *Journal of combinatorial optimization*, 20(4):399–412, 2010.

[24] G. Malkomes, M. J. Kusner, W. Chen, K. Q. Weinberger, and B. Moseley. Fast distributed k-center clustering with outliers on massive data. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1063–1071, 2015.

[25] J. McClintock and A. Wirth. Efficient parallel algorithms for k-center clustering. In *Parallel Processing (ICPP), 2016 45th International Conference on*, pages 133–138. IEEE, 2016.

[26] V. Mirrokni and M. Zadimoghaddam. Randomized composable core-sets for distributed submodular maximization. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing (STOC)*, pages 153–162. ACM, 2015.

[27] M. Noticewala and D. Vaghela. Mr-idbscan: Efficient parallel incremental dbscan algorithm using mapreduce. *International Journal of Computer Applications (IJCA)*, 93(4), 2014.

[28] V. V. Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.

Approximation Schemes for Covering and Packing in the Streaming Model

Christopher Liaw* Paul Liu† Robert Reiss‡

Abstract

The *shifting strategy*, introduced by Hochbaum and Maass [10], and independently by Baker [1], is a unified framework for devising polynomial approximation schemes to NP-Hard problems. This strategy has been used to great success within the computational geometry community in a plethora of different applications; most notably covering, packing, and clustering problems [2, 5, 7, 8, 9]. In this paper, we revisit the shifting strategy in the context of the *streaming* model and develop a streaming-friendly shifting strategy. When combined with the *shifting coreset* method introduced by Fonseca et al. [6], we obtain streaming algorithms for various graph properties of unit disc graphs. As a further application, we present the first approximation algorithms and lower bounds for the unit disc cover (UDC) problem in the streaming model.

1 Introduction

The *shifting strategy* is a unified framework for devising polynomial-time approximation schemes (PTASes) to NP-Hard problems. Originally used by Baker [1] for maximum independent set in planar graphs, the shifting strategy was modified to solve several geometric covering problems in the widely-cited paper of Hochbaum and Maass [10]. Since then, this strategy has found applications in an incredibly diverse set of domains; including facility location, motion planning, image processing, and VLSI design.

For geometric problems, the shifting strategy is based on partitioning the possible input space into disjoint regions (or windows), solving each disjoint region (either exactly or approximately), and then joining the partial solutions from each window into a candidate global solution. By choosing several partitions, and minimizing over the candidate solutions from each one, a good approximation to the problem is formed. The main observation of the shifting strategy is that the analysis of the approximation factor can be done in two independent portions; the error accumulated from dividing the space into windows, and the error from the within-window algorithm. The within-window algorithm is typically easier to design; in many problems the optimal solution size within a window is bounded by a small constant. Thus by specifying good within-window algorithms, the algorithm designer can get a global solution with only a small overhead in complexity. One of the original problems addressed by Hochbaum and Maass is the *unit disc cover* (UDC) problem: given a point set P in the plane, the problem asks for the size of the smallest set of radius r (or equivalently, unit) discs that cover P completely.[1] In this case, the partition of the input space is a tiling of the plane by identical $\ell \times \ell$ squares. Within each square, the optimal UDC is found by brute force, as the solution size is at most $\mathcal{O}(\ell^2)$. By iterating over translates (or shifts) of this tiling, Hochbaum and Maass obtain a $\left(1+\frac{1}{\ell}\right)^2$-approximation with running time $n^{\mathcal{O}(\ell^2)}$ for UDC in 2D.

Recently, there has been renewed interest in making shifting strategy algorithms practical, as the PTASes obtained by the shifting strategy are too slow to be applied in practice. In recent work by Fonseca et al. [6], the technique of *shifting coresets* is introduced, giving linear time approximations for various problems on unit disc graphs. They observe that within-window algorithms used in the shifting technique often iterate over $m^{\text{poly}(\ell)}$ candidate solutions, where m is the number of points inside the window and ℓ is the size of the window. By using coresets to approximate and sparsify point set inside the window, they mitigate the high memory and computational cost of the within-window algorithm. Their algorithms are no longer PTASes, but run in linear time and produce constant factor approximations.

Although the shifting strategy is widely used, scarce attention has been given to it in the *streaming* model. In the streaming model, the complexity of an algorithm is measured mainly by the number of passes it makes over the input data, and the amount of memory used over the duration of the algorithm. In common settings, the requirements are that the algorithm makes only one pass over the input data and uses sublinear (usually polylogarithmic) memory in the size of the input. This is difficult in the context of the shifting strategy as partitioning the input often requires the practitioner to

*Department of Computer Science, University of British Columbia, Vancouver, Canada. cvliaw@cs.ubc.ca

†Department of Computer Science, Stanford University, Stanford, USA. paul.liu@stanford.edu

‡Department of Computer Science, University of British Columbia, Vancouver, Canada. rreiss@cs.ubc.ca

[1] Actually, Hochbaum and Maass consider the problem of *finding* the smallest of discs that cover a set of points. Our problem is slightly different in that we only care about the *size* of such a cover.

keep a mapping of input points to windows within the partition, necessitating at least linear space.

In this paper, we revisit the shifting strategy in the context of the streaming model, and develop a streaming-friendly variant. Our streaming shifting strategy only relies on the algorithm designer to design a within-window streaming algorithm $\mathcal{A}$. Provided that the optimal solution within each window is bounded, the streaming shifting strategy then gives a global algorithm that only introduces a polylogarithmic overhead to the memory use of $\mathcal{A}$, with the same number of passes over the input data. The analysis is inspired by a recent algorithm of Cabello and Pérez-Lantero [4], who presents a $(3/2+\varepsilon)$ approximation for cardinality estimation of maximum independent sets (MIS) of interval graphs in $\mathcal{O}(\text{poly}(1/\varepsilon)\log n)$ memory with only one pass over the input data.

When the memory use of a within-window algorithm for a problem is small (i.e. polylogarithmic), our streaming shifting strategy gives a streaming algorithm for solving the problem globally. Due to this, our results are complementary to those given by Fonseca et al. [6], where $\mathcal{O}(1)$ memory within-window algorithms are developed for various problems on unit disc graphs. In particular, when their results are combined with our shifting strategy, we obtain streaming algorithms with polylogarithmic memory for independent set, dominating set, and minimum vertex cover on unit disc graphs.

In Section 3, we describe and analyze our streaming shifting strategy. As an application, we present in Section 4 novel approximation algorithms for the UDC problem in the streaming model. Our UDC algorithms use $\mathcal{O}(\text{poly}(1/\epsilon)\log n)$ memory, and operate in only one pass over the input data. We remark that the results of Cabello and Pérez-Lantero imply a $(3/2+\epsilon)$ approximation for the 1D UDC problem. This is due to the fact that for unit disc (i.e. unit interval) graphs in 1D, the cardinality of a maximum independent set is equal to the cardinality of a minimum disc cover. However, to the best of our knowledge, UDC has not been considered in the streaming model for 2D and above. In Section 5, we show that any one pass streaming algorithm for 2D UDC in L_2 must have approximation factor at least 2.

2 Preliminaries

We use the standard notation $[r] = \{1, \ldots, r\}$ where $r \in \mathbb{N}$. For positive numbers y, ε, δ, we use the notation $x = y(1 \pm \varepsilon) \pm \delta$ to mean $x \in [y(1-\varepsilon) - \delta, y(1+\varepsilon) + \delta]$. For simplicity, we make the minor assumption that the coordinates of the input points are bounded above by $\text{poly}(n)$ and can be represented using $\mathcal{O}(\log n)$ bits where n is the number of points.

2.1 ε-min-wise hashing

One of the key primitives in our algorithms is the ability to (approximately) sample an element from a set. To do this, we will use ε-min-wise hash functions which were introduced by Broder et al. [3]. We remark that a similar idea was also used in [4].

Let $U = V = \{0, 1, \ldots, k-1\}$ and $\mathcal{H}$ be a collection of functions $h \colon U \to V$. We will assume that k is a prime power.

Definition 1 *A family of hash functions $\mathcal{H}$ is said to be r-wise independent if for any distinct $x_1, \ldots, x_r \in U$ and any $y_1, \ldots, y_r \in V$ we have*

$$\Pr_{h \in \mathcal{H}}[h(x_1) = y_1 \wedge \ldots \wedge h(x_r) = y_r] = \frac{1}{k^r}.$$

Here, we use $\Pr_{h\in\mathcal{H}}$ to denote the probability measure where each h is drawn uniformly at random from $\mathcal{H}$. It is well-known that an r-wise independent hash family can be constructed as follows (see [14]). Let $\mathbb{F}$ be a finite field of size k (such a field exists because k is assumed to be a prime power). Let $\mathcal{H} = \{h_{a_0,a_1,\ldots,a_{r-1}} : a_i \in \mathbb{F}\}$ where $h_{a_0,a_1,\ldots,a_{r-1}}(x) = a_{r-1}x^{r-1} + \ldots + a_0$. Then $\mathcal{H}$ is an ℓ-wise independent hash family. Moreover, any element in $\mathcal{H}$ can be represented using $\mathcal{O}(r \log k)$ bits.

Definition 2 *A family of hash functions $\mathcal{H}$ is said to be (ε, s)-min-wise independent if for any $X \subseteq [k]$ with $|X| \leq s$ and $x \in X$ we have*

$$\Pr_{h \in \mathcal{H}}\left[h(x) < \min_{y \in X \setminus \{x\}} h(y)\right] = \frac{1 \pm \varepsilon}{|X|}.$$

There is a simple way to obtain (ε, s)-min-wise independent hash functions due to Indyk [11].

Theorem 1 *There are fixed constants $c, c' > 1$ such that the following holds. Let $\varepsilon > 0$ and $s \leq \varepsilon k/c$. Then any $c' \log(1/\varepsilon)$-wise independent hash family $\mathcal{H}$ is (s, ε)-min-wise independent.*

For our applications, we will have $s = n$ and $k = \max(n/\varepsilon, \text{poly}(n)^d)$. In particular, the hash functions can be represented using $\mathcal{O}(\log^2(1/\varepsilon) + d\log(1/\varepsilon)\log(n))$ bits. If $\varepsilon^{-1} \leq n$ then this quantity is $\mathcal{O}(d\log(1/\varepsilon)\log(n))$.

3 Shifting lemma

We begin by reviewing the shifting strategy of Hochbaum and Maass [10] using the UDC problem in $\mathbb{R}^d$ as an example. For simplicity, we describe the shifting strategy in the planar case $d = 2$. In the shifting strategy, we partition the plane into windows of size $2\ell \times 2\ell$ where ℓ is the "shifting parameter".[2] The windows are closed

[2] Hochbaum and Maass [10] actually partition the plane into strip of width ℓ but small variants, such as replacing strips with windows, also work for identical reasons.

on the top and left while open on the right and bottom. We further impose that the coordinates of the top left boundary point are even integers. Due to these restrictions, there are exactly ℓ^2 different ways to partition $\mathbb{R}^2$. Let $S_1, \ldots, S_{\ell^2}$ the be the ℓ^2 different partitions of the plane.

Suppose that $\mathcal{A}$ is a within-window algorithm, i.e. it (approximately) solves the covering problem within a window of size $2\ell \times 2\ell$. Hochbaum and Maass [10] proposed the following algorithm to extend $\mathcal{A}$ to a "global algorithm" $\mathcal{A}_S$. For each partition S_i, we use $\mathcal{A}$ on each of windows to compute a disc cover. Then we take the union of the disc cover on each window to produce a global solution $\mathcal{D}_i$. Having computed ℓ^2 disc covers, we output the smallest cardinality disc cover of the $\mathcal{D}_i$. The following lemma states that the approximation ratio of this scheme is not much worse than the approximation ratio of the within-window algorithm. Hence, to design a global algorithm, one only needs to design a "local algorithm".

Lemma 2 (Shifting lemma [10])

$$r_{\mathcal{A}_S} \leq \left(1 + \frac{1}{\ell}\right)^2 r_{\mathcal{A}}.$$

where $r_{\mathcal{A}}, r_{\mathcal{A}_S}$ are the approximation ratios of $\mathcal{A}, \mathcal{A}_S$, respectively.

In general, let $\mathcal{A}$ be an algorithm that approximately solves the disc cover problem in $\mathbb{R}^d$ but restricted to "windows" of size $\underbrace{2\ell \times \ldots \times 2\ell}_{d \text{ times}}$. Define $\mathcal{A}_S$ to be the algorithm that partitions $\mathbb{R}^d$ into these windows, uses $\mathcal{A}$ on each window to find a cover, then takes the smallest cover over all partitions. Then we have $r_{\mathcal{A}_S} \leq \left(1 + \frac{1}{\ell}\right)^d r_{\mathcal{A}}$. This is particularly elegant since one can focus on obtaining an approximation algorithm assuming bounded input. Once such an algorithm is developed, it can then be extended to an algorithm on the whole space.

To improve the space complexity of some of our streaming algorithms, we can use the following randomized version of the shifting lemma which we prove in Appendix A. Let $\mathcal{A}_S$ be the algorithm which randomly picks one of the ℓ^d partitions of the $\mathbb{R}^d$ as defined above, say S_i, uses $\mathcal{A}$ to compute a disc cover on each window, then outputs the union as a global disc cover.

Lemma 3 *Suppose $\ell \geq 2d$. Then with probability at least $1/2$*

$$r_{\mathcal{A}_S} \leq \left(1 + \frac{4d}{\ell}\right) r_{\mathcal{A}} \leq \left(1 + \frac{4}{\ell}\right)^d r_{\mathcal{A}}$$

where $r_{\mathcal{A}}, r_{\mathcal{A}_S}$ are the approximation ratios of $\mathcal{A}, \mathcal{A}_S$, respectively.

3.1 The shifting lemma in the streaming setting

In this section, we describe the streaming shifting strategy. For concreteness, we focus on giving a streaming variant of Lemma 3. Let $\mathcal{A}$ be a streaming algorithm which approximately solves UDC restricted to a window of size $2\ell \times 2\ell$.

We begin with a high level description of how to use the shifting strategy in the streaming setting. For now, let us fix a partition of $\mathbb{R}^2$ into windows of size $2\ell \times 2\ell$. The first issue that arrives is that one is no longer allowed to run $\mathcal{A}$ on all windows as the space would be prohibitive. To get around this, we use the following trick from [4]. Set $T = 4\ell^2$. Let γ_t be the number of windows for which $\mathcal{A}$ outputs a disc cover of size at least t. Since there is a trivial cover of size T, we can assume that $\gamma_t = 0$ for $t > T$. Then the cover obtained by running $\mathcal{A}$ on all windows is exactly $\sum_{t=1}^{T} \gamma_t$. The first key observation is that γ_1 can be interpreted as the number of windows that contain at least one point. In the language of streaming algorithm, this is exactly the distinct elements problem and can be approximated in very little space.[3] The second key observation is that, if we are able to get a random sample of the windows that contain at least one point then we can get a very good estimate of the quantity $\eta_t \coloneqq \gamma_t/\gamma_1$. We can do this approximately using min-wise hashing.

We now commence with a more formal treatment of the above ideas. Again, let us fix a partitioning of $\mathbb{R}^2$ into windows of size $2\ell \times 2\ell$. First, we can use an algorithm due to Kane, Nelson, and Woodruff [12] for distinct elements to obtain the following result.

Lemma 4 *Using $\mathcal{O}(\varepsilon^{-2} + \log(n))$ bits of space, we can obtain an estimate $\hat{\gamma_1} = (1 \pm \varepsilon)\gamma_1$ with probability at least* 0.99.

Next, we use min-wise hashing to estimate η_t for $2 \leq t \leq T$. This is formalized in the next lemma, whose proof is given in Appendix B.

Lemma 5 *Let $\mathcal{A}$ be a streaming algorithm for the disc cover problem restricted to a window of size $2\ell \times 2\ell$. Suppose that $\mathcal{A}$ uses s bits of space and let $s_h = \mathcal{O}(\log(1/\varepsilon)\log(n))$. Then using $\mathcal{O}(\varepsilon^{-2}\ell^4 \log(\ell)(s + s_h))$ bits of space, we can obtain an estimate $\hat{\eta_t} = (1 \pm \varepsilon)\eta_t \pm \varepsilon/T$ for all $t \in \{2, \ldots, T\}$ with probability at least* 0.99.

We now prove our main theorem in this section.

Theorem 6 (Streaming shifting lemma) *Let $\mathcal{A}$ be a streaming algorithm for the disc cover problem restricted to a window of size $2\ell \times 2\ell$ with approximation ratio $r_{\mathcal{A}}$. Suppose that $\mathcal{A}$ uses s bits of space and let $s_h = \mathcal{O}(\log(1/\varepsilon)\log(n))$. Then there is a streaming algorithm for the disc cover problem with approximation*

[3] Given a stream $a_1, \ldots, a_m \in [n]$, the distinct elements problem is to estimate $|\{a_1, \ldots, a_m\}|$.

ratio $(1+\varepsilon)(1+4/\ell)^2 r_{\mathcal{A}}$ *that uses* $\mathcal{O}(\varepsilon^{-2}\ell^4 \log(\ell)(s+s_h))$ *bits of space and has success probability at least* 0.99.

Proof. Fix a partition of $\mathbb{R}^2$ into $2\ell \times 2\ell$ windows. By Lemma 4, with probability at least 0.99 we obtain an estimate $\hat{\gamma_1} = (1 \pm \varepsilon)\gamma_1$. By Lemma 5, with probability at least 0.99 we obtain an estimate $\hat{\eta_t} = (1 \pm \varepsilon)\eta_t \pm \varepsilon/T$. Hence,

$$\begin{aligned}\hat{\gamma_t} &= \hat{\eta_t}\hat{\gamma_1} \\ &= [(1 \pm \varepsilon)\eta_t \pm \varepsilon/T]\,(1 \pm \varepsilon)\gamma_1 \\ &= (1 \pm 3\varepsilon)\gamma_t \pm 2\varepsilon\gamma_1/T.\end{aligned}$$

So

$$\sum_{t=1}^{T} \hat{\gamma_t} = (1 \pm 3\varepsilon)\sum_{t=1}^{T} \gamma_t \pm 2\varepsilon\gamma_1 = (1 \pm 5\varepsilon)\sum_{t=1}^{T} \gamma_t.$$

If $\varepsilon < 1/10$ then $\sum_{t=1}^{T} \gamma_t \leq (1-5\varepsilon)^{-1}\sum_{t=1}^{T} \hat{\gamma_t} \leq (1 + 20\varepsilon)\sum_{t=1}^{T} \gamma_t$. Replacing ε with $\varepsilon/20$, we have a $(1+\varepsilon)$-approximation to the disc cover computed by running $\mathcal{A}$ on all windows in the partition.

Finally, by Lemma 3, using algorithm $\mathcal{A}$ on all windows gives a $(1+4/\ell)^2 r_{\mathcal{A}}$-approximation algorithm with success probability 0.48. This can be amplified to 0.99 by running $\mathcal{O}(1)$ copies of the algorithm in parallel and taking the median.

The space complexity comes from Lemma 4 and Lemma 5. □

We remark that our strategy is very general. In fact, a straightforward extension of our strategy yields the following general theorem for unit disc covers in $\mathbb{R}^d$.

Theorem 7 *Let* $\mathcal{A}$ *be a streaming algorithm for the disc cover problem restricted to a window of size* $2\ell \times \ldots \times 2\ell$ *with approximation ratio* $r_{\mathcal{A}}$. *Suppose that* $\mathcal{A}$ *uses* s *bits of space and let* $s_h = \mathcal{O}(d \log(1/\varepsilon)\log(n))$. *Then there is a streaming algorithm for the disc cover problem with approximation ratio* $(1+\varepsilon)(1+4/\ell)^d r_{\mathcal{A}}$ *that uses* $\mathcal{O}(\varepsilon^{-2}d^{2d+2}\ell^{2d}\log(\ell d)(s+s_h))$ *bits of space and has success probability at least* 0.99.

In addition, we do not need to restrict ourselves to single-pass streaming algorithms. Theorem 6 holds whether we consider single-pass streaming algorithms or multi-pass streaming algorithms; one simply needs to use the correct streaming algorithm for $\mathcal{A}$ restricted to each window.

Using a bit more space will allow us to improve slightly on the approximation ratio in Theorem 7. This is useful when ℓ is a small constant.

Theorem 8 *Let* $\mathcal{A}$ *be a streaming algorithm for the disc cover problem restricted to a window of size* $2\ell \times \ldots \times 2\ell$ *with approximation ratio* $r_{\mathcal{A}}$. *Suppose that* $\mathcal{A}$ *uses* s *bits of space and let* $s_h = \mathcal{O}(d \log(1/\varepsilon)\log(n))$. *Then there is a streaming algorithm for the disc cover problem with approximation ratio* $(1+\varepsilon)(1+1/\ell)^d r_{\mathcal{A}}$ *that uses* $\mathcal{O}(\varepsilon^{-2}d^{2d+2}\ell^{3d}\log(\ell d)(s+s_h))$ *bits of space and has success probability at least* 0.99.

The proof of Theorem 8 is nearly identical to the proof of Theorem 7. The only difference is that instead of sampling a random partition, we maintain all partitions. Thus, the space increases by a factor of $\mathcal{O}(\ell^d)$ but for the approximation ratio, we can apply Lemma 2 instead of Lemma 3.

4 Applications of the streaming shifting lemma

In this section, we present within-window algorithms for unit disc cover and various problems on unit disc graphs. When combined with the streaming shifting strategy, these within-window algorithms give global streaming algorithms.

4.1 Unit disc cover in 2D with L_2 balls

It suffices to give an approximation algorithm for the UDC in 2D restricted to a $2\ell \times 2\ell$ window and then apply Theorem 6. Let $\delta < \frac{2/\sqrt{3}-1}{\sqrt{2}}$ be a fixed positive constant and partition the window into a uniform grid of side length $\delta \times \delta$. For each square in the grid, we keep the first point in the stream that lies in the square. Thus, we only require storing $\mathcal{O}(\ell^2)$ points and $\mathcal{O}(\ell^2 \log(n))$ bits of space for the window. We then solve the UDC problem optimally given only the points we maintain, giving us a candidate disc cover $\mathcal{C}$.Although $\mathcal{C}$ may not cover all the input points, any uncovered point is at most distance $\delta\sqrt{2}$ from a disc in $\mathcal{C}$. Hence by increasing the radius of each disc in $\mathcal{C}$ by $\delta\sqrt{2}$, we fully cover all the points in the window. By our choice of δ, each disc of radius $1+\delta\sqrt{2}$ can be completely covered by 3 unit discs (see Figure 1), giving a 3-approximation to the within-window UDC problem. Choosing $\ell = \mathcal{O}(1/\varepsilon)$ gives the following theorem.

Theorem 9 *There is a streaming algorithm that uses* $\mathcal{O}(\varepsilon^{-8}\log(1/\varepsilon)\log(n))$ *bits of space and gives a* $(3+\varepsilon)$*-approximation to the* L_2 *UDC problem in 2D.*

We note that the algorithm above can be trivially extended to higher dimensions, though we do not have a good bound on the approximation factor.

4.2 Unit disc cover in 2D with L_∞ balls

Consider as before a $2\ell \times 2\ell$ window. Recall that an L_∞ ball of unit radius corresponds to a 2×2 square in $\mathbb{R}^2$. Consider a partition of the window into ℓ horizontal strips of unit height. Then this reduces to ℓ copies of the standard 1D UDC problem. We can now use the $(3/2+\varepsilon)$-approximation for UDC in 1D (due to [4]),

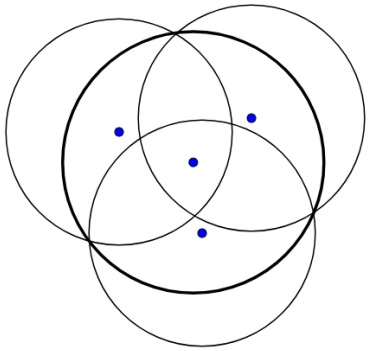


Figure 1: A covering of a radius $2/\sqrt{3}$ disc by 3 discs of radius 1.

using $\mathcal{O}(\varepsilon^{-2}\log(1/\varepsilon)\log(n))$ bits of space for each strip. Noting that any square in the optimal covering of a $2\ell \times 2\ell$ window touches at most 2 strips, this gives a $(3+\varepsilon)$-approximation to the UDC. Choosing $\ell = \mathcal{O}(1/\varepsilon)$ gives a space complexity of $\mathcal{O}(\varepsilon^{-3}\log(1/\varepsilon)\log(n))$ bits as we require ℓ runs of the 1D UDC approximation. Applying Theorem 6 gives the following theorem.

Theorem 10 *There is a streaming algorithm that uses $\mathcal{O}(\varepsilon^{-9}\log^2(1/\varepsilon)\log(n))$ bits of space and gives a $(3+\varepsilon)$-approximation to the L_∞ UDC problem in 2D.*

4.3 Streaming algorithms for unit disc graphs

Using the shifting coresets developed in Fonseca et al. [6], we obtain several streaming algorithms for unit disc graphs. In their work, they develop various $\mathcal{O}(1)$ memory within-window algorithms by computing a coreset for each window. Their coresets are similar to our within-window algorithm for UDC, in that they partition the window into squares of size $\delta \times \delta$ where δ is a fixed constant. A constant number of points is then stored in each square, and the problem is solved on the stored points. In the offline model, this gives rise to constant factor approximations for maximum weight independent set, dominating set, and minimum vertex cover on unit disc graphs.

Using the streaming shifting lemma, we obtain streaming algorithms for dominating set, minimum vertex cover, and *unweighted* maximum independent set. This is simply from using their within-window algorithms as a black box. The restriction to unweighted problems is due to our technique of subsampling windows, as subsampling may miss a small number of windows that contain large weights of the optimal solution.

5 Lower bounds

In this section, we prove lower bounds on the UDC problem via a reduction to the INDEX problem in communication complexity which is defined as follows. Let $n \in \mathbb{N}$. Alice has a vector $x \in \{0,1\}^n$ and Bob has an index $i \in [n]$. In the one-way communication model, Alice is allowed to send a single message to Bob and Bob must then compute the answer. Note that there is a trivial protocol that communicates n bits; Alice could send the whole vector x to Bob. The following theorem asserts that, up to constant factors, there is no better protocol even if it is randomized.

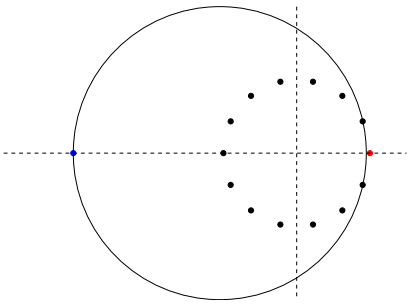


Figure 2: The lower bound construction for UDC in 2D. Alice streams in the points on the unit circle on the right. Bob streams in the point on the left to determine whether or not the rightmost point is present.

Theorem 11 ([13]) *Any one-way randomized communication protocol which solves* INDEX *with probability at least* 0.51 *requires $\Omega(n)$ bits of communication.*

Using this theorem [4] was able to show that any streaming algorithm that computes a $(1.5-\varepsilon)$-approximation to the maximum independent intervals problem in one dimension requires $\Omega(n)$ space. This essentially implies the same lower bound for UDC in any dimension.

Theorem 12 ([4]) *Fix $\varepsilon \in (0, 0.5)$. In all dimensions and for any L_p norm, if a streaming algorithm computes a $(1.5-\varepsilon)$-approximation to UDC with success probability at least* 0.51 *then it uses $\Omega(n)$ space.*

5.1 A $(2-\varepsilon)$ lower bound for L_2 UDC in 2D

Theorem 13 *Fix $\varepsilon \in (0,1)$. In dimensions two and higher, if a streaming algorithm computes a $(2-\varepsilon)$-approximation to UDC using L_2 balls with success probability at least* 0.51 *then it uses $\Omega(n)$ space.*

Proof. We will reduce from INDEX. Let $\mathcal{A}$ be a streaming algorithm, using S bits, which computes a $(2-\varepsilon)$-approximation to UDC in 2D with L_2 balls of radius 2. Let $z \in \{0,1\}^n$ be Alice's input and $i \in [n]$ be Bob's input. For simplicity, we assume that Bob's input is $i = n$; it will be apparent how to generalize to any i. If $z_j = 1$ then Alice streams the point $(\cos(2j\pi/n), \sin(2j\pi/n))$ into $\mathcal{A}$. When she is done she

sends the memory contents of $\mathcal{A}$ to Bob. Bob now streams the point $\left(\frac{1+\cos(2\pi/n)}{2} - 4, 0\right)$ and queries $\mathcal{A}$. (See also Figure 2.)

Suppose first that $z_i = 0$. Then we claim that placing a radius 2 ball with center at $\left(\frac{1+\cos(2\pi/n)}{2} - 2, 0\right)$ covers all the points. Indeed, it clearly covers Bob's point. To show that the ball covers all of Alice's points, it suffices to show that the radius 2 ball intersects the unit ball for some coordinate in $\left(\cos(2\pi/n), \frac{1+\cos(2\pi/n)}{2}\right)$. Indeed, at $x = \cos(2\pi/n)$, the y-coordinates of the radius 2 ball is at $\pm\sqrt{4 - \left(\frac{3-\cos(2\pi/n)}{2}\right)^2}$. It can be verified that the absolute value of this quantity is at least $\sin(2\pi/n)$. Indeed, for any $\theta \in \mathbb{R}$

$$\begin{aligned} &4 - \left(\frac{3-\cos(\theta)}{2}\right)^2 - \sin^2(\theta) \\ &= \frac{3}{4}\cos^2(x) - \frac{3}{2}\cos(\theta) + 3/4 \\ &= 3\left(\frac{\cos(\theta)-1}{2}\right)^2 \\ &= 3\sin^4(\theta/2) > 0, \end{aligned}$$

where in the last equality we used the identity $\sin^2(\theta/2) = (1 - \cos(\theta))/2$. Hence, the radius 2 ball covers all of Alice's points so $\mathcal{A}$ will report a quantity $\leq 2 - \varepsilon$.

On the other hand, if $z_i = 1$ then at least two points are required just to cover $(1, 0)$ and $\left(\frac{1+\cos(2\pi/n)}{2} - 4, 0\right)$ so $\mathcal{A}$ will report ≥ 2. □

6 Practical algorithms for UDC

Although the algorithms of the previous section have low approximation ratios, they involve high constant factors in their running times or memory that may make them unsuitable for practical use. In this section, we develop several streaming algorithms for unit disc cover that we believe are suitable in practice. To achieve good performance in practice, we either relax the approximation factor, or use multiple passes.

Our first algorithm for UDC is also the simplest. We cover $\mathbb{R}^d$ with an appropriate lattice of unit balls, and then apply the distinct elements algorithm of Kane, Woodruff, and Nelson [12] to count the number of balls of the lattice containing at least one input point. In the case of L_∞ in 2D, this lattice is simply a uniform grid where each square has width 2. In the case of L_2 in 2D, the lattice takes the uniform grid of L_∞ and places a unit circle on each grid point, as well as a unit circle in the center of each grid square. When a point is streamed, we compute the unit ball it belongs to and add that ball to the distinct elements data structure. If the point belongs to multiple balls (as in the L_2 case), choose any of the balls it belongs to and add it to the data structure. By choosing randomly from a family of shifted versions of such lattices, we obtain the result below whose proof is deferred to Appendix C.

Theorem 14 *There is a one pass streaming algorithm for L_2 UDC in 2D that uses $\mathcal{O}(\varepsilon^{-2} + \log(n))$ space with approximation factor $2\pi(1+\varepsilon)$ and succeeds with probability at least 0.99.*

Theorem 15 *There is a one pass streaming algorithm for L_∞ and L_1 UDC in 2D that uses $\mathcal{O}(\varepsilon^{-2} + \log(n))$ space with approximation factor 4 and succeeds with probability at least 0.99.*

Proof. The proof of this is exactly analogous to the proof of Theorem 14 but in this case it is not necessary to randomly shift the lattice. In L_∞ and L_1, we use squares instead of L_2 discs. □

6.1 Using multiple passes

By using multiple passes over the input data, we can give alternate algorithms that both improve the approximation factor and the memory of Theorems 14 and 15. One example is the following theorem, whose proof can be found in Appendix D.

Theorem 16 *There is a two pass streaming algorithm for L_∞ and L_1 UDC in 2D that uses $\mathcal{O}(\varepsilon^{-2}\log n)$ space with approximation factor 3 and succeeds with probability at least 0.99.*

Finally, we give one additional algorithm for L_1 and L_∞ UDC in $\mathbb{R}^2$. Observe that for the 1-dimensional UDC problem, if we allow around $1/\varepsilon$ passes through the data then $O(\varepsilon^{-1}\log n)$ memory suffices to solve the problem with approximation factor $1+\varepsilon$. Within each 1D window, we simply cover the leftmost uncovered point with an interval that begins at that point. By the end of a pass over the input data, we should be able to determine another leftmost uncovered point in the window or if we have covered all of the points. Since all the intervals used are disjoint, we use at most ℓ passes for a window size of ℓ. This effectively simulates the greedy offline interval covering algorithm using multiple passes. Combining this with our streaming strategy gives the following result for L_∞ UDC.

Theorem 17 *There is a $1/\varepsilon$ pass streaming algorithm for L_∞ and L_1 UDC in 2D that uses $\mathcal{O}(\varepsilon^{-7}\log(1/\varepsilon)\log(n))$ space with approximation factor $2+\epsilon$.*

Proof. We simply divide each $2\ell \times 2\ell$ window into ℓ horizontal strips, and use the 1D UDC algorithm with approximation factor $1+1/\ell$ on each strip for the within-window algorithm. Since each disc of the optimal solution can touch at most two strips, we get approximation factor $2+\epsilon$ by choosing $\ell = \mathcal{O}(1/\varepsilon)$. □

References

[1] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. In *24th Annual Symposium on Foundations of Computer Science*, pages 265–273, Nov 1983.

[2] Y. Bejerano. Efficient integration of multihop wireless and wired networks with QoS constraints. *IEEE/ACM Transactions on Networking (TON)*, 12(6):1064–1078, 2004.

[3] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher. Min-wise independent permutations. *J. Comput. Syst. Sci.*, 60(3):630–659, 2000.

[4] S. Cabello and P. Pérez-Lantero. Interval selection in the streaming model. In *Algorithms and Data Structures - 14th International Symposium, WADS 2015, Victoria, BC, Canada, August 5-7, 2015. Proceedings*, pages 127–139, 2015.

[5] T. M. Chan. A note on maximum independent sets in rectangle intersection graphs. *Information Processing Letters*, 89(1):19–23, 2004.

[6] G. D. da Fonseca, V. G. P. de Sá, and C. M. H. de Figueiredo. Linear-time approximation algorithms for unit disk graphs. In *Approximation and Online Algorithms - 12th International Workshop, WAOA 2014, Wrocław, Poland, September 11-12, 2014, Revised Selected Papers*, pages 132–143, 2014.

[7] D. Eppstein. Subgraph isomorphism in planar graphs and related problems. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '95, pages 632–640, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

[8] T. Erlebach, K. Jansen, and E. Seidel. Polynomial-time approximation schemes for geometric graphs. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 671–679, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[9] T. Erlebach and E. J. van Leeuwen. *PTAS for Weighted Set Cover on Unit Squares*, pages 166–177. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[10] D. S. Hochbaum and W. Maass. Approximation schemes for covering and packing problems in image processing and VLSI. *J. ACM*, 32(1):130–136, 1985.

[11] P. Indyk. A small approximately min-wise independent family of hash functions. *J. Algorithms*, 38(1):84–90, 2001.

[12] D. M. Kane, J. Nelson, and D. P. Woodruff. An optimal algorithm for the distinct elements problem. In *Proceedings of the Twenty-Ninth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2010, June 6-11, 2010, Indianapolis, Indiana, USA*, pages 41–52, 2010.

[13] I. Kremer, N. Nisan, and D. Ron. On Randomized one-round communication complexity. *Computational Complexity*, 8(1):21–49, 1999.

[14] S. P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012.

Appendix

A Proof of Lemma 3

Proof. Let S be a random partition of $\mathbb{R}^d$ into windows of size $2\ell \times \ldots \times 2\ell$. Consider an optimal disc cover and construct a new disc cover as follows. If a disc is in k windows then the new disc cover will have k copies of the disc, each associated with one of the windows. Note that this gives a disc cover for each of the windows.

Let us number the discs in the optimal cover, $1, \ldots,$ OPT, and let X_i be the number of windows which contain a portion of disc i. Since S is a random partition, we have that for each coordinate $j \in [d]$, disc i intersects a closed boundary of a window along coordinate j with probability $1/\ell$. If this intersection happens along k coordinates then $X_i \leq 2^k$. Hence, $\mathbb{E}X_i \leq \sum_{k\geq 0} \binom{d}{k} \left(\frac{1}{\ell}\right)^k \left(\frac{\ell-1}{\ell}\right)^{d-k} 2^k = (1+1/\ell)^d \leq 1 + 2d/\ell$ where the last inequality is because $\ell \geq 2d$.

Let $Y = \sum_{i=1}^{\mathrm{OPT}} X_i$. Then Y is an upper bound on the number of disc covers obtained by solving each window optimally. Moreover, $\mathbb{E}[Y - \mathrm{OPT}] \leq \mathrm{OPT} \cdot 2d/\ell$, so by Markov's Inequality, $Y - \mathrm{OPT} \leq 4d/\ell \cdot \mathrm{OPT}$ with probability at least $1/2$. The lemma now follows since $\mathcal{A}$ is an $r_{\mathcal{A}}$-approximate algorithm for each window. □

B Proof of Lemma 5

Proof. Let $\mathcal{H}$ be a $\mathcal{O}(\log(1/\varepsilon))$-wise independent family of hash functions. The input to the hash functions is a window (there are $\mathrm{poly}(n)$ possible windows) and the output is a number of $[\mathrm{poly}(n)]$. By Theorem 1, the family $\mathcal{H}$ is a (n, ε)-min-wise family of hash functions.

To estimate η_t we do the following. Let $r \in \mathbb{N}$ be a parameter to be chosen later and $h_1, \ldots, h_r$ be drawn from $\mathcal{H}$ uniformly and independently at random. For each $j \in [r]$, we maintain a window W_j for h_j and a copy of $\mathcal{A}$ (denoted $\mathcal{A}_j$) as follows. We initialize W_j to be a dummy window with $h_j(W_j) = \infty$. Now suppose we receive a point p in the input and let W be the window

that p belongs to. If $W = W_j$ then we stream p into $\mathcal{A}_j$. On the other hand, if $W \neq W_j$ then we have two cases. If $h_j(W) < h_j(W_j)$ then we replace W_j with the new window W, reset $\mathcal{A}_j$, and stream p into $\mathcal{A}_j$. Otherwise, if $h_j(W) > h_j(W_j)$ then we ignore p.

Fix $t \in \{2, \ldots, T\}$ and $j \in [r]$. Let X_j be the random variable which is 1 if $\mathcal{A}_j$ reports that the window minimizing h_j has a disc cover of size at least t. Otherwise, $X_j = 0$. Since $\mathcal{H}$ is an (n, ε)-min-wise family, it follows that $\mathbb{E}X_j = (1 \pm 2\varepsilon)\eta_t$. Now let $\hat{\eta}_t = \frac{1}{r}\sum_{j=1}^{r} X_j$. By Hoeffding's Inequality, we have

$$\Pr[|\hat{\eta}_t - \mathbb{E}X_j| \geq \varepsilon/T] \leq 2\exp(-2r\varepsilon^2/T^2).$$

By choosing $r \geq \mathcal{O}(T^2 \log(T)/\varepsilon^2)$, the above probability is at most $1/(100T)$. Hence, by a union bound, we have $\hat{\eta}_t = (1 \pm 2\varepsilon)\eta_t \pm \varepsilon/T$ for all t with probability at least 0.99.

Finally, it remains to analyze the space requirement of this scheme. Storing each hash function requires s_h bits of space. Hence, storing all r hash function requires $\mathcal{O}(\varepsilon^{-2}\ell^4 \log(\ell) s_h)$ bits of space. Next, we have a copy of $\mathcal{A}$ for each of the r windows we maintain. So this uses an additional $\mathcal{O}(\varepsilon^{-2}\ell^4 \log(\ell) s)$ bits of space. Hence, the total space usage is $\mathcal{O}(\varepsilon^{-2}\ell^4 \log(\ell)(s + s_h))$ bits. □

C Proof of Theorem 14

Proof. Let $\mathcal{S}_{\mathrm{OPT}}$ be the set of discs in an optimal solution. Let Γ be the lattice of unit discs described above, and let n_Γ be the maximum number of lattice discs intersecting a disc in $\mathcal{S}_{\mathrm{OPT}}$. In the worst case, the algorithm above counts n_Γ discs for each disc of $\mathcal{S}_{\mathrm{OPT}}$.

To compute the expectation from choosing a random shift of the lattice, we can view each disc of $\mathcal{S}_{\mathrm{OPT}}$ as radius 2 and the discs on the lattice as having radius 0 with lattice points on a uniform grid of side length $\sqrt{2}$. Thus n_Γ is equivalent to the expected number of lattice points that fall within a randomly placed radius 2 disc on the plane. In expectation, this is equal to the area of the disc scaled by the area of a lattice square. Hence we get that $\mathbb{E}n_\Gamma = 2\pi$. For each disc in $\mathcal{S}_{\mathrm{OPT}}$, the number of discs it intersects within the lattice is a probability distribution supported on $\{1, 2, \ldots, 16\}$. Since the mean of the distribution is 2π, running the algorithm with a randomly shifted lattice will produce at most $2\pi \cdot \mathrm{OPT}$ discs with at least a constant probability. By running multiple copies of the algorithm and taking the minimum, we get the result of Theorem 14. □

D Proof of Theorem 16

Proof. Consider the following algorithm for UDC in L_1 and L_∞. First, set the shifting parameter ℓ of Theorem 8 to be 2. For the analysis, fix a window and consider the points that fall within the window. In the first pass, the algorithm goes through the input points and maintains the smallest bounding rectangle that covers all the points. Observe that we can cover the points with 0 unit squares if and only if the input is empty and we can cover the points with 1 unit square if and only if the bounding rectangle fits inside a unit square. In either of these two cases, the second pass is not necessary. If the input points can be covered by 2 unit squares then this can be done by choosing 2 of the 4 corners of the bounding rectangle and choosing the unit squares to lie in the rectangle while covering the 2 corners. There are 6 possible ways to do this so in the second pass, we check if one these choices cover all the points. If not then the point set requires at least 3 squares to cover so we estimate it as 4. Hence, this gives a 4/3-approximation for each window. Combining this with the 9/4-approximation from using Theorem 8 with $\ell = 2$ gives the theorem. □

Formigrams: Clustering Summaries of Dynamic Data*

Woojin Kim† Facundo Mémoli‡

Abstract

When studying flocking/swarming behaviors in animals one is interested in quantifying and comparing the dynamics of the clustering induced by the coalescence and disbanding of animals in different groups.

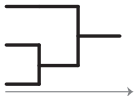
Motivated by this, we propose a summarization of time-dependent metric data which captures their time-dependent clustering features which we call *formigrams*. These set-valued functions generalize the notion of dendrogram, a prevalent object in the context of hierarchical clustering.

Also, we define a metric on formigrams for quantifying the degree of structural difference between any two given formigrams. In particular, the restriction of this metric to the collection of dendrograms recovers twice the Gromov-Hausdorff distance between the ultrametric spaces associated to the dendrograms. This fact enables us to show that constant factor approximations to the metric on formigrams cannot be obtained in polynomial time.

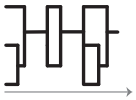
Finally, we investigate a sufficient condition for time-dependent metric spaces to be summarized into formigrams. In addition, we prove that this summarization process is stable under perturbations in the input time-dependent metric data.

1 Introduction

Given data represented as a static finite metric space (X, d_X), a hierarchical clustering method finds a hierarchical family of partitions that captures multi-scale features present in the dataset. These hierarchical families of partitions are called *dendrograms* and their visualization is straightforward (see figure on the left).



We now turn our attention to a problem of characterizing dynamic data. We model dynamic datasets as time varying finite metric spaces and study a simple generalization of the notion of dendrogram which we call *formigram* - a combination of the words formicarium[1] and diagram (see figure on the right).



Whereas dendrograms are useful for modeling situations when data points aggregate along a certain scale parameter, formigrams are better suited for representing phenomena when data points may also separate or disband and then regroup at different parameter values. One motivation for considering this scenario comes from the study and characterization of *flocking/swarming/herding* behavior of animals [1, 10, 11, 12, 19, 21, 24, 28], convoys [14], moving clusters [15], or mobile groups [13, 29].

Related work. Let $\mathcal{X}$ be a set of points having piecewise linear trajectories with time-stamped vertices in Euclidean space $\mathbf{R}^d$. Buchin and et al. [3] provided explicit algorithms for studying the grouping structure of $\mathcal{X}$. This was subsequently enriched in [18, 25, 26, 27].

From the set $\mathcal{X}$, the authors of [3] construct a Reeb graph-like structure $\mathcal{R}_{\mathcal{X}}$ which is closely related to the *formigram* derived from $\mathcal{X}$ that we introduce (Section 3 and Theorem 4). The edges of $\mathcal{R}_{\mathcal{X}}$ are labeled by *maximal groups*, and they call $\mathcal{R}_{\mathcal{X}}$ together with these labels the *trajectory grouping structure* of $\mathcal{X}$, enabling the visualization of the life span of maximal groups.

Our contributions.

1. We generalize dendrograms to *formigrams* for the analysis of clustering features of dynamic data, such as dynamic metric spaces or dynamic graphs.

2. Any dendrogram over a finite set X induces an ultrametric on X [7]. Therefore, one can quantify the structural difference between any two dendrograms by computing the Gromov-Hausdorff distance between their two induced ultrametrics [7]. We propose a distance $d_{\mathrm{I}}^{\mathrm{F}}$ between formigrams which generalizes the method above for comparing two dendrograms (Theorems 1 and 2). The desire to obtain such a precise quantification of the difference between two dynamic clusterings was already made explicit in [3, Section 6]. Also, we show that constant factor approximations to $d_{\mathrm{I}}^{\mathrm{F}}$ cannot be obtained in polynomial time (Theorem 3).

*This work was partially supported by NSF grants IIS-1422400, CCF-1526513, and DMS-1723003.

†Department of Mathematics, The Ohio State University, kim.5235@osu.edu

‡Department of Mathematics, The Ohio State University, memoli@math.osu.edu

[1] A formicarium or ant farm is an enclosure for keeping ants under semi-natural conditions [30].

3. As an application, we propose a method for turning any (tame) dynamic metric data into a formigram. This method is closely related to the construction of trajectory grouping structures [3]. In particular, this method turns out to be stable under perturbations in the input dynamic metric data under a certain notion of distance between DMSs that we introduce (Theorem 5).

2 Background

2.1 Dendrograms and treegrams

Partitions and sub-partitions. Let X be a non-empty finite set. We will call any partition P of a subset X' of X a *sub-partition* of X (in particular, any partition of the empty set is defined as the empty set). In this case we call X' the *underlying set of* P.

1. By $\mathcal{P}^{\text{sub}}(X)$, we denote the set of *all sub-partitions of* X, i.e.

$$\mathcal{P}^{\text{sub}}(X) := \{P : \exists X' \subset X \ , \ P \text{ is a partition of } X'\}.$$

2. By $\mathcal{P}(X)$, we denote the subcollection of $\mathcal{P}^{\text{sub}}(X)$ consisting solely of partitions of the *whole* X.

Given $P, Q \in \mathcal{P}^{\text{sub}}(X)$, by $P \leq Q$ we mean "P is finer than or equal to Q", i.e. for all $B \in P$, there exists $C \in Q$ such that $B \subset C$. For example, let $X = \{x_1, x_2, x_3\}$ and consider the sub-partitions $P := \{\{x_1, x_2\}\}$ and $Q := \{\{x_1, x_2\}, \{x_3\}\}$ of X. Then, it is easy to see that in this case $P \leq Q$.

Dendrograms. A *dendrogram* over a finite set X is any function $\theta_X : \mathbf{R}_+ \to \mathcal{P}(X)$ such that the following properties hold: (1) $\theta_X(0) = \{\{x\} : x \in X\}$, (2) if $t_1 \leq t_2$, then $\theta_X(t_1) \leq \theta_X(t_2)$, (3) there exists $T > 0$ such that and $\theta_X(t) = \{X\}$ for $t \geq T$, (4) for all t there exists $\epsilon > 0$ s.t. $\theta_X(s) = \theta_X(t)$ for $s \in [t, t + \epsilon]$ (right-continuity). See Figure 1 for an example.

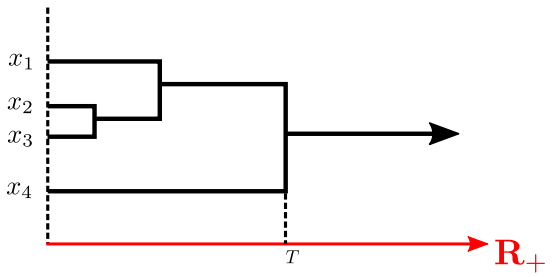


Figure 1: A dendrogram θ_X over the set $X = \{x_1, x_2, x_3, x_4\}$. Notice that $\theta_X(0) = \{\{x_1\}, \{x_2\}, \{x_3\}, \{x_4\}\}$ and $\theta_X(t) = \{X\}$ for all $t \in [T, \infty)$.

Treegrams. Dendrograms can be generalized to treegrams, a visual representation for hierarchical clustering of networks [23]. [2] A *treegram* over a finite set X is any function $\theta_X : \mathbf{R} \to \mathcal{P}^{\text{sub}}(X)$ such that the following properties hold: (1) if $t_1 \leq t_2$, then $\theta_X(t_1) \leq \theta_X(t_2)$, (2) (boundedness) there exists $T > 0$ such that $\theta_X(t) = \{X\}$ for $t \geq T$ and $\theta_X(t)$ is empty for $t \leq -T$. (3) for all t there exists $\epsilon > 0$ s.t. $\theta_X(s) = \theta_X(t)$ for $s \in [t, t+\epsilon]$ (right-continuity). See Figure 2 for an example.

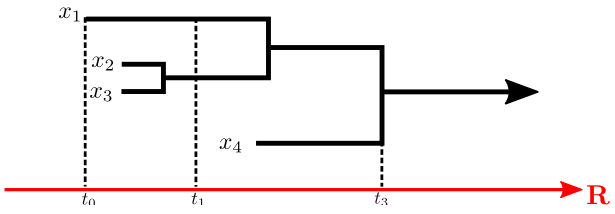


Figure 2: A treegram θ_X over the set $X = \{x_1, x_2, x_3, x_4\}$. Notice that $\theta_X(t) = \emptyset$ for $t \in (-\infty, t_0)$. Also, $\theta_X(t_0) = \{\{x_1\}\}$, $\theta_X(t_2) = \{\{x_1\}, \{x_2, x_3\}\}$, and $\theta_X(t) = \{X\}$ for all $t \in [t_3, \infty)$.

2.2 A distance between dendrograms

In this section we review the method of [7] for quantifying the structural difference between dendrograms. In short, we compare two dendrograms over sets X and Y by comparing their associated ultrametrics on X and Y, respectively.

Dendrograms and their associated ultrametrics. An *ultrametric space* (X, u_X) is a metric space satisfying the *strong triangle inequality*: for all $x, x', x'' \in X$, $u_X(x, x') \leq \max\{u_X(x, x''), u_X(x'', x')\}$.

Let X be a finite set and let $\theta_X : \mathbf{R}_+ \to \mathcal{P}(X)$ be a dendrogram over X. Recall from [7] that this θ_X induces a canonical ultrametric $u_{\theta_X} : X \times X \to \mathbf{R}_+$ on X defined by

$$u_{\theta_X}(x, x') := \inf\{\varepsilon \geq 0 : x, x' \text{ belong to the same block of } \theta_X(\varepsilon)\}.$$

For example, for the dendrogram θ_X depicted in Figure 1, it is easy to observe that $u_{\theta_X}(x_1, x_4) = T$.

Reciprocally, any ultrametric space (X, u_X) induces a dendrogram θ_X over X [7].

The Gromov-Hausdorff distance [4, Ch 7]. The Gromov-Hausdorff distance quantifies how far two compact metric spaces are from being isometric. This distance is widely used in applications such as shape comparison (for example, see [20]). In order to define the

[2] In order to regard a dendrogram $\theta_X : \mathbf{R}_+ \to \mathcal{P}(X)$ as a treegram, trivially extend θ_X to the whole $\mathbf{R}$: for $t \in (-\infty, 0)$, let $\theta_X(t) := \emptyset \in \mathcal{P}^{\text{sub}}(X)$ by definition.

Gromov-Hausdorff distance, one needs the notion of *correspondence.*

For sets X and Y, a subset $R \subset X \times Y$ is said to be a *correspondence* (between X and Y) if and and only if (1) for every $x \in X$, there exists $y \in Y$ such that $(x, y) \in R$, and (2) for every $y \in Y$, there exists $x \in X$ such that $(x, y) \in R$.

Let (X, d_X) and (Y, d_Y) be any two compact metric spaces. The Gromov-Hausdorff distance between (X, d_X) and (Y, d_Y) is defined by

$$d_{\mathrm{GH}}\left((X, d_X), (Y, d_Y)\right) := \frac{1}{2} \inf_{R} \sup_{\substack{(x,y)\in R \\ (x',y')\in R}} \left|d_X(x, x') - d_Y(y, y')\right|,$$

where the infimum is taken over all correspondences between X and Y. Note that in the case where $(X, d_X), (Y, d_Y)$ are finite metric spaces, the infimum and the supremum above can be replaced with the minimum and the maximum, respectively.

A distance between dendrograms. Let θ_X and θ_Y be dendrograms over finite sets X and Y, respectively. One defines the *Gromov-Hausdorff distance [7] between the dendrograms* θ_X and θ_Y as

$$d_{\mathrm{GH}}(\theta_X, \theta_Y) := d_{\mathrm{GH}}\left((X, u_{\theta_X}), (Y, u_{\theta_Y})\right),$$

where u_{θ_X} and u_{θ_Y} are the ultrametrics associated to the dendrograms θ_X and θ_Y, respectively.

2.3 Finest common coarsening of (sub-)partitions

For a set X, we know that there exists a canonical one-to-one correspondence between the collection of all equivalence relations on X and the collection of all partitions $\mathcal{P}(X)$ of X. We will extend this correspondence in a certain way for defining the notion of *finest common coarsening* in the collection $\mathcal{P}^{\mathrm{sub}}(X)$ of all sub-partitions of X.

Sub-equivalence relations. Let X be a non-empty set. Let $\sim$ be any equivalence relation on any subset $X' \subset X$.[3] We call the relation $\sim$ a *sub-equivalence relation on* X. We also call X' the *underlying set* of $\sim$, which is identical to $\{x \in X : (x, x) \in \sim\}$.

Clearly, any equivalence relation on X is also a sub-equivalence relation with underlying set X.

There is the canonical one-to-one correspondence between the collection of all sub-equivalence relations on X and the collection $\mathcal{P}^{\mathrm{sub}}(X)$ of all sub-partitions of X: Any sub-equivalence relation $\sim$ on X corresponds to the sub-partition P with underlying set $X' = \{x \in X : (x, x) \in \sim\}$ such that $x \sim y$ iff x and y belong to the same block $B \in P$. Reciprocally, to any sub-partition P of X, one can associate the unique sub-equivalence relation $\sim_P$ on X defined by $x \sim_P y$ if and only if x and y belong to the same block $B \in P$.

Sub-equivalence closure. Let X be a non-empty set. For an index set I, suppose that $\{\sim_i \subset X \times X : i \in I\}$ is a collection of sub-equivalence relations on X. The *sub-equivalence closure* of the collection $\{\sim_i \subset X \times X : i \in I\}$ is defined to be the transitive closure of the relation $\cup_{i\in I} \sim_i$ on X. In other words, by the *sub-equivalence closure* of the collection $\{\sim_i \subset X \times X : i \in I\}$, we mean the minimal sub-equivalence relation containing $\sim_i$ for all $i \in I$.

Finest common coarsening. Let $\{P_i\}_{i\in I}$ be any sub-collection of $\mathcal{P}^{\mathrm{sub}}(X)$. For each $i \in I$, let $\sim_i$ be the sub-equivalence relation on X corresponding to P_i. By $\bigvee_{i\in I} P_i$, we mean the sub-partition of X corresponding to the sub-equivalence closure of the collection $\{\sim_i \subset X \times X : i \in I\}$. We will refer to $\bigvee_{i\in I} P_i$ as *the finest common coarsening of the collection* $\{P_i\}_{i\in I}$.

For example, let $X := \{x, y, z, w\}$. For $P_1 = \{\{x\}, \{y\}\}, P_2 = \{\{y, z\}\}$, and $P_3 = \{\{x, w\}\}$ in $\mathcal{P}^{\mathrm{sub}}(X)$, we have:

1. $\bigvee_{i=1}^{2} P_i = \{\{x\}, \{y, z\}\} \in \mathcal{P}^{\mathrm{sub}}(X)$, and
2. $\bigvee_{i=1}^{3} P_i = \{\{x, w\}, \{y, z\}\} \in \mathcal{P}(X)$.

3 Formigrams

Although the notions of dendrogram or treegram are useful when representing the output of a hierarchical clustering method (i.e. when partitions only become coarser with the increase of a parameter), in order to represent the diverse clustering behaviors of dynamic datasets we need a more flexible concept allowing for possible refinement of partitions. Here we suggest a "zigzag like" notion of dendrograms that we call *formigram*. We allow partitions to become finer sometimes, but require that partitions defined by a formigram change only finitely many times in any finite interval for visualization.

3.1 The definition of a formigram

Formigrams. A *formigram* over a finite set X is any function $\theta_X : \mathbf{R} \to \mathcal{P}^{\mathrm{sub}}(X)$ such that:

1. (Tameness) the set $\mathrm{crit}(\theta_X)$ of points of discontinuity of θ_X is locally finite.[4] We call the elements of $\mathrm{crit}(\theta_X)$ *the critical points of* θ_X.

[3] In particular, the unique equivalence relation on the empty set $\emptyset$ is $\emptyset$.

[4] To say that $\mathrm{crit}(\theta_X)$ is locally finite means that for any bounded interval $I \subset \mathbf{R}$, the cardinality of $I \cap \mathrm{crit}(\theta_X)$ is finite. The purpose of this condition is twofold: on the one hand,

2. (Interval lifespan) for every $x \in X$, the set $I_x := \{t \in \mathbf{R} : x \in B \in \theta_X(t)\}$, said to be the *lifespan of* x, is a non-empty closed interval,

3. (Comparability) for every point $c \in \mathbf{R}$ it holds that $\theta_X(c-\varepsilon) \leq \theta_X(c) \geq \theta_X(c+\varepsilon)$ for all sufficiently small $\varepsilon > 0$.[5]

Note that the definition of formigrams generalizes those of dendrograms and treegrams.[6] In other words, every dendrogram and every treegram are formigrams. See Figure 3 for an example.

$$\theta_X(t) = \begin{cases} \{\{x_1,x_2,x_3\}\}, & t \in (-\infty, t_0) \\ \{\{x_1,x_2,x_3\},\{x_4\}\}, & t \in [t_0,t_1) \\ \{\{x_1,x_2,x_3,x_4\}\}, & t \in [t_1,t_2] \cup [t_5,\infty) \\ \{\{x_1,x_2\},\{x_3,x_4\}\}, & t \in (t_2,t_3] \cup [t_4,t_5) \\ \{\{x_1\},\{x_2\},\{x_3,x_4\}\}, & t \in (t_3,t_4). \end{cases}$$

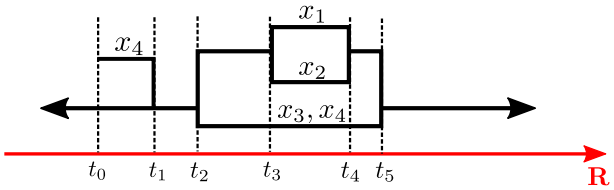


Figure 3: Top: The specification of a formigram θ_X over the set $X = \{x_1, x_2, x_3, x_4\}$. Bottom: A graphical representation of the formigram θ_X.

3.2 A distance between formigrams

In this section we introduce a (pseudo) metric on the collection of all formigrams. This metric quantifies the structural difference between two grouping/disbanding behaviors over time. In particular, when restricting this metric to the collection of dendrograms, (twice) the Gromov-Hausdorff distance between dendrograms is recovered (Theorem 2).

Partition morphisms. Before introducing a metric on formigrams, we first establish a method for interconnecting any two partitions with possibly different underlying sets. Recall that for any sets X and Y, a *multivalued map* $\varphi : X \rightrightarrows Y$ is a relation between X and Y such that for all $x \in X$, there exists (a not necessarily unique) $y \in Y$ with $(x,y) \in \varphi$.[7] For $x \in X$, the *image* $\varphi(x)$ of x is defined to be the set $\{y \in Y : (x,y) \in \varphi\}$.

For any two sets X and Y, let $P_X \in \mathcal{P}(X)$ and $P_Y \in \mathcal{P}(Y)$. Any multivalued map $\varphi : X \rightrightarrows Y$ (or map $\varphi : X \to Y$) is said to be a *partition morphism between* P_X *and* P_Y if for any $x, x' \in X$ belonging to the same block of P_X, their images $\varphi(x), \varphi(x')$ are included in the same block of P_Y (note that $\varphi(x), \varphi(x')$ can be sets containing more than one element). In this case, we write $P_X \leq_\varphi Q_Y$.

If $P_X \leq_\varphi Q_Y$, then there exists the canonical induced map $\varphi^* : P_X \to P_Y$ defined by sending each block $B \in P_X$ to the block $C \in P_Y$ such that $\varphi(B) \subset C$.

A distance between formigrams. Exploiting the fact that any formigram is a *"stack" of (sub-)partitions* of a specific set, we now introduce the *interleaving distance* $d_{\mathrm{I}}^{\mathrm{F}}$ on the collection of all formigrams. The construction of $d_{\mathrm{I}}^{\mathrm{F}}$ is inspired by the interleaving distance for Reeb graphs [9].

Let θ_X be a formigram over X and let $I \subset \mathbf{R}$ be an interval. We define $\bigvee_I \theta_X$ to be the finest common coarsening of the collection $\{\theta_X(t) : t \in I\}$ of sub-partitions of X. Also, for any $t \in \mathbf{R}$, define $[t]^\varepsilon := [t-\varepsilon, t+\varepsilon] \subset \mathbf{R}$.

Let θ_X and θ_Y be any two formigrams over X and Y, respectively. θ_X and θ_Y are said to be ε-*interleaved* if there exists a correspondence R between X and Y satisfying the following:

1. For any $(x,y) \in R$ and any $t \in \mathbf{R}$,
   (a) if x is in the underlying set of $\theta_X(t)$, then y is in the underlying set of $\bigvee_{[t]^\varepsilon} \theta_Y$.[8]
   (b) if y is in the underlying set of $\theta_Y(t)$, then x is in the underlying set of $\bigvee_{[t]^\varepsilon} \theta_X$.
2. For all $t \in \mathbf{R}$,
$$\theta_X(t) \leq_R \bigvee_{[t]^\varepsilon} \theta_Y \quad \text{and} \quad \theta_Y(t) \leq_{R^{-1}} \bigvee_{[t]^\varepsilon} \theta_X,$$
where $R^{-1} = \{(y,x) \in Y \times X : (x,y) \in R\}$.

We call any such R an ε-*correspondence between* θ_X *and* θ_Y.[9] The interleaving distance $d_{\mathrm{I}}^{\mathrm{F}}(\theta_X, \theta_Y)$ between θ_X and θ_Y is defined by the infimum of $\varepsilon \geq 0$ for which there exists an ε-correspondence between θ_X and θ_Y. If there is no ε-correspondence between θ_X and θ_Y for any $\varepsilon \geq 0$, then we declare $d_{\mathrm{I}}^{\mathrm{F}}(\theta_X, \theta_Y) = +\infty$.

we want to guarantee easy visualization, on the other hand this condition is necessary for the simplification process of formigrams via zigzag persistence theory [5]. We refer the interested readers to [16].

[5] If θ_X is not continuous at c, then at least one of the relations of $\theta_X(c-\varepsilon) \leq \theta_X(c) \geq \theta_X(c+\varepsilon)$ would be strict for small $\varepsilon > 0$. But if c is a continuity point of θ_X, then $\theta_X(c-\varepsilon) = \theta_X(c) = \theta_X(c+\varepsilon)$ for small $\varepsilon > 0$.

[6] In order to regard a dendrogram $\theta_X : \mathbf{R}_+ \to \mathcal{P}(X)$ as a formigram, trivially extend θ_X to the whole $\mathbf{R}$: for $t \in (-\infty, 0)$, let $\theta_X(t) := \emptyset \in \mathcal{P}^{\mathrm{sub}}(X)$ by definition.

[7] In particular, any correspondence R between X and Y is a multivalued map.

[8] We remark that this condition is equivalent to saying that if x is in the underlying set of $\theta_X(t)$, then there exists $t_0 \in [t]^\varepsilon$ such that y is in the underlying set of $\theta_Y(t_0)$.

[9] Note that if R is an ε-correspondence between θ_X and θ_Y, then for any $\varepsilon' > \varepsilon$, R is also an ε'-correspondence between θ_X and θ_Y.

Theorem 1 *$d_{\mathrm{I}}^{\mathrm{F}}$ is an extended pseudo-metric on formigrams.*

See Appendix A for the proof of Theorem 1. For example, consider any formigram θ_X over a finite set X and let $\tau > 0$. Define another formigram θ_X^τ as $\theta_X^\tau(t) := \theta_X(t+\tau)$ for $t \in \mathbf{R}$. Then, it is not difficult to verify that $d_{\mathrm{I}}^{\mathrm{F}}(\theta_X, \theta_X^\tau) \leq \tau$ by checking that $R_X := \{(x,x) : x \in X\}$ is a τ-correspondence between θ_X and θ_X^τ.

Theorem 2 *$d_{\mathrm{I}}^{\mathrm{F}}$ generalizes the Gromov-Hausdorff distance between dendrograms. Namely, for any dendrograms θ_X and θ_Y over X and Y respectively,*

$$d_{\mathrm{I}}^{\mathrm{F}}(\theta_X, \theta_Y) = 2\ d_{\mathrm{GH}}(\theta_X, \theta_Y).$$

Proof. Recall that by definition

$$d_{\mathrm{GH}}(\theta_X, \theta_Y) = d_{\mathrm{GH}}\left((X, u_{\theta_X}), (Y, u_{\theta_Y})\right)$$

where u_{θ_X} and u_{θ_Y} are the ultrametrics associated to the dendrograms θ_X and θ_Y, respectively. Therefore, we will show that $d_{\mathrm{I}}^{\mathrm{F}}(\theta_X, \theta_Y) = 2\ d_{\mathrm{GH}}\left((X, u_{\theta_X}), (Y, u_{\theta_Y})\right)$. First we show "$\geq$". If $d_{\mathrm{I}}^{\mathrm{F}}(\theta_X, \theta_Y) = \infty$, there is nothing to prove and hence we assume that $d_{\mathrm{I}}^{\mathrm{F}}(\theta_X, \theta_Y)$ is finite. Then, there exists an ε-correspondence $R \subset X \times Y$ between the two dendrograms θ_X and θ_Y for some $\varepsilon \geq 0$, implying that $d_{\mathrm{I}}^{\mathrm{F}}(\theta_X, \theta_Y) \leq \varepsilon$. Pick any $(x,y),(x',y') \in R$ and let $t := u_{\theta_X}(x,x')$. Then, x, x' belong to the same block of the partition $\theta_X(t)$. Since $\theta_X(t) \leq_R \bigvee_{[t]^\varepsilon} \theta_Y$, y, y' must belong to the same block of $\bigvee_{[t]^\varepsilon} \theta_Y$. Also, since θ_Y is a dendrogram, $\theta_Y(s_1) \leq \theta_Y(s_2)$ for any $s_1 \leq s_2$, and thus $\bigvee_{[t]^\varepsilon} \theta_Y = \theta_Y(t+\varepsilon)$. Therefore, y, y' belong to the same block of $\theta_Y(t+\varepsilon)$, and in turn $u_{\theta_Y}(y,y') \leq t + \varepsilon = u_{\theta_X}(x,x') + \varepsilon$. By symmetry, we also have $u_{\theta_X}(x,x') \leq u_{\theta_Y}(y,y') + \varepsilon$. Therefore, by the definition of $d_{\mathrm{GH}}((X, u_{\theta_X}), (Y, u_{\theta_Y}))$, we have $d_{\mathrm{GH}}((X, u_{\theta_X}), (Y, u_{\theta_Y})) \leq \varepsilon/2$.

Next, we prove "$\leq$". Let R be a correspondence between X and Y such that for all $(x,y),(x',y') \in R$, $|u_{\theta_X}(x,x') - u_{\theta_Y}(y,y')| \leq \varepsilon$, implying that $d_{\mathrm{GH}}((X, u_{\theta_X}), (Y, u_{\theta_Y})) \leq \varepsilon/2$. We wish to show that $\theta_X(t) \leq_R \theta_Y(t+\varepsilon)$ for all $t \in \mathbf{R}$. For $t < 0$, since $\theta_X(t) = \theta_Y(t) = \emptyset$, we trivially have $\theta_X(t) \leq_R \theta_Y(t+\varepsilon)$. Now pick any $t \geq 0$ and any $(x,y),(x',y') \in R$. Assume that x, x' belong to the same block of $\theta_X(t)$, implying that $u_{\theta_X}(x,x') \leq t$. Since $|u_{\theta_X}(x,x') - u_{\theta_Y}(y,y')| \leq \varepsilon$, we know $u_{\theta_Y}(y,y') \leq t+\varepsilon$, and hence y, y' belong to the same block of $\theta_Y(t+\varepsilon)$. Therefore, $\theta_X(t) \leq_R \theta_Y(t+\varepsilon)$ for all $t \in \mathbf{R}$. By symmetry, $\theta_Y(t) \leq_{R^{-1}} \theta_X(t+\varepsilon)$ for all $t \in \mathbf{R}$ as well, completing the proof. □

Theorem 3 (Complexity of computing $d_{\mathrm{I}}^{\mathrm{F}}$) *Fix $\rho \in (1,6)$. It is not possible to obtain a ρ approximation to the distance $d_{\mathrm{I}}^{\mathrm{F}}\big((X,\theta_X),(Y,\theta_Y)\big)$ between formigrams in time polynomial on $|X|, |Y|, |\mathrm{crit}(\theta_X)|, |\mathrm{crit}(\theta_Y)|$ unless $P = NP$.*

Proof. Pick any two dendrograms θ_X and θ_Y and invoke Theorem 2 to reduce the problem to the computation of the Gromov-Hausdorff distance

$$\Delta := d_{\mathrm{GH}}((X, u_{\theta_X}), (Y, u_{\theta_Y}))$$

between the ultrametric spaces $(X, u_{\theta_X}), (Y, u_{\theta_Y})$ associated to the dendrograms. However, according to [22, Corollary 3.8], Δ cannot be approximated within any factor less than 3 in polynomial time, unless $P = NP$. The author shows this by observing that any instance of the 3-partition problem can be reduced to an instance of the bottleneck ∞-Gromov-Hausdorff distance (∞-BGHD) problem between ultrametric spaces (see [22, p.865]). The proof follows. □

4 Application: Visualization of clustering features of dynamic metric data

In this section we explain how to extract scale dependent clustering features from time-dependent metric spaces in the form of formigrams. Furthermore, we will show that this summarization process is stable under perturbations in the input time-dependent metric spaces.

4.1 Dynamic metric spaces (DMSs)

Recall that a pseudo-metric space is a pair (X, d_X) where X is a (non-empty) set and $d_X : X \times X \to \mathbf{R}_+$ is a symmetric function which satisfies the triangle inequality, and such that $d_X(x,x) = 0$ for all $x \in X$. d_X is called the pseudo-metric. Note that one does not require that $d_X(x,x') = 0$ implies that $x = x'$ like in the case of standard metric spaces.

Dynamic metric spaces (DMSs). A *dynamic metric space* is a pair $\gamma_X = (X, d_X(\cdot))$ where X is a non-empty finite set and $d_X : \mathbf{R} \times X \times X \to \mathbf{R}_+$ satisfies:

1. For every $t \in \mathbf{R}$, $\gamma_X(t) = (X, d_X(t))$ is a pseudo-metric space.
2. There exists $t_0 \in \mathbf{R}$ such that $\gamma_X(t_0)$ is a (standard) metric space.
3. For fixed $x, x' \in X$, $d_X(\cdot)(x,x') : \mathbf{R} \to \mathbf{R}_+$ is continuous.

We refer to t as the *time* parameter. Condition 2 above is assumed since otherwise one could substitute the DMSs γ_X by another DMSs $\gamma_{X'}$ over a set X' which satisfies $|X'| < |X|$, and such that $\gamma_{X'}$ is point-wisely equivalent to γ_X.

A family of examples of DMSs is given by n particles/animals moving continuously inside an environment $\Omega \subset \mathbf{R}^d$ where particles are allowed to coalesce. If the n trajectories are $p_1(t), \ldots, p_n(t) \in \mathbf{R}^d$, then let

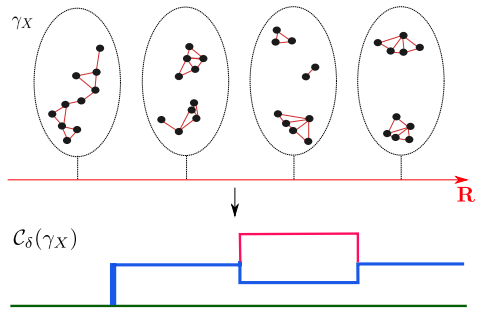


Figure 4: Top: A collection of moving particles (a DMS γ_X) is depicted over the time $\mathbf{R}$. In particular, any two points are connected by an edge if their distance does not exceed a certain $\delta > 0$. Bottom: The formigram $\mathcal{C}_\delta(\gamma_X)$ summarizes the clustering features of γ_X at the scale δ.

$P := \{1, \ldots, n\}$ and define a DMS $\gamma_P := (P, d_P(\cdot))$ as follows: for $t \in \mathbf{R}$ and $i, j \in \{1, \ldots, n\}$, let $d_P(t)(i,j) := \|p_i(t) - p_j(t)\|$, where $\|\cdot\|$ denotes the Euclidean norm.

Tame DMSs. We introduce a notion of *tameness* of DMS which will ultimately ensure that one can associate formigrams to tame DMSs. We first define *tame* functions $f : \mathbf{R} \to \mathbf{R}$: a continuous function $f : \mathbf{R} \to \mathbf{R}$ is *tame*, if for any $c \in \mathbf{R}$ and any finite interval $I \subset \mathbf{R}$, the set $f^{-1}(c) \cap I \subset \mathbf{R}$ is empty or has only finitely many connected components. For instance, polynomial functions (in particular, constant functions) and piecewise linear functions (with locally finitely many critical points) on $\mathbf{R}$ are tame. We say that a DMS $\gamma_X = (X, d_X(\cdot))$ is *tame* if for any $x, x' \in X$ the function $d_X(\cdot)(x,x') : \mathbf{R} \to \mathbf{R}_+$ is tame.

4.2 δ-clustering method for DMSs

δ-Clustering Method. Let $\delta \geq 0$. Recall (flat) single linkage clustering: Given any finite (pseudo-)metric space (X, d_X), define the partition $\mathcal{C}_\delta(X, d_X) := X/\sim_\delta$ where $\sim_\delta$ stands for the equivalence relation on X defined by $x \sim_\delta x'$ if and only if there exists a sequence $x = x_1, x_2, \ldots, x_n = x'$ of points in X such that $d_X(x_i, x_{i+1}) \leq \delta$ for each $i \in \{1, \ldots, n-1\}$.

From DMSs to Formigrams. We describe the process that, given a connectivity parameter $\delta \geq 0$, associates a formigram to any tame DMS:

Theorem 4 *Let γ_X be a tame DMS and fix $\delta \geq 0$. Then, the function $\mathcal{C}_\delta(\gamma_X) : \mathbf{R} \to \mathcal{P}(X)$ defined by $\mathcal{C}_\delta(\gamma_X)(t) = \mathcal{C}_\delta(\gamma_X(t))$ for $t \in \mathbf{R}$ is a formigram.*

See Figure 4 for an illustration. We prove Theorem 4 in Appendix A.

4.3 Stability of δ-clustering method for DMSs.

It turns out that the construction of formigrams from DMSs described in Theorem 4 is stable under perturbations in the input DMSs under a certain notion of distance between DMSs described below. Structurally, this distance is a hybrid between the Gromov-Hausdorff distance and the interleaving distance [2, 8] for Reeb graphs [9].

A distance between DMSs. Let γ_X, γ_Y be DMSs and $\varepsilon \geq 0$. We say that γ_X and γ_Y are *ε-interleaved* if there exists a correspondence R between X and Y such that $(*)$ $\forall (x,y), (x',y') \in R$, $\forall t \in \mathbf{R}$,

1. $\min_{s \in [t]^\varepsilon} d_Y(s)(y,y') \leq d_X(t)(x,x')$ and,
2. $\min_{s \in [t]^\varepsilon} d_X(s)(x,x') \leq d_Y(t)(y,y')$.

When γ_X and γ_Y are ε-interleaved we write $\gamma_X \approx_\varepsilon \gamma_Y$. The *interleaving distance* between γ_X and γ_Y is defined by $d_{\mathrm{I}}^{\mathrm{dyn}}(\gamma_X, \gamma_Y) := \inf\{\varepsilon \geq 0 : \gamma_X \approx_\varepsilon \gamma_Y\}$. If γ_X and γ_Y are not ε-interleaved for any $\varepsilon \geq 0$, declare $d_{\mathrm{I}}^{\mathrm{dyn}}(\gamma_X, \gamma_Y) = +\infty$. Also, any correspondence R satisfying $(*)$ is called an *ε-correspondence* between γ_X and γ_Y.

In Appendix B, we show that $d_{\mathrm{I}}^{\mathrm{dyn}}$ is indeed an extended metric on DMSs (Theorem 6).

Theorem 5 (Stability theorem) *For any tame DMSs γ_X, γ_Y and any $\delta \geq 0$, let $\theta_X := \mathcal{C}_\delta(\gamma_X)$ and $\theta_Y := \mathcal{C}_\delta(\gamma_Y)$ as in Theorem 4. Then,*

$$d_{\mathrm{I}}^{\mathrm{F}}(\theta_X, \theta_Y) \leq d_{\mathrm{I}}^{\mathrm{dyn}}(\gamma_X, \gamma_Y).$$

See Appendix B for the proof of Theorem 5.

5 Conclusion and Discussion

We introduced formigrams: a generalization of the notion of dendrograms that is useful for characterizing and visualizing the clustering features of DMSs. We clarified a sufficient condition (tameness) for DMSs to admit a summarization as formigrams.

We also defined the distances $d_{\mathrm{I}}^{\mathrm{F}}$ and $d_{\mathrm{I}}^{\mathrm{dyn}}$ on formigrams and on DMSs, respectively, and showed that the δ-clustering method for DMSs is stable under perturbations in the input DMSs in terms of $d_{\mathrm{I}}^{\mathrm{F}}$ and $d_{\mathrm{I}}^{\mathrm{dyn}}$. Specifically, it is noteworthy that $d_{\mathrm{I}}^{\mathrm{F}}$ generalizes the Gromov-Hausdorff distance on dendrograms.

In [17], due to the high cost of computing $d_{\mathrm{I}}^{\mathrm{F}}$, we carry out a classification task for different flocking behaviors by making use of a tractable lower bound for $d_{\mathrm{I}}^{\mathrm{F}}$. The nature of this lower bound is related to zigzag persistence theory [5, 6]: One can find theoretical details in [16].

References

[1] M. Benkert, J. Gudmundsson, F. Hübner, and T. Wolle. Reporting flock patterns. *Computational Geometry*, 41(3):111–125, 2008.

[2] P. Bubenik and J. A. Scott. Categorification of persistent homology. *Discrete & Computational Geometry*, 51(3):600–627, 2014.

[3] K. Buchin, M. Buchin, M. J. van Kreveld, B. Speckmann, and F. Staals. Trajectory grouping structure. *JoCG*, 6(1):75–98, 2015.

[4] D. Burago, Y. Burago, and S. Ivanov. *A Course in Metric Geometry*, volume 33 of *AMS Graduate Studies in Math.* American Mathematical Society, 2001.

[5] G. Carlsson and V. De Silva. Zigzag persistence. *Foundations of computational mathematics*, 10(4):367–405, 2010.

[6] G. Carlsson, V. De Silva, and D. Morozov. Zigzag persistent homology and real-valued functions. In *Proceedings of the twenty-fifth annual symposium on Computational geometry*, pages 247–256. ACM, 2009.

[7] G. Carlsson and F. Mémoli. Characterization, stability and convergence of hierarchical clustering methods. *Journal of Machine Learning Research*, 11:1425–1470, 2010.

[8] F. Chazal, D. Cohen-Steiner, M. Glisse, L. J. Guibas, and S. Oudot. Proximity of persistence modules and their diagrams. In *Proc. 25th ACM Sympos. on Comput. Geom.*, pages 237–246, 2009.

[9] V. De Silva, E. Munch, and A. Patel. Categorified reeb graphs. *Discrete & Computational Geometry*, 55(4):854–906, 2016.

[10] J. Gudmundsson and M. van Kreveld. Computing longest duration flocks in trajectory data. In *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*, pages 35–42. ACM, 2006.

[11] J. Gudmundsson, M. van Kreveld, and B. Speckmann. Efficient detection of patterns in 2d trajectories of moving points. *Geoinformatica*, 11(2):195–215, 2007.

[12] Y. Huang, C. Chen, and P. Dong. Modeling herds and their evolvements from trajectory data. In *International Conference on Geographic Information Science*, pages 90–105. Springer, 2008.

[13] S.-Y. Hwang, Y.-H. Liu, J.-K. Chiu, and E.-P. Lim. Mining mobile group patterns: A trajectory-based approach. In *PAKDD*, volume 3518, pages 713–718. Springer, 2005.

[14] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. Discovery of convoys in trajectory databases. *Proceedings of the VLDB Endowment*, 1(1):1068–1080, 2008.

[15] P. Kalnis, N. Mamoulis, and S. Bakiras. On discovering moving clusters in spatio-temporal data. In *SSTD*, volume 3633, pages 364–381. Springer, 2005.

[16] W. Kim and F. Memoli. Stable signatures for dynamic metric spaces via zigzag persistent homology. *arXiv preprint arXiv:1712.04064*, 2017.

[17] W. Kim, F. Mémoli, and Z. Smith. https://research.math.osu.edu/networks/formigrams.

[18] I. Kostitsyna, M. J. van Kreveld, M. Löffler, B. Speckmann, and F. Staals. Trajectory grouping structure under geodesic distance. In *31st International Symposium on Computational Geometry, SoCG 2015, June 22-25, 2015, Eindhoven, The Netherlands*, pages 674–688, 2015.

[19] Z. Li, B. Ding, J. Han, and R. Kays. Swarm: Mining relaxed temporal moving object clusters. *Proceedings of the VLDB Endowment*, 3(1-2):723–734, 2010.

[20] F. Mémoli and G. Sapiro. A theoretical and computational framework for isometry invariant recognition of point cloud data. *Found. Comput. Math.*, 5(3):313–347, 2005.

[21] J. K. Parrish and W. M. Hamner. *Animal groups in three dimensions: how species aggregate.* Cambridge University Press, 1997.

[22] F. Schmiedl. Computational aspects of the Gromov–Hausdorff distance and its application in non-rigid shape matching. *Discrete & Computational Geometry*, 57(4):854–880, 2017.

[23] Z. Smith, S. Chowdhury, and F. Mémoli. Hierarchical representations of network data with optimal distortion bounds. In *Signals, Systems and Computers, 2016 50th Asilomar Conference on*, pages 1834–1838. IEEE, 2016.

[24] D. J. Sumpter. *Collective animal behavior.* Princeton University Press, 2010.

[25] A. van Goethem, M. J. van Kreveld, M. Löffler, B. Speckmann, and F. Staals. Grouping time-varying data for interactive exploration. In *32nd International Symposium on Computational Geometry, SoCG 2016, June 14-18, 2016, Boston, MA, USA*, pages 61:1–61:16, 2016.

[26] M. J. van Kreveld, M. Löffler, and F. Staals. Central trajectories. *Journal of Computational Geometry*, 8(1):366–386, 2017.

[27] M. J. van Kreveld, M. Löffler, F. Staals, and L. Wiratma. A refined definition for groups of moving entities and its computation. In *27th International Symposium on Algorithms and Computation, ISAAC 2016, December 12-14, 2016, Sydney, Australia*, pages 48:1–48:12, 2016.

[28] M. R. Vieira, P. Bakalov, and V. J. Tsotras. On-line discovery of flock patterns in spatio-temporal data. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 286–295. ACM, 2009.

[29] Y. Wang, E.-P. Lim, and S.-Y. Hwang. Efficient algorithms for mining maximal valid groups. *The VLDB JournalThe International Journal on Very Large Data Bases*, 17(3):515–535, 2008.

[30] Wikipedia. Formicarium — Wikipedia, the free encyclopedia, 2017. [Online; accessed 03-June-2017].

Appendix A

Proof of Theorem 1.

Proof. Symmetry of $d_{\mathrm{I}}^{\mathrm{F}}$ is clear and thus we only show reflexivity of $d_{\mathrm{I}}^{\mathrm{F}}$ and the triangle inequality. Let X be any finite set and let θ_X be a formigram over X. Then, one can easily check that $R_X := \{(x,x) : x \in X\}$ is a 0-correspondence between two copies of θ_X, implying that $d_{\mathrm{I}}^{\mathrm{F}}(\theta_X, \theta_X) = 0$.

Let Y and Z be some finite sets and let θ_Y and θ_Z be any formigrams over Y and Z, respectively. We wish to prove that $d_{\mathrm{I}}^{\mathrm{F}}(\theta_X, \theta_Z) \leq d_{\mathrm{I}}^{\mathrm{F}}(\theta_X, \theta_Y) + d_{\mathrm{I}}^{\mathrm{F}}(\theta_Y, \theta_Z)$. We assume that $d_{\mathrm{I}}^{\mathrm{F}}(\theta_X, \theta_Y)$ and $d_{\mathrm{I}}^{\mathrm{F}}(\theta_Y, \theta_Z)$ are finite because otherwise there is nothing to prove. By this assumption, for some $0 < \varepsilon_1, \varepsilon_2 < \infty$, there are an ε_1-correspondence $R_1 \subset X \times Y$ between θ_X and θ_Y and an ε_2-correspondence $R_2 \subset Y \times Z$ between θ_Y and θ_Z. Define the set $R_2 \circ R_1 \subset X \times Z$ by

$$R_2 \circ R_1 := \{(x,z) \in X \times Z : \exists y \in Y \text{ s.t.} (x,y) \in R_1 \text{ and } (y,z) \in R_2\}.$$

It is not difficult to check that $R_2 \circ R_1$ is a correspondence between X and Z. Therefore, it suffices to prove that $R_2 \circ R_1$ is an $(\varepsilon_1 + \varepsilon_2)$-correspondence between θ_X and θ_Z.

Fix any $(x,z) \in R_2 \circ R_1$ and $t \in \mathbf{R}$. Suppose that x belongs to the underlying set of the sub-partition $\theta_X(t)$ of X. By the definition of $R_2 \circ R_1$, there exists $y \in Y$ such that $(x,y) \in R_1$ and $(y,z) \in R_2$. Since R_1 is an ε_1-correspondence between θ_X and θ_Y, y must be in the underlying set of $\bigvee_{[t]^\varepsilon} \theta_Y$. This implies that there exists $t_0 \in [t]^\varepsilon = [t-\varepsilon, t+\varepsilon]$ such that y belongs to the underlying set of $\theta_Y(t_0)$. Then, invoking that R_2 is an ε_2-correspondence between θ_Y and θ_Z, there exists $t_1 \in [t_0]^{\varepsilon_2} \subset [t]^{\varepsilon_1+\varepsilon_2}$ such that z belongs to the underlying set of $\theta_Z(t_1)$. This implies that z belongs to the underlying set of $\bigvee_{[t]^{\varepsilon_1+\varepsilon_2}} \theta_Z$. Similarly, one can check that if z belongs to the underlying set of $\theta_Z(t)$, then x belongs to the underlying set of $\bigvee_{[t]^{\varepsilon_1+\varepsilon_2}} \theta_X$.

Now, we wish to show that $\theta_X(t) \leq_{R_2 \circ R_1} \bigvee_{[t]^{\varepsilon_1+\varepsilon_2}} \theta_Z$. To this end, it suffices to show that for any $(x,z),(x',z') \in R_2 \circ R_1$, if x, x' belong to the same block of $\theta_X(t)$, then z, z' belong to the same block of $\bigvee_{[t]^{\varepsilon_1+\varepsilon_2}} \theta_Z$. Pick any $(x,z),(x',z') \in R_2 \circ R_1$ and suppose that x, x' belong to the same block of $\theta_X(t)$. By the definition of $R_2 \circ R_1$, there exist $y, y' \in Y$ such that $(x,y),(x',y') \in R_1$ and $(y,z),(y',z') \in R_2$. Since $\theta_X(t) \leq_{R_1} \bigvee_{[t]^{\varepsilon_1}} \theta_Y$, y, y' must be in the same block of $\bigvee_{[t]^{\varepsilon_1}} \theta_Y$. Recall that the sub-equivalence relation corresponding to the sub-partition $\bigvee_{[t]^{\varepsilon_1}} \theta_Y$ is the transitive closure of the relation $\cup_{s \in [t]^{\varepsilon_1}} \sim_s \subset Y \times Y$, where $\sim_s$ is the sub-equivalence relation (on Y) corresponding to the sub-partition $\theta_Y(s)$ of Y. In particular, the set $\{\sim_s : s \in [t]^{\varepsilon_1}\}$ consists of finitely many relations on Y due to the tameness of θ_Y. Therefore, there exist (finite) sequences $y = y_0, y_1, \ldots, y_n = y'$ in Y and $s_0, s_1, \ldots, s_{n-1}$ in $[t]^{\varepsilon_1}$ such that y_i, y_{i+1} belong to the same block of $\theta_Y(s_i)$ for $i = 0, \ldots, n-1$. Since R_2 is a correspondence between Y and Z, there exists a sequence $z = z_0, \ldots, z_n = z'$ in Z such that $(y_i, z_i) \in R_2$ for $i = 0, \ldots n$. Also, since $\theta_Y(s_i) \leq_{R_2} \bigvee_{[s_i]^{\varepsilon_2}} \theta_Z$, z_i, z_{i+1} belong to the same block of $\bigvee_{[s_i]^{\varepsilon_2}} \theta_Z$ for each i. Since $s_i \in [t]^{\varepsilon_1}$, we have $[s_i]^{\varepsilon_2} \subset [t]^{\varepsilon_1+\varepsilon_2}$, and in turn $\bigvee_{[s_i]^{\varepsilon_2}} \theta_Z \leq \bigvee_{[t]^{\varepsilon_1+\varepsilon_2}} \theta_Z$. Therefore, z_i, z_{i+1} belong to the same block of $\bigvee_{[t]^{\varepsilon_1+\varepsilon_2}} \theta_Z$ for each i, and hence z, z' belong to the same block of $\bigvee_{[t]^{\varepsilon_1+\varepsilon_2}} \theta_Z$. Similarly, one can verify that $\theta_Z(t) \leq_{(R_2 \circ R_1)^{-1}} \bigvee_{[t]^{\varepsilon_1+\varepsilon_2}} \theta_X$. □

Proof of Theorem 4.

Proof. We show that $\theta_X := \mathcal{C}_\delta(\gamma_X)$ satisfies the three conditions (tameness, interval lifespan, and comparability) to be a formigram. First, by the definition of $\mathcal{C}_\delta$, $\mathcal{C}_\delta(\gamma_X)$ is a function from $\mathbf{R}$ to the set of all partitions $\mathcal{P}(X) (\subset \mathcal{P}^{\mathrm{sub}}(X))$ of X. Therefore, every element $x \in X$ has the full lifespan $I_x = (-\infty, \infty)$, in θ_X.

Next we show the comparability condition. For simplicity, assume that $X = \{1, 2, \ldots, n\}$ for some $n \in \mathbf{N}$. Fix $c \in \mathbf{R}$ and consider the following two subsets of $X \times X$:

$$A(c,\delta) := \{(i,j) : \ i < j \in X, \ d_X(c)(i,j) \leq \delta\},$$

$$B(c,\delta) := \{(i,j) : \ i < j \in X, \ d_X(c)(i,j) > \delta\}.$$

The continuity of $d_X(\cdot)(i,j)$ for each $(i,j) \in X \times X$ guarantees that there exists $\varepsilon > 0$ such that

$$B(t,\delta) \supset B(c,\delta) \quad \text{for all } t \in (c-\varepsilon, c+\varepsilon)$$

and in turn

$$A(t,\delta) \subset A(c,\delta) \quad \text{for all } t \in (c-\varepsilon, c+\varepsilon)$$

since $A(t,\delta) \cup B(t,\delta) = \{(i,j) : i < j \in X\}$ for all $t \in \mathbf{R}$. This implies that the partition $\mathcal{C}_\delta(\gamma_X(c))$ is coarser than or equal to $\mathcal{C}_\delta(\gamma_X(t))$ for each $t \in (c-\varepsilon, c+\varepsilon)$, which means that $\mathcal{C}_\delta(\gamma_X)$ satisfies the comparability condition.

It remains to prove that $\mathcal{C}_\delta(\gamma_X)$ is tame. For $i, j \in X$, let $f_{i,j} := d_X(\cdot)(i,j) : \mathbf{R} \to \mathbf{R}_+$ and let $I \subset \mathbf{R}$ be any finite interval. Note that discontinuity points of the function $\mathcal{C}_\delta(\gamma_X) : \mathbf{R} \to \mathcal{P}(X)$ can occur only at endpoints of connected components of the set ${f_{i,j}}^{-1}(\delta)$ for some $i, j \in X$. Fix any $i, j \in X$. Then, since γ_X is tame, the set ${f_{i,j}}^{-1}(\delta) \cap I$ has only finitely many connected components and thus there are only finitely many endpoints arising from those components. Since the set X is finite, this implies that $\mathcal{C}_\delta(\gamma_X)$ can have only finitely many critical points in I. □

Appendix B

Isomorphic DMSs. We now introduce a notion of *equality* between two DMSs. Let $\gamma_X = (X, d_X(\cdot))$ and $\gamma_Y = (Y, d_Y(\cdot))$ be DMSs. We say that γ_X and γ_Y are *isomorphic* if there exists a bijection $\varphi : X \to Y$ such that φ is an isometry between $\gamma_X(t)$ and $\gamma_Y(t)$ across all $t \in \mathbf{R}$.

Theorem 6 *$d_{\mathrm{I}}^{\mathrm{dyn}}$ is an extended metric modulo isomorphisms between DMSs.*

We will prove Theorem 6 after showing Theorem 5.

Proof of Theorem 5.

Proof. First, note that for all $t \in \mathbf{R}$, X and Y are the underlying sets of $\theta_X(t)$ and $\theta_Y(t)$, respectively.

Let $\varepsilon \geq 0$ and assume that $R \subset X \times Y$ is any ε-correspondence between γ_X and γ_Y. It suffices to prove that R is an ε-correspondence between the formigrams θ_X and θ_Y

as well. Let $(x,y),(x',y') \in R$ and fix any $t \in \mathbf{R}$. Assume that x, x' belong to the same block of $\theta_X(t)$, meaning that there is a sequence $x = x_0, x_1, \ldots, x_n = x'$ in X such that $d_X(t)(x_i, x_{i+1}) \leq \delta$ for $0 \leq i \leq n-1$. For each $0 \leq i \leq n-1$, pick $y_i \in Y$ such that $(x_i, y_i) \in R$ where $y = y_0$ and $y' = y_n$. Since R is an ε-correspondence between γ_X, γ_Y, we have $\min_{s\in[t]^\varepsilon} d_Y(s)(y_i, y_{i+1}) \leq d_X(t)(x_i, x_{i+1}) \leq \delta$. This implies that, for each i, there is $s_i \in [t]^\varepsilon$ such that $d_Y(s_i)(y_i, y_{i+1}) \leq \delta$ and in turn y_i, y_{i+1} are in the same block of $\theta_Y(s_i)$. Also for each i, since $s_i \in [t]^\varepsilon$, one has $\theta_Y(s_i) \leq \bigvee_{[t]^\varepsilon} \theta_Y$ and in turn y_i, y_{i+1} belong to the same block of $\bigvee_{[t]^\varepsilon} \theta_Y$. Therefore, we conclude that y, y' belong to the same block of $\bigvee_{[t]^\varepsilon} \theta_Y$. We have proved that $\theta_X(t) \leq_R \bigvee_{[t]^\varepsilon} \theta_Y$. Similarly, $\theta_Y(t) \leq_{R^{-1}} \bigvee_{[t]^\varepsilon} \theta_X$ can be shown, completing the proof. □

Proof of Theorem 6.

Proof. Reflexivity and symmetry of $d_{\mathrm{I}}^{\mathrm{dyn}}$ are clear so we shall show the triangle inequality only: that for all DMSs $\gamma_X, \gamma_Y, \gamma_Z$, one has $d_{\mathrm{I}}^{\mathrm{dyn}}(\gamma_X, \gamma_Z) \leq d_{\mathrm{I}}^{\mathrm{dyn}}(\gamma_X, \gamma_Y) + d_{\mathrm{I}}^{\mathrm{dyn}}(\gamma_Y, \gamma_Z)$. We assume that $d_{\mathrm{I}}^{\mathrm{dyn}}(\gamma_X, \gamma_Y)$ and $d_{\mathrm{I}}^{\mathrm{dyn}}(\gamma_Y, \gamma_Z)$ are finite because otherwise there is nothing to prove. Let $0 < \varepsilon_1, \varepsilon_2 < \infty$ ans suppose that there are an ε_1-correspondence $R_1 \subset X \times Y$ between γ_X and γ_Y and an ε_2-correspondence $R_2 \subset Y \times Z$ between γ_Y and γ_Z. Define the correspondence $R_2 \circ R_1$ between X and Z as follows:

$$R_2 \circ R_1 := \{(x,z) \in X \times Z : \exists y \in Y \text{ s.t.} (x,y) \in R_1 \text{ and } (y,z) \in R_2\}.$$

Pick any two pairs (x,z) and (x',z') in $R_2 \circ R_1$. Then, there are $y, y' \in Y$ such that $(x,y),(x',y') \in R_1$ and $(y,z),(y',z') \in R_2$. Then for all $t \in \mathbf{R}$, it holds that

$$\min_{s\in[t]^{\varepsilon_1+\varepsilon_2}} d_Z(s)(z,z') \leq \min_{s\in[t]^{\varepsilon_1}} d_Y(s)(y,y') \leq d_X(t)(x,x'),$$
$$\min_{s\in[t]^{\varepsilon_2+\varepsilon_1}} d_X(s)(x,x') \leq \min_{s\in[t]^{\varepsilon_2}} d_Y(s)(y,y') \leq d_Z(t)(z,z').$$

Therefore, $R_2 \circ R_1$ is an $(\varepsilon_1+\varepsilon_2)$-correspondence between γ_X, γ_Z, implying that
$d_{\mathrm{I}}^{\mathrm{dyn}}(\gamma_X, \gamma_Z) \leq d_{\mathrm{I}}^{\mathrm{dyn}}(\gamma_X, \gamma_Y) + d_{\mathrm{I}}^{\mathrm{dyn}}(\gamma_Y, \gamma_Z)$, as desired.

Now, we show that $d_{\mathrm{I}}^{\mathrm{dyn}}$ is not just an (extended) pseudo-metric but an (extended) metric. Assume that $d_{\mathrm{I}}^{\mathrm{dyn}}(\gamma_X, \gamma_Y) = 0$ for some DMSs γ_X, γ_Y. Since there exist only finitely many correspondences between X and Y, there must exist a correspondence $R \subset X \times Y$ such that for any $\varepsilon > 0$, R is an ε-correspondence between γ_X and γ_Y. We claim that this R is a 0-correspondence. To this end, we need the following:

Claim. Let $f : \mathbf{R} \to \mathbf{R}$ be a continuous map and $r, t \in \mathbf{R}$. Suppose that for every $\varepsilon > 0$, $\min_{s\in[t]^\varepsilon} f(s) \leq r$. Then $f(t) \leq r$.

Proof. [Proof of Claim] For each $k \in \mathbf{N}$, take any $s_k \in [t]^{1/k}$ such that $f(s_k) \leq r$. Then $(s_k)_{k\in\mathbf{N}}$ is a sequence in $f^{-1}(-\infty, r]$ converging to t. Since f is continuous, $f^{-1}(-\infty, r]$ is a closed set and thus t must belong to $f^{-1}(-\infty, r]$, i.e. $f(t) \leq r$, as desired. □

Remember that whenever $x, x' \in X$, $y, y' \in Y$ are fixed, the distance functions $d_X(\cdot)(x,x'), d_X(\cdot)(y,y') : \mathbf{R} \to \mathbf{R}_+$ are continuous. Since R is an ε-correspondence for any $\varepsilon > 0$, it follows that for any $(x,y),(x',y') \in R$, for any $\varepsilon > 0$, and for any $t \in \mathbf{R}$,

1. $\min_{s\in[t]^\varepsilon} d_X(s)(x,x') \leq d_Y(t)(y,y')$,
2. $\min_{s\in[t]^\varepsilon} d_Y(s)(y,y') \leq d_X(t)(x,x')$.

Thus by **Claim**, for all $(x,y),(x',y') \in R$ and all $t \in \mathbf{R}$ it holds that $d_Y(t)(y,y') = d_X(t)(x,x')$. In addition, invoking that there exist $t_0, t_0' \in \mathbf{R}$ such that $\gamma_X(t_0)$ and $\gamma_Y(t_0')$ are (standard) metric spaces by the definition of DMSs, the correspondence R must be the graph of a bijection between X and Y. This implies that γ_X and γ_Y are isomorphic DMSs, as desired. □

Unfolding Low-Degree Orthotrees with Constant Refinement

Mirela Damian* Robin Flatland†

Abstract

We show that every orthotree of degree 3 or less can be unfolded with a 4×4 refinement of the grid faces. This is the first constant refinement unfolding result for orthotrees that are not required to be well-separated. Our approach shows promise of extending to arbitrary degree orthotrees.

1 Introduction

An *unfolding* of a polyhedron is obtained by cutting its surface in such a way that it can be flattened in the plane as a simple non-overlapping polygon called a *net*. An *edge unfolding* allows only cuts along the polyhedron's edges, while a *general unfolding* allows cuts anywhere on the surface. Edge cuts alone are not sufficient to guarantee an unfolding for non-convex polyhedra [BDE+03, BDD+98], however it is unknown whether all non-convex polyhedra have a general unfolding. In contrast, all convex polyhedra have a general unfolding [DO07, Sec. 24.1.1], but it is unknown whether they all have an edge unfolding [DO07, Ch. 22].

Prior work on unfolding algorithms for non-convex objects has focused on orthogonal polyhedra. This class consists of polyhedra whose edges and faces all meet at right angles. Because not all orthogonal polyhedra have edge unfoldings [BDD+98], the unfolding algorithms typically use additional non-edge cuts that follow one of two models. In the *grid unfolding model*, the surface is subdivided into rectangular *grid faces* by adding edges where axis-perpendicular planes through each vertex intersect the surface, and cuts along these added edges are also allowed. In the *grid refinement model*, each grid face under the grid unfolding model is further subdivided by an $(a \times b)$ orthogonal grid, for some positive integers $a, b \geq 1$, and cuts are also allowed along any of these grid lines.

A series of algorithms have been developed for unfolding arbitrary genus-0 orthogonal polyhedra, with each successive algorithm requiring less grid refinement. The first such algorithm [DFO07] required an exponential amount of grid refinement. This was reduced to quadratic refinement in [DDF14], and then to linear in [CY15]. These ideas were further extended in [DDFO17] to unfold arbitrary genus-2 orthogonal polyhedra with linear refinement.

The only unfolding algorithms for orthogonal polyhedra that use sublinear refinement are for specialized orthogonal shape classes. For example, there exist algorithms for unfolding orthostacks using 1×2 refinement [BDD+98] and Manhattan Towers using 4×5 refinement [DFO05]. There also exist unfolding algorithms for several classes of polyhedra composed of unit cubes. For example, orthotubes [BDD+98] and one layer block structures [LPW14] with an arbitrary number of unit holes can both be unfolded with cuts restricted to the cube edges.

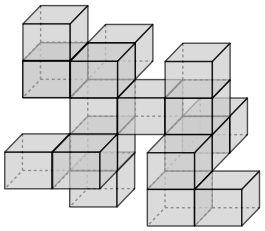


Figure 1: Orthotree of maximum degree three.

Our focus here is on the class of orthogonal polyhedra known as orthotrees. An *orthotree* $\mathcal{O}$ is composed of axis-aligned unit cubes (boxes) glued face to face, whose surface is a 2-manifold and whose dual graph $\mathcal{T}$ is a tree. (See Figure 1 for an example.) In the grid unfolding model, cuts are allowed along any of the cube edges. Each node in $\mathcal{T}$ is a box in $\mathcal{O}$ and two nodes are connected by an edge if the corresponding boxes are adjacent in $\mathcal{O}$ (i.e., if they share a face). In this paper we will use the terms *box* and *node* interchangeably. The *degree* of a box $b \in \mathcal{O}$ is defined as the degree of its corresponding node in the dual tree $\mathcal{T}$. We select any node of degree one to be the *root* of $\mathcal{T}$.

In an orthotree, each box can be classified as either a *leaf*, a *connector*, or a *junction*. A leaf is a box of degree one; a connector is a box of degree two whose two adjacent boxes are attached on opposite faces; all other boxes are junctions.

Because orthotrees are orthogonal polyhedra, they can be unfolded using the general algorithm in [CY15] with linear refinement. It is unknown whether orthotrees can be unfolded using sublinear refinement.

*Department of Computer Science, Villanova University, Villanova, PA, mirela.damian@villanova.edu

†Department of Computer Science, Siena College, Loudonville, NY, flatland@siena.edu

Prior algorithms specialized for unfolding orthotrees have been limited to orthotrees that are *well-separated*, meaning that no two junction boxes are adjacent. In [DFMO05], the authors provide an algorithm for grid unfolding well-separated orthotrees. Recent work in [HCY17] shows that the related class of well-separated orthographs (which allow arbitrary genus) can be unfolded with a 2×1 refinement.

In this paper we provide an algorithm for unfolding orthotrees of degree up to three using a 4×4 refinement of the cube faces. For each box b in $\mathcal{T}$, the algorithm unfolds b and the boxes in the subtree rooted at b recursively. Intuitively, the algorithm unfolds surface pieces of b along a carefully constructed path. When the path reaches a child box of b, the child is recursively unfolded and then the path continues on b again to the next child (if there is one). The unfolding of b and its subtree is contained within a rectangular region having two staircase-like bites taken out of it.

This is the first sublinear refinement unfolding result for orthotrees that are not required to be well-separated. Our algorithm can handle trees with adjacent junction boxes of degrees two or three, which other constant refinement algorithms are unable to do. In addition, the ideas used here show promise of extending to arbitrary degree orthotrees.

2 Terminology

For any box $b \in \mathcal{O}$, R_b and L_b are the *right* and *left* faces of b (orthogonal to the x-axis); F_b and K_b are the *front* and *back* faces of b (orthogonal to the z-axis); and T_b and B_b are the *top* and *bottom* faces of b (orthogonal to the y-axis). We use a different notation for boxes adjacent to b, to clearly distinguish them from faces: E_b and W_b are the *east* and *west* neighbors of b (adjacent to R_b and L_b, resp.); N_b and S_b are the *north* and *south* neighbors of b (adjacent to T_b and B_b, resp.); and I_b and J_b are the *front* and *back* neighbors of b (adjacent to F_b and K_b, resp.). We omit the subscript whenever the box b is clear from the context. We use combined notations to refer to the east neighbor of N as NE, the back neighbor of NE as NEJ, and so on.

If a face of a box $b \in \mathcal{O}$ is also a face of $\mathcal{O}$, we call it an *open face*; otherwise, we call it a *closed face*. On the closed face shared by b with its parent box in $\mathcal{T}$, we identify a pair of opposite edges, one called the *entry port* and the other called the *exit port* (shown in red and labeled in Figure 2). The unfolding of b is determined by an *unfolding path* that starts on b's entry port, recursively visits all boxes in the subtree $\mathcal{T}_b \subseteq \mathcal{T}$ rooted at b, and ends on b's exit port.

To make it easier to visualize the unfolding path, we use an L-shaped guide (or simply L-guide) with two orthogonal pointers, namely a HAND *pointer* and a HEAD *pointer*, as shown in Figure 2, where the circle is the HEAD and the arrow is the HAND. With very few exceptions, the unfolding path extends in the direction of one of the two pointers. Whenever the unfolding path follows the direction of the HAND, we say that it extends HAND-*first*; otherwise, it extends HEAD-*first*. Surface pieces traversed in the direction of the HAND(HEAD) will flatten out horizontally (vertically) in the plane. We denote by $\mathcal{N}_b$ the unfolding net produced by a recursive unfolding of b.

We refer the reader to Figure 2a which shows the unfolding path for the simple case of a leaf box A. The L-guide is shown positioned on top of A's parent box I at the entry port. The unfolding path extends HEAD-first around the top, back, and bottom faces of A, and ends on the bottom of I at the exit port. The resulting unfolding net $\mathcal{N}_A$ consisting of A's open faces T_A, K_A, B_A, L_A, and R_A is shown. In all unfolding illustrations, the outer surface of $\mathcal{O}$ is shown. When describing and illustrating the unfolding of a box A, we will assume without loss of generality that the box is in *standard position* (as in Figure 2a), with its parent I_A attached to its front face F_A and its entry (exit) port on the top (bottom) edge of F_A.

The *ring* r of a box b includes all the points on the surface of b (not necessarily on the surface of $\mathcal{O}$) that are within distance $0 < \delta \leq 1/4$ of the closed face shared with b's parent. Thus, r consists of four $1/4 \times 1$ rectangular pieces (which we call *ring faces*) connected in a cycle. The *entry box* b_e of b is the box containing the open face in $\mathcal{T} \setminus \mathcal{T}_b$ adjacent to b's entry port. Note that b_e may be b's parent (as in Figure 4a), but this is not necessary (see Figure 4b where b_e is the box on top of the parent I). The *entry ring* r_e of b includes all points of b_e that are within distance $1/4$ of the closed face of b_e adjacent to b's entry port. (See Figure 4.) The face e of r_e adjacent to b's entry port is the *entry ring face*. Similarly, the *exit box* b_x of b is the box containing the open face in $\mathcal{T} \setminus \mathcal{T}_b$ adjacent to b's exit port. Note that b_x is not necessarily b's parent (see Figure 4b, where b_x is the box south of the parent I). The *exit ring* r_x of b includes all points of b_x that are within distance $1/4$ of the closed face of b_x adjacent to b's exit port. The face x of r_x adjacent to b's exit port is the *exit ring face*. Note that both e and x are *open* ring faces (by definition). When unclear from context, we will use subscripts (i.e., e_b and x_b) to specify box b's entry and exit faces.

In a HEAD-first unfolding of a box b, the L-guide begins on the entry ring face with the HEAD pointing toward the entry port, and it ends on the exit ring face with the HEAD pointing away from the exit port; the HAND has the same orientation at the start and end of the unfolding. (See Figure 2a.) Similarly, in a HAND-first unfolding, the L-guide begins on the entry ring face with the HAND pointing toward the entry port, and it

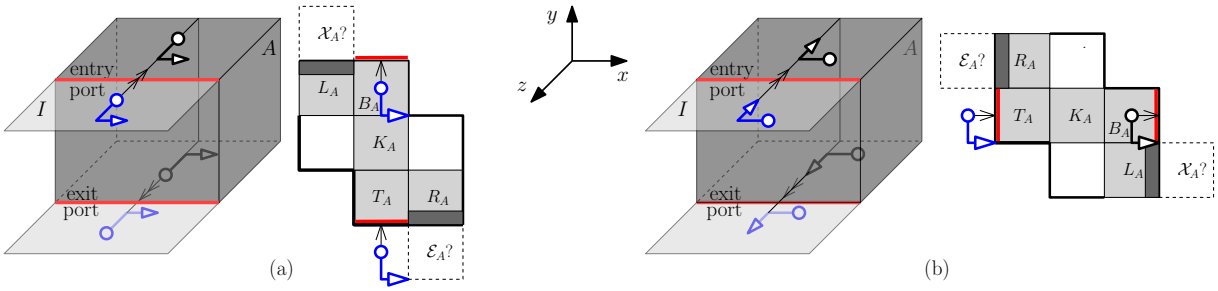


Figure 2: (a) HEAD-first and (b) HAND-first unfolding of leaf box.

ends on the exit ring face with the HAND pointing away from the exit port; the HEAD has the same orientation at the start and end of the unfolding. (See Figure 2b.) In standard position, the HAND in a HEAD-first unfolding will point either east or west. If it points east (west) we say that the unfolding is a HAND-east (west), HEAD-first unfolding. Similarly, in a HAND-first unfolding, the HEAD will either point east or west. If it points east (west), we say the unfolding is a HEAD-east (west), HAND-first unfolding.

In a HEAD-first (HAND-first) unfolding of b with entry ring face e, $\xrightarrow{e}$ is the ring face of r_e encountered immediately after e when cycling around r_e in the direction pointed to by the HAND(HEAD) of the L-guide as positioned on e at the start of b's unfolding. Similarly, in a HEAD-first (HAND-first) unfolding of b with exit ring face x, $\xleftarrow{x}$ is the ring face of r_x encountered just before x when cycling around r_x in the direction pointed to by the HAND(HEAD) of the L-guide as positioned on x at the end of b's unfolding path. Figure 4 shows $\xrightarrow{e}$ and $\xleftarrow{x}$ labeled. Note that although e and x are open ring faces by definition, $\xrightarrow{e}$ and $\xleftarrow{x}$ may not be open, as illustrated in Figure 4c.

3 Net Connections and Inductive Regions

Let $b \in \mathcal{T}$ be a box to be unfolded recursively. A HEAD-*first inductive region* for b is an orthogonally convex polygon shaped as in Figure 3a. Its bounding box is at least three units wide and at least three units tall. The lower (upper) convex vertex that lies strictly inside the bounding box is at unit vertical and horizontal distance from the lower left (upper right) corner of the bounding box, one unit away from the clockwise adjacent (reflex) vertex, and two units away from the counterclockwise adjacent (reflex) vertex. If the successor $\xrightarrow{e}$ of the entry ring face e is open, then the unit cell labeled $\mathcal{E}_b$ in Figure 3a is not part of the inductive region; otherwise, $\mathcal{E}_b$ is included as part of the inductive region. Similarly, if the predecessor $\xleftarrow{x}$ of the exit ring face x is open, then the unit cell labeled $\mathcal{X}_b$ in Figure 3a is not part of the inductive region; otherwise, $\mathcal{X}_b$ is included as part of the inductive region. (See Figure 4 for examples.) The *entry* (*exit*) port of the inductive region is the horizontal unit segment incident to the lower (upper) convex corner that lies strictly inside the bounding box of the region. A HEAD-first unfolding of b produces a net $\mathcal{N}_b$ that fits within the HEAD-first inductive region and whose entry and exit ports coincide with the entry and exit ports of the inductive region.

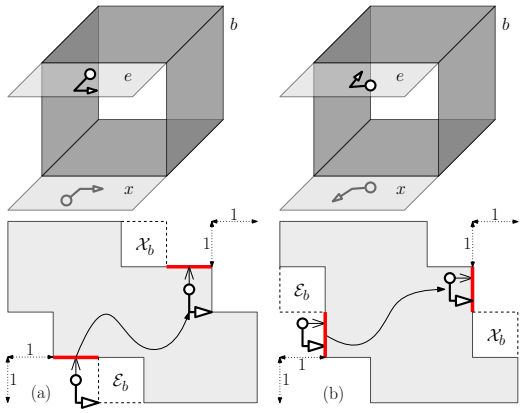


Figure 3: Inductive region for (a) HEAD-first unfolding (b) HAND-first unfolding

A HAND-*first inductive region* for b is an orthogonally convex polygon shaped as in Figure 3b. It is isometric to a HEAD-first inductive region, and one can be obtained from the other through a clockwise 90°-rotation, followed by a vertical reflection.

Lemma 1 *Let $\mathcal{N}_b$ be the unfolding net produced by a recursive* HAND*-east (west),* HEAD*-first recursive unfolding of b. If $\mathcal{N}_b$ is rotated clockwise by* 90° *and then reflected vertically, then the result is a* HEAD*-east (west),* HAND*-first recursive unfolding of b.*

Lemma 1 (whose proof is deferred to the appendix) enables us to focus the rest of our discussion on HEAD-first unfoldings only, and assume that the same results apply to the HAND-first unfoldings. Next we discuss the type of connections that each net must provide to

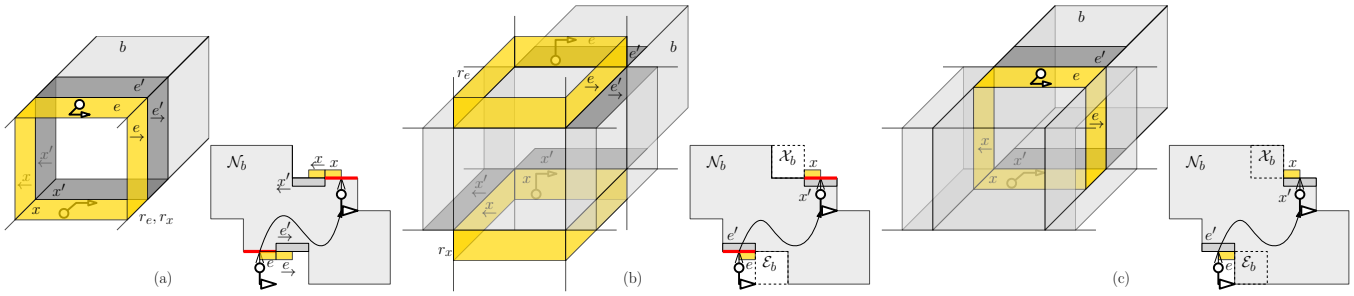


Figure 4: Entry/exit rings (in yellow) and entry/exit connections. (a) $\xrightarrow{e}$ and $\xleftarrow{x}$ open and adjacent to $\mathcal{T}_b$, type-2 entry and exit connections (type-1 entry connections would also be allowed here); (b) $\xrightarrow{e}$ and $\xleftarrow{x}$ non-adjacent to $\mathcal{T}_b$, type-1 entry and exit connections; (c) $\xrightarrow{e}$ and $\xleftarrow{x}$ closed: type-1 entry and exit connections; the squares labeled $\mathcal{E}_b$ and $\mathcal{X}_b$ belong to b's inductive region.

ensure it connects to the rest of $\mathcal{T}$'s unfolding. To do so, we need a few more definitions.

Let e' (x') be the open ring face of $\mathcal{T}_b$ that is adjacent to e (x) along the entry (exit) port. If $\xrightarrow{e}$ ($\xleftarrow{x}$) is open, let $\xrightarrow{e'}$ ($\xleftarrow{x'}$) be the open ring face adjacent to it along its side of unit length (see Figure 4). Note that, although e and $\xrightarrow{e}$ are ring faces from the same box by definition, ring faces e' and $\xrightarrow{e'}$ may be from different boxes (as in Figure 4b), and similarly for x' and $\xleftarrow{x'}$.

If b is not the root of $\mathcal{T}$, to ensure that b's net connects to the rest of $\mathcal{T}$'s unfolding, it must provide type-1 or type-2 connection pieces placed along the boundary inside its inductive region. These connections are defined as follows:

- A *type*-1 *entry connection* consists of the ring face e' placed alongside the entry port. (See Figure 4(b,c) for examples.)
- A *type*-1 *exit connection* consists of the ring face x' placed alongside the exit port. (See Figure 4(b,c) for examples.)
- A *type*-2 *entry connection* is used when the ring face $\xrightarrow{e}$ is open and adjacent to T_b. It consists of the ring face $\xrightarrow{e'}$ placed right of the entry port. (See Figure 4a for an example.)
- A *type*-2 *exit connection* is used when the ring face $\xleftarrow{x}$ is open and adjacent to $\mathcal{T}_b$. It consists of the ring face $\xleftarrow{x'}$ placed left of the exit port. (See Figure 4a for an example.)

Note that the unfolding of b begins on e (by definition) and is therefore adjacent to the entry port. The entry box b_e will provide a piece of e alongside the entry port of b's inductive region which connects to e' in b's net in a type-1 entry connection; for a type-2 entry connection, it places a piece of $\xrightarrow{e}$ next to e, which connects to $\xrightarrow{e'}$ in b's net. Examples of these pieces are shown in yellow along the boundary of the inductive regions in Figure 4. Similarly, the unfolding of b ends on x (by definition) and is therefore adjacent to the exit port. The exit box b_x will provide a piece of x alongside the exit port of b's inductive region which connects to x' in b's net in a type-1 exit connection; for a type-2 exit connection, it places a piece of $\xleftarrow{x}$ next to x, which connects to $\xleftarrow{x'}$ in b's net.

4 Inductive Hypothesis

We will make use of the following inductive hypothesis for the recursive unfolding of a box $b \in \mathcal{T}$ other than the root box:

(I1) The recursive HEAD-first (HAND-first) unfolding of b produces an unfolding net $\mathcal{N}_b$ that fits within a HEAD-first (HAND-first) inductive region and includes all open faces of $\mathcal{T}_b$, with cuts restricted to a 4×4 refinement of the box faces.

(I2) The unfolding net $\mathcal{N}_b$ provides the following entry and exit connections (see Figure 4):

  (a) If $\xrightarrow{e}$ is open and adjacent to a face in $\mathcal{T}_b$, then $\mathcal{N}_b$ provides either a type-1 or type-2 entry connection. Otherwise, $\mathcal{N}_b$ provides a type-1 entry connection.

  (b) If $\xleftarrow{x}$ is open and adjacent to a face in $\mathcal{T}_b$, then $\mathcal{N}_b$ provides either a type-1 or type-2 exit connection. Otherwise, $\mathcal{N}_b$ provides a type-1 exit connection.

(I3) Open faces of b's ring that are not used in N_b's entry and exit connections can be removed from $\mathcal{N}_b$ without disconnecting $\mathcal{N}_b$.

5 Unfolding Algorithm

Our unfolding algorithm uses an unfolding path that begins on the top face of the root box of $\mathcal{T}$, recursively

visits all nodes in the subtree rooted at the child of the root box, and ends on the bottom face of the root box. The following theorem shows how the inductive hypothesis can be used to derive our main result.

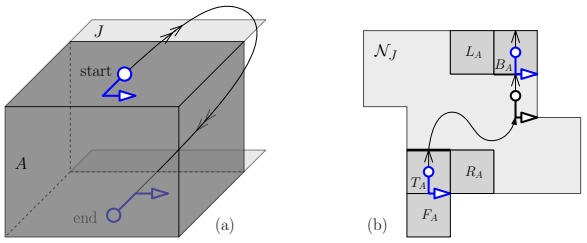


Figure 5: Head-first unfolding of root box A with back child J (a) unfolding path (b) unfolding net.

Theorem 2 *Let $\mathcal{O}$ be an orthotree of degree at most three. If the inductive hypothesis is met by all boxes in $\mathcal{T}$ other than the root box, then $\mathcal{O}$ can be unfolded into a net using a 4×4 refinement.*

Proof. Let $A \in T$ be the root of $\mathcal{T}$ (by definition, A is a node of degree one in $\mathcal{T}$). Assume A has a back child J (if this is not the case, reorient $\mathcal{O}$ to make this true). A recursive unfolding of $\mathcal{O}$ is depicted in Figure 5a: starting HEAD-first on the top face of A, the unfolding path recursively visits J and returns to the bottom face of A. The resulting net takes the shape depicted in Figure 5b.

Property (I2) applied to J tells us that $\mathcal{N}_J$ provides either type-1 or type-2 entry and exit connections. If of type-1, the entry (exit) connection attaches to T_A (B_A); otherwise, it attaches to R_A (L_A). In either case, the surface piece $\mathcal{N}_A$ depicted in Figure 5b is connected. Property (I1) applied to J tells us that $\mathcal{N}_J$ is a net that includes all open faces in the subtree $\mathcal{T}_J$ rooted at J and uses a 4×4 refinement. This along with the fact that the open faces of A attach to $\mathcal{N}_J$ without overlap shows that $\mathcal{N}_A$ is a net that uses a 4×4 refinement. □

The rest of the paper is devoted to proving that the inductive hypothesis holds for all boxes $A \in \mathcal{T}$ other than the root box. Lemma 1 allows us to restrict our attention to HEAD-first unfoldings only.

We discuss several cases depending on the node degree. The HEAD-first unfolding of a leaf node is depicted in Figure 2a, and it can be easily verified that this unfolding satisfies the inductive hypothesis. In the appendix we include a proof of this claim, and show that degree-2 nodes can be handled as degenerate cases of degree-3 nodes. Our analysis of degree-3 nodes is split into five different cases, depending on the position of A's children:

Case 3.1: E and J are children of A. The case where W and J are children of A is a vertical mirror plane reflection of this case.

Case 3.2: E and W are children of A.

Case 3.3: N and S are children of A.

Case 3.4: N and J are children of A. The case where S and J are children of A is a horizontal mirror plane reflection of this case, with the unfolding path traversed in the opposite direction.

Case 3.5: N and E are children of A. The case where N and W are children of A is a vertical mirror plane reflection of this case; the case where S and E are children of A is a horizontal mirror plane reflection of this case, with the unfolding path followed in the opposite direction; the case where S and W are children of A is a vertical mirror plane reflection of the case where S and E are children of A.

In this paper we discuss case 3.1 only, and defer the remaining cases to the appendix. Case 3.1 is handled by Lemma 5 and Theorem 6, which make use of the following two preliminary lemmas (whose proofs are deferred to the appendix).

Lemma 3 *Let $X \in \{E, W\}$ be a child of A. In a* HEAD*-first unfolding of A, if the* HAND *points in the direction of X (opposite to X), then the successor $\xrightarrow{e_A}$ (predecessor $\xleftarrow{x_A}$) of the entry (exit) ring face e_A (x_A) is open.*

Lemma 4 *Let ξ be the unfolding path and $\mathcal{N}$ the unfolding net produced by a recursive unfolding of a box in $\mathcal{T}$. Let $\overleftarrow{\xi}$ be the unfolding path traversed in reverse, starting at the exit port of $\mathcal{N}$ and ending at the entry port of $\mathcal{N}$, with the* HEAD *and* HAND *pointing in opposite direction. If $\mathcal{N}$ satisfies the inductive hypothesis, then the unfolding net induced by $\overleftarrow{\xi}$ also satisfies the inductive hypothesis.*

Lemma 5 *Let $A \in \mathcal{T}$ be a degree-3 node with parent I and children E and J. There is a* HAND*-east,* HEAD*-first unfolding of A whose net $\mathcal{N}_A$ satisfies the inductive hypothesis.*

Proof. One such unfolding is depicted in Figure 6a: starting at A's entry port, the unfolding path moves HEAD-first on T_A, then proceeds HAND-first to recursively visit E; from E's exit face on B_A, it proceeds HEAD-first to recursively visit J; from J's exit face on T_A, it moves HAND-first to L_A and B_A and then to A's exit port. We now show that, when visited in this order and laid flat in the plane, the open faces in $\mathcal{T}_A$ form a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.

We start by showing that $\mathcal{N}_A$ provides the appropriate entry and exit connection pieces. By Lemma 3, $\xrightarrow{e_A}$ is open, therefore $\mathcal{E}_A$ does not belong to A's inductive region (by definition). If $\xleftarrow{x_A}$ is closed (open), then $\mathcal{X}_A$ belongs (doesn't belong) to A's inductive region. This

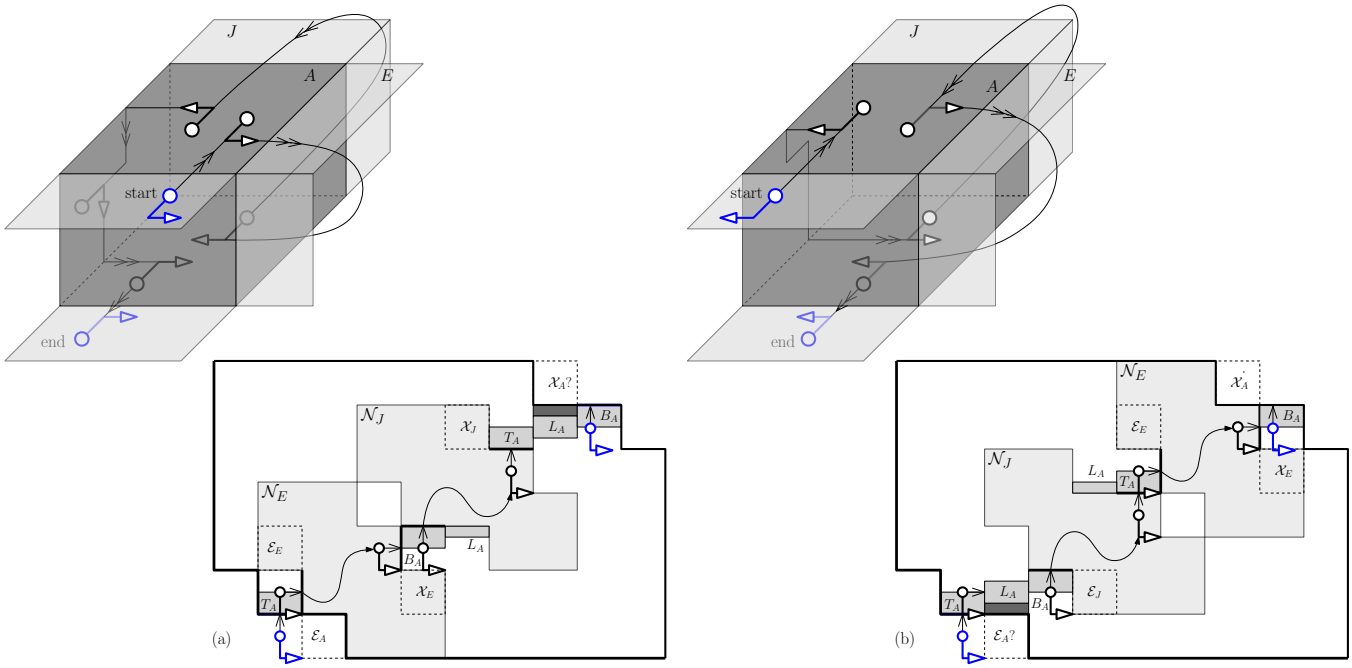


Figure 6: Box A with back and east children, HEAD-first unfolding (a) HAND pointing east (b) HAND pointing west.

dual case scenario is depicted by the cell labeled $\mathcal{X}_A?$ in Figure 6a. Observe that $\mathcal{N}_A$ provides type-1 entry and exit connections since $e'_A \in T_A$ and $x'_A \in B_A$ are positioned alongside the entry and exit ports. Thus $\mathcal{N}_A$ satisfies condition (I2) of the inductive hypothesis.

Turning now to condition (I1) of the inductive hypothesis, we begin by showing that the unfolding $\mathcal{N}_A$ from Figure 6a is connected. Note that $\xrightarrow{e_E} \in K_A$ and $\xleftarrow{x_E} \in F_A$ are both closed, so $\mathcal{E}_E$ and $\mathcal{X}_E$ belong to E's inductive region (by definition). The inductive hypothesis applied to N_E tells us that $\mathcal{N}_E$ provides type-1 entry and exit connections, and thus it connects to the pieces of $e_E \in T_A$ and $x_E \in B_A$ placed along $\mathcal{N}_E$'s boundary at the entry and exit ports.

Next we show that the net $\mathcal{N}_J$ produced by a recursive unfolding of J connects to the pieces of B_A, L_A and T_A placed alongside its boundary. First note that $\xrightarrow{e_J} \in L_A$ is open and therefore $\mathcal{E}_J$ does not belong to J's inductive region. Because $\xrightarrow{e_J}$ is adjacent to $\mathcal{T}_J$, the inductive hypothesis applied to $\mathcal{N}_J$ tells us that $\mathcal{N}_J$ provides a type-1 or type-2 entry connection. If type-1, then it connects to the piece $e_J \in B_A$; if type-2, then it connects to $\xrightarrow{e_J} \in L_A$. Also note that $\xleftarrow{x_J} \in R_A$ is closed, so by definition $\mathcal{X}_J$ is inside J's inductive region. The inductive hypothesis applied to $\mathcal{N}_J$ tells us that $\mathcal{N}_J$ provides a type-1 exit connection, which connects to $x_J \in T_A$. It follows that the net $\mathcal{N}_A$ is connected.

By the inductive hypothesis, $\mathcal{N}_E$ covers all faces in $\mathcal{T}_E$ and $\mathcal{N}_J$ covers all faces in $\mathcal{T}_J$, both using a 4×4 refinement. Observe that $\mathcal{N}_A$ includes all points in T_A, L_A and B_A (which are A's open faces) using a 4×4 refinement. Thus we conclude that $\mathcal{N}_A$ satisfies condition (I1) of the inductive hypothesis.

Turning to (I3) of the inductive hypothesis, observe that the only open ring face of A not used in A's entry or exit connections is part of L_A (this ring face is shown in dark gray in the unfolding in Figure 6a, left of the exit port). Its removal does not disconnect $\mathcal{N}_A$, and thus $\mathcal{N}_A$ satisfies condition (I3) of the inductive hypothesis. □

Theorem 6 *Any degree-3 node $A \in \mathcal{T}$ with children E and J can be unfolded into a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.*

Proof. Lemma 1 enables us to restrict our attention to HEAD-first unfoldings of A. If the HAND points east, then by Lemma 5 there is an unfolding net $\mathcal{N}_A$ that satisfies the inductive hypothesis. If the HAND points west, the unfolding follows the same path but in reverse direction (compare Figure 6a and Figure 6b) This along with Lemma 4 implies that the unfolding net from Figure 6b satisfies the inductive hypothesis. □

A complete unfolding example is included in section 10 of the appendix.

6 Conclusion

This paper presents the first result on unfolding orthotrees of degree 3 or less with a 4×4 refinement. Our preliminary investigations show promise of this approach extending to arbitrary degree orthotrees. It is open whether all orthotrees can be grid-unfolded without any refinements.

References

[BDD+98] Therese Biedl, Erik Demaine, Martin Demaine, Anna Lubiw, Mark Overmars, Joseph O'Rourke, Steve Robbins, and Sue Whitesides. Unfolding some classes of orthogonal polyhedra. In *Proceedings of the 10th Canadian Conference on Computational Geometry*, Montréal, Canada, August 1998.

[BDE+03] Marshall Bern, Erik D. Demaine, David Eppstein, Eric Kuo, Andrea Mantler, and Jack Snoeyink. Ununfoldable polyhedra with convex faces. *Computational Geometry: Theory and Applications*, 24(2):51–62, February 2003.

[CY15] Yi-Jun Chang and Hsu-Chun Yen. Unfolding orthogonal polyhedra with linear refinement. In *Proceedings of the 26th International Symposium on Algorithms and Computation, ISAAC 2015, Nagoya, Japan*, pages 415–425. Springer Berlin Heidelberg, 2015.

[DDF14] Mirela Damian, Erik D. Demaine, and Robin Flatland. Unfolding orthogonal polyhedra with quadratic refinement: the delta-unfolding algorithm. *Graphs and Combinatorics*, 30(1):125–140, 2014.

[DDFO17] Mirela Damian, Erik Demaine, Robin Flatland, and Joseph O'Rourke. Unfolding genus-2 orthogonal polyhedra with linear refinement. *Graph. Comb.*, 33(5):1357–1379, September 2017.

[DFMO05] Mirela Damian, Robin Flatland, Henk Meijer, and Joseph O'Rourke. Unfolding well-separated orthotrees. In *Abstracts from the 15th Annual Fall Workshop on Computational Geometry*, Philadelphia, PA, November 2005.

[DFO05] Mirela Damian, Robin Flatland, and Joseph O'Rourke. Unfolding Manhattan towers. In *Proceedings of the 17th Canadian Conference on Computational Geometry*, pages 211–214, Windsor, Canada, August 2005.

[DFO07] Mirela Damian, Robin Flatland, and Joseph O'Rourke. Epsilon-unfolding orthogonal polyhedra. *Graphs and Combinatorics*, 23(1):179–194, 2007.

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

[HCY17] Kuan-Yi Ho, Yi-Jun Chang, and Hsu-Chun Yen. Unfolding some classes of orthogonal polyhedra of arbitrary genus. In *The 23th International Computing and Combinatorics Conference (COCOON) 2017*, pages 275–287, 2017.

[LPW14] Meng-Huan Liou, Sheung-Hung Poon, and Yu-Jie Wei. On edge-unfolding one-layer lattice polyhedra with cubic holes. In *The 20th International Computing and Combinatorics Conference (COCOON) 2014*, pages 251–262, 2014.

Appendix

7 Proofs of Preliminary Lemmas

Lemma 1 *Let $\mathcal{N}_b$ be the unfolding net produced by a recursive* HAND*-east (west),* HEAD*-first recursive unfolding of b. If $\mathcal{N}_b$ is rotated clockwise by 90° and then reflected vertically, then the result is a* HEAD*-east (west),* HAND*-first recursive unfolding of b.*

Proof. Note that, when applied to the L-guide, this combined (90°-rotation, vertical reflection) transformation switches the HEAD and the HAND positions, so a HEAD-first orientation at the beginning (end) of the unfolding becomes HAND-first. This implies that, when applied to the unfolded net, the same transformation turns a HAND-east (west), HEAD-first unfolding into a HEAD-east (west), HAND-first unfolding. □

Lemma 3 *Let $X \in \{E, W\}$ be a child of A. In a* HEAD*-first unfolding of A, if the* HAND *points in the direction of X (opposite to X), then the successor $\xrightarrow{e_A}$ (predecessor $\xleftarrow{x_A}$) of the entry (exit) ring face e_A (x_A) is open.*

Proof. Consider first the case where the HAND points east and E is a child of A. If T_I is open, then $e_A \in T_I$ and $\xrightarrow{e_A} \in R_I$; in this case $\xrightarrow{e_A}$ is necessarily open, otherwise $IE \in \mathcal{O}$ would close a cycle (I, A, E, IE), contradicting the fact that $\mathcal{O}$ is an orthotree. If T_I is not open, then $NI \in \mathcal{O}$, $e_A \in K_{NI}$ and $\xrightarrow{e_A} \in R_{NI}$. As noted earlier, $IE \notin O$. This along with the fact that $\mathcal{O}$ is a 2-manifold implies that R_{NI} is open (otherwise $NIE \in \mathcal{O}$ either meets E at an edge or closes a cycle (NIE,EN,E,A,I,NI)). It follows that $\xrightarrow{e_A}$ is open. Similar arguments hold for the other cases. □

Lemma 4 *Let ξ be the unfolding path and $\mathcal{N}$ the unfolding net produced by a recursive unfolding of a box in $\mathcal{T}$. Let $\overleftarrow{\xi}$ be the unfolding path traversed in reverse, starting at the exit port of $\mathcal{N}$ and ending at the entry port of $\mathcal{N}$, with the* HEAD *and* HAND *pointing in opposite direction. If $\mathcal{N}$ satisfies the inductive hypothesis, then the unfolding net induced by $\overleftarrow{\xi}$ also satisfies the inductive hypothesis.*

Proof. The unfolding net $\overleftarrow{\mathcal{N}}$ induced by $\overleftarrow{\xi}$ is a diagonal flip (180°-rotation) of $\mathcal{N}$. It can be verified that the inductive hypothesis is invariant under 180°-rotations, therefore it holds for $\overleftarrow{\mathcal{N}}$ as well. □

8 Unfolding Degree-3 Nodes

In this section we discuss the unfoldings of cases 3.2 through 3.5 listed in Section 5.

Lemma 7 *Let $A \in \mathcal{T}$ be a degree-3 node with parent I and children E and W. There is a* HAND*-east,* HEAD*-first unfolding of A whose net $\mathcal{N}_A$ satisfies the inductive hypothesis.*

Proof. One such unfolding is depicted in Figure 7a. The unfolding path is similar to the one from Figure 6a, with the only difference that the recursive unfolding of J in Figure 6a is replaced with a straight path across the back face of A in Figure 7a, and the straight path across the west face of A in Figure 6a is replaced with a recursive unfolding of W in Figure 7a. We now show that, when visited in this order and laid flat in the plane, the faces in $\mathcal{T}_A$ form a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.

We start by considering the left/right boundaries of A's inductive region and show that A provides the appropriate entry and exit connection pieces. By Lemma 3 $\xrightarrow{e_A}$ and $\xleftarrow{x_A}$ are both open, therefore $\mathcal{E}_A$ and $\mathcal{X}_A$ are outside A's inductive region. Observe that $\mathcal{N}_A$ provides type-1 entry and exit connections, since $e'_A \in T_A$ and $x'_A \in B_A$ are placed alongside the entry and exit ports. Thus $\mathcal{N}_A$ satisfies condition (I2) of the inductive hypothesis.

Next we turn to condition (I1) of the inductive hypothesis. We begin by showing that the unfolding $\mathcal{N}_A$ from Figure 7a is connected. First note that $\xrightarrow{e_E} \in K_A$ is open, therefore $\mathcal{E}_E$ is outside E's inductive region. Because $\xrightarrow{e_E}$ is open and adjacent to $\mathcal{T}_E$, the inductive hypothesis applied to $\mathcal{N}_E$ tells us that $\mathcal{N}_E$ provides a type-1 or type-2 entry connection: if a type-1 entry connection, then it connects to $e_E \in T_A$; if a type-2 entry connection, then it connects to $\xrightarrow{e_E} \in K_A$. Also note that $\xleftarrow{x_E} \in F_A$ is closed, therefore $\mathcal{X}_E$ is inside E's inductive region. The inductive hypothesis applied to $\mathcal{N}_E$ tells us that $\mathcal{N}_E$ provides a type-1 exit connection, which connects to $x_E \in B_A$.

Next we show that the net $\mathcal{N}_W$ produced by a recursive unfolding of W connects to the pieces T_A, B_A and K_A placed alongside its boundary. First note that $\xrightarrow{e_W} \in F_A$ is closed, therefore $\mathcal{E}_W$ is inside W's inductive region. The inductive hypothesis applied to $\mathcal{N}_W$ tells us that $\mathcal{N}_W$ provides a type-1 entry connection, which connects to $e_W \in T_A$. Also note that $\xleftarrow{x_W} \in K_A$ is open, therefore $\mathcal{X}_W$ is outside W's inductive region. Because $\xleftarrow{x_W}$ is open and adjacent to $\mathcal{T}_W$, the inductive hypothesis applied to $\mathcal{N}_W$ tells us that $\mathcal{N}_W$ provides a type-1 or type-2 exit connection: if a type-1 exit connection, then it connects to $x_W \in B_A$; if a type-2 exit connection, then it connects to $\xleftarrow{x_W} \in K_A$. It follows that the entire net $\mathcal{N}_A$ is connected.

By the inductive hypothesis $\mathcal{N}_E$ covers all faces in $\mathcal{T}_E$ and $\mathcal{N}_W$ covers all faces in $\mathcal{T}_W$, both using a 4×4 refinement. Observe that $\mathcal{N}_A$ includes all points in T_A, K_A and B_A (which are A's open faces) using a 4×4 refinement. Thus we conclude that $\mathcal{N}_A$ includes all open

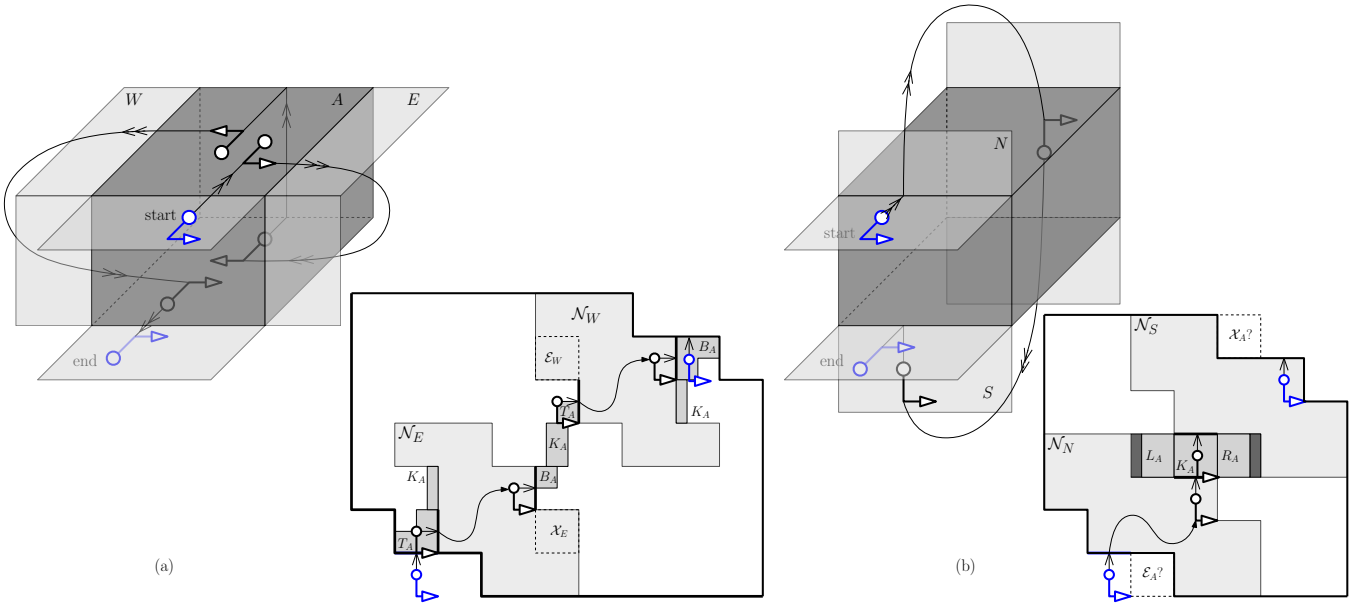


Figure 7: Box A of degree 3, HEAD-first unfolding (a) case when A has east and west children (b) case when A has north and south children.

faces of $\mathcal{T}_A$ and satisfies condition (I1) of the inductive hypothesis.

Condition (I3) of the inductive hypothesis is trivially satisfied, because only two ring faces of A are open, and they are both used in A's entry and exit connections. This concludes the proof. □

Theorem 8 *Any degree-3 box $A \in \mathcal{T}$ with children E and W can be unfolded into a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.*

Proof. Lemma 1 enables us to restrict our attention to HEAD-first unfoldings of A. If the HAND points east, then by Lemma 7 there is an unfolding net $\mathcal{N}_A$ that satisfies the inductive hypothesis. Observe that the case when the HAND points west is symmetric, so arguments analogous to those in Lemma 7 show that its net also satisfies the inductive hypothesis. □

Lemma 9 *Let $A \in \mathcal{T}$ be a degree-3 node with parent I and children N and S. There is a* HAND*-east,* HEAD*-first unfolding of A whose net $\mathcal{N}_A$ satisfies the inductive hypothesis.*

Proof. One such unfolding is depicted in Figure 7b: starting at the entry port on A, the unfolding path proceeds HEAD-first to recursively visit N, then crosses N's exit face K_A and proceeds HEAD-first to recursively visit S, ending at A's exit port. We now show that, when visited in this order and laid flat in the plane, the open faces in $\mathcal{T}_A$ form a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.

We start by considering the left/right boundaries of A's inductive region and show that A provides the appropriate entry and exit connection pieces. Note that the entry ports of A and N coincide, therefore the left boundaries of their inductive regions also coincide. If $\xrightarrow{e_A} = \xrightarrow{e_N} \in R_I$ is closed (open), then $\mathcal{E}_A = \mathcal{E}_N$ belongs (does not belong) to A and N's inductive region. This dual case scenario is depicted by the free cell labeled $\mathcal{E}_A$ in Figure 7b. Because $\xrightarrow{e_N}$ is not adjacent to $\mathcal{T}_N$, the inductive hypothesis tells us that N will provide a type-1 entry connection, which also serves as a type-1 entry connection for A since $e'_N = e'_A$. By analogous arguments, the right boundaries of A and S coincide, $\mathcal{X}_A = \mathcal{X}_S$ may or may not belong to their inductive regions, and S will provide a type-1 exit connection that also serves as a type-1 exit connection for A. Thus $\mathcal{N}_A$ satisfies condition (I2) of the inductive hypothesis.

Next we turn to condition (I1) of the inductive hypothesis. We begin by showing that the unfolding $\mathcal{N}_A$ from Figure 7b is connected. First note that $\xleftarrow{x_N} \in L_A$ is open, therefore $\mathcal{X}_N$ is outside N's inductive region. Because $\xleftarrow{x_N}$ is open and adjacent to $\mathcal{T}_N$, the inductive hypothesis applied to $\mathcal{N}_N$ tells us that $\mathcal{N}_N$ provides a type-1 or type-2 exit connection: if a type-1 exit connection, then it connects to $x_N \in K_A$; if a type-2 exit connection, then it connects to $\xleftarrow{x_N} \in L_A$.

Similarly, $\xrightarrow{e_S} \in R_A$ is open, therefore $\mathcal{E}_S$ is outside S's inductive region. Because $\xrightarrow{e_S}$ is open and adjacent to $\mathcal{T}_S$, the inductive hypothesis applied to $\mathcal{N}_S$ tells us that $\mathcal{N}_S$ provides a type-1 or type-2 entry connection: if a type-1 entry connection, then it connects to $e_S \in K_A$; if a type-2 entry connection, then it connects to $\xrightarrow{e_S} \in R_A$.

It follows that the entire net $\mathcal{N}_A$ is connected.

By the inductive hypothesis $\mathcal{N}_N$ covers all faces in $\mathcal{T}_N$ and $\mathcal{N}_S$ covers all faces in $\mathcal{T}_S$, both using a 4×4 refinement. Observe that $\mathcal{N}_A$ includes L_A, K_A and R_A (which are A's open faces) using a 4×4 refinement. Thus we conclude that $\mathcal{N}_A$ includes all open faces of $\mathcal{T}_A$ and satisfies condition (I1) of the inductive hypothesis.

Finally we turn to condition (I3) of the inductive hypothesis. Note that the open ring faces of A not involved in A's entry and exit connections are the dark-shaded pieces of L_A and R_A from Figure 7b, whose removal does not disconnect $\mathcal{N}_A$. Thus $\mathcal{N}_A$ satisfies (I3) as well. □

Theorem 10 *Any degree-3 node $A \in \mathcal{T}$ with children N and S can be unfolded into a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.*

Proof. Lemma 1 enables us to restrict our attention to HEAD-first unfoldings of A. If the HAND points east, then by Lemma 9 there is an unfolding net $\mathcal{N}_A$ that satisfies the inductive hypothesis. Observe that the case when the HAND points west is symmetric, so arguments analogous to those in Lemma 9 show that its net also satisfies the inductive hypothesis. □

Lemma 11 *Let $A \in \mathcal{T}$ be a degree-3 node with parent I and children N and J. If R_I is open, there is a* HAND*-east,* HEAD*-first unfolding of A whose net $\mathcal{N}_A$ satisfies the inductive hypothesis.*

Proof. One such unfolding is depicted in Figure 8a: starting from the entry face T_I, the unfolding path moves HAND-first to R_I, HEAD-first to R_A, then it cycles HAND-first around A to L_A; from there it proceeds HEAD-first to recursively visit J and then HAND-first to recursively visit N; from N's exit face L_A it moves HAND-first to B_A and then to A's exit port. We now show that, when visited in this order and laid flat in the plane, the open faces in $\mathcal{T}_A$ form a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.

We start by considering the left/right boundaries of A's inductive region and show that A provides the appropriate entry and exit connection pieces. By the lemma statement $\xrightarrow{e_A} \in R_I$ is open, therefore $\mathcal{E}_A$ is outside the inductive region for A. Because $\xrightarrow{e_A} \in R_I$ is adjacent to T_A, the inductive hypothesis allows $\mathcal{N}_A$ to provide a type-2 entry connection, which it does in the form of $\xrightarrow{e'_A} \in R_A$ placed right of $\mathcal{N}_A$'s entry port. If $\xleftarrow{x_A} \in L_I$ is open (closed), then $\mathcal{X}_A$ is outside (inside) the inductive region for A. This dual case scenario is depicted by the free cell labeled $\mathcal{X}_A$ in Figure 8a. Note that $\mathcal{N}_A$ provides a type-1 exit connection, because $x'_A \in B_A$ is placed alongside its exit port. Thus $\mathcal{N}_A$ satisfies condition (I2) of the inductive hypothesis.

Next we turn to condition (I1) of the inductive hypothesis. We begin by showing that the unfolding $\mathcal{N}_A$ from Figure 8a is connected. First note that $\xrightarrow{e_J} \in T_A$ is closed, therefore $\mathcal{E}_J$ belongs to the inductive region for J. The inductive hypothesis applied to $\mathcal{N}_J$ tells us that $\mathcal{N}_J$ provides a type-1 entry connection, which connects to $e_J \in L_A$. Also note that $\xleftarrow{x_J} \in B_A$ is open and adjacent to T_J. In this case the inductive hypothesis applied to $\mathcal{N}_J$ tells us that $\mathcal{N}_J$ provides a type-1 or type-2 exit connection: if type-1, it connects to $x_J \in R_A$; if type-2, it connects to $\xleftarrow{x_J} \in B_A$.

Next we show that the net $\mathcal{N}_N$ produced by a recursive unfolding of N connects to the pieces of R_A and L_A placed alongside its boundary. Since $\xrightarrow{e_N} \in F_A$ and $\xleftarrow{x_N} \in K_A$ are closed, $\mathcal{E}_N$ and $\mathcal{X}_N$ are inside the inductive region for N. The inductive hypothesis applied to N tells us that $\mathcal{N}_N$ provides type-1 entry and exit connections, therefore it connects to the pieces of $e_N \in R_A$ and $x_N \in L_A$ placed alongside $\mathcal{N}_N$'s entry and exit ports. It follows that the entire net $\mathcal{N}_A$ is connected.

By the inductive hypothesis $\mathcal{N}_J$ covers all faces in $\mathcal{T}_J$ and $\mathcal{N}_S$ covers all faces in $\mathcal{T}_S$, both using a 4×4 refinement. Observe that $\mathcal{N}_A$ includes all points in L_A, B_A and R_A (which are A's open faces) using a 4×4 refinement. Thus we conclude that $\mathcal{N}_A$ includes all open faces of $\mathcal{T}_A$ and satisfies condition (I1) of the inductive hypothesis.

Finally we turn to condition (I3) of the inductive hypothesis. Note that the only open ring face of A not involved in A's entry and exit connections is the dark-shaded piece of L_A from Figure 8a, whose removal does not disconnect $\mathcal{N}_A$. Thus $\mathcal{N}_A$ satisfied (I3) as well. □

Lemma 12 *Let $A \in \mathcal{T}$ be a degree-3 node with parent I and children N and J. If R_I is closed, there is a* HAND*-east,* HEAD*-first unfolding of A whose net $\mathcal{N}_A$ satisfies the inductive hypothesis.*

Proof. One such unfolding is depicted in Figure 8b: starting at the entry port on A, the unfolding path moves HEAD-first to F_N and cycles HAND-first around N to K_N; it then proceeds HEAD-first to recursively visit NN, HAND-first to recursively visit NW, and then HEAD-first to recursively visit J; from J's exit face B_J it moves HEAD-first to A's exit port. We now show that, when visited in this order and laid flat in the plane, the open faces in $\mathcal{T}_A$ form a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.

We start by considering the left/right boundaries of A's inductive region and show that A provides the appropriate entry and exit connection pieces. By the lemma statement $\xrightarrow{e_A} \in R_I$ is closed, therefore $\mathcal{E}_A$ is inside the inductive region for A. If $\xleftarrow{x_A} \in L_I$ is closed (open), then $\mathcal{X}_A$ is inside (outside) the inductive region for A. Note that $\mathcal{N}_A$ provides a type-1 entry and exit connection, because $e'_A \in F_N$ and $x'_A \in B_A$ are placed alongside its entry and exit ports. Thus $\mathcal{N}_A$ satisfies condition (I2) of the inductive hypothesis.

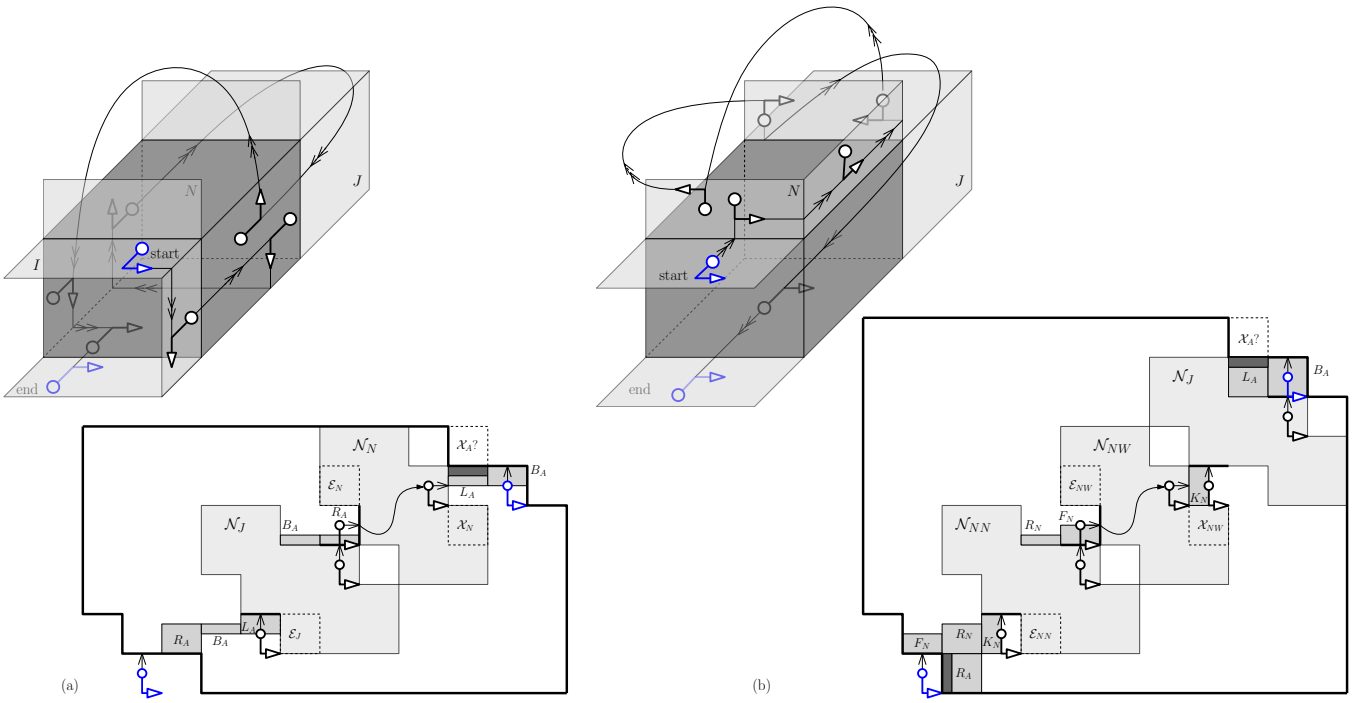


Figure 8: Box A of degree 3 with north and back children, HEAD-first unfolding (a) right face of I open (b) right face of I closed (so right face of N open).

Next we turn to condition (I1) of the inductive hypothesis. We begin by showing that the unfolding $\mathcal{N}_A$ from Figure 8b is connected. Here we assume the subtrees rooted at NN and NW are non-degenerate and thus consist of at least one box. (Handling cases in which one or both are degenerate requires only minor modifications.) By the lemma statement $\xrightarrow{e_{NN}} \in L_N$ is closed (and so $\mathcal{E}_{NN}$ is inside the inductive region for NN). The inductive hypothesis applied to $\mathcal{N}_{NN}$ tells us that $\mathcal{N}_{NN}$ provides a type-1 entry connection, which connects to $e_{NN} \in K_N$ placed alongside its entry port. Also note that $\xleftarrow{x_{NN}} \in R_N$ is open and adjacent to $\mathcal{T}_{NN}$ (and so $\mathcal{X}_{NN}$ is outside the inductive region for NN). In this case the inductive hypothesis applied to $\mathcal{N}_{NN}$ tells us that $\mathcal{N}_{NN}$ provides a type-1 or type-2 exit connection: if type-1, it connects to $x_{NN} \in F_N$; if type-2, it connects to $\xleftarrow{x_{NN}} \in R_N$.

Next we show that the net $\mathcal{N}_{NW}$ produced by a recursive unfolding of NW forms a connected component with the pieces of F_N and K_N placed alongside its boundary. Since $\xrightarrow{e_{NW}} \in B_N$ is closed, $\mathcal{E}_{NW}$ is inside the inductive region for NW. The inductive hypothesis applied to NW tells us that $\mathcal{N}_{NW}$ provides a type-1 entry connection, which connects to the piece of $e_{NW} \in F_N$ placed alongside its entry port. By the lemma statement $\xleftarrow{x_{NW}} \in T_N$ is closed (and so $\mathcal{X}_{NW}$ is inside the inductive region for NW). In this case the inductive hypothesis applied to $\mathcal{N}_{NW}$ tells us that $\mathcal{N}_{NW}$ provides a type-1 exit connection, which connects to $x_{NW} \in K_N$ placed alongside its exit port.

Next we show that the net $\mathcal{N}_J$ produced by a recursive unfolding of J connects to pieces of K_N, B_A, and L_A placed alongside its boundary. Note that $\xrightarrow{e_J} \in R_N$ is open (and so $\mathcal{E}_J$ is outside the inductive region for J) but not adjacent to T_J. The inductive hypothesis applied to J tells us that $\mathcal{N}_J$ provides a type-1 entry connection, which connects to $e_J \in K_N$ placed alongside its entry port. Also note that $\xleftarrow{x_J} \in L_A$ is open (and so $\mathcal{X}_J$ is outside J's inductive region) and adjacent to $\mathcal{T}_J$. The inductive hypothesis applied to $\mathcal{N}_J$ tells us that $\mathcal{N}_J$ provides a type-1 or type-2 exit connection: if type-1, it connects to $x_J \in B_A$; if type-2, it connects to $\xleftarrow{x_J} \in L_A$. It follows that the entire net $\mathcal{N}_A$ is connected.

By the inductive hypothesis $\mathcal{N}_{NN}$ covers all faces in $\mathcal{T}_{NN}$, $\mathcal{N}_{NW}$ covers all faces in $\mathcal{T}_{NW}$, and $\mathcal{N}_J$ covers all faces in $\mathcal{T}_J$, all using a 4×4 refinement. Observe that $\mathcal{N}_A$ includes all points in F_N, R_N and K_N (which are N's open faces) and in L_A, B_A and R_A (which are A's open faces), also using a 4×4 refinement. Thus we conclude that $\mathcal{N}_A$ includes all open faces of $\mathcal{T}_A$ and satisfies condition (I1) of the inductive hypothesis.

Finally we turn to condition (I3) of the inductive hypothesis. Note that the only open ring faces of A that are not involved in A's entry and exit connections are the dark-shaded piece of R_A and L_A from Figure 8b, whose removal does not disconnect $\mathcal{N}_A$. Thus $\mathcal{N}_A$ satisfied (I3) as well. □

Theorem 13 *Any degree-3 node $A \in \mathcal{T}$ with parent I*

and children N and J can be unfolded into a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.

Proof. Lemma 1 enables us to restrict our attention to HEAD-first unfoldings of A. If the HAND points east and R_I is open, then by Lemma 11 there is an unfolding net $\mathcal{N}_A$ that satisfies the inductive hypothesis; if R_I is closed, then by Lemma 12 there is an unfolding net $\mathcal{N}_A$ that satisfies the inductive hypothesis. Observe that the case when the HAND points west is symmetric, so arguments analogous to those in Lemma 11 (when L_I is open) and Lemma 12 (when L_I is closed) show that the theorem holds for this case as well. □

Lemma 14 *Let $A \in \mathcal{T}$ be a degree-3 node with parent I and children N and E. If K_N is open, there is a* HAND*-east,* HEAD*-first unfolding of A whose output net $\mathcal{N}_A$ satisfies the inductive hypothesis.*

Proof. One such unfolding is depicted in Figure 9a: starting at the entry port on A, the unfolding path proceeds HEAD-first to recursively visit N; from N's exit port on K_N it moves HAND-first to R_N, then proceeds HEAD-first to recursively visit E; from E's exit face B_A it moves in the direction opposite the HAND to K_A, then HEAD-first to L_A, HAND-first B_A and then to A's exit port. We now show that, when visited in this order and laid flat in the plane, the open faces in $\mathcal{T}_A$ form a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.

We start by considering the left/right boundaries of A's inductive region and show that A provides the appropriate entry and exit connection pieces. Observe that the entry ports of A and N coincide, and thus the left boundaries of their inductive regions also coincide. By Lemma 3, $\xrightarrow{e_N} = \xrightarrow{e_A} \in R_I$ is open, therefore $\mathcal{E}_N = \mathcal{E}_A$ does not belong to the inductive region of N and A. Because $\xrightarrow{e_N}$ is not adjacent to T_N, the inductive hypothesis tells us that N will provide a type-1 entry connection, which is also a type-1 entry connection for A (since $e'_N = e'_A$). Now consider the right side of A's inductive region. Observe that $\xleftarrow{x_A}$ may or may not be open, therefore $\mathcal{X}_A$ may or may not belong to the inductive region of A, as indicated by the free cell in Figure 9a. As shown in Figure 9a, A places ring face $x'_A \in B_A$ adjacent to its exit port, and thus provides a type-1 exit connection. Therefore, $\mathcal{N}_A$ satisfies condition (I2) of the inductive hypothesis.

We now show that the net $\mathcal{N}_A$ is itself connected. First note that $\mathcal{X}_N$ is not part of N's inductive region because $\xleftarrow{x_N} \in L_A$ is open. Because $\xleftarrow{x_N}$ is adjacent to $\mathcal{T}_N$, it seems at first that N could provide a type-1 or type-2 exit connection. However, we can argue in this situation N must provide a type-1 exit connection. Specifically, if N is of degree 1 or 2, then it has a type-1 exit connection because all degree 1 and 2 unfoldings do [1]. If N is of degree 3, then it must be Case 3.1. To see this, orient N in standard position[2] and observe that it can only have a west and back child; its other faces are either open or attached to its parent. Observe that all Case 3.1 unfoldings have type-1 exit connections. This implies that N has a type-1 exit connection which tells us that $x'_N \in K_N$ is positioned as shown under N's exit port. This attaches to the piece of R_N (taken from N) on the right. By (I3) of the inductive hypothesis applied to N, it is safe to remove this piece of R_N from $\mathcal{N}_N$ without disconnecting $\mathcal{N}_N$.

Next we show that the net $\mathcal{N}_E$ connects to R_N, K_A, and B_A placed around its boundary. First note that $\xrightarrow{e_E} \in F_N$ and $\xleftarrow{x_E} \in K_A$ are open, so $\mathcal{E}_E$ and $\mathcal{X}_E$ are not part of E's inductive region. Because $\xrightarrow{e_E}$ is not adjacent to $\mathcal{T}_E$, E has a type-1 entry connection, and therefore $e_E \in R_N$ placed along E's entry port is sufficient to make the connection to $\mathcal{N}_E$. Now consider $\xleftarrow{x_E}$. It is adjacent to $\mathcal{T}_E$, so E may have a type-1 or a type-2 exit connection. Therefore to ensure $\mathcal{N}_E$ is connected to the rest of the unfolding, both $x_E \in B_A$ and $\xleftarrow{x_E} \in K_A$ are placed as shown alongside E's inductive region. Thus we have shown that N_A is connected.

By the inductive hypothesis, $\mathcal{N}_N$ covers all faces in $\mathcal{T}_N$ (except for the piece of R_N used by A) and $\mathcal{N}_E$ covers all faces in $\mathcal{T}_E$, both using a 4×4 refinement. Observe that $\mathcal{N}_A$ includes all pieces of L_A, K_A, and B_A (which are A's open faces), also using a 4×4 refinement. Thus we conclude that $\mathcal{N}_A$ includes all open faces of $\mathcal{T}_A$ and satisfies (I1) of the inductive hypothesis.

For I3, observe that the only open ring face of A not used in A's entry or exit connections is part of L_A, shown in dark gray in Figure 9a. Its removal does not disconnect $\mathcal{N}_A$ so (I3) is satisfied. □

Lemma 15 *Let $A \in \mathcal{T}$ be a degree-3 node with parent I and children N and E. If K_N is closed, there is a* HAND*-east,* HEAD*-first unfolding of A whose output net $\mathcal{N}_A$ satisfies the inductive hypothesis.*

Proof. First note that K_N closed implies that K_E is open, because otherwise box EJ exists and shares an edge with NJ and so either box AJ or EJN exists (because the orthotree is a manifold), which implies a cycle in the orthotree. The unfolding for this case is depicted in Figure 9b: from the entry ring face on T_I, the unfolding path moves HAND-first to R_I, HEAD-first to E, then proceeds HAND-first to recursively visit ES, then HEAD-first to recursively visit EE; from EE's exit face F_E it moves HAND-first to T_E, then proceeds HEAD-first to recursively visit N; from N's exit face L_A it moves HAND-first to K_A, then HEAD-first to B_A, and finally to A's exit port. We now show that, when visited in this

[1] check this

[2] defined in Section 1: entry and exit ports are the top and bottom edges of the front face.

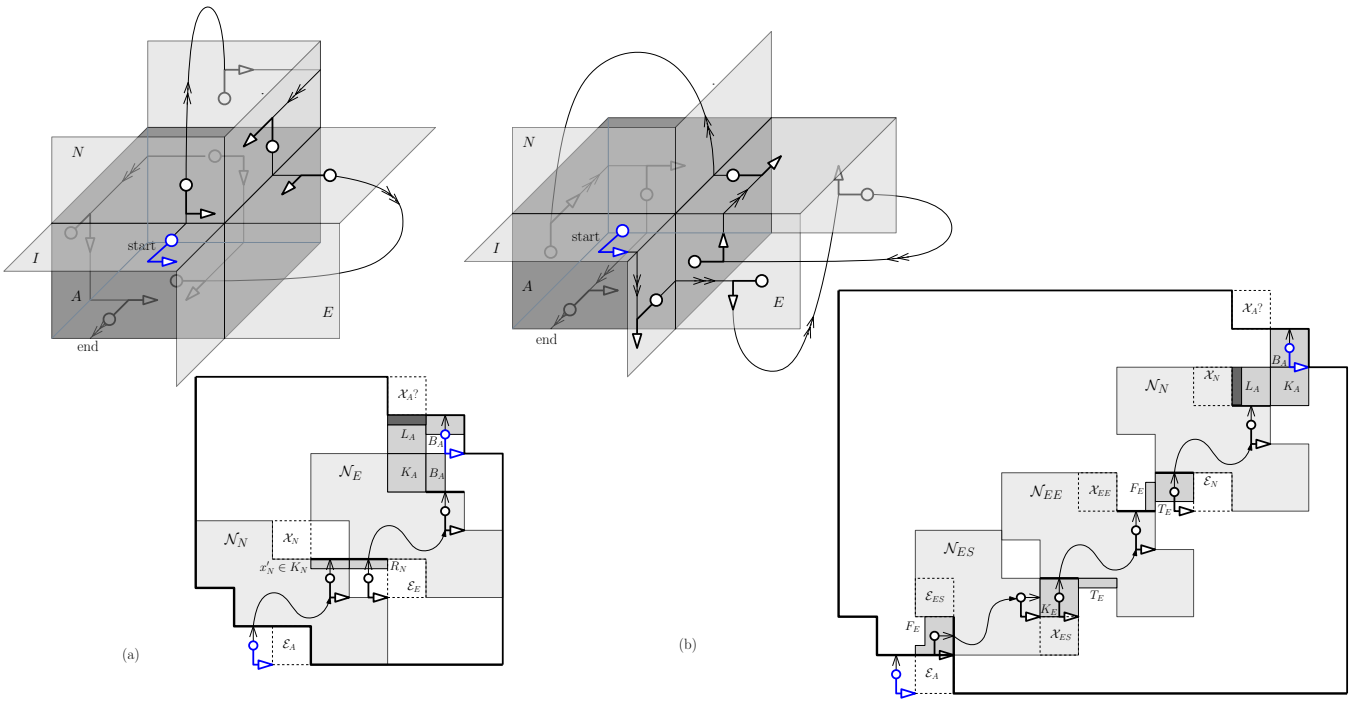


Figure 9: Box A of degree 3 with north and east children, HEAD-first unfolding, HAND pointing east (a) K_N open (b) K_N closed (and so K_E open).

order and laid flat in the plane, the open faces in $\mathcal{T}_A$ form a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.

We start by considering the left/right boundaries of A's inductive region and show that A provides the appropriate entry and exit connection pieces. By Lemma 3, $\xrightarrow{e_A} \in R_I$ is open, therefore $\mathcal{E}_A$ does not belong to A's inductive region. Because $\xrightarrow{e_A}$ is adjacent to T_A, the inductive hypothesis tells us that $\mathcal{N}_A$ can provide a type-2 entry connection, which it does in the form of $\xrightarrow{e'_A} \in F_E$ placed just to the right of the entry port. Now consider the right side of A's inductive region. Observe that $\xleftarrow{x_A}$ may or may not be open, therefore $\mathcal{X}_A$ may or may not belong to the inductive region of A, as indicated by the free cell in Figure 9b. Notice that A places the ring face $x'_A \in B_A$ adjacent to its exit port, and thus provides a type-1 exit connection. Therefore, $\mathcal{N}_A$ satisfies condition (I2) of the inductive hypothesis.

We now show that the net $\mathcal{N}_A$ is itself connected. We will assume that the subtrees rooted at ES and EE are non-degenerate and thus consist of at least one box. (Handling cases in which one or both are degenerate requires only minor modifications.) First note that $\mathcal{E}_{ES}$ and $\mathcal{X}_{ES}$ are both part of ES's inductive region because $\xrightarrow{e_{ES}} \in R_E$ and $\xleftarrow{x_{ES}} \in L_E$ are closed. The inductive hypothesis applied to ES tells us that $\mathcal{N}_{ES}$ provides type-1 entry and exit connections, which connect to pieces of $e_{ES} \in F_E$ and $x_{ES} \in K_E$.

Next we show that the net $\mathcal{N}_{EE}$ connects to K_E, T_E, and F_E placed around its boundary. First note that $\xrightarrow{e_{EE}} \in T_E$ is open, so $\mathcal{E}_{EE}$ is not part of EE's inductive region. Because $\xrightarrow{e_{EE}}$ is adjacent to $\mathcal{T}_{EE}$, the inductive hypothsis tells us that EE provides either a type-1 or type-2 entry connection, which attaches to $e_{EE} \in K_E$ (if type-1) or $\xrightarrow{e_{EE}} \in T_E$ (if type-2). Because $\xleftarrow{x_{EE}} \in B_E$ is closed, $\mathcal{X}_{EE}$ is part of EE's inductive region. The inductive hypothsis tells us that EE provides a type-1 exit connection, which attaches to the piece of $x_{EE} \in F_E$.

Next we show that the net $\mathcal{N}_N$ connects to T_E and L_A placed around its boundary. First note that $\xrightarrow{e_N} \in K_E$ is open, so $\mathcal{E}_N$ is not part of N's inductive region. Because $\xrightarrow{e_N}$ is not adjacent to $\mathcal{T}_N$, the inductive hypothsis tells us that $\mathcal{N}_N$ provides a type-1 entry connection, which attaches to $e_N \in T_E$. Because $\xleftarrow{x_N} \in F_A$ is closed, $\mathcal{X}_N$ is part of N's inductive region. The inductive hypothsis tells us that $\mathcal{N}_N$ provides a type-1 exit connection, which attaches to $x_N \in L_A$. Thus we have shown that N_A is connected.

By the inductive hypothesis, $\mathcal{N}_{ES}$, $\mathcal{N}_{EE}$, and $\mathcal{N}_N$ cover all faces in $\mathcal{T}_{ES}$, $\mathcal{T}_{EE}$ and $\mathcal{T}_N$, all using a 4×4 refinement. Observe that $\mathcal{N}_A$ includes all pieces of L_A, K_A, B_A, F_E, T_E, and K_E (which are A's and E's open faces), also using a 4×4 refinement. Thus we conclude that $\mathcal{N}_A$ includes all open faces of $\mathcal{T}_A$ and satisfies (I1) of the inductive hypothesis.

For condition (I3) of the inductive hypothesis, observe that the only open ring face of A not used in A's entry or exit connections is part of L_A, shown in dark gray

in Figure 9b. Its removal does not disconnect $\mathcal{N}_A$ so (I3) is satisfied. □

Lemma 16 *Let $A \in \mathcal{T}$ be a degree-3 node with parent I and children N and E. If B_I is open, then there is a* Hand*-west,* Head*-first unfolding of A whose output $\mathcal{N}_A$ satisfies the inductive hypothesis.*

Proof. One such unfolding is depicted in Figure 10a: from the entry port on A, the unfolding path proceeds Head-first to recursively visit N; from N's exit face K_A it moves Hand-first to L_A, Head-first to B_A, in the direction opposite the Hand to K_A, then proceeds Head-first to recursively visit E; from E's exit port it moves Head-first to R_I and then Hand-first to the exit face B_I. We now show that, when visited in this order and laid flat in the plane, the open faces in $\mathcal{T}_A$ form a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.

We start by considering the left/right boundaries of A's inductive region and show that A provides the appropriate entry and exit connection pieces. Observe that the entry ports of A and N coincide, and thus the left boundaries of their inductive regions also coincide. Also note that $\xrightarrow{e_N} = \xrightarrow{e_A} \in L_I$ may be open or closed, therefore $\mathcal{E}_N = \mathcal{E}_A$ may or may not belong to the inductive region of N and A, as indicated by the free cell in Figure 10a. Because $\xrightarrow{e_N}$ is not adjacent to T_N, the inductive hypothesis applied to N tells us that N will provide a type-1 entry connection $e_N \in F_N$, which is also a type-1 entry connection for A (since $e'_N = e'_A$).

Now consider the right side of A's inductive region. Observe that $\xleftarrow{x_A} \in R_I$ is open, therefore $\mathcal{X}_A$ does not belong to the inductive region of A. Because $\xleftarrow{x_A}$ is adjacent to $\mathcal{T}_A$, A can provide either a type-1 or type-2 exit connection, and in this case A provides a type-2 connection. To see this, observe that the right boundaries of E's and A's inductive regions overlap along E's exit port, which is also where A would place a type-2 connection piece. Because $\xleftarrow{x_E} \in T_I$ is open but not adjacent to $\mathcal{T}_E$, the inductive hypothesis tells us that E provides a type-1 exit connection $x'_E \in F_E$ placed under its exit port. Because $x'_E = \xleftarrow{x'_A}$, this piece also serves as a type-2 connection for A. Therefore, $\mathcal{N}_A$ satisfies condition (I2) of the inductive hypothesis.

We now show that the net $\mathcal{N}_A$ is itself connected. First note that $\mathcal{X}_N$ is part of N's inductive region because $\xleftarrow{x_N} \in R_A$ is closed. The inductive hypothesis applied to N tells us that N provides a type-1 exit connection, which attaches to the piece $x_N \in K_A$.

Next we show that the net $\mathcal{N}_E$ connects to K_A and B_A placed along its left boundary. First note that $\xrightarrow{e_E} \in B_A$ is open, so $\mathcal{E}_E$ is not part of E's inductive region. Because $\xrightarrow{e_E}$ is adjacent to $\mathcal{T}_E$, $\mathcal{N}_E$ may have a type-1 or type-2 entry connection which will connect to $e_E \in K_A$ (if type-1) or $\xrightarrow{e_E} \in B_A$ (if type-2). Thus we have shown that N_A is connected.

By the inductive hypothesis, $\mathcal{N}_N$ covers all faces in $\mathcal{T}_N$ and $\mathcal{N}_E$ covers all faces in $\mathcal{T}_E$, both using a 4×4 refinement. Observe that $\mathcal{N}_A$ includes all pieces of L_A, K_A, and B_A (which are A's open faces), also using a 4×4 refinement. Thus we conclude that $\mathcal{N}_A$ includes all open faces of $\mathcal{T}_A$ and satisfies (I1) of the inductive hypothesis.

For (I3), observe that the only open ring face of A not used in A's entry or exit connections is part of L_A, shown in dark gray in Figure 10a. Its removal does not disconnect $\mathcal{N}_A$ so (I3) is satisfied. □

Lemma 17 *Let $A \in \mathcal{T}$ be a degree-3 node with parent I and children N and E. If T_N is open, then there is a* Hand*-west,* Head*-first unfolding of A whose output $\mathcal{N}_A$ satisfies the inductive hypothesis.*

Proof. One such unfolding is depicted in Figure 10b: from the entry port on A, the unfolding path moves Head-first across F_N to T_N and then proceeds Hand-first to recursively visit NW; from NW's exit face L_A it moves Hand-first to B_A and Head-first to K_A, then proceeds Head-first to recursively visit NJ; from NJ's exit face T_N it moves Hand-first to R_N and then proceeds Hand-first to recursively visit E; and from E's exit face it reaches directly A's exit port. We now show that, when visited in this order and laid flat in the plane, the faces in $\mathcal{T}_A$ form a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.

We start by considering the left/right boundaries of A's inductive region and show that A provides the appropriate entry and exit connection pieces. Because $\xrightarrow{e_A} \in L_I$ may be open or closed, $\mathcal{E}_A$ may or may not belong to A's inductive region, as indicated by the free cell in Figure 10b. Because $\xleftarrow{x_A} \in R_I$ is open, $\mathcal{E}_A$ does not belong to A's inductive region. Observing that A provides type-1 entry and exit connections $e'_A \in F_N$ and $x'_A \in B_A$, we conclude that $\mathcal{N}_A$ satisfies condition (I2) of the inductive hypothesis.

We now show that $\mathcal{N}_A$ is itself connected. We will assume that the subtrees rooted at NW and NJ are non-degenerate and thus consist of at least one box. (Handling cases in which one or both are degenerate requires only minor modifications.) First note that $\mathcal{E}_{NW}$ and $\mathcal{X}_{NW}$ are both part of NW's inductive region because $\xrightarrow{e_{NW}} \in K_N$ and $\xleftarrow{x_{NW}} \in F_A$ are closed. The inductive hypothesis applied to $\mathcal{N}_{NW}$ tells us that it provides type-1 entry and exit connections, which connect to pieces $e_{NW} \in T_N$ and $x_{NW} \in L_A$ placed alongside its entry and exit ports.

Next we show that $\mathcal{N}_{NJ}$ connects to the pieces of K_A and T_N placed along its boundary. First note that $\xrightarrow{e_{NJ}} \in R_A$ and $\xleftarrow{x_{NJ}} \in L_N$ are closed, so $\mathcal{E}_{NJ}$ and $\mathcal{X}_{NJ}$ are part of NJ's inductive region. By the inductive

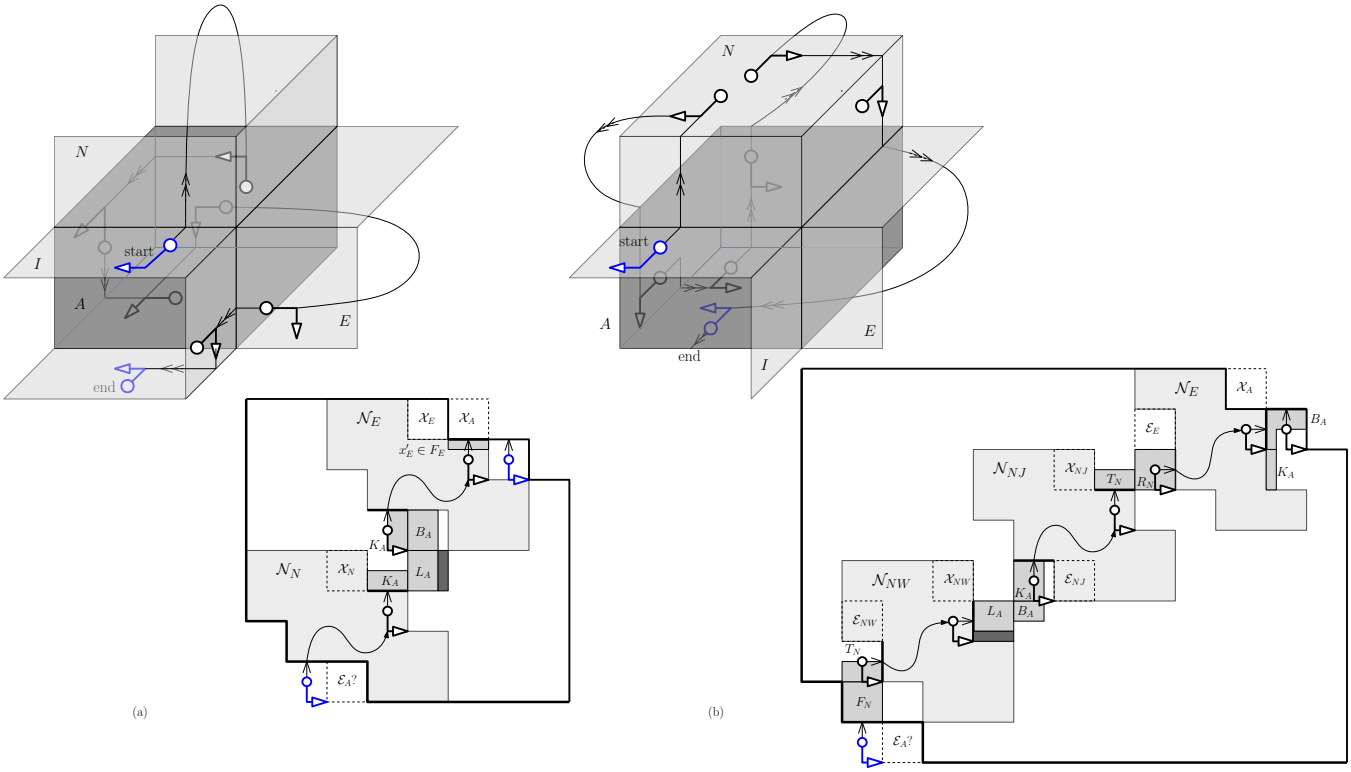


Figure 10: Box A of degree 3 with north and east children, HEAD-first unfolding, HAND pointing west. (a) B_I open (b) T_N open.

hypothesis, $\mathcal{N}_{NJ}$ provides type-1 entry and exit connections, which connect to $e_{NJ} \in K_A$ and $x_{NJ} \in T_N$.

Next we show that $\mathcal{N}_E$ connects to the pieces of R_N, B_A, and K_A placed along its boundary. First note that $\xrightarrow{e_E} \in F_N$ and $\xleftarrow{x_N} \in K_A$ are open, so $\mathcal{E}_E$ and $\mathcal{X}_E$ are not part of E's inductive region. Because $\xrightarrow{e_E}$ is not adjacent to $\mathcal{T}_E$, the inductive hypothesis tells us that E provides a type-1 entry connection, which connects to $e_E \in R_N$. Because $\xleftarrow{x_E}$ is adjacent to $\mathcal{T}_E$, the inductive hypothesis tells us that E provides a type-1 or type 2 exit connection: if type-1 it connects to $x_E \in B_A$ and if type-2 it connects to $\xleftarrow{x_E} \in K_A$. Thus we have shown that N_A is connected.

By the inductive hypothesis, $\mathcal{N}_{NW}$, $\mathcal{N}_{NJ}$, and $\mathcal{N}_E$ cover all faces in $\mathcal{T}_{NW}$, $\mathcal{T}_{NJ}$ and $\mathcal{T}_E$, all using a 4×4 refinement. Observe that $\mathcal{N}_A$ includes all pieces of L_A, K_A, B_A, F_N, T_N, and R_N (which are A's and N's open faces), also using a 4×4 refinement. Thus we conclude that $\mathcal{N}_A$ includes all open faces of $\mathcal{T}_A$ and satisfies (I1) of the inductive hypothesis.

For condition (I3) of the inductive hypothesis, observe that the only open ring face of A not used in A's entry or exit connections is part of L_A, shown in dark gray in Figure 10b. Its removal does not disconnect $\mathcal{N}_A$ so (I3) is satisfied. □

Lemma 18 *Let $A \in \mathcal{T}$ be a degree-3 node with parent I and children N and E. If T_N and B_I are both closed and K_N is open, there is a* HAND*-west,* HEAD*-first unfolding of A whose output $\mathcal{N}_A$ satisfies the inductive hypothesis.*

Proof. One such unfolding is depicted in Figure 11a: from the entry port on A, the unfolding path moves HEAD-first to F_N, then proceeds HAND-first to recursively visit NW, then HEAD-first to recursively visit NN; from NN's exit face F_N it moves HAND-first across R_N to K_N, HEAD-first to K_A, HAND-first to L_A, HEAD-first to B_A, in the direction opposite the HAND across K_A to K_N, HEAD-first to R_N, HAND-first to T_E, then HEAD-first across F_E to B_E; it then proceeds HAND-first to recursively visit EE, then HEAD-first to recursively visit EJ; finally, from EJ's exit face B_E, it moves across B_A to the exit port. We now show that, when visited in this order and laid flat in the plane, the faces in $\mathcal{T}_A$ form a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.

Arguments identical to the ones used in the proof of Lemma 17 show that A provides the appropriate entry and exit connection pieces. We now show that the net $\mathcal{N}_A$ is itself connected. We will assume that NW is not degenerate and therefore it consists of at least one box. (Handling the case where NW is degenerate requires only minor modifications.) First note that $\xrightarrow{e_{NW}} \in T_N$

and $\xleftarrow{x_{NW}} \in B_N$ are closed, so $\mathcal{E}_{NW}$ and $\mathcal{X}_{NW}$ are part of NW's inductive region. The inductive hypothesis tells us that NW provides type-1 entry and exit connections, which connect to pieces of $e_{NW} \in F_N$ (along the entry port) and $x_{NW} \in K_N$ (along the exit port).

Next we show that the net $\mathcal{N}_{NN}$ connects to the pieces of K_E, R_N, and F_N placed along its boundary. First note that $\xrightarrow{e_{NN}} \in R_N$ is open and adjacent to $\mathcal{T}_{NN}$, so $\mathcal{E}_{NN}$ is not part of NN's inductive region. The inductive hypothesis applied to NN tells us that $\mathcal{N}_{NN}$ provides either a type-1 or type-2 entry connection. If type-1, it connects to $e_{NN} \in K_N$ and if type-2, it connects to $\xrightarrow{e_{NN}} \in R_N$. Because $\xleftarrow{x_{NN}} \in L_N$ is closed, $\mathcal{X}_{NN}$ is part of NN's inductive region. The inductive hypothesis tells us that NN provides a type-1 exit connection, which attaches to $x_{NN} \in F_N$.

We now consider $\mathcal{N}_{EE}$ and $\mathcal{N}_{EJ}$ and show they connect to the pieces of B_E, T_E, and F_E placed along their boundaries. We will assume that their subtrees are non-degenerate and thus consist of at least one box. (Handling cases in which one or both are degenerate require only minor modifications.) First note that $\mathcal{E}_{EE}$ and $\mathcal{E}_{EJ}$ are part of their respective inductive regions because $\xrightarrow{e_{EE}} \in K_E$ and $\xrightarrow{e_{EJ}} \in L_E$ are closed. Similarly, $\mathcal{X}_{EJ}$ is part of EJ's inductive region because $\xleftarrow{x_{EJ}} \in R_E$ is closed. The inductive hypothesis applied to these two nets tells us that $\mathcal{N}_{EE}$ has a type-1 entry connection (which attaches to $e_{EE} \in B_E$), and $\mathcal{N}_{EJ}$ has a type-1 entry and exit connection (which attach to $e_{EJ} \in T_E$ and $x_{EJ} \in B_E$). Also note that $\mathcal{X}_{EE}$ is not part of EE's inductive region because $\xleftarrow{x_{EE}} \in F_E$ is open. The inductive hypothesis applied to $\mathcal{N}_{EE}$ tells us that it provides a type-1 or type-2 exit connection, which connects $x_{EE} \in T_E$ (if type-1) or $\xleftarrow{x_{EE}} \in F_E$ (if type-2). Thus we have shown that N_A is connected.

By the inductive hypothesis, $\mathcal{N}_{NW}$, $\mathcal{N}_{NN}$, $\mathcal{N}_{EE}$, and $\mathcal{N}_{EJ}$ cover all faces in $\mathcal{T}_{NW}$, $\mathcal{T}_{NN}$, $\mathcal{T}_{EE}$ and $\mathcal{T}_{EJ}$, all using a 4×4 refinement. Observe that $\mathcal{N}_A$ includes all open faces of A, N, and E, also using a 4×4 refinement. Thus we conclude that $\mathcal{N}_A$ includes all open faces of $\mathcal{T}_A$ and satisfies (I1) of the inductive hypothesis.

For condition (I3) of the inductive hypothesis, observe that the only open ring face of A not used in A's entry or exit connections is part of L_A, shown in dark gray in Figure 11a. Its removal does not disconnect $\mathcal{N}_A$ so (I3) is satisfied. □

Lemma 19 *Let $A \in \mathcal{T}$ be a degree-3 node with parent I and children N and E. If T_N, B_I and K_N are all closed, then there is a* HAND*-west,* HEAD*-first unfolding of A whose output $\mathcal{N}_A$ satisfies the inductive hypothesis.*

Proof. First note that T_{NJ} is open, because otherwise boxes $\{NJN, NN, N, NJ\}$ form a cycle. Also note that R_{NJ} is open, because otherwise box NJE exists and is edge-adjacent to E; this implies that box EJ must also exist (to ensure that the orthotree is a manifold), but this creates the cycle $\{NJ, NJE, EJ, E, A, N\}$.

An unfolding that satisfies the conditions of the lemma is depicted in Figure 11b: from A's entry port the unfolding path moves to F_N and proceeds HAND-first to recursively visit NW, then HAND-first to recursively visit NJ; from R_{NJ} it moves HEAD-first to T_{NJ} and proceeds HAND-first to recursively visit NN; from NN's exit face F_N it moves in the direction opposite the HEAD to R_N and then proceeds HAND-first to recursively visit E; and from E's exit face B_A it reaches A's exit port. We now show that, when visited in this order and laid flat in the plane, the faces in $\mathcal{T}_A$ form a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.

Arguments identical to the ones used in the proof of Lemma 17 show that A provides the appropriate entry and exit connection pieces. We now show that the net $\mathcal{N}_A$ is itself connected. We will assume that NW is not degenerate and therefore consists of at least one box. (Handling the case where $\mathcal{T}_{NW}$ is empty requires only minor modifications.) First note that $\xrightarrow{e_{NW}} \in T_N$ is closed, so $\mathcal{E}_{NW}$ is part of NW's inductive region. The inductive hypothesis tells us that $\mathcal{N}_{NW}$ provides a type-1 entry connection, which connects to a piece of $e_{NW} \in F_N$ placed adjacent to its entry port. Also note that $\xleftarrow{x_{NW}} \in B_{NJ}$ is open but not adjacent to NW, so $\mathcal{X}_{NW}$ is not part of the inductive region. The inductive hypothesis applied to NW tells us that $\mathcal{N}_{NW}$ provides a type-1 exit connection.

Now we show that the net $\mathcal{N}_{NJ}$ connects to $\mathcal{N}_{NW}$'s type-1 exit connection and to the piece of T_{NJ} placed along its boundary. First note that $\xrightarrow{e_{NJ}} \in T_{NW}$ is open (so $\mathcal{E}_{NJ}$ is not part of NJ's inductive region) but not adjacent to $\mathcal{T}_{NJ}$. The inductive hypothesis applied to NJ tells us that $\mathcal{N}_{NJ}$ provides a type-1 entry connection $e'_{NJ} \in L_{NJ}$, which is adjacent to $\mathcal{N}_{NW}$'s type-1 exit connection $x'_{NW} \in K_{NW}$, so the two nets are connected to each other. Because $\xleftarrow{x_{NJ}} \in B_N$ is closed, $\mathcal{X}_{NJ}$ is part of NJ's inductive region. The inductive hypothesis applied to NJ tells us that $\mathcal{N}_{NJ}$ provides a type-1 exit connection $x'_{NJ} \in R_{NJ}$, which attaches along the bottom of the ring face piece of T_{NJ} extracted from $\mathcal{N}_{NJ}$. Because this ring face piece is not used in the entry or exit connections of $\mathcal{N}_{NJ}$, removing it from $\mathcal{N}_{NJ}$ does not disconnect $\mathcal{N}_{NJ}$, by the inductive hypothesis (I3) applied to NJ.

We now consider $\mathcal{N}_{NN}$ and $\mathcal{N}_E$ and show they connect to the pieces of T_{NJ}, F_N, R_N, B_A and K_A placed along their boundaries. First note that $\mathcal{E}_{NN}$ and $\mathcal{E}_E$ are not part of their respective inductive regions because $\xrightarrow{e_{NN}} \in L_{NJ}$ and $\xrightarrow{e_E} \in F_N$ are open. In addition, $\xrightarrow{e_{NN}}$ is not adjacent to $\mathcal{T}_{NN}$ and $\xrightarrow{e_E}$ is not adjacent to $\mathcal{T}_E$. By the inductive hypothesis, $\mathcal{N}_{NN}$ and $\mathcal{N}_E$ both provide type-1 entry connections, which connect to the piece of

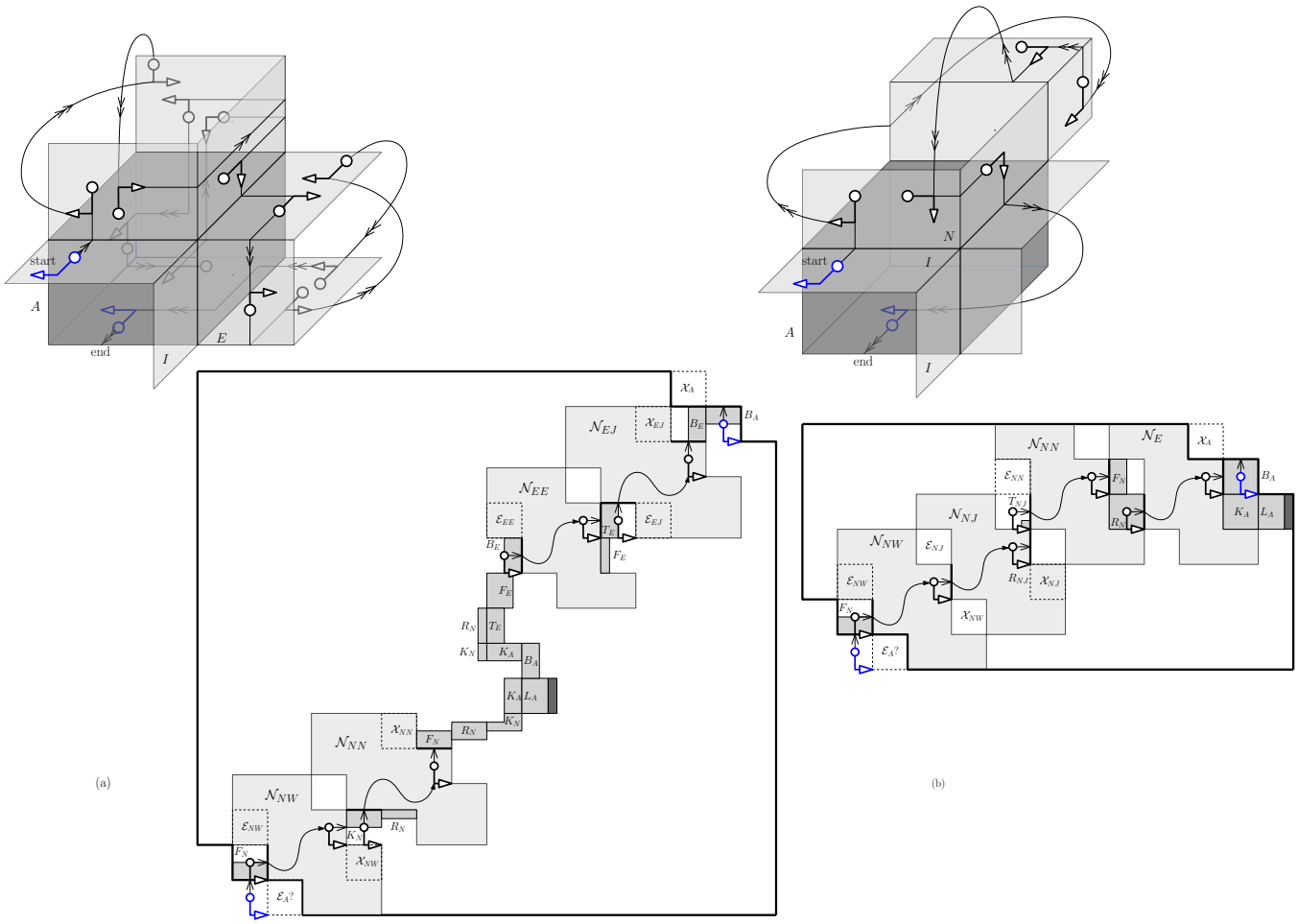


Figure 11: Box A of degree 3 with north and east children, HEAD-first unfolding, HAND pointing west, B_I closed (so B_E open) and T_N closed. (a) K_N open (b) K_N closed (and so T_{NJ} and E_{NJ} open).

$e_{NN} \in T_{NJ}$ and $e_E \in R_N$, respectively. Similarly, $\mathcal{X}_{NN}$ and $\mathcal{X}_E$ are also not part of their respective inductive regions because $\overleftarrow{x_{NN}} \in R_N$ and $\overleftarrow{x_E} \in K_A$ are both open. In addition, $\overleftarrow{x_{NN}}$ is adjacent to $\mathcal{T}_{NN}$ and $\overleftarrow{x_E}$ is adjacent to $\mathcal{T}_E$. By the inductive hypothesis, $\mathcal{N}_{NN}$ and $\mathcal{N}_E$ provide either type-1 or type-2 exit connections, therefore $\mathcal{N}_{NN}$ connects to F_N (if type-1) and R_N (if type-2), and $\mathcal{N}_E$ connects to B_A (if type-1) and K_A (if type-2). Thus we have shown that N_A is connected.

By the inductive hypothesis, $\mathcal{N}_{NW}$, $\mathcal{N}_{NJ}$, $\mathcal{N}_{NN}$, and $\mathcal{N}_E$ cover all faces in $\mathcal{T}_{NW}$, $\mathcal{T}_{NJ}$, $\mathcal{T}_{NN}$ and $\mathcal{T}_E$ using a 4×4 refinement, except for the piece of T_{NJ} that A uses. Observe that $\mathcal{N}_A$ includes all open faces of A and N, also using a 4×4 refinement. Thus we conclude that $\mathcal{N}_A$ includes all open faces of $\mathcal{T}_A$ and satisfies (I1) of the inductive hypothesis.

For condition (I3) of the inductive hypothesis, observe that the only open ring face of A not used in A's entry or exit connections is part of L_A, shown in dark gray in Figure 11b. Its removal does not disconnect $\mathcal{N}_A$ so (I3) is satisfied. □

Theorem 20 *Any degree-3 box $A \in \mathcal{T}$ with children N and E can be unfolded into a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.*

Proof. Lemma 1 enables us to restrict our attention to HEAD-first unfoldings of A. Consider first the case with the HAND pointing east. We discuss two cases, depending on whether K_N is open or closed. If K_N is open, by Lemma 14 there is an unfolding net $\mathcal{N}_A$ that satisfies the inductive hypothesis, so the theorem holds. This unfolding is depicted in Figure 9a. If K_N is closed, by Lemma 15 there is an unfolding net $\mathcal{N}_A$ that satisfies the inductive hypothesis, so the theorem holds as well. This unfolding is depicted in Figure 9b.

Consider now the case with the HAND pointing west. We discuss four cases, depending on whether B_I, T_N and K_N are open or closed. If B_I is open, the theorem holds by Lemma 16; the unfolding for this case is depicted in Figure 10a. If T_N is open, the theorem holds by Lemma 17; the unfolding for this case is depicted in Figure 10b. If B_I and T_N are both closed and K_N

is open, the theorem holds by Lemma 18; the unfolding for this case is depicted in Figure 11a. Finally, if B_I and T_N are both closed and K_N is closed, the theorem holds by Lemma 19; the unfolding for this case is depicted in Figure 11b. □

8.1 Unfolding Degree-2 Nodes

In this section we turn our attention to degree-2 nodes in $\mathcal{T}$ and show that they can be unfolded into nets that satisfy the inductive hypothesis. We discuss three cases, depending on whether the degree-2 box has a back child, a north child or an east child. The cases where A has a south child or a west child are symmetric.

Theorem 21 *Any degree-2 box $A \in \mathcal{T}$ with back child J can be unfolded into a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.*

Proof. Lemma 1 enables us to restrict our attention to HEAD-first unfoldings of A. One such unfolding is depicted in Figure 12a. Note that this unfolding is very similar to the unfolding of the root box from Figure 5. Because $\xrightarrow{e_A}$ ($\xleftarrow{x_A}$) may be open or closed, $\mathcal{E}_A$ ($\mathcal{X}_A$) may or may not belong to A's inductive region, as indicated by the free cells from Figure 12a. Since $\xrightarrow{e_J}\in R_A$ is open, the inductive hypothesis applied to $\mathcal{N}_J$ tells us that $\mathcal{N}_J$ provides a type-1 or type-2 entry (exit) connection, which attaches to either T_A or R_A (B_A or L_A). Thus the net $\mathcal{N}_A$ is connected. By the inductive hypothesis, $\mathcal{N}_J$ covers all open faces in $\mathcal{T}_J$ using a 4×4 refinement. Noting that $\mathcal{N}_A$ includes the open faces of A, we conclude that $\mathcal{N}_A$ includes all open faces of $\mathcal{T}_A$ and satisfies (I1) of the inductive hypothesis. Note that $\mathcal{N}_A$ provides type-1 entry and exit connections, therefore it satisfies (I2) of the inductive hypothesis. Finally, the open ring faces of A not used in its entry and exit connections (dark-shaded in Figure 12a) can be removed from $\mathcal{N}_A$ without disconnecting $\mathcal{N}_A$, therefore $\mathcal{N}_A$ satisfies (I3) of the inductive hypothesis. □

Theorem 22 *Any degree-2 box $A \in \mathcal{T}$ with north child N can be unfolded into a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.*

Proof. Lemma 1 enables us to restrict our attention to HEAD-first unfoldings of A. One such unfolding is depicted in Figure 12b. Note that this is a degenerate case of the unfolding of the box with north and south children from Figure 7b. The only difference is that the net $\mathcal{N}_S$ for the south child from Figure 7b has been replaced by the south face B_A in Figure 12b. Arguments similar to the ones used in the proof of Theorem 10 show that $\mathcal{N}_A$ satisfies the inductive hypothesis. □

Theorem 23 *Any degree-2 box $A \in \mathcal{T}$ with east child E can be unfolded into a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.*

Proof. Lemma 1 enables us to restrict our attention to HEAD-first unfoldings of A. These unfoldings are depicted in Figure 13a for the case with the HAND pointing east, and in Figure 13b for the case with the HAND pointing west. Note that these unfoldings are degenerate cases of the unfoldings of the box with east and back children from Figure 6. One difference is that the unfolding net $\mathcal{N}_J$ for the back child from Figure 6 is replaced by the back face K_A in Figure 13. A few more minor modifications are necessary to accommodate for the fact that, in the HAND-east unfolding, $\xrightarrow{e_E}\in K_A$ is open (and so $\mathcal{E}_E$ does not belong to the inductive region for E) and therefore $\mathcal{N}_E$ may provide a type-1 or a type-2 entry connection. Similarly, in the HAND-west unfolding, $\xleftarrow{x_E}\in K_A$ is open (and so $\mathcal{X}_E$ does not belong to the inductive region for E) and thus $\mathcal{N}_E$ may provide a type-1 or a type-2 exit connection. These accommodations are reflected in Figure 13. Arguments similar to the ones used in the proof of Theorem 6 show that the nets $\mathcal{N}_A$ from Figure 6 satisfy the inductive hypothesis. □

9 Unfolding Leaf Nodes

Lemma 24 *Let $A \in \mathcal{T}$ be leaf box with parent I. There is an unfolding of A whose net $\mathcal{N}_A$ satisfies the inductive hypothesis.*

Proof. Lemma 1 enables us to restrict our attention to HEAD-first unfoldings of A. Consider the unfolding depicted in Figure 2a: starting at A's entry port, the unfolding path simply moves HEAD-first until it reaches A's exit port. We now show that, when laid flat in the plane, the open faces of A form a net $\mathcal{N}_A$ that satisfies the inductive hypothesis.

If $\xrightarrow{e}$ is closed (open), then $\mathcal{E}_A$ belongs (doesn't belong) to the inductive region for A. This dual case scenario is depicted by the free cell labeled $\mathcal{E}_A$ in Figure 2a. Similarly, if $\xleftarrow{x}$ is closed (open), then $\mathcal{X}_A$ belongs (doesn't belong) to the inductive region for A. Thus condition (I1) of the inductive hypothesis is trivially satisfied.

To check that (I2) is satisfied, note that $\mathcal{N}_A$ provides type-1 entry and exit connections since $e' \in T_A$ and $x' \in B_A$ are positioned alongside the entry and exit ports.

Turning to (I3) of the inductive hypothesis, observe that the open ring faces of A not used in A's entry or exit connections are the dark-shaded pieces from Figure 2a, whose removal does not disconnect $\mathcal{N}_A$. Thus $\mathcal{N}_A$ also satisfies condition (I3) of the inductive hypothesis. □

10 A Complete Example

Figure 14 illustrates a complete unfolding example for an orthotree composed of 9 boxes. The root A of the

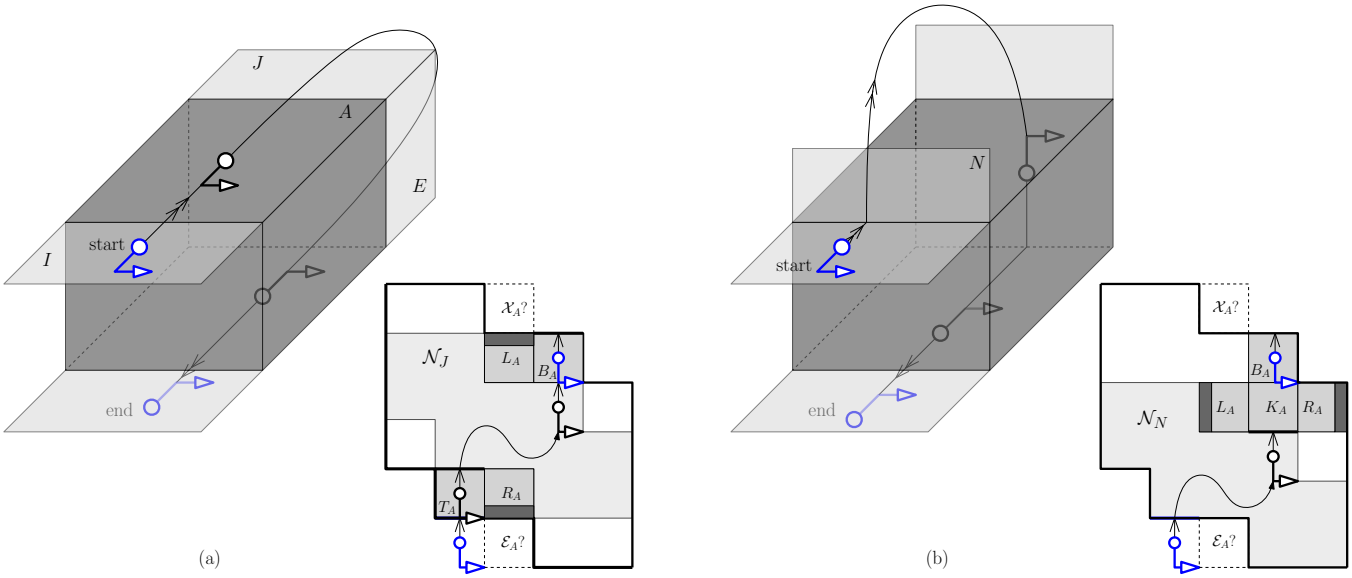


Figure 12: Box A of degree 2 with (a) back child (b) north child.

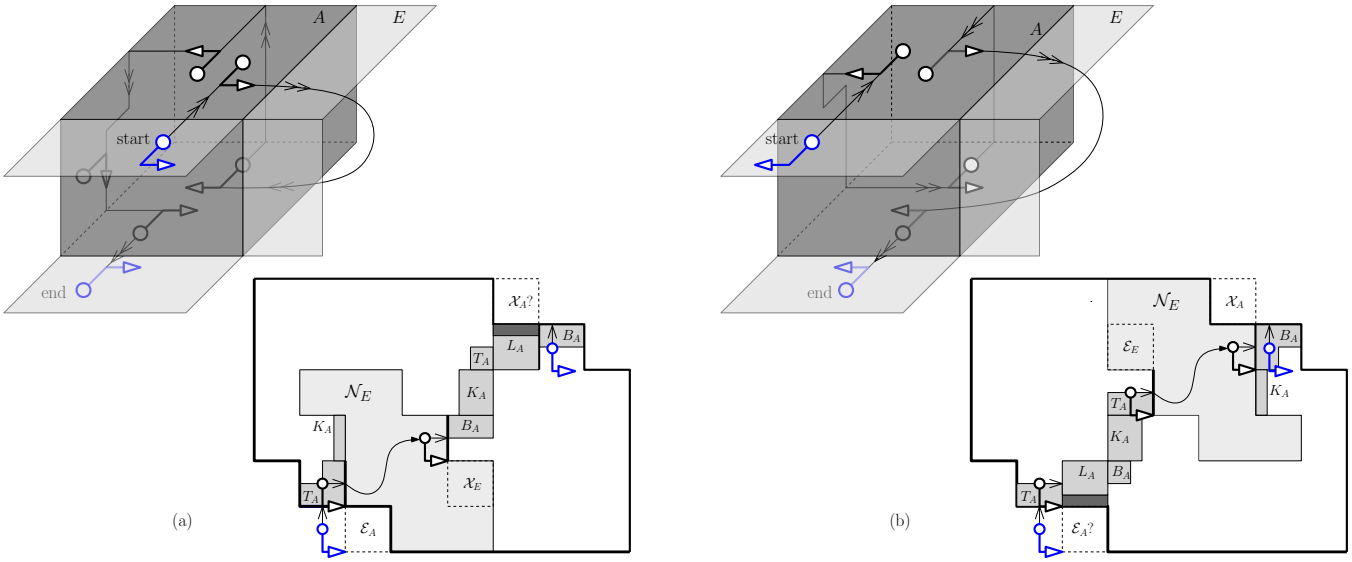


Figure 13: Box A of degree 2 with east child (a) HAND pointing east (b) HAND pointing west.

the unfolding tree is a degree-1 box with back child J, which is unfolded recursively. The unfolding of J follows the pattern depicted in Figure 9b, slightly adjusted to accommodate for the fact that J does not have a south-east child. The east-east child of J (labeled C in Figure 14) follows the unfolding pattern depicted in Figure 13a. The north child of J (labeled F in Figure 14) follows the unfolding pattern from Figure 10b, traversed on reverse (note that the subtree rooted at F is a horizontal mirror plane reflection of the case depicted in Figure 10b, after a clockwise 90°-rotation about a vertical axis followed by a clockwise 90°-rotation about a horizontal axis, to bring it in standard position). Finally, the leaves are unfolded as in Figure 2.

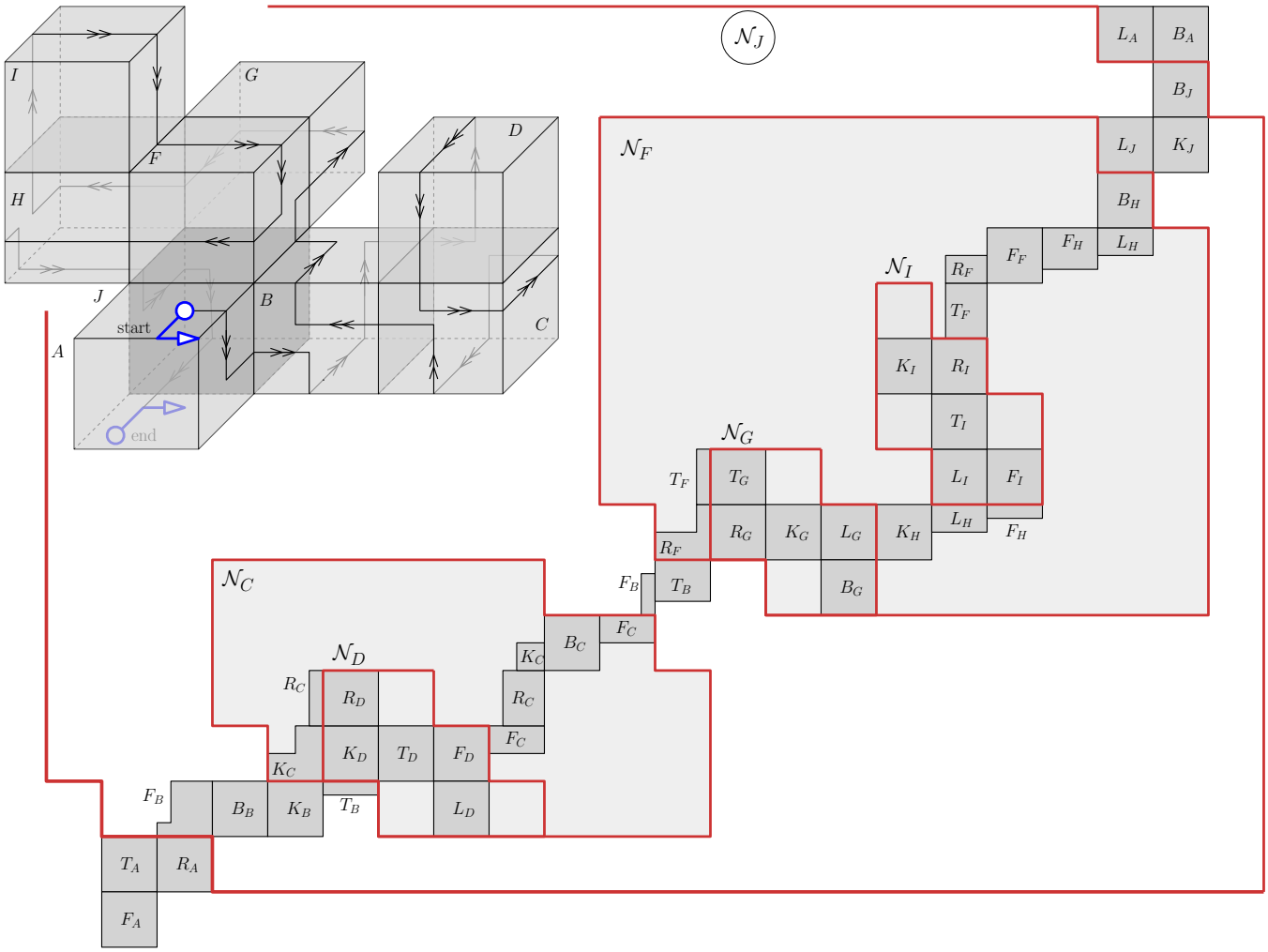


Figure 14: A complete unfolding example.

Dihedral Rigidity and Deformation

Nina Amenta* Carlos Rojas†

Abstract

We consider defining the embedding of a triangle mesh into $\mathbb{R}^3$, up to translation, rotation, and scale, by its vector of dihedral angles. Theoretically, we show that locally, almost everywhere, the map from realizable vectors of dihedrals to mesh embeddings is one-to-one. We experiment with a heuristic method for mapping straight-line interpolations in dihedral space to interpolations between mesh embeddings and produce smooth and intuitively appealing morphs between three-dimensional shapes.

1 Introduction

Frequently, a polygon mesh is represented by its mesh combinatorics and a vector of 3D vertex coordinates, specifying the immersion of the mesh into $\mathbb{R}^3$. A polygon mesh is *rigid* when the only motions of the vertex coordinates for which the faces are not deformed in any way are the rigid motions (rotation and translation). Non-rigid polyhedra do exist [Bri97, Con77, Con79], although they are rare; a non-rigid polyhedron has some *flexing* motion in which the faces move but do not deform. That is, the dihedral angles between faces change continuously while the faces themselves remain rigid. In 1974 Herman Gluck [Glu75] proved

Theorem 1 (Gluck) *A generic immersion of any mesh topology homeomorphic to the sphere is rigid.*

By *generic* we mean all vectors of vertex coordinates, except some "degenerate" subset of measure zero. So, for example, if you construct the edge-skeleton of triangulated computer graphics model, with stick edges held together at flexible joints, it almost certainly would be rigid.

Mesh deformations, in which, typically, both edge lengths and dihedrals change, is an important topic in computer graphics, computer vision and scientific shape analysis. A deformation defines a path in the space of discrete metrics - a continuous change in the vector of edge lengths. This has been quite well studied, especially for the subset of discrete conformal transformations, eg. [SA07, BLL15, ZLL+15]. But discrete metrics do not correspond to shape in any precise sense: rigidity theory tells us that a vector of edge lengths typically has multiple discrete realizations as a rigid mesh, and that it might even correspond to a flexible polyhedron.

We are interested in the other possibility: characterizing a deformation by the change in its vector of dihedral angles. The first mathematical question one might ask is whether there are motions in which all the dihedral angles stay the same, but the edge lengths change. Indeed, this is trivially possible; consider a cube deforming into an arbitrary box. But notice that during such a deformation the inner face angles (the plane angles) remain unchanged. So next we ask if there are deformations in which the dihedral angles remain fixed, but the inner angles change; we call this a *dihedral flex*. We say that a polyhedron which does *not* allow a dihedral flex is *dihedral-rigid*. It is not known if dihedral-flexible polyhedra exist. Here, we prove the following analog of Gluck's theorem:

Theorem 2 *A generic immersion of any triangle mesh homeomorphic to the sphere is dihedral-rigid.*

While interesting as a result in rigidity theory, of course this is only a small step towards a theoretical justification of the idea of representing mesh embeddings by their dihedral vectors. We also give some experimental evidence that the dihedral representation is useful and natural. We compute a morphs between two embeddings by heuristically mapping the straight line segment connecting their dihedral vectors in (Euclidean) dihedral-space onto a path in the space of embeddings. We find smooth paths connecting very different shapes, and observe that the resulting morphs seem quite natural.

2 Related work

In 1968, Stoker [Sto68] conjectured that a convex polyhedron is uniquely defined by its combinatorics and dihedral angles (and thus that it is dihedral-rigid). This would be the dihedral version of Cauchy's theorem on the rigidity of convex polyhedra [Cau13]. A fairly simple proof of Stoker's conjecture for triangulated convex polyhedra was given by Pogorelov [Pog02]; we draw on his work as well as that of Gluck. Only recently was a complete proof of Stoker's conjecture provided by Mazzeo and Montcouquiol [MM+11], using much more

*Department of Computer Science, University of California, Davis, amenta@cs.ucdavis.edu

†Department of Computer Science, University of California, Davis, crojas@ucdavis.edu

sophisticated techniques and applying to the interesting case of ideal hyperbolic polyhedra as well.

In computer graphics, there is an ongoing interest in constructing shape spaces in which geodesic paths correspond to physically natural-looking morphs, which can be used for applications such as morphing, shape exploration, deformation and deformation transfer. These spaces tend to be curved and difficult to deal with, eg. [KMP07]. A recent series of papers [HRS+14, HRS+16, ZHRS15] explores the curved shape space implied by the elastic model of deformation. They prove that it forms a Riemannian manifold, and produce shape averages, principal components and splines in this "shell space". Each of these operations proves to be challenging, both mathematically and computationally.

There is a practically successful line of work [BLL15, KG08, WDAH10] on interpolating mesh embeddings by interpolating both their dihedral angles and their edge lengths, and then doing some sort of least-squares reconstruction to produce an interpolating mesh. These methods cannot realize both the dihedrals and the edge lengths exactly - there are roughly $6n$ parameters and $3n$ degrees of freedom in the embedding, where n is the number of mesh vertices - but they are fairly simple and they provide very nice-looking results.

The space of dihedral angles was proposed recently as a representation for deformation by Paille et al. [PRP+15], albeit for a tetrahedralized volume. Here again, we find that the number of dihedrals in a tetrahedralization is much larger than the dimension of the space of realizable meshes. Finally, [IGG01] showed that ignoring edge length and just using connectivity to reconstruct shapes is surprisingly successful.

3 Infinitesimal rigidity

One's first instinct when considering the possibility of a dihedral flex is to consider the vertex positions p_i as functions $p_i(t)$ of some parameter t, and consider the derivatives of the inner angles $\beta_{j,i,j+1}$ and dihedrals $\alpha_{i,j}$ with respect to t. At any point along any traditional (edge length) flex, the length derivatives $l'_{i,j} = 0$ at every edge, while at least some of the $\alpha'_{i,j}$ are non-zero. Similarly along any dihedral flex (if such a thing exists!) we expect to find an infinitesimal motion such that all $\alpha'_{i,j} = 0$, while there are inner angles for which the derivatives $\beta'_{j,i,j+1}$ are non-zero. We call a polyhedron which admits such a motion *dihedral infinitesimally non-rigid*.

A polyhedron which is dihedral non-rigid must be dihedral infinitesimally non-rigid. It may well be possible, however, for a polyhedron to be dihedral infinitesimally non-rigid while being rigid; there are many polyhedra which are (length) infinitesimally non-rigid, but actually rigid. Following Gluck, we prove that a generic immersion of a mesh forms a polyhedron which is dihedral infinitesimally rigid, and hence dihedral rigid.

4 Dihedral infinitesimal rigidity as a matrix equation

There is a very nice relationship between the derivatives of the dihedral angles α' and the derivatives of the triangle inner angles β', which Gluck used in his theorem on length rigidity. We have, going around the one-ring of any vertex p_i,

$$\sum_j \alpha'_{ij}\vec{e}_{ij} + \sum_j \beta'_{j,i,j+1}\vec{n}_{j,i,j+1} = \mathbf{0}$$

where $\vec{n}_{j,i,j+1}$ is the normal to triangle $t_{j,i,j+1}$, and $\vec{e}_{ij} = (p_i - p_j)/||p_i - p_j||$ is the unit vector in the direction of edge e_{ij}. This equation expresses the fact that the instantaneous angular velocities in the one-ring have to change in a coordinated fashion for the one-ring to continue to "hold together". Their derivation appears in Appendix A. Since the edge and normal vectors have three coordinates each, we have three equations at each vertex, for a total of $3V$. Let's call these the vertex equations.

Gluck considered the case in which we assume that the change in edge lengths, and hence the inner angle derivatives β', are all zero, so that the length infinitesimally non-rigid configurations were those with

$$\sum_j \alpha'_{ij}\vec{e}_{ij} = \mathbf{0}$$

This system has $3V$ equations in $3V - 6$ variables.

We make the opposite assumption, that the dihedral angles α remain unchanged, so we are interested in non-zero solutions to

$$\sum_j \beta'_{j,i,j+1}\vec{n}_{j,i,j+1} = \mathbf{0}$$

In our case we have $3V$ equations in the $6V-12$ variables β'. There are additional constraints on the β' variables which determine the validity of the mesh.

One is that the sum of the inner angles of any triangle add up to π. Taking the derivative of this condition is gives us

$$\beta'_i + \beta'_j + \beta'_{j+1} = 0$$

We call these the face equations.

Finally, the Law of Sines implies the following differential cotangent formula for the triangles around a given one-ring

$$\sum_j \cot\beta_{i,j,j+1}\beta'_{i,j,j+1} - \cot\beta_{i,j+1,j}\beta'_{i,j+1,j} = 0$$

The derivation of this equation appears in Appendix B. We call these the cotangent equations. Together, the

vertex equations, face equations and cotangent equations form a system with $3V + 2V - 4 + V = 6V - 4$ equations in $6V - 12$ variables.

$$M\beta' = \mathbf{0}$$

A mesh is dihedral infinitesimally non-rigid if this system M has some non-zero solution for the βs, that is, if there is an infinitesimal motion of the mesh that leaves the dihedrals fixed but allows the inner angles to flex somehow, while maintaining a valid mesh.

5 Condition for a solution

Following Gluck, we observe that there is a non-zero solution for β if and only if the coefficient matrix M has rank less than $6V - 12$. And for this to be true, it must be the case that every $6V - 12 \times 6V - 12$ submatrix of M has zero determinant. We can write this condition on the coefficient matrix itself as a system of $\binom{6V-4}{8}$ polynomials in the matrix elements; call this system F. In Gluck's proof, he dealt with a matrix whose coefficients were themselves polynomials in the vertex coordinates of the mesh, and this allowed him to argue that the resulting variety formed a set of measure zero.

In our case, the coefficients are the face normals, ones, and the cotangents of the inner angles. These are not all polynomials in the vertex coordinates. To get around this, we treat the normals and cotangents as variables themselves; for notational clarity, let's write $c_{j,i,j+1} = \cot \beta_{j,i,j+1}$. The c and n variables are not independent of each other. The normals must all have length one; for $n_{j,i,j+1} = (n_x, n_y, n_z)$, we have

$$n_x^2 + n_y^2 + n_z^2 = 1 \tag{1}$$

In addition, the normal and cotangent variables are conveniently related to each other, and to the vertex coefficients, by the following formula.

$$\begin{aligned}&[(p_i - p_j) \times (p_i - p_{j+1})]\, c_{j,i,j+1} = \\ &[(p_i - p_j) \cdot (p_i - p_{j+1})]\, n_{j,i,j+1}\end{aligned} \tag{2}$$

This formula relates the cotangent to the scaling of the cross-product to form the triangle normal; an (easy) derivation appears in Appendix C. Note that since the cross-product and dot-product are both polynomial functions, this is a polynomial as well.

In order for a mesh configuration to be be dihedral infinitesimally non-rigid, we need Equations 1 and 2 to be true for every angle, as well as for all of the subdeterminants of M to be zero. These conditions are all polynomial, and they define a variety (the intersection of their zero-sets) in the space of the p, n, c variables.

An arbitrary assignment of values to p, n, c does not correspond to an immersion of the mesh; the p-variables are all free, but the n and c will not obey Equations 1 and 2. Given a choice of p variables, the n and c variables of that embedding uniquely satisfy 1 and 2 (the normal is indeed the cross-product, scaled as required). So there is unique lifting of the Euclidean space defined by the vertex coordinate space p into (p, n, c)-space. Let $\tilde{P}$ be this lifting of the the vertex coordinate space, which is Euclidean. The space $\tilde{P}$ is similarly simply connected and $3n$-dimensional.

If we have a connected component of an algebraic variety and we add an additional polynomial constraint to the system, either the the whole component satisfies the new equation, or the dimension of the new variety is reduced by the intersection with the new equation. Thus, if there is any point of $\tilde{P}$ which does *not* also satisfy the system F saying that all of the sub-determinants have to be zero, the set of common zeros (the space of dihedral infinitesimally non-rigid polyhedra) has smaller dimension than $\tilde{P}$, and forms a subset of measure zero.

So all we need to do to show that the dihedral infinitesimally non-rigid polyhedra form a set of measure zero is to display some point in $\tilde{P}$ which is *not* in F; that is, a dihedral infinitesimally rigid polytope. As it happens, we can do this for every mesh topology; the results of Pogorelov [Pog02] and Mazzeo and Montcouquiol [MM$^+$11] show that every convex polyhedron is dihedral infinitesimally rigid, and we know that every mesh topology can be realized as a convex polyhedron (this is Steinitz' theorem).

This proves Theorem 2.

6 Experiments with dihedral parameterization

In this section we experiment with treating the dihedral vector for a given mesh topology as a Euclidean shape space. First, we consider interpolating between different embeddings of the same mesh by interpolating their dihedral angles. We find that this produces smooth morphs between quite different shapes. For instance, in Figure 1, we get a smooth morph from a dinosaur to a camel.

We morph between the two input embeddings by connecting their two dihedral vectors with a straight line segment in dihedral space, and reconstructing a series of embeddings corresponding to uniformly-spaced points along the segment. Since the dihedrals are scale-invariant, we normalize the scale of the embedding as well as its rotation and translation. This means that the space of possible embeddings had dimension $3n - 7$, where n is the number of mesh vertices ($3n$ possible vertex coordinates, normalized for the seven-dimensional transformation space). A mesh homeomorphic to the sphere has $3n - 6$ dihedral angles, however, so we do not

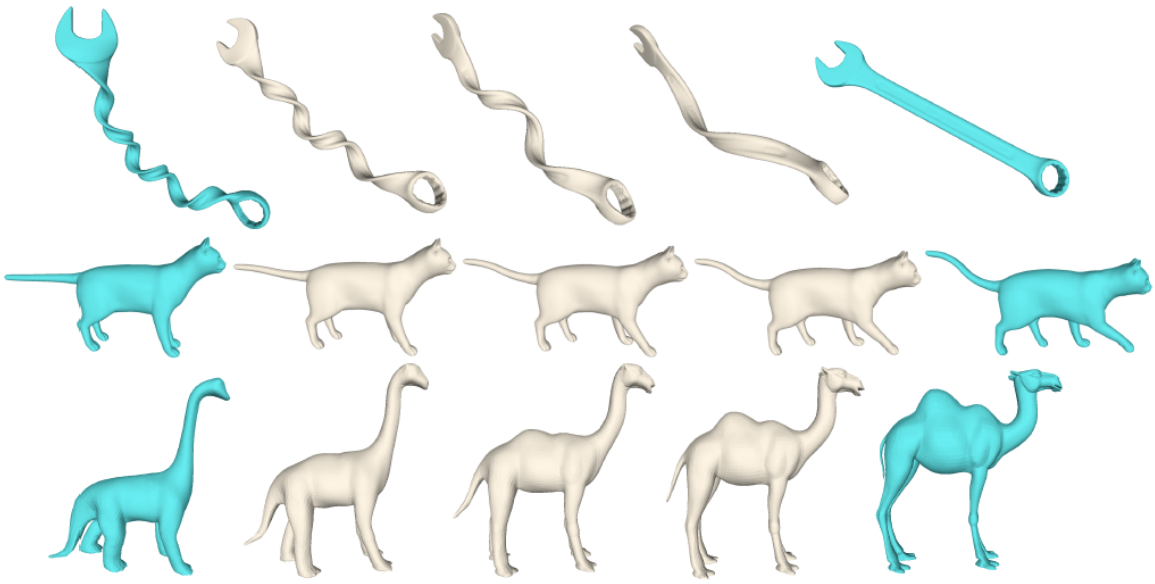


Figure 1: Three examples of morphs between different shapes. The twisting of the wrench is handeled nicely by considering dihedrals. The poses of the cat are interpolated naturally, without distortion in the intermediate shapes. The dinosaur and the camel have the same mesh topology, but are very different embeddings.

expect the intermediate dihedral vectors to exactly correspond to embeddings. Instead, we use a least-squares algorithm to compute embeddings that lie as close as possible to the dihedral-space line segment connecting the input shapes.

At each interpolated point, we use an optimization algorithm to find a mesh embedding that comes as close as possible to realizing the desired dihedrals. The optimization algorithm consists of an initialization step and a refinement step. The initialization fits an embedding to both interpolated dihedrals and interpolated edge lengths. Then a refinement step alternates between computing a set of normal vectors $\vec{n}_k$ which realize the given dihedrals, and a set of mesh vertex positions p_i which realize the $\vec{n}_k$ as well as possible. The initialization, and each iteration of the refinement process, consists of a linear least-squares solve.

The initialization step is similar to the mesh interpolation algorithms of Baek et al. [BLL15] and Kircher and Garland [KG08]. We first reconstruct the one-ring of each vertex, given the interpolated edge lengths and dihedral, and then we combine the one-rings, using least-squares, to produce a set of vertex positions. We describe this in more detail in Appendix D.

The refinement phase is more novel. We define an energy function E for a mesh, which considers both the normal vectors $\vec{n}_k$ and the vertices p_i.

$$E = \alpha \sum_{\text{adjacent triangles } k,l} ||M_{kl}\vec{n}_k - \vec{n}_l||_F^2 + \beta \sum_{\substack{\text{edge } i,j \\ \text{triangle } k}} ||L_{ijk}(p_i - p_j) - \vec{n}_k||^2 \tag{3}$$

F indicates the Frobenious norm. Here $\vec{n}_k$ is the normal of triangle k and $\vec{n}_l$ is the normal of triangle l, adjacent across edge i, j. The matrix M_{kl} is a rotation by exactly the desired dihedral angle δ_{ij} between $\vec{n}_k$ and $\vec{n}_l$, with the axis of rotation $\vec{e}_{ij} = (p_i - p_j)/||p_i - p_j||$. Thus the first energy term measures how well the normals achieve the dihedral angles at every edge. The second term measures how well the normals and vertices agree with each other. The matrix $L_{i,j,k}$ takes edge i, j into the normal of one of its adjacent triangles k. It is the product of a rotation by $\pi/2$, along with a scaling to normalize the length. The weights are $\alpha = 0.6$ and $\beta = 0.4$.

At each step, we recompute M and L from the current mesh, solve for new $\vec{n}_k$ while keeping the p_i fixed, and finally solve for new values of p_i.

We see that this algorithm succeeds in reducing the dihedral error at of the interpolations by about half. We define the dihedral error simply as the Euclidean difference between the desired interpolated dihedral vector and the actual dihedrals achieved by our embedding; an example appears in Figure 2. As noted above, we do not expect to be able to achieve a dihedral error of zero.

Videos of these smooth morphs can be seen at `https://vimeo.com/270302684`.

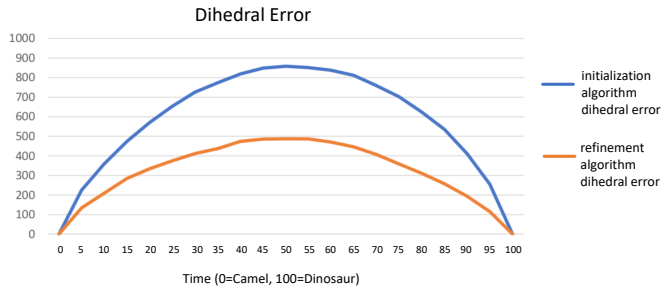


Figure 2: Dihedral error reduction due to the refinement step, for the camel-to-dinosaur morph.

In earlier experiments [RTH+14], we found that attempting to optimize the embedding towards the interpolated edge lengths, rather than the interpolated dihedrals, produced morphs with discontinuities and glitches. We believe that this is because there are many possible embeddings realizing a given set of edge lengths, while we suspect that at most one embedding per vector of dihedrals.

7 Shape analysis

We also considered using the space of dihedral angles as a method for shape analysis. This idea is appealing because if we treat dihedral space as Euclidean, we can use off-the-shelf techniques and software.

As an example, we analyze the ground-truth subset of the MPI FAUST dataset. The entire dataset include 300 human 3D laser scans in a wide range of poses [BRLB14], and it is intended as a benchmark for registration methods. Its ground truth subset is given as a set of embeddings of a single topological mesh, representing 10 subjects each in 10 different poses, labeled by subject and pose. Each mesh has approximately 7,000 vertices.

In the dihedral space, we used PCA to reduce the dimensions of the ground-truth dataset; a scatterplot on these first two principal coordinates is shown at the top in Figure 3. There are two distinct 460 clusters that correspond to gender, demonstrating the fact that in dihedral space the most salient features are those that reflect body shape rather than pose or size; this is not true, for example, in the Euclidean space formed by the $3n$ vertex coordinates. Variation of the body shape along the first principal component in dihedral space is shown at the bottom of Figure 3. The meshes in this visualization were created using the initial approximate least-squares reconstruction from dihedrals and edge lengths of Appendix D, without optimization, using in every case the average edge lengths over the entire corpus of 100 scans and only varying the input dihedrals along the principal component. This shows that quite large changes in shape can be visualized without changing the input edge lengths.

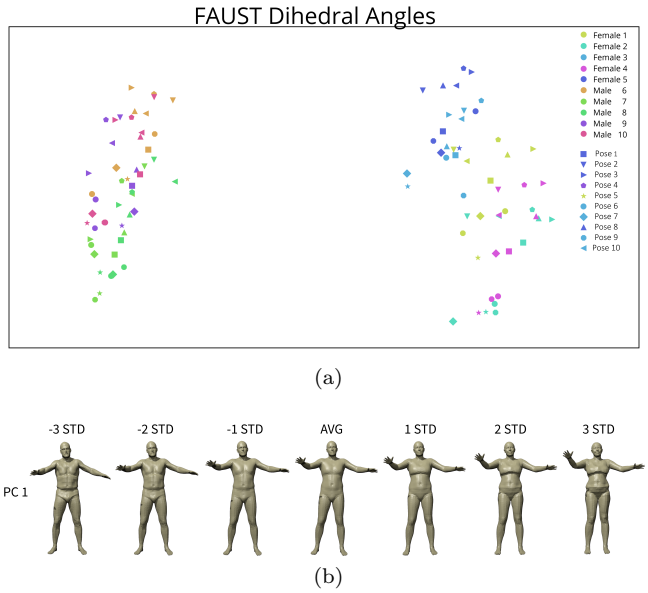


Figure 3: The top principal component in dihedral shape space for the FAUST human body shape data describes the fundamental shape difference between male and female bodies. When we plot the first two principal components (a) we clearly see the separation between the group of male and female subjects. In Figure (b) we warp the average shape in the direction of the first principal component, by adding multiples of it to the average shape. All of the figures are reconstructed using the same edge lengths. Each edge length is averaged over the whole input set.

8 Discussion

There is a clear gap between the very basic level of our mathematical understanding of the dihedral vectors of mesh embeddings and the potential reflected in our experimental work. This suggests several conjectures, perhaps the most important being,

Conjecture 1 *There is at most one set of inner face angles consistent with an embedding of a mesh realizing a given vector of dihedral angles.*

References

[BLL15] Seung-Yeob Baek, Jeonghun Lim, and Kunwoo Lee, *Isometric shape interpolation*, Computers & Graphics **46** (2015), 257–263.

[Bri97] Raoul Bricard, *Mémoire sur la théorie de l'octaèdre articulé*, Journal de

Mathématiques pures et appliquées **3** (1897), 113–148.

[BRLB14] Federica Bogo, Javier Romero, Matthew Loper, and Michael J Black, *Faust: Dataset and evaluation for 3d mesh registration*, Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 3794–3801.

[Cau13] Augustin Louis Cauchy, *Sur les polygones et polyedres*, J. Ec. Polytechnique **16** (1813), 87–99.

[Con77] Robert Connelly, *A counterexample to the rigidity conjecture for polyhedra*, Publications Mathématiques de l'Institut des Hautes Études Scientifiques **47** (1977), no. 1, 333–338.

[Con79] ———, *The rigidity of polyhedral surfaces*, Mathematics Magazine **52** (1979), no. 5, 275–283.

[Glu75] Herman Gluck, *Almost all simply connected closed surfaces are rigid*, Geometric topology, Springer, 1975, pp. 225–239.

[HRS+14] Behrend Heeren, Martin Rumpf, Peter Schröder, Max Wardetzky, and Benedikt Wirth, *Exploring the geometry of the space of shells*, Computer Graphics Forum, vol. 33, Wiley Online Library, 2014, pp. 247–256.

[HRS+16] ———, *Splines in the Space of Shells*, Computer Graphics Forum, vol. 35, Wiley Online Library, 2016, pp. 111–120.

[IGG01] Martin Isenburg, Stefan Gumhold, and Craig Gotsman, *Connectivity shapes*, Proceedings of the conference on Visualization'01, IEEE Computer Society, 2001, pp. 135–142.

[KG08] Scott Kircher and Michael Garland, *Free-form motion processing*, ACM Transactions on Graphics (TOG) **27** (2008), no. 2, 12.

[KMP07] Martin Kilian, Niloy J Mitra, and Helmut Pottmann, *Geometric modeling in shape space*, ACM Transactions on Graphics (TOG), vol. 26, ACM, 2007, p. 64.

[LSLCO05] Yaron Lipman, Olga Sorkine, David Levin, and Daniel Cohen-Or, *Linear rotation-invariant coordinates for meshes*, ACM Transactions on Graphics (TOG) **24** (2005), no. 3, 479–487.

[Mao86] Jianqin Mao, *Optimal orthonormalization of the strapdown matrix by using singular value decomposition*, Computers & mathematics with applications **12** (1986), no. 3, 353–362.

[MM+11] Rafe Mazzeo, Grégoire Montcouquiol, et al., *Infinitesimal rigidity of cone-manifolds and the stoker problem for hyperbolic and euclidean polyhedra*, Journal of Differential Geometry **87** (2011), no. 3, 525–576.

[Pog02] AV Pogorelov, *On a problem of stoker*, Dokl. Akad. Nauk **385** (2002), no. 1, 25–27.

[PRP+15] Gilles-Philippe Paillé, Nicolas Ray, Pierre Poulin, Alla Sheffer, and Bruno Lévy, *Dihedral angle-based maps of tetrahedral meshes*, ACM Transactions on Graphics (TOG) **34** (2015), no. 4, 54.

[RTH+14] Carlos Rojas, Alex Tsui, Stewart He, Lance Simons, Shengren Li, and Nina Amenta, *Edge length interpolation*, ACM Symposium on Solid and Physical Modeling, 2014, poster paper.

[SA07] Olga Sorkine and Marc Alexa, *As-rigid-as-possible surface modeling*, Symposium on Geometry processing, vol. 4, 2007, p. 30.

[Sto68] James J Stoker, *Geometrical problems concerning polyhedra in the large*, Communications on pure and applied mathematics **21** (1968), no. 2, 119–168.

[WDAH10] Tim Winkler, Jens Drieseberg, Marc Alexa, and Kai Hormann, *Multi-scale geometry interpolation*, Computer graphics forum, vol. 29, Wiley Online Library, 2010, pp. 309–318.

[ZHRS15] Chao Zhang, Behrend Heeren, Martin Rumpf, and William AP Smith, *Shell PCA: Statistical shape modelling in shell space*, Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1671–1679.

[ZLL+15] Zhibang Zhang, Guiqing Li, Huina Lu, Yaobin Ouyang, Mengxiao Yin, and Chuhua Xian, *Fast as-isometric-as-possible shape interpolation*, Computers & Graphics **46** (2015), 244–256.

Appendix A Derivation of the vertex equation

First, we need to review some material on the derivatives of the rigid motions. To warm up, let us consider translation. As t goes from zero to one, the coordinate vector P_0 changes to $P_0 + b$, meaning the vector (b_x, b_y, b_z) is added to every component of P_0. At time t, we have $P(t) = P_0 + tb$. The derivative then is $dP(t)/dt = b$.

Rotations are more interesting. Let the matrix $R(\alpha)$ represent the rotation through the axis (r_x, r_y, r_z) (through the origin) by angle α. At time t, we have $P(t) = R(t\alpha)P_0$, that is, the angle of rotation increases with t but the axis stays the same. So

$$\frac{dP(t)}{dt} = \frac{dR(t\alpha)}{dt}P_0 + R(t\alpha)\frac{dP_0}{dt} \quad (4)$$

$$= \frac{dR(t\alpha)}{dt}P_0. \quad (5)$$

Interestingly, the derivative

$$\frac{dR(t\alpha)}{dt} = SR(t\alpha) = (\alpha r_x, \alpha r_y, \alpha r_z) \times R(\alpha t)$$

is a 3×3 matrix, where S is the matrix

$$\begin{vmatrix} 0 & -\alpha r_z & \alpha r_y \\ \alpha r_z & 0 & -\alpha r_x \\ -\alpha r_y & -\alpha r_x & 0 \end{vmatrix}$$

which performs the cross-product. The vector $\omega = (\alpha r_x, \alpha r_y, \alpha r_z)$ is the axis of rotation of R, is known as the angular velocity vector; the actual angular velocity of a point p at time t undergoing the rotation, however, is represented by the value of the derivative at t, $(\omega \times R(\alpha t))p_0$.

Notice that $(R_a\omega_b) \times p \neq R_aS_bp$; this is easy to see since the matrix on the right-hand-side does not have zero diagonal, and the matrix on the left does. In fact, the correct transformation is $(R_a\omega_b) \times p = R_aS_bR_a^Tp$, and we can say $(R_a\omega_b)R_a = R_aS_b$, since $R_aR_a^T = I$. This property comes in handy when working with the derivative of a series of rotations, as follows. Say

$$R_d = R_aR_bR_c$$

Then

$$\begin{aligned} \frac{dR_d}{dt} &= \frac{dR_a}{dt}R_bR_c + R_a\frac{dR_b}{dt}R_c + R_aR_b\frac{dR_c}{dt} \\ S_dR_d &= S_aR_aR_bR_c + R_aS_bR_bR_c + R_aR_bS_cR_c \\ \omega_d \times R_d &= \omega_a \times R_aR_bR_c + (R_a\omega_b) \times R_aR_bR_c + \\ &\quad (R_aR_b\omega_c) \times R_aR_bR_c \end{aligned}$$

and hence

$$\omega_d = \omega_a + (R_a\omega_b) + (R_aR_b\omega_c)$$

Notice that the vectors ω are given in the local coordinate system, so that the multiplications by the preceding rotations in the equation above are transforming them into the global coordinate system.

Appendix B Derivation of the cotangent equation at a vertex

Let p_i be a vertex, and consider the vertices of its one-ring, p_j, p_{j+1}, etc. Using the Law of Sines, we have

$$\frac{\sin\beta_{i,j,j+1}}{\sin\beta_{i,j+1,j}} = \frac{l_{i,j+1}}{l_{i,j}}$$

Going around the one-ring,

$$\prod_j \frac{l_{i,j+1}}{l_{i,j}} = 1$$

and so

$$\prod_j \frac{\sin\beta_{i,j,j+1}}{\sin\beta_{i,j+1,j}} = 1$$

Taking the natural logarithm, we have

$$\sum_j \ln\sin\beta_{i,j,j+1} - \ln\sin\beta_{i,j+1,j} = 0$$

Next we take the derivative. We have $(\ln x)' = 1/x$ and $(\sin x)' = \cos x$, so we get

$$\sum_j \frac{\cos\beta_{i,j,j+1}}{\sin\beta_{i,j,j+1}}\beta'_{i,j,j+1} - \frac{\cos\beta_{i,j+1,j}}{\sin\beta_{i,j+1,j}}\beta'_{i,j+1,j} = 0$$

or

$$\sum_j \cot\beta_{i,j,j+1}\beta'_{i,j,j+1} - \cot\beta_{i,j+1,j}\beta'_{i,j+1,j} = 0$$

Appendix C Derivation of Equation 2

We know that

$$(p_i - p_j) \times (p_i - p_{j+1}) = ||p_i - p_j||||p_i - p_{j+1}||\sin\beta_{j,i,j+1}n_{j,i,j+1}$$

We also know that

$$(p_i - p_j) \cdot (p_i - p_{j+1}) = ||p_i - p_j||||p_i - p_{j+1}||\cos\beta_{j,i,j+1}$$

So we can write

$$[(p_i - p_j) \times (p_i - p_{j+1})]\cos\beta_{j,i,j+1} =$$

$$[(p_i - p_j) \cdot (p_i - p_{j+1})]\sin\beta_{j,i,j+1}n_{j,i,j+1}$$

and hence

$$[(p_i - p_j) \times (p_i - p_{j+1})]\cot\beta_{j,i,j+1} =$$

$$[(p_i - p_j) \cdot (p_i - p_{j+1})]n_{j,i,j+1}$$

Appendix D Initialization algorithm

We can initialize the dihedral angle morph in Section 6 by computing an approximate embedding of the interpolated point.

We approximate the embedding by linearly interpolating the dihedral angles and the edge lengths between the two inputs; we can construct a mesh that satisfies both in a least-squares sense. The method we use combines the ideas of [BLL15, KG08, LSLCO05, WDAH10]. At each vertex p_i, we construct a least-squares approximation to its star (the set of triangles containing p_i), achieving the desired dihedrals but introducing error in the edge lengths opposite p. We also define an arbitrary canonical coordinate system F_i at each vertex p_i. For every two stars at p_i and p_j connected by an edge e_{ij}, we find the three dimensional rotation R_{ij} that takes F_i to F_j when the two stars are merged. This gives us a relative rotation along each edge. We use these to solve for a global orientation at each vertex:

$$\min \sum_{e_{ij}} ||G_i R_{ij} - G_j||_F^2$$

where $|| \cdot ||_F$ indicates the Frobrenious norm. This is a least-squares solve for the R_{ij}. Because of numerical error and the poor conditioning of the system, we may end up with "rotations" R_{ij} that are not actually orthonormal. Following [Mao86], we correct these using the singular value decomposition. This produces a set of global rotations aligning the coordinate frames at every vertex. Given a good set of G_i matrices, we can then use them to reconstruct vertex positions, using

$$\min \sum_{\vec{e}_{ij}} ||(p_i - p_j) + G_i p_{ij}||^2$$

Here p_{ij} represents the position of the copy of vertex p_j in the original coordinate frame at p_i; as transformed by the global rotation, it should be equal to $p_i - p_j$. Again, this is a least-squares computation.

Vertex Unfoldings of Orthogonal Polyhedra: Positive, Negative, and Inconclusive Results

Luis A. Garcia * Andres Gutierrrez * Isaac Ruiz * Andrew Winslow *

Abstract

We obtain results for three questions regarding vertex unfoldings of orthogonal polyhedra. The (positive) first result is a simple proof that all genus-0 and genus-1 orthogonal polyhedra have grid vertex unfoldings. The (negative) second result is an orthogonal polyhedron that is not vertex-unfoldable. The (inconclusive) third result is a vertex unfolding of an orthogonal polyhedron that cannot be arranged orthogonally, evidence that deciding whether an orthogonal polyhedron has a vertex unfolding may not lie in NP.

1 Introduction

The study of unfolding polyhedral surfaces can be traced to as early as the 1500s when Albrecht Dürer considered unfoldings of convex polyhedra via cuts only along edges (called *edge unfoldings*).

To explore unfoldings of non-convex polyhedra, Biedl et al. [3] considered *orthogonal* polyhedra, in which all faces are perpendicular to one of the three axes, providing examples of edge ununfoldable orthogonal polyhedra as well as methods for unfolding some classes of orthogonal polyhedra. Subsequent work has further explored the boundary between unfoldable and ununfoldable orthogonal polyhedra by varying both the types of unfoldings permitted and classes of orthogonal polyhedra under consideration.

For instance, one line of work has considered broadening permitted unfoldings via *refinement*: adding a regular grid of (potential cut) edges to each face. Damian, Flatland, and O'Rourke [8] proved that exponential refinement was sufficient to unfold any orthogonal polyhedron; this refinement was later reduced to quadratic [7], and then linear [6].

To unfold the edge ununfoldable examples of Biedl et al. [3], it is sufficient to add *grid edges* formed by the intersection of orthogonal planes intersecting each vertex of the polyhedron. Such *grid edge unfoldings* have been found for several classes of orthogonal polyhedra [3, 5, 10, 13].

*Department of Computer Science, University of Texas Rio Grande Valley, Edinburg, TX. luis.a.garcia01@utrgv.edu, andres.a.gutierrez01@utrgv.edu, isaac.ruiz02@utrgv.edu, andrew.winslow@utrgv.edu

Extending grid unfoldings to allow cuts meeting at (but excluding) a vertex yields *grid vertex unfoldings*. Vertex unfoldings were first introduced for general polyhedra [11], and grid vertex unfoldings have been shown to exist for some classes of orthogonal polyhedra [12], including all genus-0 orthogonal polyhedra [9].

Here we obtain three new results on grid vertex unfoldings of orthogonal polyhedra:

- Every genus-0 and genus-1 orthogonal polyhedron has a grid vertex unfolding (extending a previous result of Damian, Flatland, and O'Rourke [9] to include genus 1 and providing an alternative, simpler proof).
- There exists an orthogonal polyhedron with faces homeomorphic to disks that does not have a vertex unfolding (complementing a vertex-ununfoldable topologically convex polyhedron of Abel, Demaine, and Demaine [2] and vertex-ununfoldable orthogonal polyhedra with faces not homeomorphic to disks by Biedl et al. [3]).
- There exists a "maximally cut" vertex unfolding of an orthogonal polyhedron that cannot be made orthogonal, raising the question of whether deciding if an orthogonal polyhedron has a vertex unfolding is in NP (in contrast with the trivial containment in NP of deciding whether an orthogonal polyhedron has an *edge* unfolding [1]).

2 Definitions

This work considers orthogonal polyhedra, i.e. polyhedra where all edges are parallel to the x-, y-, or z-axis. A *gridded* polyhedron has edges everywhere that an xy-, xz-, or yz-plane intersects a vertex of the polyhedron, resulting in faces that are edge-adjacent rectangles. A *polycube* is a special case of gridded orthogonal polyhedra whose faces are unit squares.

An *unfolding* is a connected planar arrangement of the surface of a polyhedron by the addition of cuts. If the cuts are restricted to the polyhedron's edges, then the resulting unfolding is an *edge unfolding* if the surface remains strongly connected, and a *vertex unfolding* otherwise. A surface that permits no additional edges or vertices to be cut without disconnecting the surface is *maximally cut*.

Unfoldings that are (possibly weakly connected) orthogonal polygons are also called *orthogonal*. In the case of vertex unfoldings, point connectivity on the surface yields a "hinge" or "joint" allowing portions of the surface to rotate relative to each other. Thus vertex unfoldings are weakly connected polygons that may either be orthogonal or not.

The flexibility around vertices allows for faces in a vertex unfolding to (potentially) not appear in the same clockwise order around a common vertex as they do on the original surface; here we allow such "rearrangement" of vertex-adjacent faces (see [11] for further discussion).

3 A Simple Proof that Genus-≤ 1 Orthogonal Polyhedra have Grid Vertex Unfoldings

Here we prove the existence of vertex unfoldings for low-genus polycubes using an approach reminiscent of a proof of a similar result [11] for polyhedra with (possibly intersecting) triangular faces. This implies a corollary (Corollary 2) for grid vertex unfoldings that extends the previous result by Damian, Flatland, and O'Rourke [9] on grid vertex unfoldings to include genus-1 polyhedra.

Theorem 1 *Every genus-0 and genus-1 polycube has a vertex unfolding.*

Proof. First, we prove that the face dual graph (obtained by creating a vertex for every face of the polycube and connecting pairs of edge-adjacent faces) of every such polycube has a Hamiltonian path. Afterwards, we show how to use a Hamiltonian path through the faces to obtain a vertex unfolding.

Bodini and Lefranc [4] prove that polycube face dual graphs are 4-connected. Intuitively, this is due to the four directions that must be traversed to obtain a disconnecting cut of the surface of a polycube (i.e. non-contractable cycle on the surface). As they observe, Tutte [15] proves that all genus-0 4-connected graphs are Hamiltonian, implying that all face dual graphs of genus-0 polycubes are Hamiltonian. Recently, Thomas, Yu, and Zang [14] proved that all genus-1 4-connected graphs have a Hamiltonian path.

The vertex unfolding consists of a path of (vertex) connected faces appearing in the same order as on the Hamiltonian path previously proved to exist. These faces are arranged left to right, and remain either edge-connected (if the previous face is left of the next face) or cut to become vertex-connected and rotated by $\pm 90°$ (if the previous face is above or below the next face). These three cases are seen in Figure 1.

Only these three cases need be considered due to maintaining the invariant that before the edge connecting the previous (gray) face to the next (green) face is cut, the next face is above, below, or to the right of the previous face. □

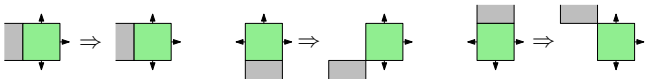


Figure 1: The three cases for vertex unfolding a sequence of consecutive faces along a Hamiltonian path of the face dual graph. The invariant maintained is that each subsequent face (in green) begins attached to the above, below, or right of the previous face (indicated as arrows). In the right two cases, the face is rotated $\pm 90°$ to maintain the invariant.

Observe that the resulting unfolding is also orthgonal. Moreover, because the resulting unfolding can be partitioned into vertical strips each containing one face of the polycube, a similar result holds if the faces are edge-adjacent rectangles, as they are in all "gridded" orthogonal polyhedra:

Corollary 2 *Every genus-0 or genus-1 orthogonal polyhedron has an orthogonal grid vertex unfolding.*

4 A Vertex-Ununfoldable Orthogonal Polyhedron

Theorem 3 *There is an orthogonal polyhedron with simple faces that cannot be vertex-unfolded.*

Proof. The vertex-ununfoldable polyhedron consists of a box with a thin "ridge" through two adjacent faces of the box (see Figure 2).

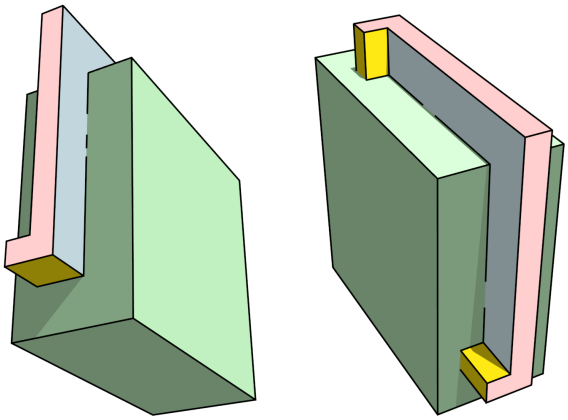


Figure 2: An orthogonal polyhedron that cannot be vertex unfolded.

We prove that the polyhedron is ununfoldable by considering only a portion of the surface consisting of two large *base faces* containing the ridge (green in Figure 2), and the *ell*, *putt*, and *bungie faces* on the ridge (blue, pink, and yellow, respectively, in Figure 2).

In the unfolding, at least one of the two (symmetric) putt faces must be connected to a base face via a sequence of faces that excludes the other putt face. For

the remainder of the proof, we consider only this sequence of faces, proving that any such sequence cannot be arranged without overlap.

Since the boundary of the putt shares no boundary with the base face on the surface, the sequence must contain either a bungie or ell face. Regardless of the sequence of faces connecting the putt to the base face, the end of the face sequence is either:

1. ell → base, or
2. putt → bungie → base, or
3. ell → bungie → base.

Case 1: ell → base. The first case implies overlap between the ell face and base face (see Figure 3), as these two faces are connected by either the unique edge they share on the surface or a vertex of this edge.

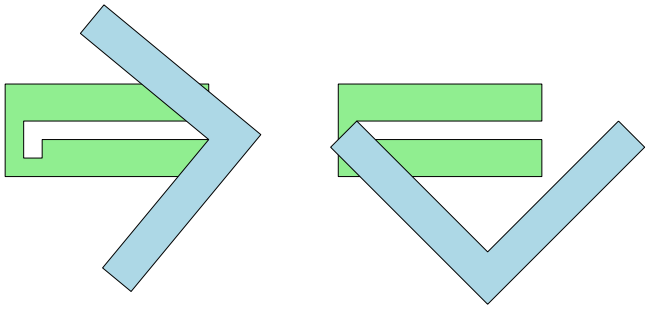


Figure 3: Any attached ell and base face must overlap.

For the other two cases, we simplify the analysis by considering the bungie faces as a zero-width curve of length between 0 and the maximum distance between two locations on a sequence of connected bungie faces (see Figure 4) of $\approx 7.30 < 8$. That is, we suppose they behave as an elastic "bungie cord".

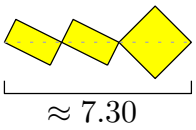


Figure 4: The maximum length of a connected sequence of bungie faces is $\sqrt{5} + \sqrt{5} + \sqrt{8} \approx 7.30$.

Case 2: putt → bungies → base face. In the unfolding, a bungie face connects to the putt at boundary location(s) limited to those drawn in red in Figure 5. Any non-overlapping arrangement of the putt and base faces either has the entire notch filled by the putt (leaving no available location for any bungie face in the unfolding) or has no portion of the boundary of the putt to which the bungie faces can attach more than distance 1 from the entrance of the notch. In the latter case, the minimum distance between the bungie faces' connection to the base face and putt's boundary is at least 8, exceeding the maximum distance that can be spanned by the bungie faces.

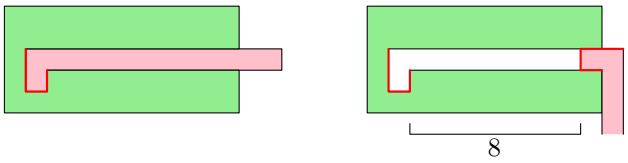


Figure 5: The two potential putt placements for Case 2. The red boundary portions denote where bungie faces must connect (to both the putt and base faces).

Case 3: ell → bungies → base face. This case is proved similarly to Case 2. Since ell faces are 2 units wide, any optimal non-overlapping arrangement of the ell and base faces forms a right triangle consisting of a portion of the ell face, with the 90° vertex on the ell and two remaining vertices at the entrance of the notch (see Figure 6).

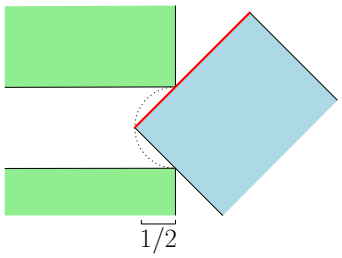


Figure 6: Arranging an ell face as deep in the notch as possible.

By Thales's theorem, the 90° vertex lies on a circle whose diameter is the notch entrance. Then since this circle has radius 1/2, the 90° vertex (and all other locations on the ell) has distance at most 1/2 from the entrance of the notch. So as in Case 2, the minimum distance between the bungie faces' connection to the base face and putt's boundary exceeds the maximum distance that can be spanned by the bungie faces. □

5 Evidence that Vertex Unfolding Orthogonal Polyhedra is not in NP

Finally, we consider the complexity of deciding whether a polycube has a vertex unfolding. Specifically, whether the problem lies in NP.

As Abel and Demaine [1] observe, the same problem limited to edge unfoldings is easily seen to be in NP. One proof uses the following simple algorithm: nondeterministically select a set of maximal set of polycube

faces to cut that leaves the face dual graph connected (i.e. a set of cut edges that yields a tree-shaped face dual graph). Then check whether the resulting surface is indeed an unfolding, i.e. has no overlaps.

This NP algorithm for edge unfolding relies in part on the uniqueness of the induced unfolding. However, in vertex unfoldings, faces may be connected by a single vertex, allowing infinitely many angles at which these two faces may be arranged. Thus the same algorithmic approach for vertex unfoldings requires efficiently determining whether a maximally set of cut edges yields a surface that can be arranged into a vertex unfolding (by careful selection of adjacent face angles).

One natural approach to resolving this issue is to prove that any maximally cut polycube surface has a vertex unfolding only if there is such an unfolding that is orthogonal, i.e. where all adjacent face angles from the set $\{0°, 90°, 180°, 270°\}$. Here, we prove by example that this is not the case.

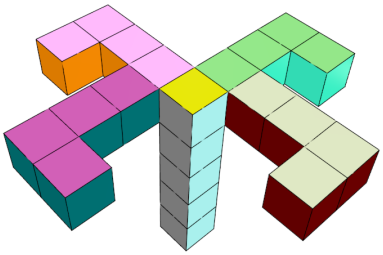


Figure 7: A polycube with a maximally cut vertex unfolding that cannot be arranged orthogonally.

Theorem 4 *There is a maximally cut vertex unfolding of a polycube with no orthogonal arrangement.*

Proof. The maximally cut unfolding with no orthogonal arrangement is seen in Figure 8. The four L-shaped regions adjacent to face c (colored fushia, eggshell, light green, and pink in Figure 8) are *claws*. The two regions attached by a single vertex to face c consist of face sets $A = \{a_1, \ldots, a_5\}$ and $B = \{b_1, \ldots, b_5\}$

For the remainder of the proof, we assume face c is orthogonal. In any orthogonal arrangement of the unfolding, each of six face adjacent to c lies in one of the eight orthogonal locations adjacent to c (see Figure 9). Moreover, faces a_1 and b_1 must lie in locations 2, 3, or 4.[1]

Claw arrangements up to symmetry. Observe that the claws come in symmetric pairs (colored fushia/pink and eggshell/green in Fig. 8) and each pair is also symmetric. Without loss of generality, assume that the

[1]Recall that we allow vertex-adjacent faces to appear in a different clockwise ordering around the vertex than they appear on the surface; the proof holds even when such unfolding are permitted.

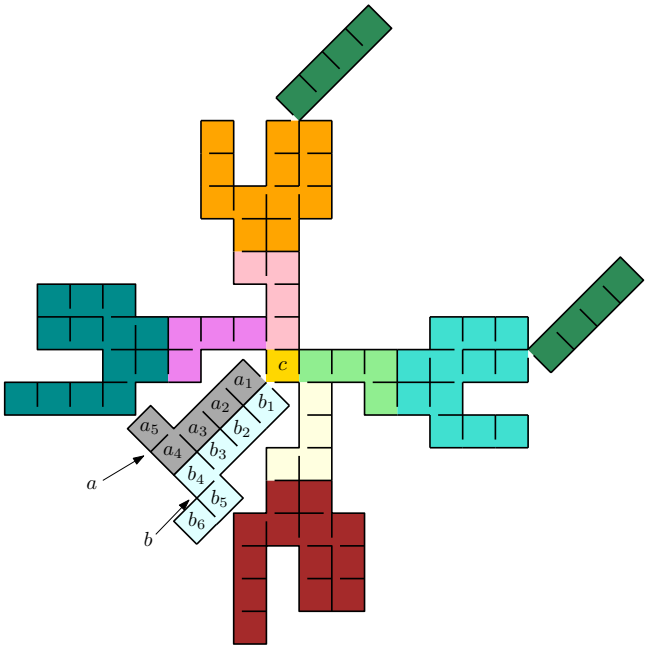


Figure 8: A maximally cut vertex unfolding of the polycube in Figure 7 that cannot be arranged orthogonally. The solid black lines represent cuts.

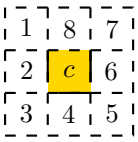


Figure 9: The eight possible locations for faces adjacent to c in orthogonal unfolding.

claws appear in the relative order around c seen in Figure 8 (i.e. in clockwise order, fushia, pink, light green, eggshell).

Then the fushia and pink claws must have faces in locations 1 and 8, respectively, since otherwise overlap occurs between a_1, b_1, c and two fushia faces sharing a common vertex (see Figure 10). By symmetry, the eggshell and light green claws must lie in locations 5 and 6.

Next, consider the arrangement of the fushia claw faces. Figure 11 enumerates the five possible arrangements. The remainder of the proof is dedicated to proving that each arrangement leads to overlap.

Arrangements 4 and 5. Both arrangements cause overlap due to more than four faces sharing a common vertex location. For arrangement 4, these faces consist of two fushia faces, c, a_1, and b_1. For arrangement 5, these faces consist of four fushia faces, one pink face, and c.

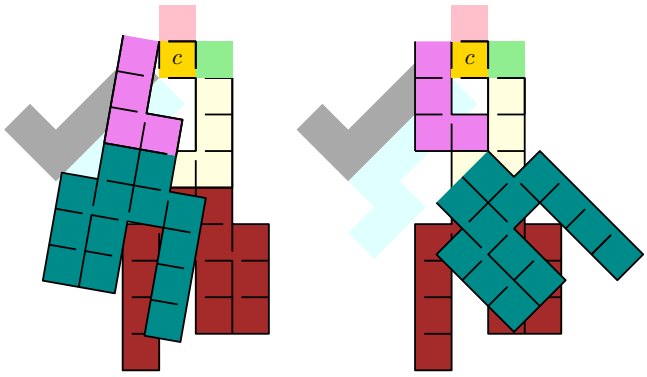


Figure 10: Attemping to place the first face of the fushia claw into location 2 causes overlap.

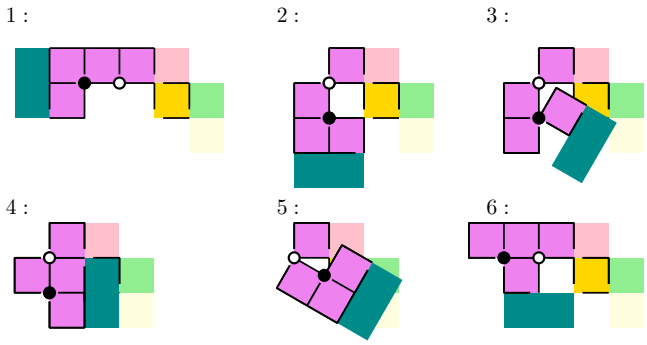


Figure 11: The five possible arrangements of the fushia claw faces.

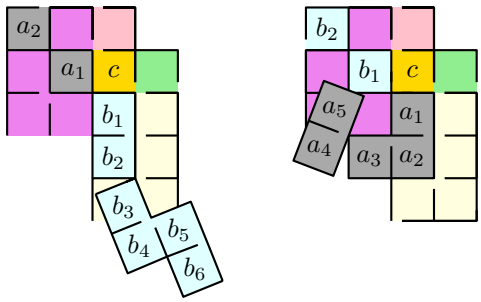


Figure 12: Two possible configurations if crossings are allowed.

Arrangement 3. The faces of A or B may be arranged so that the first face (a_1 or b_1) lies in location 2 as seen in Figure 12. As shown the figure, either option causes overlap.

Arrangements 1 and 2. Here we consider placing A and B. Either a_1 or b_1 must be placed in location 2 or 4. By symmetry (and ignoring the existence of b_6), it suffices to consider two cases:

1. a_1 is placed in location 2.
2. b_1 is placed in location 2.

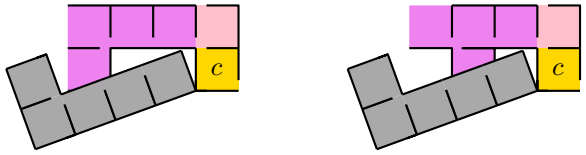


Figure 13: Attempting to unfold A and B by placing a_1 in location 2.

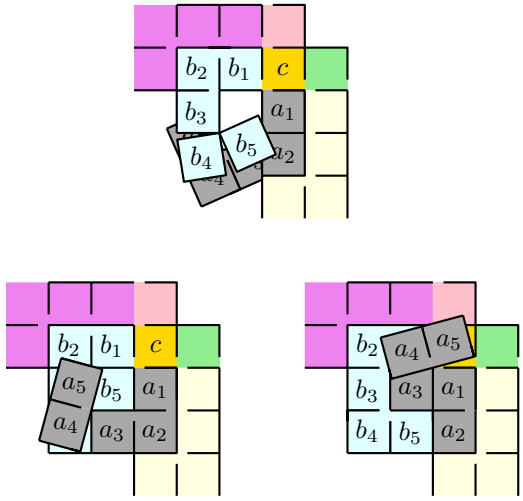


Figure 14: Top: required arrangement of faces b_1, b_2, b_3, a_1, a_2. Bottom: the two options for arranging b_4, b_5, and a_3 that further avoid overlap (in both cases, overlap involving a_4 and a_5 still occurs).

We consider the cases in order. In the case that a_1 is in location 2, A overlaps with the fushia claw, since due to the cuts, a_2 is left of a_1 or overlaps the fushia claw, and likewise a_3 is either left or above a_2.

Next, consider the case that b_1 is placed in location 2. In this case, there are unique placements of faces b_2, b_3, a_1, and a_2 that avoid overlap (the top portion of Figure 14) and only two placements of b_4, b_5 and a_3 that also avoid overlap (the bottom portion of Figure 14). In both cases, a_4 and a_5 overlap with other faces due to a vertex incident to more than four faces. □

6 Open Problems

Each of our results leads directly to a natural open problem in the same direction. Since all genus-0 and genus-1 orthogonal polyhedra have grid vertex unfoldings, what about genus-2?

Open Problem 1 *Does every genus-2 orthogonal polyhedron have a grid vertex unfolding?*

Since there is an orthogonal polyhedron with simple faces and no vertex unfolding, does the same hold for simple faces that are also edge-incident rectangles?

Open Problem 2 *Does every orthogonal polyhedron (of any genus) have a grid vertex unfolding?*

Finally, we provided an example of a maximally cut polycube with an unfolding, but no orthogonal unfolding,[2] demonstrating that the orthogonal unfoldings of maximally cut grided orthogonal polyhedra do not characterize the (unrestricted) unfoldings of maximally cut gridded orthogonal polyhedra (eliminating one particularly simple proof that deciding whether an orthogonal polyhedron has a vertex unfolding is in NP). Thus the following two related problems remain open:

Open Problem 3 *Is deciding if an orthgonal polyhedron is grid vertex-unfoldable in* NP*? Is the problem* NP*-hard?*

Additionally, the relationship between orthogonal grid vertex-unfoldings and (unrestricted) grid vertex-unfoldings also remain open:

Open Problem 4 *Does there exist a grid vertex-unfoldable orthogonal polyheron with no orthogonal grid vertex-unfolding?*

To our knowledge, the same question also remains open for non-grid vertex-unfoldings:

Open Problem 5 *Does there exist a vertex-unfoldable orthogonal polyhedron no orthogonal vertex-unfolding?*

Acknowledgments

We would like to thank Erik Demaine for discussion of known results related to vertex unfoldings, and anonymous reviewers whose suggestions improved the readability and correctness of the paper.

References

[1] Z. Abel and E. D. Demaine. Edge-unfolding orthogonal polyhedra is strongly NP-complete. In *Proceedings of the 23rd Canadian Conference on Computational Geometry (CCCG)*, 2011.

[2] Z. Abel, E. D. Demaine, and M. L. Demaine. A topologically convex vertex-ununfoldable polyhedron. In *Proceedings of the 23rd Canadian Conference on Computational Geometry (CCCG)*, 2011.

[3] T. Biedl, E. Demaine, M. Demaine, A. Lubiw, M. Overmars, J. O'Rourke, S. Robbins, and S. Whitesides. Unfolding some classes of orthogonal polyhedra. In *Proceedings of the 10th Canadian Conference on Computational Geometry (CCCG)*, pages 70–71, 1998.

[2] Note that the polycube from which the example is obtained is indeed vertex (and also edge) unfoldable, and so is not a potential counterexample to Open Problem 2.

[4] O. Bodini and S. Lefranc. How to tile by dominoes the boundary of a polycube. In *Proceedings of the 13th International Conference on Discrete Geometry for Computer Imagery (DGCI)*, volume 4245 of *LNCS*, pages 630–638. Springer, 2006.

[5] E. W. Chambers, K. A. Sykes, and C. M. Traub. Unfolding rectangle-faced orthostacks. In *Proceedings of the 24th Canadian Conference on Computational Geometry (CCCG)*, pages 23–28, 2012.

[6] Y.-J. Chang and H.-C. Yen. Improved algorithms for grid-unfolding orthogonal polyhedra. *International Journal of Computational Geometry & Applications*, 27(1):33–56, 2017.

[7] M. Damian, E. D. Demaine, and R. Flatland. Unfolding orthogonal polyhedra with quadratic refinement: The delta-unfolding algorithm. *Graphs and Combinatorics*, 30(1):25–40, 2014.

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

[9] M. Damian, R. Flatland, and J. O'Rourke. Grid vertex-unfolding orthogonal polyhedra. *Discrete & Computational Geometry*, 39(1–3), 2008.

[10] M. Damian and H. Meijer. Grid edge-unfolding orthostacks with orthogonally convex slabs. In *Proceedings of the 14th Annual Fall Workshop on Computational Geometry*, pages 20–21, 2004.

[11] E. D. Demaine, D. Eppstein, J. Erickson, G.W. Hart, and J. O'Rourke. Vertex-unfoldings of simplicial manifolds. In *Proceedings of the 18th Symposium on Computational Geometry (SoCG)*, pages 237–243. ACM Press, 2002.

[12] E. D. Demaine, J. Iacono, and S. Langerman. Grid vertex-unfolding orthostacks. *International Journal of Computational Geometry & Applications*, 20(3):245–254, 2010.

[13] J. O'Rourke. Unfolding orthogonal terrains. https://arxiv.org/abs/0707.0610, 2007.

[14] R. Thomas, X. Yu, and W. Zang. Hamilton paths in toroidal graphs. *Journal of Combinatorial Theory, Series B*, 94:214–236, 2005.

[15] W. T. Tutte. A theorem on planar graphs. *Transactions of the American Mathematical Society*, 82:99–116, 1956.

Approximate Free Space Construction and Maximum Clearance Path Planning for a Four Degree of Freedom Robot

Chloe Arluck* Victor Milenkovic† Elisha Sacks‡

Abstract

We present an algorithm for constructing an inner approximation of the free space for a polyhedral robot with four degrees of freedom. The robot rotates about a fixed axis and translates in three dimensions with respect to a fixed polyhedral obstacle. We approximate the free space by subdividing the rotation dimension into short angle ranges, generating a three dimensional free space for each angle range, and constructing a graph for navigation in the four dimensional space. We also present an algorithm for path planning that is complete in the approximated space. The path planning algorithm produces paths that are guaranteed to be collision free and approximately maximizes obstacle clearance, ensuring safe and practical paths.

1 Introduction

We present a method of approximating the free space of a polyhedron R that rotates around a fixed axis, without loss of generality the z-axis, and translates freely relative to a polyhedron O. For example, R models a drone helicopter and O models a warehouse. Further, we present a path planning algorithm that demonstrates the benefit of free space construction for fast and efficient navigation: after a one-time computation to construct the free space, we can quickly construct paths between any two configurations or determine that none exists.

There are cases where traditional probabilistic road map planners have poor performance, particularly cases where the path must traverse through narrow passages. Additionally, PRM planners test for collision along local paths by taking discrete samples, and therefore may miss collisions when obstacles are very thin. While many sampling methods have been developed to address the narrow passage problem [1] [2] [5], and collision detection can be guaranteed using adaptive resolution [15], the challenge of choosing the appropriate variety of PRM planner for the problem adds additional tuning complexity for the user. We present a path planning algorithm that requires minimal tuning, handles narrow passages, and produces paths that are always collision free.

*Department of Computer Science, University of Miami c.arluck@cs.miami.edu

†Department of Computer Science, University of Miami vjm@cs.miami.edu

‡Department of Computer Science, Purdue University elisha.sacks@gmail.com

2 Constructing the Free Space

We approximate the four dimensional free space S by subdividing the rotation dimension into short angle ranges and generating a three dimensional free space S_i for each angle range. If n is the number of angle ranges, each S_i is an inner approximation of the free space with rotation $[\frac{2\pi i}{n}, \frac{2\pi(i+1)}{n}]$. Meaning, for all $\theta \in [\frac{2\pi i}{n}, \frac{2\pi(i+1)}{n}]$ and for all $t \in S_i$, R does not intersect O at (t, θ), where (t, θ) denotes a rotation of R by θ then a translation of R by t. To generate S_i, we construct R', a polyhedral approximation of R swept through the rotation $[0, \frac{2\pi}{n}]$ about the z axis. Then $S_i = \overline{O \oplus -R'_i}$ where R'_i is R' rotated $\frac{2\pi i}{n}$ about the z axis, $\oplus$ denotes the Minkowski sum, minus denotes the negation of each vertex, and over line denotes the complement.

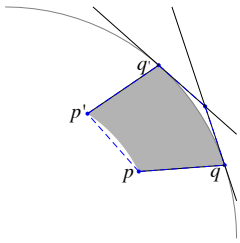
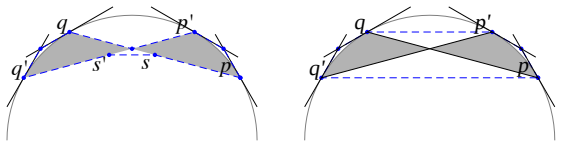


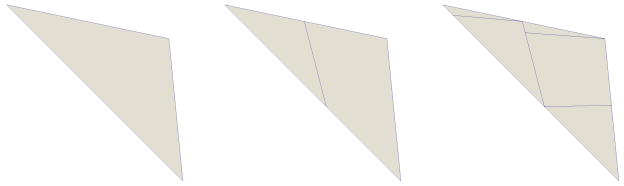
Figure 1: The containing polygon of a segment pq rotated in the plane.

To construct the outer approximation of the sweep of R, we reduce the problem to sweeping segments in the plane. Let p and q be endpoints of a segment in the plane where $|p| \leq |q|$. If p' and q' are p and q rotated θ about the origin, then an outer approximation of the sweep for segment pq is the pentagon with endpoints p, p', q, q', and the intersection of tangent lines at q and q' (Fig. 1). If the nearest point on pq to the origin is not an endpoint, we split at the nearest point and take the sweep of the two segments individually. Fig. 2 shows that not splitting at the nearest point results in a poor approximation.



(a) (b)

Figure 2: (a) The union of polygons containing the sweep of ps and qs, where s is the point on pq closest to the origin, and (b) a single polygon containing the sweep of pq.

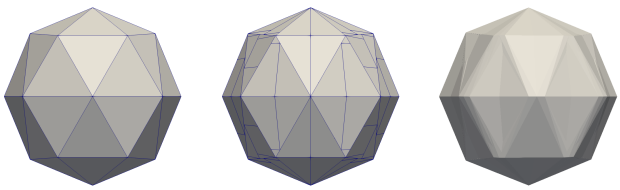


(a) (b) (c)

Figure 3: (a) An input face F and the result of splitting F by (b) its tangent plane and then by (c) xy-planes passing through each vertex.

This two dimensional method is used to construct the outer approximation of the sweep of R as follows. Let F be a face of triangulated polyhedron R. We subdivide F in order to consider it as a set of segments pq that lie on planes parallel to the xy-axis. We split F by the plane containing the nearest point of each segment (Fig. 3(b)) to avoid the poor approximation shown in Fig. 2. Next, we split by planes parallel to the xy-plane going through each vertex (Fig. 3(c)). The result is a set of trapezoids each with their top and bottom edges being segments parallel to the xy-plane.

The outer approximation of the sweep of a trapezoid is the union of the 2D sweep approximations for each z cross section. Sides of this union are ruled but not necessarily planar. We introduce cross section segments separated by at most θ in angle and $r\theta$ in z, where r is the radius of the robot, and we connect adjacent segments by their convex hull. The outer approximation of the sweep of R is its union with the outer approximation of each trapezoid sweep.

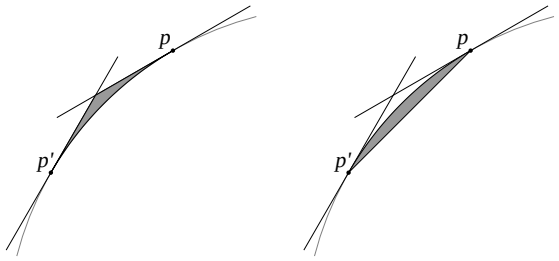


(a) (b) (c)

Figure 4: (a) A triangulated robot R, (b) its tetrahedralization and (c) its sweep polyhedron R'.

Given the three dimensional free spaces S_i, we construct a graph for navigation in the four dimensional space S. A node represents a connected component of a space S_i. If components of S_i and S_{i+1} intersect, then their corresponding nodes are neighbors. By providing the relationship between the inner approximated subspaces S_i, this graph defines an inner approximation of S. After performing the one-time computation to construct the inner approximation of S and the graph, we can query the graph to quickly construct paths between any two configurations, or determine that none exists.

3 Error Bound on the Sweep Polyhedron



(a) (b)

Figure 5: (a) A *cap* and (b) a *cup* at point p.

We wish to find an upper bound on the excess generated by this sweep approximation. For a sweep of a segment the excess is composed of *caps* and *cups*. A cap at a point p is the region between a circle of radius $|p|$ and the tangent lines at p and p', where p' is p rotated θ about the origin (Fig. 5). The area of a cap is $p^2(\tan\frac{\theta}{2} - \frac{\theta}{2})$, which has third degree Taylor series approximation $p^2\frac{\theta^3}{24}$. A cup at p is the region between segment pp' and a circle of radius $|p|$ (Fig. 5). A cup at p has area $p^2\frac{\theta-\sin\theta}{2}$ and Taylor series approximation $p^2\frac{\theta^3}{12}$.

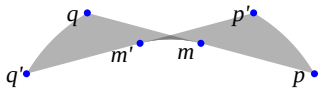


Figure 6: The pseudo area when the midpoint m is the closest point on pq to the origin. In this case equivalent to the actual sweep area.

To find an upper bound on the excess area of the sweep, we consider the worst case. For convenience, we define the *pseudo area*, a measure that is always less than or equal to the actual area swept by segment pq. If m is the midpoint of segment pq, then the pseudo area is defined to be

$$(q^2 + p^2 - 2m^2)\frac{\theta}{2} - m^2\frac{\theta^3}{24}.$$

Intuitively, the pseudo area is the sum of the areas swept by the two subsegments on either side of m without a cap at m. When m is the closest point on pq to the origin, the pseudo area is equivalent to the actual sweep area (Fig. 6) and we will show that it is always less than or equal to the actual sweep area.

If p is the point on the segment closest to the origin, then the area swept by pq is

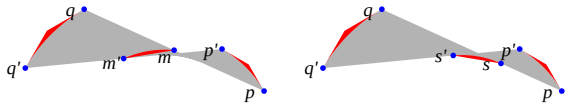
$$\int_{|p|}^{|q|} \theta r dr = (q^2 - p^2)\frac{\theta}{2}.$$

Since p is the closest point on pq to the origin, $|m| > |p|$. Hence $q^2 - p^2 > q^2 + (p^2 - m^2) - m^2$ and the pseudo area must be smaller than the actual area in this case.

If the closest point on pq to the origin is some internal point s, then the area swept by the segment is the sum of the areas swept by the two sub-segments without a cap at s, which is swept by both sub-segments.

$$(q^2 + p^2 - 2s^2)\frac{\theta}{2} - s^2\frac{\theta^3}{24}$$

Since, $|m| \geq |s|$, this area must be greater than or equal to the pseudo area.



(a) (b)

Figure 7: The (a) pseudo excess and (b) actual excess generated when approximating the sweep of segment pq, where m is the midpoint and s is the point closest to the origin.

Similar to the pseudo area, we define the *pseudo excess*, a measure that is always greater than or equal to the excess area generated by the outer approximation of sweeping segment pq. The pseudo excess is a cap at endpoints p and q and a cup at midpoint m (Fig. 7).

$$(q^2 + p^2 + 2m^2)\frac{\theta^3}{24}$$

If p is the point closest to the origin, then the excess is a cap at q and a cup at p.

$$(q^2 + 2p^2)\frac{\theta^3}{24}.$$

Since $|m| \geq |p|$, this value must be smaller than the pseudo excess. If s is the closest point on pq to the origin, then the excess is a cap at p, a cap at q, and a cup at s.

$$(q^2 + p^2 + 2s^2)\frac{\theta^3}{24}$$

Since $|m| > |s|$, this value is greater than or equal to the pseudo excess.

Now, given triangulated polyhedron R, consider an input face. The outer approximation algorithm splits by a plane parallel to the xy-axis at each vertex, so the input face is divided into two triangles sharing an edge pq with length l. We wish to find the vertex positions that result in the smallest pseudo area and the largest pseudo excess, while fixing the z-components of each vertex and the length of segment pq.

Without loss of generality we consider only one of the two triangles. Let t be the vertex opposite edge pq and consider an arbitrary segment $\hat{p}\hat{q}$ where triangle pqt intersects a plane parallel to the xy-plane. Since the z-components of p, q, and t and the length of pq are fixed, the length of $\hat{p}\hat{q}$ is also fixed. Let $\hat{m}$ be the midpoint of $\hat{p}\hat{q}$. If the distance from the z-axis of every $\hat{m}$ does not decrease, then for segment $\hat{p}\hat{q}$ the pseudo area does not increase and the pseudo excess does not decrease. We will perform a series of operations that do not decrease the distance between $\hat{m}$ and the z-axis and that fix the z-components of all vertices and the length of pq.

1. Rotate t about the z-axis until it is aligned with the midpoint of pq. Since t is moving towards the midpoint of pq, $\hat{m}$ is moving away from the z-axis.

2. Rotate p about q and until $|p| = |q|$ while rotating t about the z-axis to remain aligned with the midpoint of pq. Since p is moving away from the z-axis and and t remains aligned with the midpoint, $\hat{m}$ is moving away from the origin.

3. Scale the x and y components of t until t is distance r from the z-axis, where r is radius of the robot.

4. Translate p and q to distance r from the z-axis such that the distance l between p and q stays the same. Then $m^2 = r^2 - \frac{l^2}{4}$, the pseudo area is $\frac{l^2\theta}{4} + (\frac{l^2}{4} - r^2)\frac{\theta^3}{24}$, and the pseudo excess is $(4r^2 - \frac{l^2}{2})\frac{\theta^3}{24}$. As r increases, the pseudo area decreases and the pseudo excess increases.

The result is a scalene triangle where all vertices are at the maximum distance from the origin and t is equidistant to p and q. Changing the position of any vertices either reverses one of the above operations, exceeds the radius of the robot, or violates the original conditions that the z-components and length of pq are fixed. Hence this triangle achieves the minimal allowable pseudo area and the maximal allowable pseudo excess. Since t is equidistant to p and q, the midpoint of each segment is the nearest point, so the pseudo excess and area are equal to the true excess and area. Since the pseudo excess and area are always worst than the true area and excess, this must be the worst case for approximating a face of R. We will integrate to find the volume of the swept triangle and its excess. Let H be the difference in z-component between t and segment pq. The

length of the segment being swept as a function of h, where $h \in [0, H]$ is $L(h) = \frac{l}{H}h$. Each segment on pqt has its nearest point on the midpoint. The swept area of such a segment is the sum of the areas of the two sub-segments minus a cap at its midpoint. This area is bounded below by the area swept by one of the sub-segments $A_{min}(h) = L^2(h)\frac{\theta}{8}$. Then a lower bound on the swept volume is

$$V_{min} = \int_0^H A_{min}(h)dh = \frac{l^2\theta H}{24}.$$

The excess area generated by each segment is $A_{ex}(h) = \frac{\theta^3}{24}(4r^2 - \frac{L^2(h)}{2})$. So the excess volume generated by triangle pqt is

$$V_{ex} = \int_0^H A_{ex}(h)dh = \frac{\theta^3}{24}H(4r^2 - \frac{l^2}{6}).$$

So, an upper bound on the ratio of excess volume to sweep volume is

$$\frac{V_{ex}}{V_{min}} = \frac{24r^2 - l^2}{6l^2}\theta^2$$

where r is the radius of the robot and l is no smaller than minimum altitude of all input triangles. Hence, the error on the polyhedral approximation of the sweep is $O(\theta^2)$ and produces a close approximation when θ is small.

Additional excess is introduced when we approximate ruled surfaces by a sequence of convex hulls, but because they are only necessary for cap and cup segments, which have length $O(\theta)$, by construction these have $O(\theta^4)$ volume and there are $O(\frac{1}{\theta})$ of them, and so the error they introduce is also $O(\theta^2)$.

4 Path Panning

Suppose we want to navigate between configurations (t_a, θ_a) and (t_b, θ_b). Then θ_a is contained in $[\frac{2\pi i}{n}, \frac{2\pi(i+1)}{n}]$ for some i and t lies in some connected component of S_i. This defines a node that contains (t_a, θ_a) and, similarly, a node that contains (t_b, θ_b).

Given the graph we constructed, we can perform a breadth first search between these nodes to quickly determine whether such a path exists. Since the graph defines an inner approximation of S, we may return a false negative but not a false positive. If a path exists, our search will return a sequence of free space components $\{C_1, C_2, ..., C_n\}$ connecting (t_a, θ_a) and (t_b, θ_b) if one exists.

Given a method of finding paths between two points in the same component, we can construct a valid path from (t_a, θ_a) to (t_b, θ_b) as follows: Navigate inside C_1 from t_a to some point t_2 in $C_1 \cap C_2$. Rotate from θ_a to some θ_2 in the range of C_2. Similarly, for each $i > 2$, navigate inside C_{i-1} from t_{i-1} to some point t_i in $C_{i-1} \cap C_i$ and rotate from θ_{i-1} to some θ_i in the range of C_i. Lastly, navigate in C_n from t_n to t_b and rotate from θ_n to θ_b. The result is a valid path from (t_a, θ_a) to (t_b, θ_b).

For generating a valid path, the choice of t_i is arbitrary. However, a more methodical choice of t_i can reduce path length. Working backwards, we assign t_n to the nearest point in $C_{n-1} \cap C_n$ to t_b and, iteratively, assign t_i to the nearest point in $C_{i-1} \cap C_i$ to t_{i+1}. While the problem is symmetrical, working backwards from the terminal point results in a more intuitive path: the robot will approach an obstacle and then rotate to maneuver around it.

5 Finding Paths in a Free Space Component

The described path planning algorithm requires a method of finding a path between two points in a given component. In general, the problem of finding the shortest path between two points among polyhedral obstacles is NP-Hard [3], so instead we seek a valid and reasonable path. The shortest path can be approximated in polynomial time using a visibility graph [6] [9] [13]. The more densely the graph is constructed, the closer the approximation. However, in practice, achieving a good approximation is expensive.

We instead reduce the problem to finding paths along the surface of the obstacle. The problem of finding shortest paths on the surface is much simpler and can be solved in polynomial time by wavefront propagation [7] or by partitioning of the obstacle surface [16] [11] [12]. We implement a simpler algorithm for finding paths on the surface.

We reduce the problem to finding paths on the surface as follows: To find a path from p to q, which lie in component C, we find all the points where the segment pq intersects with a face of C. If any portion of pq lies outside of C, we replace it with a path on the surface of C. The result is a path that is fairly intuitive: the robot will move directly towards its destination and maneuver around obstacles as it encounters them.

We find a path from s to t on the surface of a triangulated connected component C as follows. First, we use breadth first search to find a sequence of neighboring faces $\{T_1, T_2, ..., T_n\}$ that connect the containing faces of s and t (Fig. 8(a)). Let t_i be the transformation that rotates T_{i+1} about its shared edge with T_i so that the two triangles lie in the same plane. By applying $t_1 \circ t_2 \circ ... \circ t_{i-1}$ to each T_i, the faces are 'unfolded' to all lie in the same plane (Fig. 8(b)). Since the common edges of the triangles are unchanged, the shortest path through all the common edges is the same in the planar problem as it is in the original problem.

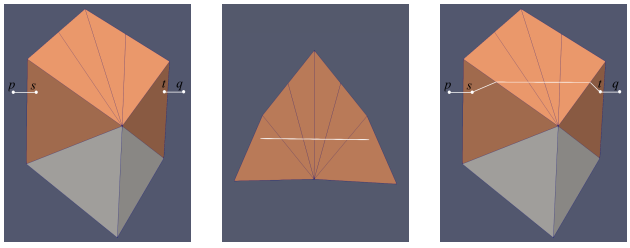
We use a funnel algorithm to find the shortest path

(a) (b) (c)

Figure 8: (a) The sequence of triangles returned by breadth first search for a path from s to t, (b) the sequence unfolded into a 2D and the resulting shortest path, and (c) the 2D shortest path converted back to 3D.

through every common edge of the planar triangles in linear time [4]. We add each common edge, finding the convex hull of the right hand vertices by making only right hand turns and the convex hull of the left hand vertices by making only left hand turns. Whenever the right path becomes left of the left path, move elements from the beginning of the left path to the end of the output path. Whenever the left path becomes right of the right path, move elements from the beginning of the right path to the end of the output path.

Given the shortest path in the planar problem, we find the intersection of the path with each common edge. For each intersection point, we find the point that is the same distance along the equivalent three dimensional edge. The result is the shortest path from s to t through $\{T_1, T_2, ..., T_n\}$ (Fig. 8(c)).



(a) (b) (c)

Figure 9: (a) The sequence of triangles found by going the other way around the vertex shown in Fig. 8, (b) the sequence unfolded into 2D and the resulting shortest path, and (c) the 2D shortest path converted back to 3D.

Since the resulting path is dependent on a sequence of faces $\{T_1, T_2, ..., T_n\}$, we iteratively modify the sequence of faces until we reach a local optimum. For each point where the path goes through a vertex v, the path may improve by going the other way around v. We replace all T_i that are incident on v with faces on the other side of v and perform the two dimensional shortest path algorithm again (Fig. 9). The resulting path is either shorter or unchanged. We repeat this process until the path remains unchanged for every vertex on the path. The result is a locally optimal path on the surface of C.

6 Approximate Maximization of Path Clearance

Given a graph of free space components and the described method of generating paths within a component, we can navigate between any two connected points in the free space. However, the resulting path may contain points on the boundary of the free space, where the robot would scrape against an obstacle. In practice, it is preferable to generate paths with sufficient distance between the robot and any obstacle, for safety and for the maneuverability of the robot.

We expand the graph to account for path clearance by structuring it in levels, where deeper nodes represent free space components with more clearance. The user selects the unit of clearance d and the number of levels l. For each level $l > 0$, S_i^l is the subset of S_i where the robot has at least $d \cdot 2^{l-1}$ clearance from O, generated by taking the Minkowski difference of S_i with a sphere of radius $d \cdot 2^{l-1}$. As before, each component of S_i^l is represented by a node in the graph and edges are placed between intersecting components belonging to neighboring angles ranges. So, at each level, the graph is a representation of the four dimensional free space with clearance of at least $d \cdot 2^{l-1}$.

Additionally, each component is connected to its children: the components of the succeeding level that are contained in it. When a component of S_i^l is narrowed to produce a space with more clearance, it may be eliminated or split into multiple components of S_i^{l+1}. Hence a component may have zero, one, or multiple children.

Given this new graph, we now have the capacity to search for paths in the free space at multiple clearance values. Suppose we want to navigate between components C_a and C_b. A path that traverses deeper nodes corresponds to a path with more obstacle clearance, so we search for a path that maximizes node depth. A simple algorithm to find the deepest path is to visit each node, starting at level 0, and remove the node if doing so does not disconnect C_a and C_b.

A maximal depth search has the advantage of maximizing the clearance on a local basis. The robot can traverse through high clearance components in parts of the path where space is available and also squeeze through tight passages. We choose to increase the clearance unit exponentially in order to capture multiple resolutions with relatively few levels. This results in better paths for problems where the tightness varies greatly at different points in the workspace.

We can further optimize the clearance of the output path by adjusting the paths between two points within a

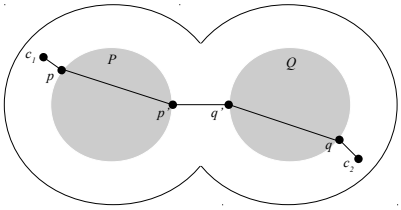


Figure 10: A path from c_1 to c_2 within a component that traverses through its nearest children, P and Q.

component. Suppose we need a path between points c_1 and c_2 within component C, and C has children. Then the children are contained in C and have more clearance. So a path through the children of C is preferable to a path through C. We wish to generate a path though C that avoids the space outside of its children. We first find P and Q, the nearest children to c_1 and c_2, respectively. Next, we find p and q, the nearest point in P to c_1 and nearest point in Q to c_2. If p' and q' are the nearest pair of points between P and Q then $\{c_1, p, p', q', q, c_2\}$ is a path through C that avoids the space outside its children (Fig. 10). If P or Q also have children, we recursively use the same algorithm to find paths in P or Q. This algorithm minimizes the distance the robot must travel in components with low clearance. For faster path finding, we find and store p' and q' for each pair of nodes that share a parent during graph construction.

7 Implementation Details

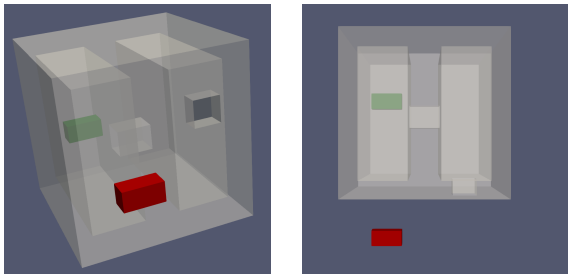
All computation is implemented using the adaptive precision controlled perturbation robustness library [14] [1]. The library ensures results are accurate to the user specified error bound. The Minkowski sums, which are required to generate each three dimensional subspace, are computed using a convolution based approach [8].

For convenience, we save each each subspace and the graph of free space components to files to be used for future path queries. Doing so requires converting the polyhedra with high precision coordinates to meshes with floating point coordinates. The vertices are rounded using a geometric rounding algorithm that preserves the topology of the mesh [10].

8 Results

In the problem depicted in Fig. 11, the obstacle is a box with two inner chambers. There are narrow paths connecting the first and second chamber and connecting the second chamber to the outside. The two narrow paths have opposite orientation and navigating from the start position inside the first chamber to the

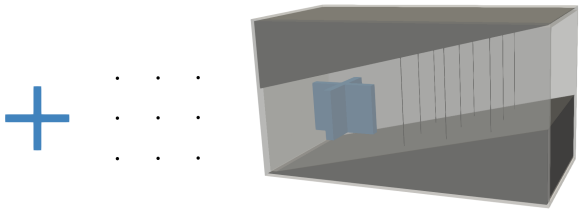
[1] `http://www.cs.miami.edu/home/vjm/robust/`



(a) (b)

Figure 11: Start (green) and end (red) positions for a rectangular robot navigating around a two chamber obstacle.

end position outside requires the robot to make two 90 degree turns. We generate a path for this problem that approximately maximizes the robot's distance from the walls. An animation of that path is available at: `http://web.cs.miami.edu/home/arluck/tworoom/`[2]. The animation shows the scene from three different views: one in each chamber and one outside.



(a) (b)

Figure 12: A cross shaped robot navigating through (a) a 2D integer lattice and (b) a 3D integer lattice of narrow poles on an incline.

Next, consider the problem in Figure 12. There is a cross shaped robot inside a room open on one end. Between the robot and the exit is an integer lattice of narrow poles with a radius of 10^{-5}. This case is difficult for traditional PRM, since it would require a very small step size in order to detect the collision with each pole. We generate a path that rotates back and forth to weave through the lattice while rising to accommodate the incline. An animation of that path is available at: `http://web.cs.miami.edu/home/arluck/lattice/`.

For both problems, we divide the rotation dimension into 40 angle ranges. On one core of a machine with an Intel Xeon E7 CPU, we generate the 40 free spaces in just under 10 minutes for the first problem and 25 for the second problem. We construct the no-clearance graph in about 15 and 30 minutes and each addition level of clearance in 40 minutes and 1 hour. After the one-time computation of the free space, we can generate paths between any two configurations in only 5-10 seconds.

[2] Animation created by Hal Milenkovic

Acknowledgments

Arluck and Milenkovic are supported by NSF grant CCF-1526335. Sacks is supported by NSF grant CCF-1524455.

References

[1] N. M. Amato, O. B. Bayazit, and L. K. Dale. OBPRM: An obstacle-based PRM for 3D workspaces, 1998.

[2] V. Boor, M. H. Overmars, and A. F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 2, pages 1018–1023 vol.2, 1999.

[3] J. Canny and J. Reif. New lower bound techniques for robot motion planning problems. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 49–60, Oct 1987.

[4] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1):209–233, Nov 1987.

[5] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In *Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics on Robotics : The Algorithmic Perspective: The Algorithmic Perspective*, WAFR '98, pages 141–153, Natick, MA, USA, 1998. A. K. Peters, Ltd.

[6] K. Jiang, L. S. Seneviratne, and S. W. E. Earles. Finding the 3D shortest path with visibility graph and minimum potential energy. In *Intelligent Robots and Systems '93, IROS '93. Proceedings of the 1993 IEEE/RSJ International Conference on*, volume 1, pages 679–684 vol.1, Jul 1993.

[7] S. Kapoor. Efficient computation of geodesic shortest paths. In *Proceedings of the Thirty-first Annual ACM Symposium on Theory of Computing*, STOC '99, pages 770–779, New York, NY, USA, 1999. ACM.

[8] M.-H. Kyung, E. Sacks, and V. Milenkovic. Robust polyhedral Minkowski sums with GPU implementation. *Computer-Aided Design*, 6768:48–57, 2015.

[9] T. Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, Oct. 1979.

[10] V. Milenkovic and E. Sacks. Geometric Rounding and Feature Separation in Meshes. *ArXiv e-prints*, May 2018.

[11] D. Mount. On finding shortest paths on convex polyhedra. page 35, 05 1985.

[12] J. O'Rourke, S. Suri, and H. Booth. Shortest paths on polyhedral surfaces. In K. Mehlhorn, editor, *STACS 85*, pages 243–254, Berlin, Heidelberg, 1984. Springer Berlin Heidelberg.

[13] C. H. Papadimitriou. An algorithm for shortest-path motion in three dimensions. *Information Processing Letters*, 20(5):259 – 263, 1985.

[14] E. Sacks and V. Milenkovic. Robust cascading of operations on polyhedra. *Computer-Aided Design*, 46:216–220, Jan. 2014.

[15] F. Schwarzer, M. Saha, and J.-C. Latombe. *Exact Collision Checking of Robot Paths*, pages 25–41. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[16] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, pages 144–153, New York, NY, USA, 1984. ACM.

Integral Unit Bar-Visibility Graphs

Therese Biedl* Ahmad Biniaz* Veronika Irvine* Philipp Kindermann* Anurag Murty Naredla*
Alexi Turcotte*

Abstract

In this paper, we take another look at unit bar-visibility representations, that is bar-visibility representations where every bar has the same width. Motivated by some applications in textile construction, we restrict the graphs further to integral unit bar-visibility representations (IUBVR), that is a bar-visibility representation where the bar of every vertex is a horizontal line segment $[i-1, i]$, for some positive integer i, at some real-value y position.

We study which graph classes do/don't have an IUBVR, both in the weak model and in the strong model. In the weak model, we show that it is NP-hard to test whether a graph has an IUBVR. We also present recursive algorithms to create IUBVRs for some graph classes, such as 2-connected outerplanar graphs with maximum degree 4. In the strong model, we provide a polynomial-time algorithm to test for the existence of a strong IUBVR. In the event of a positive answer, the algorithm also generates such a strong IUBVR.

1 Introduction

The topic of *bar-visibility representations* is well-studied in the graph drawing community. We want to represent a graph by assigning a horizontal line segment (*bar*) to every vertex in such a way that no two bars share a point and for every edge (a, b), the two bars assigned to a and b are visible to each other in the sense that some vertical segment drawn from a reaches b without crossing any other vertices—we call this vertical segment a *line-of-sight*. There are some variations of this idea. Sometimes a line-of-sight is required to exist along a positive-width strip (ε-*visibility*) [11, 3]. Also, in the *strong model*, if a line-of-sight exists between two intervals, then there *must* be a corresponding edge in the graph, whereas in the *weak model* such edges may or may not exist.

It is well-understood which graphs have a bar-visibility representation (we give a detailed review below). Researchers have therefore turned their attention to versions where the bars are further restricted. Of particular interest to us are *unit bar-visibility representations*, where every bar has unit width. Motivated by some applications in textile construction, we take this one step further and study in this paper an *integral unit bar-visibility representation* (IUBVR), which is a bar-visibility representation where the bar of every vertex has the form $[i_v-1, i_v] \times y_v$ for some $i_v \in \mathbb{N}$ and $y_v \in \mathbb{R}$.

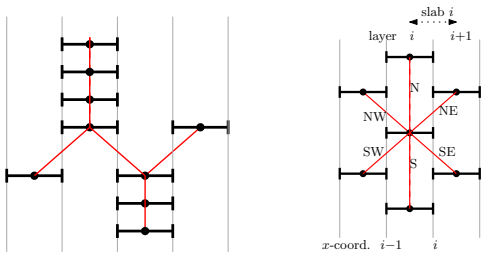
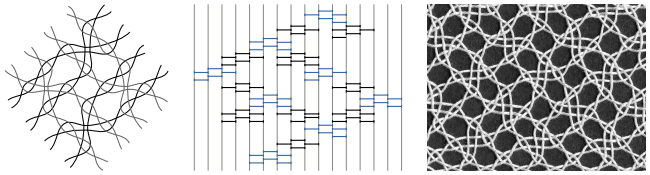


Figure 1: A strong IUBVR of a tree, and six edge-directions at a vertex.

Motivation: We first came across the idea of IUBVR in a problem we studied related to bobbin lace. In bobbin lace, and other methods of creating textiles such as macramé or friendship bracelets, the standard setup is several strings hanging down in parallel. The artist picks up a few (typically in multiples of two) consecutive strings and first braids (or knots) them together, then releases the strings and creates another braid elsewhere. A braid can only be executed if the strings involved hang freely, i.e., the braid must be below all previously executed braids involving any of the strings in this subset. This can be modeled as a graph as follows: Define the vertical line with x-coordinate $i \in \mathbb{N}$ to represent one of the strings. Now draw a horizontal bar $[i-1, i]$ to represent a braid that involves the strings at $i-1$ and i; the y-coordinate of the bar represents the relative order of the braid.

Notice that the strong bar-visibility representation induced by these bars is an IUBVR. If we direct all edges in the resulting graph downward, then the possible topological orders of the resulting digraph correspond exactly to the orders in which we can execute the braids. Because we want to maximize the number of crossings that can be made using the threads (or a subset of the threads) already in the artist's hand, a function of braiding-order, we became interested in the types of graphs that could be represented in such a fash-

*University of Waterloo, Canada. Research of TB, AB and VI supported by NSERC. This research was initiated at the Algorithmic Problem Session group at the University of Waterloo; many thanks to Craig Kaplan and Anna Lubiw for helpful input.



(a) (b) (c)

Figure 2: Bobbin lace motivation: (a) Drawing of one repeat, (b) sIUBVR of pattern, (c) several repeats worked in cotton thread

ion.

Our contributions: In this paper, we study which graph classes do/don't have an IUBVR, both in the weak model and in the strong model. In the weak model, we show that it is NP-hard to test whether a plane graph has an IUBVR; we denote a weak IUBVR by wIUBVR. We also present recursive algorithms to create IUBVRs for some graph classes, such as 2-connected outerplanar graphs with maximum degree 4.

We then turn to the strong model. As a warm-up, we argue exactly which trees have a strong IUBVR, denoted by sIUBVR. Then, we turn to the most intricate result of this paper and provide a polynomial-time algorithm to test for the existence of an sIUBVR. In the event of a positive answer, the algorithm also generates such an sIUBVR.

1.1 Related work

The primary application of bar-visibility graphs is to generate a compact layout for a printed VLSI circuit board. The research in this area covers two main topics: 1) characterizing all graphs that have a bar-visibility representation and 2) determining whether a specific graph supports a bar-visibility representation.

Every planar graph has a weak bar-visibility representation [4] and that representation can be found in linear time [8, 9, 11]. Determining whether a 3-connected planar graph has a strong bar-visibility representation was shown to be NP-complete by Andreae [1]. However, for maximal planar and 4-connected planar graphs, there exist $O(|V|)$ and $O(|V|^3)$ algorithms, respectively, for computing a strong visibility representation [11].

Melnikov introduced the idea of ε-visibility [7] as the model most germane to VLSI layout. Duchet showed that every maximal planar graph has an ε-visibility representation [4] which Thomassen extended to all 3-connected planar graphs [12]. Wismath [14] as well as Tamassia and Tollis [11] independently characterized bar-visibility graphs under the ε-model as planar graphs having all cutpoints on a single face. This can be tested, and the representation constructed, in linear time [11].

In addition to bars, axis-aligned rectangles which admit a horizontal as well as a vertical line-of-sight have been explored [2].

Unit bar-visibility representations The concept of bar-visibility graphs with uniform length bars was first studied by Dean and Veytsel [3] under the ε-model. They characterized the existence of such representations for several graph classes including trees, complete bipartite, outerplanar and triangulated graphs. Wiglesworth [13] characterized graphs with a bar-visibility representation with reach (maximum distance between the left and right bar coordinates) less than 2.

Layered drawings IUBVRs turn out to be closely related to so-called *layered drawings*, see e.g. Suderman [10] and the references therein. A *layered drawing* in the most general sense is a planar straight-line drawing where every vertex is assigned to a *layer* or *level*, i.e., a vertical line with integer x-coordinate.[1] There are a number of different models, depending on what types of edges are allowed. In this paper, we consider *short layered drawings*: every edge (v, w) must satisfy that its ends are either in the same layer or in adjacent layers. Any IUBVR naturally gives rise to a short layered drawing, see below. The second type of layered drawing that we will need is called *proper layered drawing* in [10] but we use the term *leveled-planar drawing* from [6]; here, for every edge (v, w) the two vertices must be on adjacent layers. Heath and Rosenberg [6] showed that it is NP-hard to determine whether a planar graph has a leveled-planar drawing. Suderman [10] studied various models of layered drawings with the objective of obtaining such drawings with few layers for trees. It is also known that for various models of layered drawings minimizing the number of layers is fixed-parameter tractable in the number of layers [5].

2 Preliminaries

Since a bar visibility representation can only exist for a planar graph, we assume throughout the paper that G is a planar graph, i.e., can be drawn without crossings in the plane. We usually assume that G is *plane*, i.e., we have fixed the clockwise order of edges at every vertex (which determines the *faces*) and the outer-face of G.

Fix an IUBVR D of G where, as before, vertex v has bar $[i_v-1, i_v] \times y_v$. We can created a drawing Γ from D by placing vertex v at point $(i_v - \frac{1}{2}, y_v)$ and drawing edges as straight-line segments. It is straightforward to verify that Γ is a planar short layered drawing; we call this the *associated layered drawing of* D. (Not all short layered drawings come from an IUBVR.) Based on the associated layered drawing, the following terminology is natural: Vertex v is said to reside in *layer* i *of* D if $i_v{=}i$.

[1] Some references use horizontal lines instead; this is the same after a rotation.

An edge e *spans slab* i *of* D if the ends of e are in layers i and $i{+}1$. We furthermore need the following crucial observation (see also Fig. 1):

Observation 1 *In an IUBVR D, every vertex v has at most 6 incident edges which can be classified as follows:*

- *There can be at most one* N-edge *connecting v to a neighbour w with $i_w{=}i_v$ and $y_w{>}y_v$.*
- *There can be at most one* NE-edge *connecting v to a neighbour w with $i_w{=}i_v{+}1$ and $y_w{>}y_v$.*
- *There can be at most one* NW-edge *connecting v to a neighbour w with $i_w{=}i_v{-}1$ and $y_w{>}y_v$.*
- *Symmetrically there can be at most one* S-edge, SE-edge *and* SW-edge *in which the condition "$y_w{>}y_v$" is replaced by "$y_w{<}y_v$".*

Proof. Assume that there are two NE-edges, say (v, w) and (v, x). So $i_w = i_x = i_v + 1$ and (say) $y_v < y_w < y_x$. But then the bar of w blocks all lines of sight from v to x. So there is at most one NE-edge. □

Fix some $\alpha \in \{\mathrm{NW}, \mathrm{N}, \mathrm{NE}, \mathrm{SE}, \mathrm{S}, \mathrm{SW}\}$. In an IUBVR, we say that a directed edge $v \to w$ has *orientation* α if it is the α-edge at v. We use the term also for an undirected edge, meaning that it becomes an α-edge when directed suitably. We also call edges *vertical*, *diagonal*, *upward-diagonal* and *downward-diagonal* in the obvious way. A vertex v may or may not have an α-edge. When creating drawings, we sometimes use the term α-*port* for the possibility of adding an edge at v in that orientation.

We say that two IUBVRs, D and D', are *equivalent* if (possibly after a translation) the layers contain the same vertices in the same top-to-bottom order, and (after imposing arbitrary directions) the edges have the same orientations in D and D'. We say that a graph has a *unique* IUBVR if all its IUBVRs are equivalent up to a rotation by π.

3 Graph classes that admit weak IUBVRs

In this section, we show that all trees and 2-connected outerplanar graphs of maximum degree 4 admit wIUBVR.

Theorem 1 *Every tree T of maximum degree 4 has a wIUBVR.*

Proof. Created a rooted tree from T by selecting any leaf of T as the root. We prove the result for any rooted subtree T_x of T, by induction on the height of T_x. We created two wIUBVRs for T_x, one where the drawing resides within $R^{\lceil}(x)$ and one that resides within $R^{\lrcorner}(x)$ (see Fig. 3a for the definition of these shapes; they are meant to extend rightward to infinity). We only explain how to construct the wIUBVR in $R^{\lceil}(x)$; the other construction is symmetric. If x has no children then the bar of x alone gives the representation, so assume that x has children. Since T has maximum degree 4 and T is a rooted tree, every vertex has at most 3 children; we assume here that x has exactly three children a, b, c (we can pad the tree with some dummy-children if it has fewer). If the given bar for x is in layer i, then place unit bars for a and b in layer $i{+}1$, with $y_a{<}y_x{<}y_b$. Place a unit bar for c in layer i with $y_x{<}y_c$. By the inductive hypothesis we can obtain representations of T_a, T_b, and T_c in $R^{\lrcorner}(a)$, $R^{\lceil}(b)$, and $R^{\lceil}(c)$, respectively. Putting these representations together, we obtain a representation of T_x in $R^{\lceil}(x)$ as depicted in Fig. 3c. □

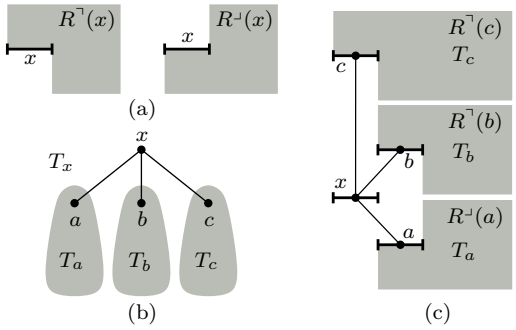
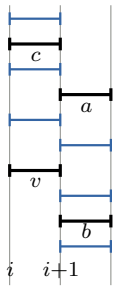


Figure 3: Illustration of the proof of Theorem 1: (a) Regions $R^{\lceil}(x)$ and $R^{\lrcorner}(x)$, (b) a rooted tree T_x, and (c) a wIUBVR of T_x in $R^{\lceil}(x)$.

Theorem 2 *There are trees of maximum degree 5 without a wIUBVR.*



Proof. Consider a tree T with a degree-5 vertex v that is adjacent to five degree-5 vertices. Assume for contradiction that T has a wIUBVR. Of the 6 ports at v (cf. Observation 1), exactly one is unused. Up to symmetry we may assume that the unused port is either the SW-port or the S-port. Let the NE-edge, SE-edge and N-edge be (v, a), (v, b) and (v, c). Observe that a cannot have a NW-edge, because any such neighbour would need to be below c and hence block the line-of-sight for (v, c). It also cannot have an S-edge because such a neighbour, regardless of its position, would either block the line-of-sight for (v, b) or for (v, a). Therefore $\deg(a) \leq 4$, a contradiction. □

The following will be useful later:

Corollary 3 *Let G be a graph with an IUBVR D and an edge (v, a) where $\deg(v){=}6$ and $\deg(a) \geq 5$ and a, v have no common neighbor. Then (v, a) is vertical in D.*

Proof. Assume for contradiction that (v, a) is diagonal, say it is the NE-edge of v. By $\deg(v) = 6$ it has a SE-

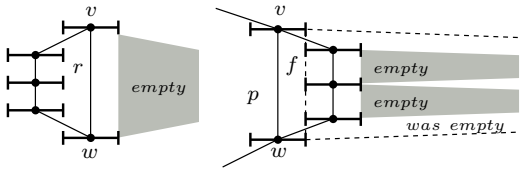


Figure 4: Creating a representation for max degree 3.

edge (v, b) and a N-edge (v, c). As in the previous proof then $\deg(a) \leq 4$, a contradiction. □

Now we turn to 2-connected outer-planar graphs, i.e., planar graphs whose outer-face is a simple cycle that contains all vertices. The *weak dual* G^* of such a graph is obtained by creating a vertex f^* for every inner face f and connecting two such vertices (f^*, g^*) with an edge in G^* if they share an edge (which is necessarily a chord of the outer-face cycle). It is well-known that the weak dual of a 2-connected outer-planar graph is a tree.

Theorem 4 *Every 2-connected outerplanar graph of maximum degree 4 has a wIUBVR.*

Proof. For brevity, we prove the case when the maximum degree is 3; the proof for maximum degree 4 is significantly more involved and is given in the appendix. Root the weak dual tree G^* at a face r^* that is a leaf. We now add the faces of G following their pre-order in the dual tree. We maintain the invariant that any chord e is drawn vertically and, as long as only one of the faces incident to e has been drawn, nothing is drawn to the right of e.

We start with the root r^* and let (v, w) be the unique chord of r (recall that r^* is a leaf of G^*). Draw r as a rotated trapezoid, with (v, w) on the long side in the right layer. Clearly the invariant holds. Now consider some face f and assume that the parent p^* of f^* has already been drawn, with the common chord (v, w) of p and f drawn vertically and without anything to its right. Draw f as a rotated trapezoid, with (v, w) as a unique edge on its left and all other vertices in the layer to the right of (v, w). The created SE-edge of v cannot be a chord because v already has three incident edges and therefore no other inner face can be incident here. Likewise the NE-edge of w cannot be a chord. So the invariant holds, and repeating for all faces gives a wIUBVR. □

4 Recognition in the weak model

In this section, we show that testing whether a plane graph admits a wIUBVR is NP-hard, by reducing from the NP-hard problem [6] of testing whether a given plane graph has a leveled-planar drawing.

In this reduction we will represent edges by a rigid structure. Consider the *rigid block* B depicted in Fig. 5a. One can easily show that B has a unique wIUBVR. Namely, since $\deg(a) = \deg(b) = 6$, by Corollary 3, (a, b) must be drawn vertically, say a above b. All other edge-orientations are then determined by the planar embedding since all ports at a and b are used.

We now combine M rigid blocks $B_1, \ldots, B_M$ to form an *M-tube*, see Fig. 5b. Since each rigid block has a unique wIUBVR, so does the M-tube. Note that an M-tube spans exactly $2M+1$ layers.

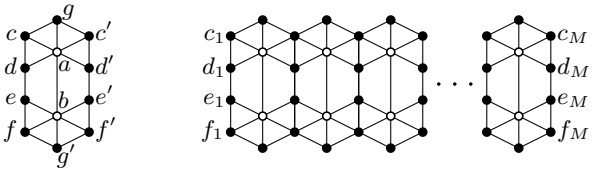
For representing the vertices, we make use of a different structure. Consider the *node block* L depicted in Fig. 6a. A node block can be connected to an M-tube as depicted in Fig. 6b. The wIUBVR of a node block is not unique; in particular it can be "bent" at the white squares.

Now fix a plane graph $G=(V_G, E_G)$ for which we wish to find a leveled-planar drawing. Create a plane graph H for which we wish to find a wIUBVR as follows:

Each edge e in G gives rise to an M-tube M_e in H, where M is chosen sufficiently large ($M=24n$ should do). Each vertex v of G is replaced by a *node gadget* N_v that consists of a cycle of d (where $d=\deg(v)$) node blocks $L_1, \ldots, L_d$ that are connected by identifying each vertex b_i with the vertex a_{i+1}; see Fig. 6c. Enumerating the edges around v as $e_1, \ldots, e_d$, we attach L_i to one end of M-tube M_{e_i} as depicted in Fig. 6b.

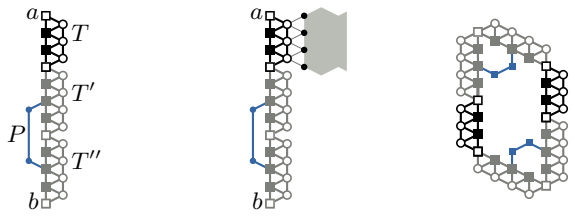
It is quite easy to see that if G has a leveled-planar drawing Γ, then H has an wIUBVR D, essentially by mapping level h of Γ to the layers from $h(2M+8) - 1$ to $h(2M+8) + 5$ in D, arguing that the node gadget N_v can be placed in those layers, and placing the M-tube of edge (v, w) in the $2M+1$ layers that are between the layers of its endpoints; see Fig. 7 for an example. For the other direction, we argue that there is in fact no other way than to lay out the node gadgets in these layers, so we can obtain a level-assignment that gives a leveled-planar drawing of G. The appendix has details. We conclude:

Theorem 5 *It is NP-hard to test whether a given plane graph H has a wIUBVR.*



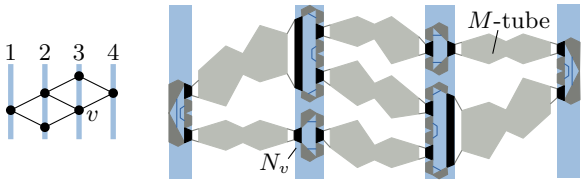
(a) A rigid block (b) An M-tube

Figure 5: The rigid structures used to represent edges.



(a) (b) (c)

Figure 6: (a) A node block (b) connected to an M-tube and (c) the node gadget of a degree-2 vertex.



(a) G (b) H

Figure 7: An example of a drawing of H obtained from a drawing of G. The node gadgets lie completely inside the shaded rectangles and the M-tubes lie in between.

5 Strong model

Now we turn to the strong model, where the existence of a line-of-sight implies that the corresponding edge *must* exist in G. As a simple warm-up result, we have:

Theorem 6 *A tree T has an sIUBVR if and only if it is a subdivision of a caterpillar of maximum degree 3.*

Proof. If T has an sIUBVR, then it also has a strong unit bar-visibility representation. As shown by Dean and Veytsel [3], it then has maximum degree 3 and is a subdivided caterpillar, i.e., it contains a path S (the spine) such that $T - S$ consists of paths (the subdivided legs). Vice versa, for any subdivided caterpillar of maximum degree 3 it is easy to create an sIUBVR; Fig. 1 illustrates the construction. □

The rest of this section is devoted to showing that more generally, we can test for any plane graph G whether it has an sIUBVR. We may assume that G is connected, else test each component separately. We assume for now that G is 2-connected.

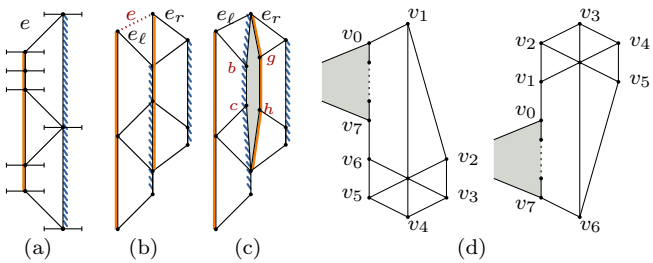
It will be convenient to direct outer-face edges so that the outer-face is to their left. Observe that in any IUBVR, the topmost diagonal edge that spans a slab is on the outer-face, with the outer-face above it; with the above direction therefore its orientation is NE or SE.

Let's start by outlining the idea. We create an auxiliary directed graph H that has a super-source s, a super-sink t, and a vertex $v(e, \alpha)$ for each *configuration* (e, α), where e is an edge on the outer-face of G and $\alpha \in \{\text{SE}, \text{NE}\}$. Vertex $v(e, \alpha)$ expresses the possibility of an sIUBVR where e has orientation α and e is the topmost edge in the slab that it spans. Crucially, fixing (e, α) determines the entire drawing within this slab. We can also define conditions under which two configurations, (e_ℓ, α_ℓ) and (e_r, α_r), could occur in consecutive slabs; if they are met, add an arc $v(e_\ell, \alpha_\ell) \to v(e_r, \alpha_r)$ to H. Likewise we can add arcs $s \to v(e, \alpha)$ or $v(e, \alpha) \to t$ if (e, α) could occur in the leftmost/rightmost slab. Testing whether an sIUBVR exists then amounts to finding a directed path from s to t in H.

To explain the details, we need a few observations.

Lemma 7 *In any sIUBVR D, any internal face f spans exactly one slab of D.*

Proof. In the strong model, the vertices within one layer i form an induced path connecting two vertices on the outer-face. Hence any face is either to the left of i or to the right of i, and so it cannot span two slabs. □



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

Figure 8: (a) The sub-drawing $D(e, \text{SE})$. $L(e, \text{SE})$ is orange (bold), $R(e, \text{SE})$ is blue (dashed). (b and c) Two examples of slabs that are not compatible: (b) The combined drawing creates an unwanted edge e. (c) $R(e_\ell, \text{NE}) \cup L(e_r, \text{SE})$ is not an induced path, leading to parts of G (gray) that are not represented. (d) Example that has non-unique sIUBVRs.

In fact, as illustrated in Fig. 8a, inner faces in an sIUBVR have a special form. We say that an inner face f *forms a trapezoid* (in some sIUBVR) if the edges of f can be enumerated (in clockwise or counterclockwise order) as $e_1, e_2, \ldots, e_d$ such that e_1 is upward diagonal, e_2 is vertical, e_3 is downward diagonal, and $e_4, \ldots, e_d$, if they exist, are vertical (Note that we allow the trapezoid to degenerate to a triangle).

Lemma 8 *Fix an arbitrary sIUBVR D. Then any internal face f forms a trapezoid.*

Proof. Assume f spans slab i and let e_b, e_t be the bottommost/topmost edges in slab i that belong to f. One can easily argue that e_b and e_t must have different orientations, else the face would be split by another diagonal edge that spans slab i and is between e_b and e_t. Up

to symmetry e_t is upward-diagonal and e_b is downward-diagonal. The right ends of e_t and e_b are connected to each other by a path that runs along level $i+1$. If this path contains any vertices other than the right ends of e_t and e_b, these additional vertices would necessarily be below the top end of e_t and above the bottom end of e_b. At least one of these vertices would be adjacent to a vertex on the left side of f (the bottom end of e_t or the top end of e_b or some vertex in-between). This again would split face f, a contradiction. So there are no vertices in this path, meaning that the right ends of e_b and e_t are connected by a single vertical edge as desired. □

As a consequence, fixing the topmost edge of a face and its orientation fixes the orientation for all edges of that face. We can propagate this to all faces of a slab, which gives the crucial insight for our algorithm.

Lemma 9 *Let D, D' be two sIUBVRs of a graph G. Assume that some outer-face edge e has the same orientation $\alpha \in \{\text{NE,SE}\}$ in D and D', and is the topmost edge in its slab in both D and D'. Then the slab of e in D and the slab of e in D' contain exactly the same faces and edges, in exactly the same order from top to bottom.*

Proof. Set $e_1{=}e$ and let f_1 be the unique inner face adjacent to e. Since e_1 is topmost and has orientation α, we know exactly the trapezoidal shape that f_1 must take, and therefore, the unique other edge e_2 that is diagonal and on f_1. Further, e_2 has the opposite orientation of e_1. Now repeat with e_2. Generally, once e_i is fixed, let f_i be the face incident to e_i that is not f_{i-1}. If f_i is the outer-face then stop. Else the orientation of e_i determines the trapezoidal shape of f_i and hence the unique other diagonal edge e_{i+1} on f_i and its orientation. Repeating the process determines all edges and faces that intersect the slab. □

We use $D(e,\alpha)$ to denote the subgraph formed by the inner faces $f_1, f_2, \dots$ in the above proof, and equip the edges of $D(e,\alpha)$ with the orientation as they are determined in the process. From the proof of Lemma 9, it follows that we can determine $D(e,\alpha)$ from the planar embedding of G alone, without needing to know an sIUBVR. If there exists some sIUBVR with e in orientation α as topmost edge of a slab, then $D(e,\alpha)$ expresses the part of it within that slab. Furthermore, in this case the edges $C(e,\alpha) := \{e_1, e_2, \dots\}$ (see proof of Lemma 9) span the slab and $D(e,\alpha){-}C(e,\alpha)$ is the union of two induced paths (along the sides of the slab). Let $L(e,\alpha)$ be the path that contains the left end of e and let $R(e,\alpha)$ be the other one. If anything goes wrong while determining $L(e,\alpha)$ and $R(e,\alpha)$ (e.g. if some edge obtains two contradictory directions for $D(e,\alpha)$, or if $D(e,\alpha){-}C(e,\alpha)$ is not the union of two paths) then we discard the node $v(e,\alpha)$ since it cannot lead to an sIUBVR.

Now we add an arc $v(e_\ell,\alpha_\ell){\to}v(e_r,\alpha_r)$ if (e_ℓ,α_ℓ) is *compatible* with (e_r,α_r). The latter means that $D(e_\ell,\alpha_\ell)$ and $D(e_r,\alpha_r)$ could occur on two consecutive slabs of an sIUBVR of G. It is not hard to test this in linear time: The two partial representations fix all the (downward) directions and orientations of all the involved edges. If this results in contradicting direction for edges, then no such sIUBVR of G can exist. Otherwise we can uniquely determine the relative position of bars for all involved vertices and simply test that these bars created no unwanted lines-of-sights. Finally the vertices in $R(e_\ell,\alpha_\ell) \cup L(e_r,\alpha_r)$ must induce a path in G, else putting the two partial representations would skip some inner faces or represent some vertices twice. See Fig. 8b and the appendix for details.

To finalize H, we add an arc $s \to v(e,\alpha)$ if $D(e,\alpha)$ could be the leftmost slab; this can be read directly from the planar embedding since then $L(e,\alpha)$ must consist of outer-face edges. Similarly we add an arc $v(e,\alpha) \to t$ if $D(e,\alpha)$ could be rightmost slab. This finishes the construction of H.

One can now easily show that G has an sIUBVR D if and only if H has a directed path s to t. Namely, given D we can find the vertices $v(e,\alpha)$ for which e is the topmost edge in some slab and has orientation α and argue that these form a path in H. Vice versa, if there is such a path, then each node $v(e,\alpha)$ on it defines a partial drawing $D(e,\alpha)$, and we can glue these partial drawings together since the arcs ensure compatibility. One can then argue that the result exactly represents G. Details are in the appendix.

Clearly, our approach gives a polynomial-time algorithm to test whether a 2-connected graph has an sIUBVR. We argue in the appendix how with a bit more care we can achieve a run-time of $O(n^2)$ for 2-connected graphs. We also discuss how to handle cutvertices in the appendix by splitting the graph into its 2-connected components. Overall, we achieve:

Theorem 10 *Let G be a plane graph with n vertices. Then we can test in $O(n^2)$ time whether G has an sIUBVR.*

Uniqueness? Our testing algorithm relies strongly on the fact that once the topmost edge and its orientation are determined, the representation within one slab is unique (up to moving bars up or down). Once one such slab is fixed, often the adjacent slab is fixed as well. In light of this, it may come as a surprise that an sIUBVR is not always unique. Indeed, we can construct an example where for one slab we have fixed the topmost edge and its orientation, and still at an adjacent slab we can have choices as to which edges and faces cross the slab. See Fig. 8d.

6 Conclusion and open problems

In this paper, we studied IUBVRs. We showed that recognizing whether a graph has a weak IUBVR is NP-hard, but in contrast testing whether it has a strong one is polynomial. We also showed that trees and 2-connected outer-planar graphs with maximum degree 4 have a weak IUBVR. We leave some open problems:

- In macramé, it is possible to knot more than two strings together, but typically no more than a small constant. What graphs are possible if vertex bars have the form $[i_v - k, i_v] \times y$ for, say, $k \leq 4$?
- For some graphs, the existence of an sIUBVR depends on the embedding, e.g. see Fig. 9. Can we test whether a planar graph (without fixed embedding) has an sIUBVR?

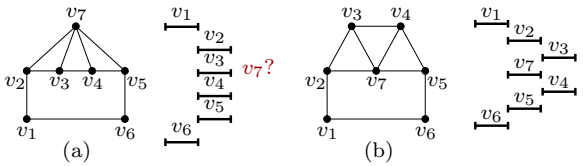


Figure 9: (a) Embedding for G that has no sIUBVR (b) A different embedding of the same G and its sIUBVR

References

[1] T. Andreae. Some results on visibility graphs. *Discrete Applied Mathematics*, 40(1):5–17, 1992.

[2] P. Bose, A. Dean, J. Hutchinson, and T. Shermer. On rectangle visibility graphs. In *International Symposium on Graph Drawing*, pages 25–44. Springer, 1996.

[3] A. M. Dean and N. Veytsel. Unit bar-visibility graphs. *Congressus Numerantium*, 160:161–176, 2003.

[4] P. Duchet, Y. Hamidoune, M. Las Vergnas, and H. Meyniel. Representing a planar graph by vertical lines joining different levels. *Discrete Mathematics*, 46(3):319–321, 1983.

[5] V. Dujmovic, M. Fellows, M. Kitching, G. Liotta, C. McCartin, N. Nishimura, P. Ragde, F. Rosamond, S. Whitesides, and D. Wood. On the parameterized complexity of layered graph drawing. *Algorithmica*, 52:267–292, 2008.

[6] L. S. Heath and A. L. Rosenberg. Laying out graphs using queues. *SIAM J. Comput.*, 21(5):927–958, 1992.

[7] L. A. Melnikov. Problem at the sixth Hungarian Colloquium on Combinatorics, 1981.

[8] R. H. J. M. Otten and J. van Wijk. Graph representations in interactive layout design. In *IEEE Internat. Symp. on Circuits and Systems, 1978*, pages 914–918, 1978.

[9] P. Rosenstiehl and R. E. Tarjan. Rectilinear planar layouts and bipolar orientations of planar graphs. *Discrete & Computational Geometry*, 1(4):343–353, 1986.

[10] M. Suderman. Pathwidth and layered drawings of trees. *International Journal of Computational Geometry & Applications*, 14(3):203–225, 2004.

[11] R. Tamassia and I. G. Tollis. A unified approach to visibility representations of planar graphs. *Discrete & Computational Geometry*, 1(4):321–341, 1986.

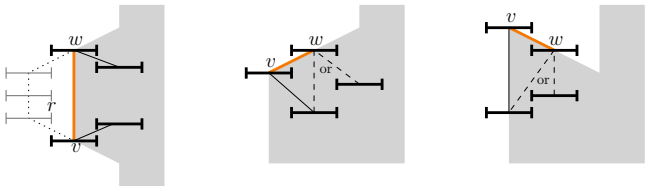
[12] C. Thomassen. Plane representations of graphs. *Progress in graph theory*, pages 43–69, 1984.

[13] L. W. Wiglesworth. *A study of unit bar-visibility graphs*. PhD thesis, University of Louisville, 2008.

[14] S. K. Wismath. Characterizing bar line-of-sight graphs. In *Symposium on Computational Geometry*, pages 147–152. ACM, 1985.

Appendix A wIUBVR for outerplanar graphs

We now explain how to create a wIUBVR of a 2-connected outerplanar graph with maximum degree 4. As before, we root the weak dual G^* (which is a tree) at a leaf r^*. For any face $f^* \neq r^*$ corresponding to face f, the *parent-face* is the face p corresponding to the parent p^* of f^*. Let (v, w) be the edge that f shares with its parent-face; it will be convenient to direct (v, w) so that f is to its right. The *subgraph* $G_{v,w}$ *attached at* (v, w) is the graph formed by the faces in the subtree rooted at f^*. We create up to 5 possible drawings of $G_{v,w}$, where the β-drawing, for $\beta \in \{\text{NW}, \text{N}, \text{NE}, \text{SE}, \text{SW}\}$, satisfies the following (see also Fig. 10):

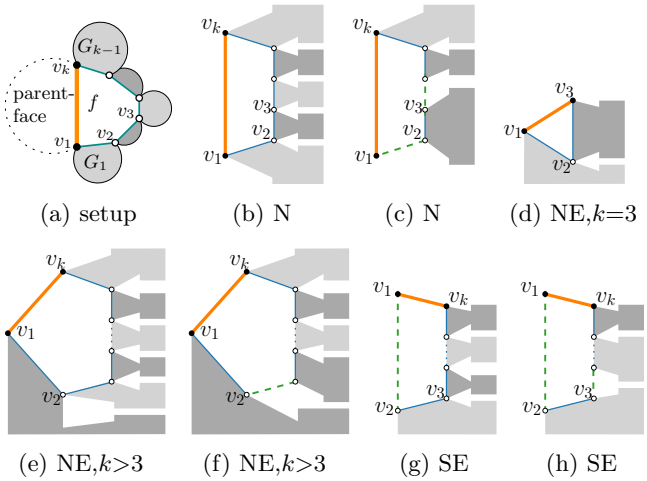


(a) N (b) NE (c) SE

Figure 10: Drawing types. The bold orange edge is the edge shared with the parent-face, and the entire drawing must reside within the gray region (extended infinitely rightward). (a) also illustrates how to add root r at the end.

- (v, w) is the β-edge at v.
- One of v, w is in layer 1, i.e., the leftmost layer.
- If $\beta = \text{N}$, then v has a NE-edge, w has a SE-edge, and layer 1 contains no other vertices.
- If $\beta = \text{NE}$, then v has a SE-edge and w has a S-edge or a SE-edge (or both). Layers 1 and 2 are empty above v, w.
- If $\beta = \text{SE}$, then w has a S-edge or a SW-edge or both, and it has no NE-edge. Layers 1 and 2 are empty above v, w. (This in particular implies that SE-drawings can exist only if $\deg(v) = 2$; we will ensure that this holds.)
- $\beta = \text{NW}$ is symmetric with $\beta = \text{NE}$ and $\beta = \text{SW}$ is symmetric with $\beta = \text{SE}$; flip the corresponding drawings in Fig. 10 upside-down.

We create such drawings by going bottom-up in tree G^*. So let $f^* \neq r^*$ be a node of G^*, corresponding to face f. Enumerate the vertices of f as $v_1, \ldots, v_k$ in counter-clockwise order such that $v_1 \to v_k$ is the edge that f shares with its parent-face. We want to draw the subgraph G_{v_1,v_k} attached at (v_1, v_k). For $s = 1, \ldots, k-1$, denote by G_s the subgraph $G_{v_s,v_{s+1}}$ attached at (v_s, v_{s+1}) (it is empty if (v_s, v_{s+1}) is on the outer-face). Since v_1 has degree at most 4 and it is adjacent to $v_k \notin G_1$ and has one further neighbour in the parent-face of f, we can conclude that v_1 has degree at most 2 in G_1.



(a) setup (b) N (c) N (d) NE,k=3

(e) NE,k>3 (f) NE,k>3 (g) SE (h) SE

Figure 11: Drawing face f and merging subgraphs. We only show the corresponding layered drawing. Dashed green edges are outer-face edges. We show some of the cases where G_s is empty and therefore more ports are available for G_{s-1} and G_{s+1}.

We explain how to create a β-drawing of G_{v_1,v_k} only for $\beta = \text{N}, \text{NE}, \text{SE}$, the cases $\beta = \text{NW}, \text{SW}$ are symmetric (flip the drawings for NE and SE upside-down); see Fig. 11.

Case 1: $\beta = \text{N}$. We draw f as a trapezoid with (v_1, v_k) as long vertical edge in layer 1. Directing the other edges as $v_1 \to v_2 \to \cdots \to v_k$, their orientations (in order) are $\text{NE}, \text{N}, \ldots, \text{N}, \text{NW}$. With this we have the required NE-edge for v_1 and SE-edge for v_k.

For each $1 \leq s \leq k-1$, recursively find the β'-drawing of G_s where β' is the orientation that we just assigned to $v_s \to v_{s+1}$. The goal is to merge these drawings; for this we need to argue that no port at a vertex v_s is used repeatedly (it could be used by f or by G_{s-1} or by G_s).

One easily verifies that there is no conflict between f and G_s due to the restrictions on the drawing type for G_s; for example; if G_s is a N-drawing, then its left-most layer contains only (v_s, v_{s+1}) and so it uses no port at v_s that was used by f. But it is less obvious that G_{s-1} and G_s, presuming they are both non-empty, could not both use a port of v_s. Recall that G_{s-1} uses a NE-drawing or a N-drawing, so v_s has a S-edge or a SE-edge in G_{s-1} that is not (v_{s-1}, v_s). Likewise G_s uses a NW-drawing or a N-drawing, and v_s has a N-edge or a NE-edge in G_s that is not (v_s, v_{s+1}). This gives four distinct edges at v_s, so by $\deg(v_s) \leq 4$ there are no others. So all edges of v_s in G_{s-1} go southward while all its edges in G_s go northward and there is no conflict among the ports.

With this, we can merge the drawings of the attached subgraphs into the drawing of the face. Because no ports at any v_s can be used by two subgraphs, this does not lead to overlap as long as we compact the drawing of G_s vertically so that it occupies only minimal space below v_s and above v_{s+1}. Also, if G_1 is non-empty then it uses a NE-drawing and so v_1 has a SE-edge other than (v_1, v_2). Since v_1 has at no other neighbours in G_1 as argued earlier, it has no S-edge and layer 1 is empty below v_1. Similarly layer 1 is empty above v_2. So we have created the desired N-drawing of G_{v_1,v_k}.

Case 2: $\beta = \text{NE}$ and $k = 3$. We draw f as a triangle with $v_1 \to v_3$ as NE-edge of v_1; thus $v_1 \to v_2$ is a SE-edge and $v_2 \to v_3$ is a N-edge, giving the edges required for a NE-drawing. As before, for each attached subgraph find the drawing that respects these orientations (this is feasible for G_1 since v_1 has at most two neighbours in G_1). If both G_1 and G_2 are non-empty, then v_2 has a NE-edge in G_2 and a S-edge or SW-edge in G_1, and so there is no port-conflict at v_2. Therefore, we can merge the drawings of the sub-graphs. One easily verifies that layer 1 and 2 remain empty above v_1 and v_k, so we obtain a NE-drawing.

Case 3: $\beta = \text{NE}$ and $k > 3$. We draw f as a pentagon with $v_1 \to v_3$ as NE-edge of v_1; $v_1 \to \cdots \to v_k$ receive orientations (in order) $\text{SE}, \text{NE}, \text{N}, \ldots, \text{N}, \text{NW}$. As before, for each attached subgraph G_s find the drawing that respects these orientations. The argument that there is no conflict among ports is the same as for Case 1, except at vertex v_2. Here (presuming both subgraphs G_1 and G_2 are non-empty) v_2 receives a S-edge or SW-edge in G_1 and a SE-edge in G_2, and by maximum degree 4 it has no other edges and there are no port-conflicts. Therefore we can merge the drawings of the sub-graphs. One easily verifies that layer 1 and 2 remain empty above v_1 and v_k, so we obtain a NE-drawing.

Case 4: $\beta = \text{SE}$. We know that this happens only if $\deg(v_1) = 2$. We draw f as a trapezoid with $v_1 \to v_k$ as SE-edge, so the edges $v_1 \to \cdots \to v_k$ receive orientation $\text{S}, \text{NE}, \text{N}, \ldots, \text{N}$. Since $\deg(v_1) = 2$ there is no subgraph attached at (v_1, v_2). For $2 \le s < k-1$, find the drawing of G_s that respects the assigned orientations. Verify as for Case 1 that this cannot lead to conflict among the ports. Therefore we can merge the drawings of the subgraphs. One easily verifies that layer 1 and 2 remain empty above v_1 and v_k.

We must argue that v_k has no NE-edge. We know that v_k has at most three neighbours in G_{v_1,v_k} (because it has one more in the parent-face of f). If it has three neighbours, then the neighbour x other than v_1 and v_k must be in subgraph G_{k-1}. But G_{k-1} uses a N-drawing or a NE-drawing; either way it has an edge different from (v_{k-1}, v_k) that is a S-edge or SE-edge of v_k. So (v_k, x) has southerly orientation, as do (v_k, v_1) and (v_k, v_{k-1}). Hence v_k has no NE-edge and we obtain a SE-drawing.

With this, we can draw any subgraph that corresponds to a strict subtree of G^*. To finish off, as before let r^* be the root of G^* and let (v, w) be the unique chord of r (recall that r^* is a leaf). Find a N-drawing of the graph $G_{v,w}$ attached at (v, w), which places (v, w) as vertical edge in the leftmost layer. We can now add r as a trapezoid with (v, w) as long edge on the right and all other vertices in one layer further left. This gives the desired wIUBVR of G.

Appendix B Details of the NP-hardness

We start by describing the components used later to form a vertex gadget. We define a *k-zigzag* to be the graph that consists of a $(2k-1)$-cycle $v_1, \ldots, v_k, u_{k-1}, \ldots, u_1$ with chords (v_i, u_j) for $i = 2, \ldots, k-1$ and $j \in \{i-1, i\}$; see Fig. 12a. We call vertices $v_1, \ldots, v_k$ *squared*, vertices $u_1, \ldots, u_{k-1}$ *circular*, and the vertices v_1 and v_k *end vertices*. Since every interior face of a k-zigzag T is a triangle, choosing the orientation of a single edge of T fixes the orientation of all of its edges. Note that the edges on the path $v_1, \ldots, v_k$ are all drawn with the same orientation β; we say that T *is drawn with orientation* β or that it is a *β-k-zigzag* or just *β-zigzag*.

We construct a *node block* L as follows; see Fig. 12b. Let T be a 3-zigzag, and let T' and T'' be two 4-zigzags. Identify vertex v_4 of T with vertex v'_1 of T' and vertex v'_5 of T' with vertex v''_1 of T''. Finally, add a path $P{=}(v'_3, x, y, v''_3)$, (called a *fixating path*). Let a be vertex v_1 of T and let b be vertex v''_5 of T''. A node block $L_i(v)$ consists of a 3-zigzag T_i, two 4-zigzags T'_i and T''_i and a fixating path P_i.

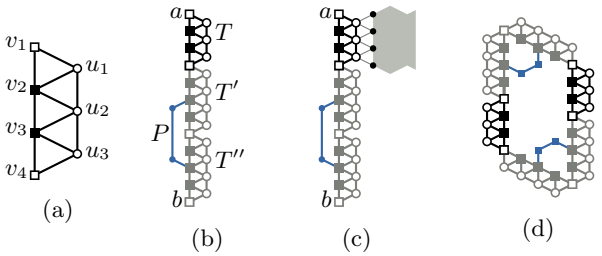


Figure 12: (a) A 3-zigzag. (b) A node block (c) connected to a tube and (d) the node gadget of a degree-2 vertex.

Let $n = |V_G|$ and set $M = 24n$. For any vertex $v \in V_G$ with $\deg(v){=}d$, the *node gadget* N_v of v consists of a cycle of d node blocks $L_1(v), \ldots, L_d(v)$ in H obtained by identifying vertex b_i of $L_i(v)$ with vertex a_{i+1} of $L_{i+1}(v)$ for $i{=}1, \ldots, d$ (where $L_{d+1}{=}L_1$). See Fig. 12d. Let $u_1, \ldots, u_d$ be the neighbours of v in G

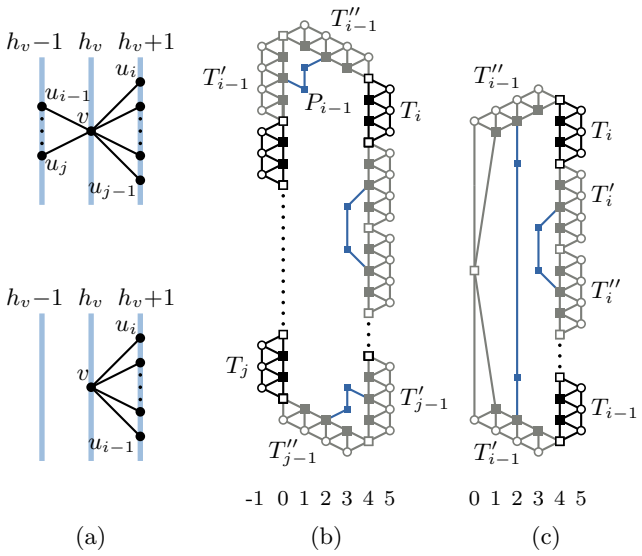


Figure 13: (a) A node v in G; (b)–(c) the node gadget of v in H when v has neighbours on (b) both sides and (c) only to the right. In this and the following figures, we omit the (v) identifier of the zigzags and we label the layers without the addition of $h_v \cdot (2M+8)$ to reduce clutter.

in clockwise order as defined by the embedding. Assign edge (v, u_i) in G to the 3-zigzag $T_i(v)$ in H, $1 \le i \le d$.

Now every edge (v, u) of E_G is assigned to exactly two 3-trapezoids in H, say $T_i(v)$ and $T_j(u)$. Attach an M-tube M_e to $T_i(v)$ and $T_j(u)$ as depicted in Fig. 12c. This completes the construction of H.

From G to H. We now show that we can construct a wIUBVR D of H from a leveled-planar drawing Γ of G. Enumerate the levels of Γ from left to right as $0, 1, \ldots, m$. Let v be a node of G with $\deg(v)=d$ that is drawn in level h_v. We draw N_v in the layers $h_v \cdot (2M+8)-1$ to $h_v \cdot (2M+8)+5$ as follows. Let (v, u_i) be the edge assigned to $T_i(v)$, $1 \le i \le d$.

First, assume that v has at least one neighbour in both level h_v+1 and level h_v-1. Let u_i be the top-most neighbour of v in level h_v+1 and let u_j be the bottom-most neighbour of v in level h_v-1; see Fig. 13a (top). We draw the trapezoids $T_i(v), T'_i(v) \ldots, T_{j-1}(v), T'_{j-1}(v)$ with S-orientation such that all their squared vertices lie in layer $h_v \cdot (2M+8)+4$ and all their circular vertices lie in layer $h_v \cdot (2M+8)+5$; see Fig. 13b. The interior vertices of the fixating paths $P_i, \ldots, P_{j-2}$ are placed one layer left of their endpoints, that is, in the layer $h_v \cdot (2M+8)+3$. Symmetrically, we place the trapezoids $T_j(v), T'_j(v) \ldots, T_{i-1}(v), T'_{i-1}(v)$ with N-orientation such that all their squared vertices lie in layer $h_v \cdot (2M+8)$ and all their circular vertices lie in layer $h_v \cdot (2M+8)-1$. The interior vertices of $P_j, \ldots, P_{i-2}$ are symmetrically placed in layer $h_v \cdot (2M+8)+1$. The trapezoid $T''_{i-1}(v)$ is drawn with SE-orientation in the layers $h_v \cdot (2M+8)$ to $h_v \cdot (2M+8)+4$. By this construction, the endpoints of P_{i-1} lie in the layers $h_v \cdot (2M+8)$ and $h_v \cdot (2M+8)+2$, so we can place its interior vertices in layer $h_v \cdot (2M+8)$. Symmetrically, the trapezoid $T''_{j-1}(v)$ is drawn with NW-orientation in the layers $h_v \cdot (2M+8)$ to $h_v \cdot (2M+8)+4$ and the interior vertices of P_{j-1} are placed in layer $h_v \cdot (2M+8)+3$.

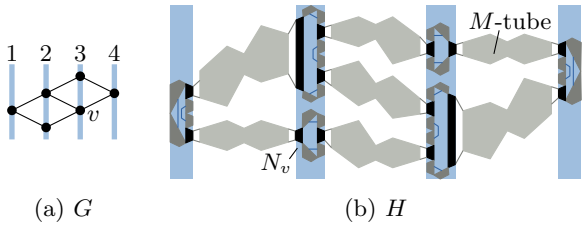


Figure 14: An example of a drawing of H obtained from a drawing of G. The node gadgets lie completely inside the shaded rectangles and the M-tubes lie in between.

Second, assume that all neighbours of v lie in level h_v+1 as shown in Fig. 13a (bottom). Again let u_i be the top-most neighbour of v. We place the trapezoids $T_i(v), T'_i(v) \ldots, T_{i-1}(v)$ with S-orientation such that all their squared vertices lie in layer $h_v \cdot (2M+8)+4$ all their circular vertices lie in layer $h_v \cdot (2M+8)+5$; see Fig. 13c. As in the previous case, the interior vertices of $P_i, \ldots, P_{i-2}$ are placed in layer $h_v \cdot (2M+8)+3$. The trapezoid $T'_{i-1}(v)$ is placed with NW-orientation and $T''_{i-1}(v)$ is placed with NE-orientation in the layers $h_v \cdot (2M+8)+4$ to $h_v \cdot (2M+8)$ This way, the endpoints of P_{i-1} both lie in layer $h_v \cdot (2M+8)+2$, so we can draw the path vertically in this layer.

Finally, if all neighbours of v lie in level h_v-1, then we place the vertices symmetrical to the previous case by choosing u_i as the bottom-most neighbour of v such that the circular vertices of the trapezoids $T_i(v), T'_i(v) \ldots, T_{i-1}(v)$ lie in layer $h_v \cdot (2M+8)-1$.

By this construction, if any edge between a node v in level h_v and a node u in level h_v+1 is assigned to v_i and u_j in H, then the circular vertices of $T_i(v)$ are drawn in layer $h_v \cdot (2M+8)+5$ and the circular vertices of $T_j(u)$ are drawn in layer $(h_v+1) \cdot (2M+8)-1$, so there are exactly $2M+1$ layers between them that we use to place the M-tube that is connected to $T_i(v)$ and $T_j(u)$. This completes the construction; see Fig. 14 for an example.

From H to G. Next, we show how to construct a leveled-planar drawing Γ of G from a wIUBVR D of H. We enumerate the layers of D from left to right as $0, 1, \ldots, m$. For any vertex $u \in V_H$, let $\ell(u)$ be the layer of u in D.

Let $v \in V_G$ with $\deg(v)=d$ and let $e=(v, u) \in E_G$ be assigned to the trapezoids $T_i(v)$ and $T_j(u)$. Since $T_i(v)$

and $T_j(v)$ are connected to an M-tube M_e, both have to be drawn with S-orientation or with N-orientation. Furthermore, $T_i(v)$ is a S-zigzag if and only if $T_j(u)$ is a N-zigzag. Assume w.l.o.g. that $T_i(v)$ is a S-zigzag. We aim to show that N_u lies completely to the right of N_v, that is, $\ell(u^*) > \ell(v^*)$ for every $u^* \in N_u$ and $v^* \in N_v$. The node gadget N_v consists of d node blocks, so it contains $24d \leq 24n$ vertices. Analogously, N_u consists of at most $24n$ vertices. Hence, both N_v and N_u lie in at most $24n$ consecutive layers. Let $v' \in T_i(v)$ and $u' \in T_j(u)$. Since $T_i(v)$ and $T_j(u)$ are connected to the same M-tube M_e, we have $\ell(u') \geq \ell(v')+2M+2$. Furthermore, for any vertex $v^* \in N_v$ we have $\ell(v^*) \leq \ell(v') + 24n$ and for any vertex $u^* \in N_u$ we have $\ell(u^*) \geq \ell(u') - 24n$. Hence,

$$\begin{aligned}\ell(u^*) &\geq \ell(u') - 24n \geq \ell(v') + 2M + 2 - 24n \\ &\geq \ell(v^*) + 2M + 2 - 48n \\ &= \ell(v^*) + 48n + 2 - 48n > \ell(v^*).\end{aligned}$$

Let $v \in G$ and let $M_i(v)$ be the M-tube attached to the 3-zigzag $T_i(v)$ in N_v, $i = 1, \ldots, d = \deg(v)$. Since D is planar, the M-tubes $M_1(v), \ldots, M_d(v)$ either all leave N_v to the right, all leave N_v to the left, or there is some $1 \leq i, j \leq d$ such that $M_i(v), \ldots, M_{j-1}(v)$ leave N_v to the right and $M_j(v), \ldots, M_{i-1}(v)$ leave N_v to the left. It follows that N_v has one of the following properties.

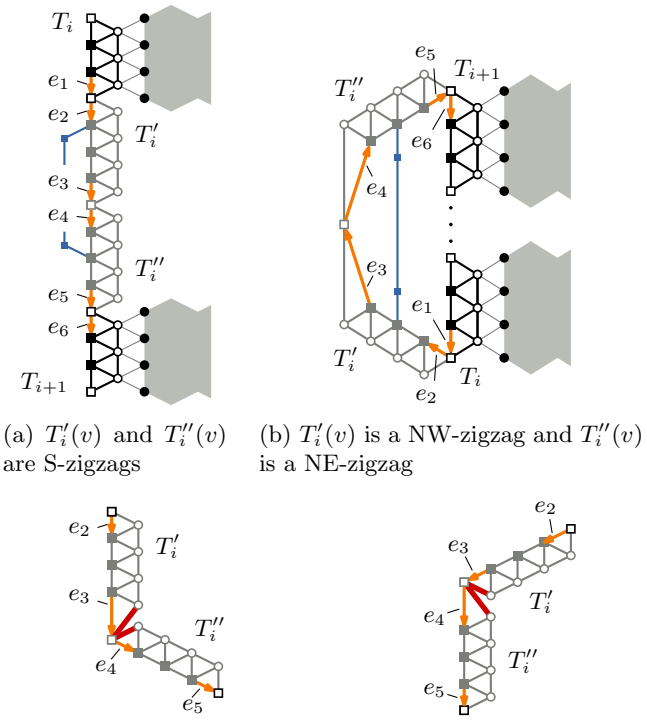
(P1) Every $T_i(v)$, $1 \leq i \leq d$, is a S-zigzag. There is some $1 \leq j \leq d$ such that $M_j(v), \ldots, M_d(v), M_1(v), \ldots, M_{j-1}(v)$ are ordered from top to bottom in this order.

(P2) Every $T_i(v)$, $1 \leq i \leq d$, is a N-zigzag. There is some $1 \leq j \leq d$ such that $M_j(v), \ldots, M_d(v), M_1(v), \ldots, M_{j-1}(v)$ are ordered from bottom to top in this order.

(P3) There is some i, $1 \leq i \leq d$, and some $j \neq i$, $1 \leq j \leq d$, such that $T_i(v), T_{i+1}(v), \ldots, T_{j-1}(v)$ are S-zigzags, $M_i(v), M_{i+1}(v), \ldots, M_{j-1}(v)$ are ordered from top to bottom in this order, $T_j(v), T_{j+1}(v), \ldots, T_{i-1}(v)$ are N-zigzags, and $M_j(v), M_{j+1}(v), \ldots, M_{i-1}(v)$ are ordered from bottom to top in this order.

Lemma 11 *Let N_v be a node gadget. The circular vertices of all S-3-zigzags of N_v lie in the same layer $\overrightarrow{\ell_v}$ and the circular vertices of all N-3-zigzags of N_v lie in the same layer $\overleftarrow{\ell_v}$ with $\Delta\ell = \overrightarrow{\ell_v} - \overleftarrow{\ell_v} = 6$. Furthermore, the drawing of N_v spans at least 6 layers between $\overleftarrow{\ell_v}$ and $\overrightarrow{\ell_v}$.*

Proof. Consider first the case that D is drawn with property (P1), that is, Every $T_i(v)$, $1 \leq i \leq d$, is a S-zigzag and there is some $1 \leq j \leq d$ such that $M_j(v), \ldots, M_d(v), M_1(v), \ldots, M_{j-1}(v)$ are ordered from top to bottom in this order. Let $1 \leq i \leq n, i \neq j-1$. Then, T_i and T_{i+1} are S-zigzags and $M_{i+1}(v)$ lies below $M_i(v)$. We will argue that $T'_i(v)$ and $T''_i(v)$ are also S-zigzags. Intuitively, between two zigzags in counterclockwise order, there can never be a "rightwards bend" because of the order of the edges around each squared vertex, as otherwise at least one port has to be used twice.



(a) $T'_i(v)$ and $T''_i(v)$ are S-zigzags

(b) $T'_i(v)$ is a NW-zigzag and $T''_i(v)$ is a NE-zigzag

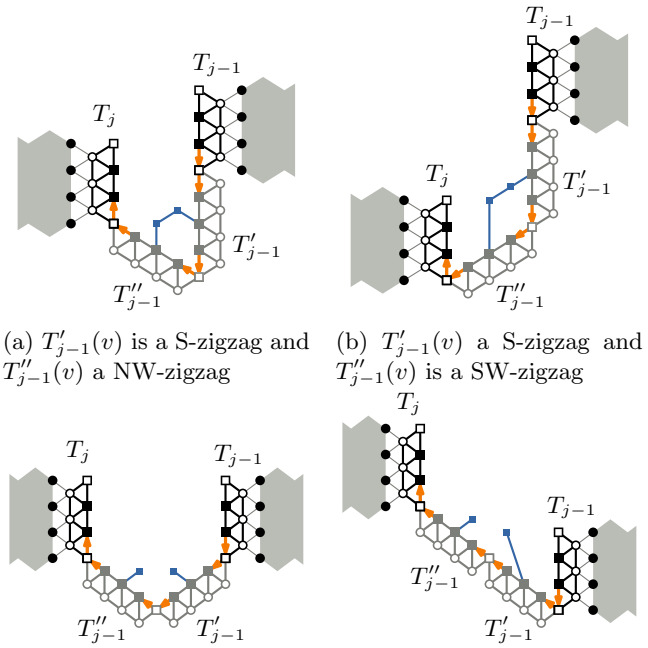
(c) $T'_i(v)$ is a S-zigzag and $T''_i(v)$ is a SE-zigzags

(d) $T'_i(v)$ is a SW-zigzag and $T''_i(v)$ is a S-zigzag

Figure 15: $T_i(v)$ and $T_{i+1}(v)$ are S-zigzags.

Consider the (directed) edges $e_1, \ldots, e_6$ in Fig. 15. By construction, e_1 and e_6 are S-edges. Since there are two edges between e_1 and e_2 around their common vertex in clockwise order, e_2 and thus e_3 has to be a S-, SW-, or NW-edge. Symmetrically, e_5 and thus e_4 has to be a S-, SE-, or NE-edge. However, since there are two edges between e_3 and e_4 around their common vertex in clockwise order, there are only two compatible configurations: either both are a S-edge (see Fig. 15a), or e_3 is a NW-edge and e_4 is a NE-edge (see Fig. 15b); otherwise, at least one port between e_3 and e_4 has to be used twice (see Figures 15c and 15d. However, the second case is a contradiction to the face that $M_i(v)$ lies completely above $M_{i+1}(v)$. Hence, $e_1, \ldots, e_6$ are S-edges and thus $T_i(v)$, $T'_i(v)$, $T''_i(v)$, and T_{i+1} are S-zigzags.

Now, let $i=j-1$. In this case, the argument is exactly the same; with the only difference that now $M_i(v)=M_{j-1}(v)$ is the bottom-most one M-tube, so



(a) $T'_{j-1}(v)$ is a S-zigzag and $T''_{j-1}(v)$ a NW-zigzag

(b) $T'_{j-1}(v)$ a S-zigzag and $T''_{j-1}(v)$ is a SW-zigzag

(c) $T'_{j-1}(v)$ is a SW-zigzag and $T''_{j-1}(v)$ is a NW-zigzag

(d) $T'_{j-1}(v)$ and $T''_{j-1}(v)$ are NW-zigzags

Figure 16: $T_{j-1}(v)$ is a S-zigzag and $T_j(v)$ is a N-zigzag.

$M_{i+1}(v)$=$M_j(v)$ lies above it. Hence, the case that e_3 and e_4 are S-edges is a contradiction to this property. Thus, $T'_i(v)$ is a NW-zigzag and $T''_i(v)$ is a NE-zigzag; see Fig. 15b. Hence, N_v spans exactly the 6 layers between $\overrightarrow{\ell_v}-5$ and $\overrightarrow{\ell_v}$.

This shows that $T_j(v), T'_j(v), T''_j(v), \ldots, T_{j-1}(v)$ are all S-zigzags. Hence, the circular vertices of the 3-zigzags $T_1(v), \ldots, T_d(v)$, which are all S-3-zigzags, all lie in the same layer $\overrightarrow{\ell_v}$.

The case that D is drawn with property (P2) is completely symmetric.

Consider now the case that D is drawn with property (P3), so there is some $1 \leq i, j \leq d$, $i \neq j$, such that $T_i(v), \ldots, T_{j-1}(v)$ are S-zigzags and $T_j(v), \ldots, T_{i-1}(v)$ are N-zigzags. With the same argument as above, we show that the circular vertices of $T_i(v), \ldots, T_{j-1}(v)$ all lie in the same layer $\overrightarrow{\ell_v}$ and that the circular vertices of $T_j(v), \ldots, T_{i-1}(v)$ all lie in the same layer $\overleftarrow{\ell_v}$. It remains to show that $\Delta\ell$=6.

Consider the S-zigzag $T_{j-1}(v)$, the N-zigzag $T_j(v)$, and the (directed) edges $e_1, \ldots, e_6$ in Fig. 16. Following the same arguments as above, $T'_{j-1}(v)$ has to be a S-, SW-, or NW-zigzag and $T''_{j-1}(v)$ has to be a N-, SW-, or NW-zigzag. However, the fixating path $P_{j-1}(v)$ forces either $T'_{j-1}(v)$ to be a S-zigzag or $T''_{j-1}(v)$ to be a N-zigzag; see Fig. 16a and Fig. 16b for the former case, the latter case is symmetric. In particular, assume that $T'_{j-1}(v)$ and $T''_{j-1}(v)$ are both drawn SW- or NW-zigzags; see Fig. 16c and Fig. 16d, the remaining cases are symmetric. Then, there are three layers between the endpoints of $P_{j-1}(v)$, but $P_{j-1}(v)$ only has two interior vertices. Hence, it is impossible to draw $P_{j-1}(v)$. Thus, $T'_{j-1}(v)$ and $T''_{j-1}(v)$ have to be drawn as in Fig. 16a and Fig. 16b (up to symmetry) and one can easily see that this implies $\Delta\ell$=6 and N_v spans exactly the 7 layers between $\overleftarrow{\ell_v}$ and $\overrightarrow{\ell_v}$. □

We now show that every node gadget N_v lies completely inside the layers $h_v\cdot(2M+8)-1$ to $h_v\cdot(2M+8)+5$ for some $0 \leq h_v \leq n$.

First, we show that all vertices in layer 0 belong to a node gadget. Assume to the contrary that there is some vertex u in layer 0 that belongs to an M-tube. By construction, every M-tube is connected to a vertex of a node gadget on both sides; hence, there has to be at least one vertex that completely lies to the left of the M-tube. This is a contradiction to layer 0 being the leftmost layer that contains a vertex.

Let $s \in V_G$ be a node such that some vertex of N_s lies on layer 0 in D. For any node $v \in V_G$, let d_v be the length of the shortest path between s and v in G. We now analyze the layers that contain the vertices of node gadgets.

Lemma 12 *For any node $v \in V_G$, $\overleftarrow{\ell_v}$=$h_v \cdot (2M+8)-1$ and $\overrightarrow{\ell_v}$=$h_v \cdot (2M+8)+5$ for some $0 \leq h_v \leq d_v$.*

Proof. We prove the lemma by induction over d_v.

If d_v=0, then v=s. Let h_s=0. By choice of s, the leftmost vertex of N_s lies in layer $0 > 0 \cdot (2M+8)-1$. Since there are no node gadgets that lie to the left of N_s, it is drawn as depicted in Fig. 13c. Hence, the N_s is drawn with property (P1) and by Lemma 11 all circular vertices of its S-3-zigzags lie in layer 5=$h_v\cdot(2M+8)+5$.

Now, assume that the lemma holds for all vertices $w \in V_G$ with $d_w \leq k \geq 0$. Let $v \in V_G$ with d_v=k+1 and let (u, v) be the last edge on the shortest path from s to v in G. Let u_i and v_j be the vertices the edge (u, v) is assigned to.

Assume first that $T_i(u)$ is a S-3-zigzag and $T_j(v)$ is a N-3-zigzag. By induction, u_i lies in a layer $\overrightarrow{\ell_u}$=$h_u \cdot (2M+8)+5$ for some $0 \leq h \leq d_u$. The vertices u_i and v_j are connected to the same M-tube. Since an M-tube spans exactly $2M+1$ layers, it follows that $\ell(v_j)$=$\overrightarrow{\ell_u}$ + $(2M+2)$=$h_v \cdot (2M+8)-1$ for h_v=h_u+1. Hence, by Lemma 11, $\overleftarrow{\ell_v}$=$h_v\cdot(2M+8)-1$ and $\overrightarrow{\ell_v}$=$h_v\cdot(2M+8)+5$.

If that $T_i(u)$ is a N-3-zigzag and $T_j(v)$ is a S-3-zigzag, then analogously $\ell(v_j)$=$\overleftarrow{\ell_u}$−$(2M+2)$=$h_v \cdot (2M+8)+5$ for h_v=h_u+1 and thus $\overleftarrow{\ell_v}$=$h_v \cdot (2M+8)-1$ and $\overrightarrow{\ell_v}$=$h_v \cdot (2M+8)+5$. □

We are now ready to create the drawing Γ of G. We draw every node $v \in V_G$ in level h_v in Γ. Let $v_1, \ldots, v_k$

be the vertices in level h, $0 \le h \le n$. By Lemma 11 and 12, each node gadget $N_{v_1}, \ldots, N_{v_k}$ contains at least one vertex in layer $h_v \cdot (2M+8)$. Since each of these node gadgets is connected to at least one M-tube, there is some order, say $v_1, \ldots, v_k$, such that the vertices in layer belong to $N_{v_1}, \ldots, N_{v_k}$ from bottom to top. We draw each node v_i, $1 \le i \le k$ at coordinate (h, i). Since D is planar, the obtained drawing Γ of G is also planar. Let $(v, u) \in E_G$ be assigned to vertices v_i of N_v and u_j of N_v. Assume w.l.o.g. that $T_i(v)$ is a S-zigzag and $T_j(u)$ is a N-zigzag. Since v_i and u_j are connected to a common M-tube, we have $\ell(v_i) = \overrightarrow{\ell_u} = h_u \cdot (2M+8) + 5$ and

$$\begin{aligned} h_u \cdot (2M+8) - 1 = \overleftarrow{\ell_u} &= \ell(u_j) \\ &= h_v \cdot (2M+8) + 5 + 2M + 2 \\ &= (h_v{+}1) \cdot (2M+8) - 1, \end{aligned}$$

so $h_u = h_v{+}1$. Hence, v is placed in level h_v and u is placed in level $h_v{+}1$, so Γ is a leveled-planar drawing.

This completes the proof of Theorem 5.

Appendix C Details of testing for an sIUBVR

C.1 Compatibility of configurations

We first explain in more detail how to test whether two configurations (e_ℓ, α_ℓ) and (e_r, α_r) are compatible, i.e., what properties must hold if some sIUBVR has $D(e_\ell, \alpha_\ell)$ and $D(e_r, \alpha_r)$ in adjacent slabs, say in slabs $i{-}1$ and i corresponding to layers $i{-}1, i$ and $i{+}1$.

In the following, we consider paths $R(e_\ell, \alpha)$ and $L(e_r, \alpha_r)$ to be directed downward, i.e., from the end in e_ℓ/e_r to the other end. We claim that the following conditions are necessary for compatibility:

- The vertices U of $R(e_\ell, \alpha_\ell) \cup L(e_r, \alpha_r)$ form an induced path P of G that can be directed such that it is consistent with the directions of $R(e_\ell, \alpha_\ell)$ and $L(e_r, \alpha_r)$. This holds because the union of the two paths is drawn on level i as a vertical path.
- In particular, the vertices I in $R(e_\ell, \alpha_\ell) \cap L(e_r, \alpha_r)$ must form a subpath of P and the edges among them must be directed the same in $R(e_\ell, \alpha_\ell)$ and $L(e_r, \alpha_r)$. Also, the vertices in $U - I = R(e_\ell, \alpha_\ell) \oplus L(e_r, \alpha_r)$ must occur at the beginning or end of P.
- If α_ℓ=SE, then $R(e_\ell, \alpha_\ell)$ is a prefix of P, i.e., no vertex of P comes before the first vertex of $R(e_\ell, \alpha_\ell)$. This holds because the left end v_ℓ of e_ℓ is above the right end w_ℓ, and w_ℓ is the topmost vertex of $R(e_\ell, \alpha_\ell)$. If any vertices of $L(e_r, \alpha_r)$ were above w_ℓ, then there would be an edge from them to v_ℓ (or higher up), and hence e_ℓ would not be on the outer-face.
- Similarly we have three more requirements at the bottom/top ends to avoid unwanted edges.
  - if α_r=NE, then no vertex in P comes before $L(e_r, \alpha_r)$.
  - Let $\overline{e_\ell}$ be the bottommost diagonal edge of $D(e_\ell, \alpha_\ell)$. If its direction (in $D(e_\ell, \alpha_\ell)$) is NE, then no vertex in P comes after $R(e_\ell, \alpha_\ell)$.
  - Let $\overline{e_r}$ be the bottommost diagonal edge of $D(e_r, \alpha_r)$. If its direction (in $D(e_r, \alpha_r)$) is SE, then no vertex in P comes after $R(e_r, \alpha_r)$.

It is not hard to see that these conditions are also sufficient. If they are satisfied, then draw P, in order, in level i from top to bottom. In level $i{-}1$ place the vertices of $L(e_\ell, \alpha_\ell)$ so that their position relative to the vertices in $R(e_\ell, \alpha_\ell)$ is the same as it was in $D(e_\ell, \alpha_\ell)$. Observe that this results in exactly the same set of edges as the set that spans slab $i{-}1$. Additional edges could only come from vertices above/below $R(e_\ell, \alpha_\ell)$ in P but such vertices either require e_ℓ to be directed NE or $\overline{e_\ell}$ to be directed SE, neither of which results in a line-of-sight. Similarly, we place the vertices of $R(e_r, \alpha_r)$ in level $i{+}1$ to obtain the desired representation of the two slabs.

For later use, we note one more property:

Observation 2 *Assume that (e_ℓ, α_ℓ) is compatible with (e_r, α_r). Then the edges in $R(e_\ell, \alpha_\ell) \oplus L(e_r, \alpha_r)$ are on the outer-face.*

Proof. Consider just the edges of $R(e_\ell, \alpha_\ell) - L(e_r, \alpha_r)$, the others are symmetric. Continuing in the notations introduced above, we saw that all these edges are drawn vertically in layer i, either above or below $D(e_r, \alpha_r)$. Say they are above. Then no diagonal edge that spans slab i is above them, thus they can reach the outer-face through slab i. □

C.2 Correctness of the construction

Recall that we combined all $D(e_i, \alpha_i)$ for some path $s \to v(e_1, \alpha_1) \to \cdots \to v(e_k, \alpha_k) \to t$. Let D be the sIUBVR that is induced by the resulting bars. For $i{=}2, \ldots, k$, set $I_i := R(e_{i-1}, \alpha_{i-1}) \cap L(e_i, \alpha_i)$. By Observation 2, I_i is a path that connects two outer-face vertices. We can therefore split the graph G into subgraphs by splitting at all paths $I_2, \ldots, I_k$. More precisely, using $I_1 = L(e_1, \alpha_1)$ and $I_{k+1} = R(e_k, \alpha_k)$, we set G_i to be the graph formed by all faces that can reach the inner face at e_i along a path of inner faces without crossing I_i or I_{i+1}.

It is now straightforward to show by induction that the first i slabs of D (i.e., what we obtain when putting together $D(e_1, \alpha_1) \cup \cdots \cup D(e_i, \alpha_i)$) is an sIUBVR of $G_1 \cup \cdots \cup G_i$. This is straightforward for $i{=}1$ since $D(e_1, \alpha_1)$ represents exactly G_1. When adding in $D(e_{i+1}, \alpha_{i+1})$, we add exactly the faces of G_{i+1} since $D(e_{i+1}, \alpha_{i+1})$ covers them, and we do not add extra

edges since the compatibility-condition ensures that vertices in $L(e_{i+1}, \alpha_{i+1})$ (if any) that are added in layer i+1 do not add edges to layer i.

C.3 Run-time

We now turn towards the time-complexity of testing whether a 2-connected plane graph G has a sIUBVR. There are $O(n)$ edges on the outer-face of G, hence H has $O(n)$ vertices. As we will argue below, it also has $O(n)$ arcs. Computing the directed path in H (if any) hence takes $O(n)$ time, and we can extract the sIUBVR from it in $O(n)$ time as well.

However, computing the arcs of H is non-trivial and takes cubic time if done in a straightforward way, and quadratic time if we are careful. As a first step, compute sets $L(e, \alpha)$ and $R(e, \alpha)$ for all configurations (e, α) where e is on the outer-face and $\alpha \in \{\text{NE}, \text{SE}\}$; this can be done in $O(n)$ time per configuration and hence overall quadratic time. While doing this we can easily check whether (e, α) is compatible with the left/right boundary and hence find all arcs incident to s and t.

The remaining arcs all connect $v(e, \alpha)$ to $v(e', \beta)$ for some outer-face edges e, e' and directions $\alpha, \beta \in \{\text{NE}, \text{SE}\}$. To find such an arc, we do four tests for each configuration (e, α):

- Walk clockwise along the outer-face starting at e until you encounter the first edge e' that does not belong to $R(e, \alpha)$. Test for both β=NE and β=SE whether (e, α) is compatible with (e', β).
- Walk counter-clockwise along the outer-face starting at e until you encounter the first edge e'' that does not belong to $L(e, \alpha)$. Test for both γ=NE and γ=SE whether (e'', γ) is compatible with (e, α).

To see that this suffices, observe that if (e, α) it compatible with (e', β), then the clockwise path Q from e to e' on the outer-face belongs to either $R(e, \alpha)$ or $L(e', \beta)$. (This holds by Observation 2: The edges in $R(e, \alpha) \oplus L(e', \beta)$ are on the outer-face, and because G is 2-connected, path Q cannot include any edges not in them.) If Q belongs to $R(e, \alpha)$, then our first test will use exactly this e' and hence detect the compatibility. If Q belongs to $L(e, \alpha)$, then at the time of performing the test for configuration (e', β), the second test will use e as e'' and hence detect compatibility. So this determines all arcs of H as needed, and there are $O(n)$ of them. Notice that one test of compatibility can be done in $O(n)$, and so the overall run-time is quadratic.

C.4 Dealing with cutvertices

So far, we assumed that G is a 2-connected plane graph. If G has a cutvertex, then we will argue that we can process each 2-connected component (*blocks*) separately. For this, we need to argue some restrictions on the structure near cutvertices. We assume in the following that G is not a path, else it trivially has an sIUBVR. We need two definitions. First, recall that the edges at a vertex can be classified as NE-edge etc. by their relative directions; we say that two edges incident to a vertex v *use consecutive ports* if their directions are consecutive in the cyclic order $\{\text{N}, \text{NE}, \text{SE}, \text{S}, \text{SW}, \text{NW}\}$. Second, define an *subdivided leg* of graph G to be a maximal induced path for which one end has degree 1 in G. The other end (which necessarily has degree at least 3 in G) is called the *attachment point* of the subdivided leg.

Lemma 13 *Let G be a plane graph that has an sIUBVR D. Let v be a cutvertex of G that is in layer i.*

1. *v is on the outer-face of G.*
2. *If w_1, w_2 are two vertices in the same layer but in different cut-components of v, then they are both in layer i with v between them.*
3. *If e_1, e_2 are two incident edges of v in different cut-components of v, then they do not use consecutive ports at v.*
4. *v has at most three cut-components.*
5. *If v has exactly three cut-components, then one of them is a subdivided leg.*
6. *For any block B containing v the IUBVR D_B of B induced by D is a strong IUBVR and contains v as topmost or bottommost vertex in layer i.*

Proof. 1. Recall that an inner face forms a trapezoid, hence is drawn convex in the associated layered drawing. But at least one face at v contains v repeatedly and so cannot be convex. So v must be adjacent to the outer-face.

2. The vertices within one layer induce a path. So if w_1, w_2 are in the same layer, they are connected by the path of the vertices that are between them on the layer. Any such path must contain v since w_1, w_2 are in different cut-components.

3. Inspection of all cases shows that if two such edges use consecutive ports, then their other endpoints are connected by an edge, contradicting that they are in different cut-components.

4. v has only 6 ports, and we must skip one port whenever we switch from one cut-component to the next in the order of edges around v.

5. The three cut-components must use 3 ports at v without using consecutive ones; up to symmetry these are the N-,SE-, and SW-port. Then the cut-component that uses the N-port must entirely lie within layer i to avoid having an edge to the other two components. So it forms an induced path and its topmost vertex has degree 1, hence it is a subdivided leg.

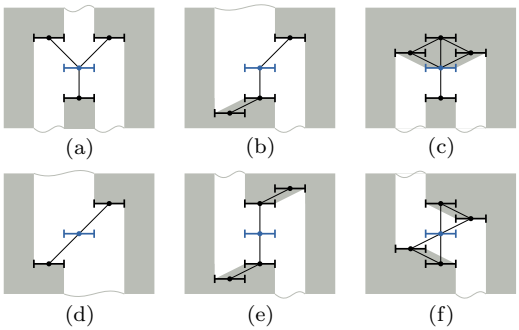


Figure 17: Possible cutvertex configurations

6. Let x, y be two vertices in B and assume that they have a line-of-sight in D_B, but not in D. The vertex z that blocked the line-of-sight in D must share a level with at least one of them, say z and x are in the same level. Since z is in a different 2-connected component, this must be level i and vertex v must be between z and x. But then vertex v would block the line-of-sight in D_B. So D_B is a strong IUBVR. If there were vertices of B both above and below v in layer i, then the N-edge and S-edge at v belong to B. All other edges at v hence would then use consecutive ports with an edge in B, contradicting that v is a cutvertex.

□

We assume in the following that G satisfies conditions (1),(4) and (5), i.e., the conditions that do not depend on the choice of the sIUBVR.

Lemma 14 *Let G be a plane graph that has an sIUBVR. Let G' be the graph obtained from G by removing all subdivided legs. Then the blocktree of G' is a simple path.*

Furthermore, the blocktree can be enumerated as $B_0 - w_1 - B_1 - \cdots - B_{k-1} - w_k - B_k$ for blocks $B_0, \ldots, B_k$ and cutvertices $w_1, \ldots, w_k$ such that $\ell(w_1) \leq \cdots \leq \ell(w_k)$ (where $\ell(w_i)$ denotes the layer of w_i), and all vertices of B_i lie within $[\ell(w_i), \ell(w_{i+1})]$ (where $\ell(w_0) := 1$ and $\ell(w_{k+1}) := \infty$).

Proof. Call a cutvertex of G *non-trivial* if it has at least two cut-components that contain cycles; these are the same as the cutvertices of G'. Notice that a non-trivial cutvertex has exactly two cut-components with cycles (which correspond to cut-components of G') by Lemma 13.

Fix a non-trivial cutvertex v and the two cut-components C_1, C_2 of v that have cycles. We claim that, up to renaming, all vertices in C_1 must be in the level of v and farther left while all vertices in C_2 must be in the level of v or farther right. For otherwise, since both cut-components have cycles, they would both use the same adjacent layer of v, leading to an edge from $C_1 - v$ to $C_2 - v$, a contradiction. We say that v *separates* its cut-components that have cycles.

Now consider a block B that is not a bridge and hence has cycles and occupies at least two layers. Assume two non-trivial cutvertices v_1, v_2 belong to B and are in the same layer. Since v_1 separates B from some other cut-component at v_1, it must be in the leftmost or rightmost layer of B. Say v_1 and v_2 are in the leftmost layer of B. Then the other cut-components C_1 and C_2 at v_1 and v_2 must be to the left of v_1 and v_2. We may choose C_1 and C_2 to contain cycles, so they must use layers strictly to the left of v_1 and v_2. But then there is an edge from $C_1 - v_1$ to $C_2 - v_2$, an impossibility. So for any non-bridge block B, no two non-trivial cutvertices can be in the same layer, and if there are two, they must be in the leftmost and rightmost layer of B. We say that B is *between* its non-trivial cutvertices. In particular, this implies that any non-bridge block has at most two non-trivial cutvertices. Putting things together, therefore every block and cutvertex of G' has at most two incident cutvertices/blocks, which means that the blocktree of G' is as desired.

The second claim follows almost immediately. Let ℓ be the leftmost and rightmost level that contain vertices of G'. If all of G' is drawn within layer ℓ, then the second claim holds trivially. So assume G' uses some layers further right, and let B be a block of G' that spans slab ℓ. Observe that there cannot be two such blocks B, B', because otherwise we could find a cycle that contains edges of both blocks by using the paths within layers ℓ and $\ell{+}1$ and the diagonal edges in the two blocks that span the slab. Observe further that B cannot have a non-trivial cutvertex in ℓ. For otherwise both of its incident cut-components in G' would have to use layer $\ell{+}1$, leading to an edge between them, a contradiction. Since B lies between its non-trivial cutvertices, therefore B has only one non-trivial cutvertex, in its rightmost layer. Thus B is a leaf of the blocktree of G'; call it B_0 and enumerate the rest of the blocktree correspondingly. In particular w_1 is the (unique) non-trivial cutvertex of B_0 and lies in its rightmost column. Block B_1 cannot use layers to the left of w_1 since w_1 separates B_0 and B_1. So B_1 is either drawn entirely within $\ell(w_1)$ (then it is necessarily a bridge) or it is drawn in $\ell(w_1)$ and further right, and its unique other non-trivial cutvertex lies in its rightmost level. Either way we obtain $\ell(w_1) \leq \ell(w_2)$ and B_1 lies only within these layers. Repeating the argument for the remaining blocks and cutvertices of G' gives the claim. □

We note here that this lemma mirrors nicely the characterization of T that have an sIUBVR: We know that this exists if and only if T is a subdivided caterpillar, which means that it, after removing subdivided legs, is a path (and hence its blocktree is also a path).

We need one last characterization of how subdivided legs can be drawn.

Observation 3 *Let G be a plane graph that has an sIUBVR D. Let G' be the graph obtained from G by removing subdivided legs and let D' be its induced sIUBVR. Let P be a subdivided leg whose attachment point v is not in the leftmost or rightmost level of D'. Then P is drawn vertically in the level of v, and either immediately above v or immediately below v.*

Proof. Let i be the level of v, and assume for contradiction that P contains a diagonal edge, say vertex w_P of P is in level $i{+}1$. By assumption some vertex w' of G' also resides in level $i{+}1$. This contradicts Lemma 13 since w_P and w' are in different cut-components of v. So P must reside within level i, and be immediately above or below v to create the edge between v and its neighbour in P. □

Now we can explain the full algorithm. First, detect all subdivided legs (this can be done in linear time by extending paths from vertices of degree 1) and remove them while marking their attachment point. So we have G' and compute its blocktree of G'. This must split into a path $B_0 - w_1 - B_1 - \cdots - w_k - B_k$, else G has no sIUBVR. Note that if G has an sIUBVR, then we may without loss of generality assume that none of the vertices of B_0 are farther right than the vertices of B_k, for otherwise we can rotate the representation by 180°. We hence can require the levels of the cutvertices to satisfy $\ell(w_1) \leq \ell(w_2) \leq \cdots \leq \ell(w_k)$.

For each block B_i of G' that is not a bridge, let H_i be the auxiliary graph computed as before, with super-source s_i and super-sink t_i. We modify H_i slightly to remove some arcs that clearly cannot lead to a solution. Namely, assume that H_i has an arc $a{=}v(e_\ell, \alpha_\ell) \to v(e_r, \alpha_r)$. If arc a is used in a solution, then the resulting sIUBVR contains $D(e_\ell, \alpha_\ell)$ and $D(e_r, \alpha_r)$ in consecutive slabs, and in particular, fixes exactly the vertices $U{=}R(e_\ell, \alpha_\ell) \cup L(e_r, \alpha_R)$ that are in the common layer of the slabs (say layer j). It also fixes the direction of incident edges of U. We remove arc a from H_i if this placement of U contradicts restrictions from Lemma 13 or Observation 3. In particular we remove a if

- U contains cutvertex w_i or w_{i+1}. (This would contradict that these two cutvertices are the leftmost or rightmost within their 2-connected component, while arc a implies that there are vertices both left and right of layer j.)
- U contains a cutvertex $w \neq w_i, w_{i+1}$ of G, and w is not the topmost or bottommost vertex of U. (Note that we are studying here cutvertices of G, not G', so such a cutvertex w can exist if it is the attachment point for some subdivided leg.)
- U contains a cutvertex $w \neq w_i, w_{i+1}$ of G, but the edges to U use ports such that we cannot attach the subdivided leg vertically at w without using consecutive ports and while respecting the planar embedding.

With this, any path from s_i to t_i in H_i leads to an sIUBVR of B_i to which we can add all subdivided legs whose attachment point is not in the leftmost or rightmost layer.

Now we want to create an auxiliary graph for $B_i \cup B_{i+1}$. Assume first that neither B_i nor B_{i+1} is a bridge. We then combine the two auxiliary graphs H_i and H_{i+1}, by eliminating vertices t_i and s_{i+1} and adding arcs between some of their neighbours. Consider any (e, α) and (e', β) such that we had arcs $(e, \alpha) \to t_i$ and $s_{i+1} \to (e', \beta)$ in H_i, i.e., we could have had these configurations in the rightmost/leftmost slab in representations of B_i/B_{i+1}. We can eliminate any such vertices if $w_{i+1} \notin R(e, \alpha)$ or $w_{i+1} \notin L(e', \beta)$, since we know that this is required in an sIUBVR of G'. If w_{i+1} occurs in both sets, then this determines a unique way of merging $D(e, \alpha)$ and $D(e', \beta)$, and with it, the direction of all edges incident to $R(e, \alpha) \cup L(e'\beta)$. We add an arc $a{=}v(e, \alpha) \to v(e', \beta)$ if this gives a feasible representations that allows adding subdivided legs. More precisely, we add arc a only if

- no port at w_i is used by edges in both partial drawings,
- no two consecutive ports at w_i are used by edges to B_i and B_{i+1},
- for any $w \in R(e, \alpha) \cup L(e', \beta)$ that is an attachment point of a subdivided leg, the incident edges in B_i and/or B_{i+1} use ports such that it is possible to add the subdivided leg vertically at w without using consecutive ports or violating the planar embedding.

(It may sound as if we could create $\Omega(n^2)$ arcs here, but similarly as in the 2-connected case we need not test all combinations of (e, α) and (e', β); we can read from the planar embedding and the outer-face path of G' a constant-size set of edges e' that need to be tested for each configuration (e, α).)

Thus if neither B_i nor B_{i+1} is a bridge, then we can combine their auxiliary graphs. If B_i is a bridge (w_i, w_{i+1}), then we create an auxiliary graph for $B_i \cup B_{i+1}$ similarly. Namely, let in this case H_i consist of vertices s_i and t_i and four more vertices $v(e, \mathrm{N}), v(e, \mathrm{NE}), v(e, \mathrm{SE})$ and $v(e, \mathrm{S})$, representing the possibility of drawing (w_i, w_{i+1}) while respecting $\ell(w_i) \leq \ell(w_{i+1})$. Each vertex determines the representation of B_i in its entirety and so we can combine this

graph with H_{i+1} as above, adding arcs only if this allows for merging of subdivided legs. Similarly we deal with the case where B_{i+1} is a bridge.

With this, any directed path from s_i to t_{i+1} in the combined graph leads to an sIUBVR of $B_i \cup B_{i+1}$ where we can add all subdivided legs whose attachment point is not in the leftmost and rightmost layer. Repeatedly merging, we obtain one graph H where any path from s_0 to t_k corresponds to an sIUBVR of G' where we can add all subdivided legs whose attachment point is not in the leftmost or rightmost layer.

As a final step, we must modify H to deal with subdivided legs whose attachment points are in the leftmost layer (the rightmost layer is dealt with similarly). This is slightly different from before since Observation 3 does not apply. In fact, a subdivided leg that attaches at (say) the topmost vertex v in the leftmost layer may well use the NW-port at v, allowing the NE-port of v to be used by G'. More precisely, let $a{=}s_0 \to v(e, \alpha)$ be an arc in H_0 (and hence H). If arc a is used for a solution, then this determines the layout of $L(e, \alpha)$ in the leftmost layer. Let v_t and v_b be the topmost and bottommost vertex of $L(e, \alpha)$. We must remove a from H if one of the following happens:

- $L(e, \alpha)$ contains a cutvertex w of G that is neither v_t nor v_b.
- $v_t \neq v_b$, both v_t and v_b are cutvertices of G, v_t uses the NE-port and v_b uses the SE-port for edges in B_0. (In this case, neither of the attached subdivided legs at v_t and v_b can be drawn vertically, so they both must go to the left, but this would create an edge between them which is not allowed.)

It should be clear from the construction that if G has an sIUBVR, then there exists a path from s_0 to t_k in the final constructed auxiliary graph. For the other direction, assume we have such a path. This implies a path from s_i to t_i in each subgraph H_i and so an sIUBVR D_i for each block B_i. Furthermore, we eliminated sufficiently many arcs such that $D_0 \cup \ldots D_k$ can be combined into one, and all subdivided legs can be attached without creating unwanted edges. So we obtain an sIUBVR of G as desired. Therefore testing for an sIUBVR equals finding a path from s_0 to t_k in a directed graph, which takes linear time. Building H can be done in $O(n^2)$, since for each of the $O(n)$ arcs the conditions for eliminating or adding it can be tested in $O(n)$. Theorem 10 follows.

Continuous Terrain Guarding with Two-Sided Guards

Wei-Yu Lai* Tien-Ruey Hsiang†

Abstract

We consider the continuous two-sided guarding on a 1.5-dimensional(1.5D) terrain T. To our knowledge, this is the first work on this problem. Specificially, we aim at selecting a minimum number of guards such that every point on the terrain can be seen by a guard to its left, and another guard to its right. A vertex v sees a point p on T if the line segment connecting v to p is on or above T. We demonstrate that the continuous 1.5D terrain guarding problem can be transformed to the discrete terrain guarding problem with a finite point set X and that if X is two-sided guarded, then T is also two-sided guarded. Through this transformation, we provide an optimal algorithm determining a guard set with minimum cardinality that completely two-sided guards the terrain.

1 Introduction

A 1.5 dimensional(1.5D) terrain T is an x-monotone polygonal chain in $\mathbb{R}^2$ specified by n vertices $V(T) = \{v_1, ..., v_i, ..., v_n\}$, where $v_i = (x_i, y_i)$. The vertices induce $n-1$ edges $E(T) = \{e_1, ..., e_i, ..., e_{n-1}\}$ with $e_i = \overline{v_i v_{i+1}}$.

A point p sees or guards q if the line segment $\overline{pq}$ lies above or on T, or more precisely, does not intersect the open region bounded from above by T and from the left and right by the downward vertical rays emanating from v_1 and v_n.

There are two types of terrain guarding problems: (1) continuous terrain guarding (CTG) problem, with objective of determining a subset of T with minimum cardinality that guards T, and (2) discrete terrain guarding problem, with the objective of determining a subset of U with minimum cardinality guarding X, given that the two point sets U and X are on T.

Many studies have referred to applications of 1.5D terrain guarding in real world [1, 2, 3]. The examples include guarding or covering a road with security cameras or lights and using line-of-sight transmission networks for radio broadcasting.

*Department of Computer Science and Information Engineering, National Taiwan University of Science Technology, D10115005@mail.ntust.edu.tw

†Department of Computer Science and Information Engineering, National Taiwan University of Science Technology, trhsiang@csie.ntust.edu.tw

1.1 Related Work

Ample research has focused on the 1.5D terrain guarding problem, which can be divided into the general terrain guarding problem and the orthogonal terrain guarding problem.

In a 1.5D terrain, King and Krohn [4] proved that the general terrain guarding problem is NP-hard through planar 3-SAT.

Initial studies on the 1.5D terrain guarding problem discussed the design of a constant-factor approximation algorithm. Ben-Moshe et al. [5] gave the first constant-factor approximation algorithm for the terrain guarding problem and left the complexity of the problem open. King [6] gave a simple 4-approximation, which was later determined to actually be a 5-approximation. Recently, Elbassioni et al. [7] gave a 4-approximation algorithm.

Finally, Gibson et al. [8] considered the discrete terrain guarding problem by finding the minimal cardinality from candidate points that can see a target point [8] and proved the presence of a planar graph that appropriately relating the local and global optima; thus, the discrete terrain guarding problem allows a polynomial time approximation scheme (PTAS) based on local search. Friedrichs et al. [9] revealed that for the continuous 1.5D terrain guarding problem, finite guard and witness sets (G and X, respectively) can be constructed such that an optimal guard cover $G'' \subseteq G$ that covers terrain T is present and when these guards monitor all points in X, the entire terrain is guarded. According to [8], the continuous 1.5D terrain guarding problem can apply PTAS by constructing a finite guard and witness set with the former PTAS.

Some studies have considered orthogonal terrain T. T is called an orthogonal terrain if each edge $e \in E(T)$ is either horizontal or vertical. An orthogonal terrain has four vertex types. If v_i is a vertex of orthogonal terrain and the angle $\angle v_{i-1} v_i v_{i+1} = \pi/2$, then v_i is a convex vertex, otherwise it is a reflex vertex. A convex vertex v_i is left(right) convex if $\overline{v_{i-1}v_i}(\overline{v_i v_{i+1}})$ is vertical. A reflex vertex v_i is left(right) reflex if $\overline{v_{i-1}v_i}(\overline{v_i v_{i+1}})$ is horizontal.

Katz and Roisman [10] gave a 2-approximation algorithm for the problem of guarding the vertices of an orthogonal terrain. The authors constructed a chordal graph demonstrating the relationship of visibility between vertices. On the basis of [11], [10] gave a 2-approximation algorithm and used the minimum clique

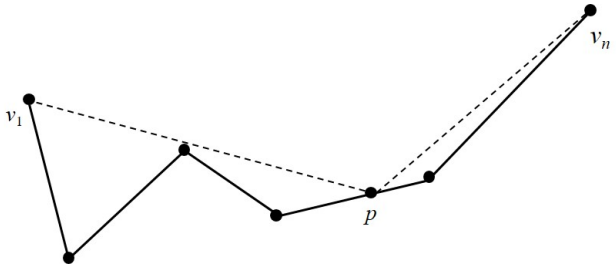


Figure 1: Point p is two-sided guarded by v_1 and v_n.

cover of a chordal graph to solve the right(left) convex vertex guarding problem.

Lyu and Üngör [12] gave a 2-approximation algorithm for the orthogonal terrain guarding problem that runs in $O(n \log m)$, where m is the output size. The authors also gave an optimal algorithm for the right(left) convex vertex guarding problem. On the basis of the vertex type of the orthogonal terrain, the objective of the subproblem is to determine a minimum cardinality subset of $V(T)$ guarding all right(left) convex vertices of $V(T)$; furthermore, the optimal algorithm uses stack operations to reduce time complexity.

The $O(n \log m)$ time 2-approximation algorithm has previously been considered the optimal algorithm for the orthogonal terrain guarding problem. However, some studies have used alternatives to the approximation algorithm.

Durocher et al. [13] gave a linear-time algorithm for guarding the vertices of an orthogonal terrain under a directed visibility model, where a directed visibility mode considers the different visibility for types of vertex. If u is a reflex vertex, then u sees a vertex v of T, if and only if every point in the interior of the line segment uv lies strictly above T. If u is a convex vertex, then u sees a vertex v of T, if and only if $\overline{uv}$ is a nonhorizontal line segment that lies on or above T. Khodakarami et al. [14] considered the guard with guard range. They presented a fixed-parameter algorithm that found the minimum guarding set in time $O(4^k \cdot k^2 \cdot n)$, where k is the terrain guard range.

1.2 Result and Problem Definition

In this paper, we define the CTG problem with two-sided guards and propose an optimal algorithm for the 1.5D CTG problem with two-sided guards. To the best of our knowledge, the 1.5D CTG problem with two-sided guards has never been examined.

Definition 1 (Two-Sided Guarding). A point p on a 1.5D terrain is two-sided guarded if there exist two distinct guards u, which is on or to the left of p, and v, which is on or to the right of p, such that p can be seen by both u and v. Furthermore, the guards u and v are called a left-guard and a right-guard of p.

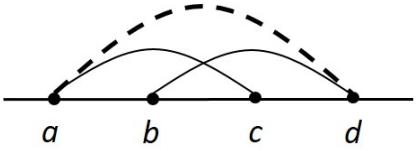


Figure 2: Schematic of Lemma 1.

Fig. 1 illustrates an example where vertex v_1 left-guards p and v_n right-guards p. In this paper, we define the following problem:

Definition 2(CTGTG: Continuous Terrain Guarding with Two-Sided Guards) Given a 1.5D terrain T, find a vertex guard set S of minimum cardinality such that every point of T can be two-sided guarded.

1.3 Paper Organization

Section 2 presents preliminaries, Section 3 demonstrates how to create a finite point set for the CTGTG model, Section 4 gives an algorithm for the CTGTG, along with its proof, and Section 5 presents our conclusions.

2 Preliminaries

Let p and q be two points on a 1.5D terrain, we write $p \prec q$ if p is on the left of q. We denote the visible region of p by $vis(p) = \{v \in V(T) | v \text{ sees } p\}$. For a $vis(p)$, let $L(p)$ be the leftmost vertex in $vis(p)$ and $R(p)$ be the rightmost vertex in $vis(p)$.

Given a CTGTG instance, let $OPT = \{o_1, o_2, ..., o_m\}$ be an optimal guard set, where $o_k \prec o_{k+1}$ for $k = 1, ..., m-1$. For a point p on the terrain, let $O_R(p)$ and $O_L(p)$ be the subsets of OPT such that p is right-guarded by every guard in $O_R(p)$ and left-guarded by every guard in $O_L(p)$. We also define N_i^R as the rightmost point on the terrain that is not right-guarded by $\{o_i, o_{i+1}, ..., o_m\}$ and N_i^L as the leftmost point on the terrain that is not left-guarded by $\{o_1, o_2, ..., o_i\}$.

An important visible property on 1.5D terrains is as follows:

Lemma 1 (Order Claim[5]) *Let a, b, c and d be four points on a terrain T such that $a \prec b \prec c \prec d$. If a sees c and b sees d, then a sees d.*

Fig. 2 is a schematic of Lemma 1. Because T is an x-monotone chain, we use a straight line to demonstrate the relation between x-coordinate of points and an arc to show the visible relation among points on T. In this paper, we use a straight line to simplify the explanations.

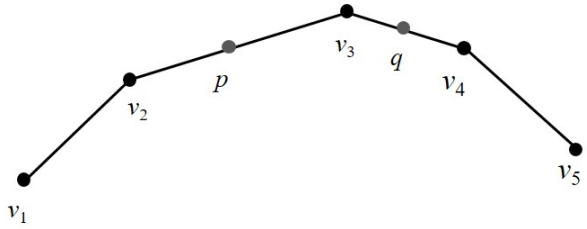


Figure 3: $V(T)$ is right-guarded and left-guarded by $\{v_1, v_2, v_4, v_5\}$, but not T.

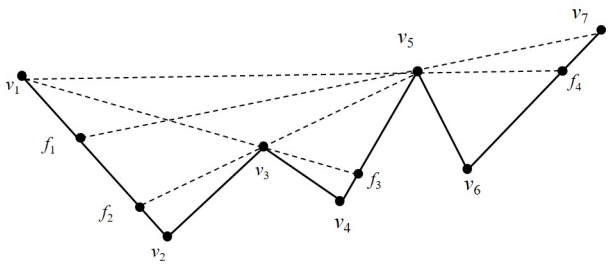


Figure 4: Points f_1, f_2, f_3 and f_4 are boundary points on T.

Observation 1 *Let e_j be an edge of the terrain and p is on e_j. If p is left-guarded by a guard v, then v also completely guards $\overline{pv_{j+1}}$.*

Observation 2 *Let e_j be an edge of the terrain and p is on e_j. If p is right-guarded by a guard v, then v also completely guards $\overline{v_jp}$.*

3 Discretization

Although $V(T)$ are right-guarded and left-guarded, T is not necessarily right-guarded and left-guarded. In Fig. 3, $V(T)$ is right-guarded and left-guarded by $\{v_1, v_2, v_4, v_5\}$ with minimal cardinality. The vertices v_1 and v_2 are left-guarded by v_1 and right-guarded by v_2. Vertices v_4 and v_5 are left-guarded by v_4 and right-guarded by v_5. Vertex v_3 is left-guarded and right-guarded by v_2 and v_4, respectively. Only v_3 can right-guard p and left-guard q where p is on e_2 and q is on e_3, but $v_3 \notin \{v_1, v_2, v_4, v_5\}$. In our example, we must create a point set X such that if X is right-guarded and left-guarded, then T is also.

Definition 3 (Boundary Point). If line $\overline{v_iv_j}$ and e_k have an intersection point $f \notin \{v_k, v_{k+1}\}$, and v_i and v_j can see f then f is the boundary point.

In Fig. 4, we provide an example with four boundary points: f_1, f_2, f_3 and f_4. Boundary point f_1 is from v_7, f_2 is from v_5; and boundary points f_3 and f_4 are from v_1. We say e_1 has two boundary points, f_1 and f_2; each of e_4 and e_6 has a boundary point.

Lemma 2 *For an edge e_i on terrain T, there exist at most two non-endpoints p and q such that e_i is complete two-sided guarded if p and q are two-sided guarded.*

Proof. According to the number of boundary points on e_i, we may consider the proof under the following cases: edge e_i does not have boundary point or has one, two, or k boundary points (where $k \geq 3$).

In the first case, we assume e_i does not have boundary point. Let point $p \notin \{v_i, v_{i+1}\}$ be on edge e_i. If p is right-guarded and left-guarded, then edge e_i is also right-guarded and left-guarded.

In the second case, we assume e_i has a boundary point f. We split the edge into two line segments $\overline{v_if}$ and $\overline{fv_{i+1}}$. Then, the first case can be applied to the line segments $\overline{v_if}$ and $\overline{fv_{i+1}}$. Therefore, we create two points $p \notin \{v_i, f\}$ on line segment $\overline{v_if}$ and $q \notin \{f, v_{i+1}\}$ on line segment $\overline{fv_{i+1}}$. If p and q are right-guarded and left-guarded, then e_i is also right-guarded and left-guarded.

In the third case, we assume e_i has two boundary points f_1 and f_2. We split the edge into three line segments $\overline{v_if_1}$, $\overline{f_1f_2}$ and $\overline{f_2v_{i+1}}$. The line segments $\overline{v_if_1}$ and $\overline{f_2v_{i+1}}$ can be reduced to the first case. Therefore, we create two points $p \notin \{v_i, f_1\}$ on line segment $\overline{v_if_1}$ and $q \notin \{f_2, v_{i+1}\}$ on line segment $\overline{f_2v_{i+1}}$. If p and q are left-guarded and right-guarded, then line segment $\overline{f_1f_2}$ is also left-guarded and right-guarded.

In the final case, we assume e_i has k boundary points $f_1, ..., f_k$. We split the edge into $k+1$ line segments $L = \{\overline{v_if_1}, \overline{f_1f_2}, ..., \overline{f_kv_{i+1}}\}$. The line segments $\overline{v_if_1}$ and $\overline{f_kv_{i+1}}$ can be reduced to the first case. Therefore, we create two points: $p \notin \{v_i, f_1\}$ on line segment $\overline{v_if_1}$ and $q \notin \{f_k, v_{i+1}\}$ on line segment $\overline{f_kv_{i+1}}$. If p and q are left-guarded and right-guarded, then each line segment $\overline{f_cf_{c+1}} \in L$ is also left-guarded and right-guarded. □

From the construction of Lemma 2, in order to completely two-sided guard a terrain, it is sufficient to first select a finite subset X of positions from the terrain to be two-sided guarded, such that $|X| \leq 2(n-1)$.

4 An Optimal Algorithm for CTGTG

In this section, we present an optimal algorithm for the CTGTG. The idea of the algorithm follows from Observation 3. In each step of our algorithm, we add a vertex v_i to our result S such that if $v_i \notin OPT$ then v_i can replace a vertex $v_j \in OPT$ and $|S| = |OPT|$.

Observation 3 *The optimal solution of the CTGTG includes v_1 and v_n.*

This is because in the CTGTG for right-guarded and left-guarded T, only v_1 can left-guard v_1 and only v_n can right-guard v_n.

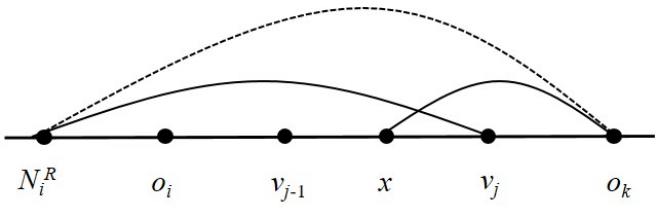


Figure 5: Position of $v_j \in R(N_i^R) \cup O_R(N_i^R)$.

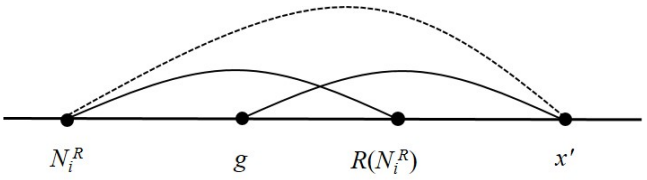


Figure 6: If $g \in O_R(N_i^R)$ left-guard x', then x_k and N_i^R see each other.

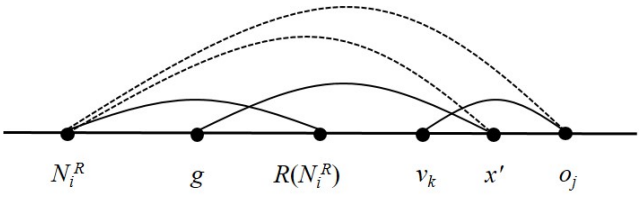


Figure 7: If x' and N_i^R see each other, then o_j right-guards N_i^R.

Lemma 3 *$R(N_i^R)$ and any guard in $O_R(N_i^R)$ do not lie on the right side of o_i.*

Proof. Let x be a point on the edge e_{j-1} such that $N_i^R \prec x$. We assume that $v_j \in R(N_i^R) \cup O_R(N_i^R)$ is on the right side of o_i. We know that x is right-guarded by o_k and o_k is on the right side of v_j. According to Lemma 1, if o_k right-guards x, then N_i^R is right-guarded by o_k. This contradicts the definition of N_i^R and o_k sees N_i^R. The schematic of Lemma 3 is given in Fig 5. □

Lemma 4 *If $R(N_i^R) \notin O_R(N_i^R)$, then any guard in $O_R(N_i^R)$ cannot left-guard $x' \in \{x \in X | R(N_i^R) \prec x\}$.*

Proof. Let g be a guard in $O_R(N_i^R) \setminus R(N_i^R)$ and let x' be a point such that $R(N_i^R) \prec x'$. Therefore, $N_i^R \prec g \prec R(N_i^R) \prec x'$. Consider x' on the edge $e_k = \overline{v_k v_{k+1}}$, there exists a guard o_j that right-guards x'. According to Lemma 1, if x' and g see each other, then x' and N_i^R also see each other. This is illustrated in Fig. 6. Because o_j right-guards x' and sees v_k, if x' sees N_i^R then o_j right-guard N_i^R too, as illustrated in Fig. 7. □

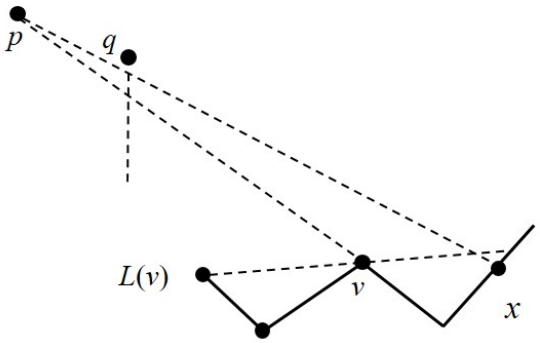


Figure 8: If $L(v)$ cannot see x and v sees x, then $v = L(x)$.

Lemma 5 *If $R(N_i^R) \notin O_R(N_i^R)$, $x \in X$ is right-guarded by o_j and $i \leq j \leq m$, then x cannot lie between $g \in O_R(N_i^R)$ and $R(N_i^R)$.*

Proof. We assume that the point x is on the $e_k = \overline{v_k R(N_i^R)}$ and x is right-guarded by o_j. We know that o_j right-guards v_k by Observation 2. According to Lemma 1, if x is right-guarded by o_j, then $N_R(o_i)$ is right-guarded by o_j. Therefore, we know that if $R(N_i^R) \notin O_R(N_i^R)$, then x cannot lie between $g \in O_R(N_i^R)$ and $R(N_R(o_i))$. □

By Lemma 3, Lemma 4 and Lemma 5, we have the following theorem.

Theorem 6 *If $R(N_i^R) \notin O_R(N_i^R)$, then $R(N_i^R)$ can replace any guard in $O_R(N_i^R)$.*

Proof. Based on Lemma 3, Lemma 4 and Lemma 5, if $R(N_i^R) \notin O_R(N_i^R)$, then $g \in O_R(N_i^R)$ cannot left-guard $x_k \in \{x_j \mid N_i^R \prec x_j\}$. Due to $g \prec R(N_i^R))$, we know $vis(R(N_i^R)) \supseteq vis(g)$ by Lemma 1. □

Similarly, $L(N_i^L)$ can replace any guard in $O_L(N_i^L)$.

Theorem 7 *If $O_L(N_i^L) \notin L(N_i^L)$, then $L(N_i^L)$ can replace any guard in $O_L(N_i^L)$.*

5 Complexity

Because our approach has two phases, we must first discuss the complexity of discretization. We obtain boundary points for a vertex v on $E(T)$ in $O(n)$ time by [15]. Therefore, we compute all boundary points for each vertex of $V(T)$ on each edge $e \in E(T)$ in $O(n^2)$ time. We obtain at most $2|V(T)|$ boundary points in $O(n^2)$ time.

Next, we demonstrate how to compute an optimal solution for the CTGTG. In step 1, we add v_1 and v_n to our solution. In step 2, we compute the $vis(v_1)$ and $vis(v_n)$. In step 3, we add $R(x)$ to our solution, where

$x \in X$ is the nonright-guarded rightmost point. If a point x exists that is not right-guarded, then repeat step 3 until X is right-guarded. In step 4, we add $L(x)$ to our solution, where x is the nonleft-guarded leftmost point. If there exists a point x that is not yet left-guarded, then repeat Step 4 until it is left-guarded. Thus, all points on the terrain are successfully guarded from both sides.

We show our algorithm for the CTGTG runs in $O(n)$ time using two steps. Before the algorithm begins, we can compute $R(x)$ and $L(x)$ for each point of X in $O(n)$ time. After this computation, we proceed to the algorithm in $O(n)$ time. Therefore, our proposed algorithm for the CTGTG runs in $O(n)$ time.

Algorithm 1: Compute all $L(x)$

```
Input: T: terrain, X: point set
Output: { L(x)|x ∈ X }
Q ← X ∪ V(T)
for q_i ∈ Q processed from left to right do
    q_j = q_{i−1}
    while L(q_i) = ∅ do
        if q_i sees L(q_j) then
            if L(q_j) is not v_1 then
                q_j = L(q_j)
            else
                L(q_i) = v_1
        else
            L(q_i) = q_j
for x ∈ X processed from left to right do
    Return L(x)
```

Lemma 8 *Let v and x be two points on a terrain T such that $v \prec x$. If $L(v)$ cannot see x and v sees x then $v = L(x)$.*

Proof. Let p, v and x be three points on T such that $p \prec L(v) \prec v \prec x$. We assume that $L(v)$ cannot see x and v can see x. If p sees x and cannot see v, then a vertex q exists and lie above line $\overline{vL(v)}$ and $p \prec q \prec L(v)$, as illustrated in Fig. 8. However, the assumption that $L(v) \neq q$ is contradictory. □

We propose Algorithm 1 to compute $L(x)$ for all points x in X according to Lemma 8 and Lemma 1. We prove that the running time of Algorithm 1 is $O(n)$.

Theorem 9 *Algorithm 1 runs in $O(n)$ time.*

Proof. We count the number of times q_i sees $L(q_j)$ in the algorithm. If q_i sees $L(q_j)$, then the algorithm does not visit the vetrices between q_i and $L(q_j)$. Therefore, the number of times q_i sees $L(q_j)$ is at most once for each point of Q. If q_i does not see $L(q_j)$, then q_i has found $L(q_i)$. Therefore, the number of times q_i does not see $L(q_j)$ is at most once for each point Q. □

After computing $L(x_i)$ and $R(x_i)$ for X, we reach the algorithm for the CTGTG in $O(n)$ time. We divided our algorithm into left-guarding and right-guarding, and therefore we provide the algorithm for left-guarding that can be implemented in $O(n)$ time.

Algorithm 2: Left-guarding

```
Input: T: terrain, X: point set
Output: S_L : left-guarding set
S_L is null;
Add v_1 to S_L;
V(T′)=V(T);
for x_i ∈ X processed from left to right do
    while g(x_i) is null do
        s is rightmost vertex in S_L ∩ V(T′);
        if x_i is guarded by s then
            g(x_i) is s;
            Remove the vertices between x_i and s
             from V(T′);
        else if s ≺ L(x_i) then
            g(x_i) be the vertex L(x_i) ;
            Add g(v_i) to S_L;
            Remove the vertics between x_i and
             L(x_i) from V(T′);
        else
            Remove s from V(T′);
return S_L
```

Theorem 10 *Algorithm 2 runs in $O(n)$ time.*

Proof. For each x_i, we examine whether x_i is guarded by $s \in S_L$ from x_i to $g(x_i)$. If $g(x_i) = v_j$, then Algorithm 2 will not visit the point and vertex between x_i and v_j. We count the number of times x_i is not seen by S_L. We can check s from x_i to $L(x_i)$. If s does not see x_i, then we will not check s for $\{x_k \mid x_i \prec x_k\}$. The number of times X is not seen by S_L is $|V(T)|$, and the number of times X is seen by S_L is $|X|$. Therefore, the algorithm visits the point and vertex at most $2|X| + |V(T)|$ times. After computing all $L(x_i)$, Algorithm 2 runs in $O(n)$ time. □

6 Conclusion

In this paper, we considered the CTGTG problem and devised an algorithm that can determine the minimal cardinality vertex that guards T under two-sided guarding. We showed that the CTGTG problem can be reduced to the discrete terrain guarding problem with at most $2|V(T)|$ points in $O(n^2)$ time and solved the problem using our devised algorithm in $O(n)$ time where n is the number of vertices on T.

References

[1] P. Ashok, F. V. Fomin, K. Sudeshna, S. Saurabh, M. Zehavi, Exact algorithms for terrain guarding, in: 33rd International Symposium on Computational Geometry, 2017.

[2] H. Eliş, A finite dominating set of cardinality $O(k)$ and a witness set of cardinality $O(n)$ for 1.5d terrain guarding problem, Annals of Operations Research (2017) 1–10.

[3] F. Khodakarami, F. Didehvar, A. Mohades, A fixed-parameter algorithm for guarding terrains, Theoretical Computer Science 595 (2015) 134–142.

[4] J. King, E. Krohn, Terrain guarding is np-hard, SIAM Journal on Computing 40 (5) (2011) 1316–1339.

[5] B. Ben-Moshe, M. J. Katz, J. S. Mitchell, A constant-factor approximation algorithm for optimal 1.5 d terrain guarding, SIAM Journal on Computing 36 (6) (2007) 1631–1647.

[6] J. King, A 4-approximation algorithm for guarding 1.5-dimensional terrains, in: Latin American Symposium on Theoretical Informatics, Springer, 2006, pp. 629–640.

[7] K. Elbassioni, E. Krohn, D. Matijević, J. Mestre, D. Ševerdija, Improved approximations for guarding 1.5-dimensional terrains, Algorithmica 60 (2) (2011) 451–463.

[8] M. Gibson, G. Kanade, E. Krohn, K. Varadarajan, An approximation scheme for terrain guarding, in: Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques, Springer, 2009, pp. 140–148.

[9] S. Friedrichs, M. Hemmer, C. Schmidt, A ptas for the continuous 1.5d terrain guarding problem, in: Canadian Conference on Computational Geometry, 2014.

[10] M. J. Katz, G. S. Roisman, On guarding the vertices of rectilinear domains, Computational Geometry 39 (3) (2008) 219–228.

[11] F. Gavril, Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph, SIAM Journal on Computing 1 (2) (1972) 180–187.

[12] Y. Lyu, A. Üngör, A fast 2-approximation algorithm for guarding orthogonal terrains, in: Canadian Conference on Computational Geometry, 2016.

[13] S. Durocher, P. C. Li, S. Mehrabi, Guarding orthogonal terrains., in: Canadian Conference on Computational Geometry, 2015.

[14] F. Khodakarami, F. Didehvar, A. Mohades, 1.5d terrain guarding problem parameterized by guard range, Theoretical Computer Science 661 (2017) 65–69.

[15] M. Löffler, M. Saumell, R. I. Silveira, A faster algorithm to compute the visibility map of a 1.5d terrain, in: Proc. 30th European Workshop on Computational Geometry, 2014.

Finding Minimum Witness Sets in Orthogonal Polygons

I. Aldana-Galván*† C. Alegría-Galicia*‡ J.L. Álvarez-Rebollar*§ N. Marín-Nevárez*¶
E. Solís-Villarreal*‖ J. Urrutia** C. Velarde††

Abstract

A *witness set* W in a polygon P is a subset of P such that any set $G \subset P$ that guards W is guaranteed to guard P. We study the problem of finding a minimum witness set for an orthogonal polygon under three models of orthogonal visibility: rectangular, staircase and k-periscope visibility.

Under the traditional line-segment visibility, it is known that not all simple polygons admit a finite witness set and, when a polygon admits a finite minimal witness set, the witnesses must lie on the boundary of the polygon [3].

In this paper, we prove that every orthogonal polygon with n vertices admits a finite witness set which has $O(n^2)$ witnesses under rectangular, staircase and k-periscope visibility. We also show that there exist orthogonal polygons which require $\Omega(n^2)$ witnesses under staircase visibility. Furthermore, we show that there exist orthogonal polygons for which the boundary is not a witness set for any of the three considered visibility models. Finally, we describe an $O(n^4)$ time algorithm to find a minimum witness set for a given orthogonal polygon under the rectangular and staircase visibility models.

1 Introduction

The Art Gallery Problem (AGP) is a classical problem in Computational Geometry that has been widely studied since it was proposed in 1973 by V. Klee [2]: given a polygon P, the Art Gallery Problem consists in finding a minimum set of points G such that each point in P is guarded by at least one element of G. G is called a *guard set*.

Several variants of AGP arise by imposing restrictions on the type of guards, the visibility model or the shape of the gallery. Many results on problems related to AGP can be found in the book by O'Rourke [9] and the surveys by Shermer [12] and Urrutia [13].

Under line-segment visibility, it is well known that finding minimum guard sets for simple and orthogonal polygons is an NP-hard problem, see Lee and Lin [7] and Schuchardt and Hecker [10], respectively.

In this paper we are interested in three kinds of orthogonal visibility: rectangular, staircase and k-periscope visibility. For rectangular visibility, a minimum guard set can be found in polynomial time if the polygon has no holes. An algorithm with complexity $O(n^{17})$ is given by Worman and Keil in [14]. However, if the polygon has holes, then AGP is NP-hard under rectangular visibility, as proven by Biedl and Mehrabi in [1]. For staircase visibility, Motwani et al. [8] prove that a minimum guard set can be found in $O(n^8)$ time in orthogonal polygons without holes. It remains as an open problem to determine if AGP is NP-hard in polygons with holes under staircase visibility. Finally, for k-periscope visibility, Gewali and Ntafos proved in [6] that AGP can be solved in $O(n^3)$ time for $k = 1$ in a restricted class of orthogonal polygons.

The *Witness Problem* is a variant of AGP that consists in finding a set W in a given polygon, such that if W is guarded by a set of guards G, then the polygon is guaranteed to be guarded by G. The set W is called a *witness set*. A motivation behind this research is that a witness set allows us to quickly verify if a polygon is guarded by a set of points.

The Witness Problem under line-segment visibility in simple polygons was studied by Chwa et al. in [3]. They proved that not all simple polygons admit a finite witness set. They also proved that, if a simple polygon P admits a finite minimal witness set, then all the witnesses must lie on the boundary of P. In addition, they gave an $O(n^2 \log n)$ time algorithm that computes a minimum witness set for P if it exists, or it reports the non-existence of a finite witness set otherwise.

*Research supported by PAEP from Universidad Nacional Autónoma de México

†Universidad Nacional Autónoma de México, ialdana@ciencias.unam.mx.

‡Posgrado en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México, calegria@uxmcc2.iimas.unam.mx

§Posgrado en Ciencias Matemáticas, Universidad Nacional Autónoma de México, chepomich1306@gmail.com

¶Posgrado en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México, mnjn16@uxmcc2.iimas.unam.mx

‖Posgrado en Ciencia e Ingeniería de la Computación, Universidad Nacional Autónoma de México, solis_e@uxmcc2.iimas.unam.mx

**Research supported by PAPIIT IN102117 from Universidad Nacional Autónoma de México, urrutia@matem.unam.mx

††Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, velarde@unam.mx

In this paper, we study the Witness Problem in orthogonal polygons under rectangular, staircase and k-periscope visibility. First, we show that there exist orthogonal polygons such that their boundary is not a witness set under the visibility models considered here. Then, we prove that we can always find a finite witness set with $O(n^2)$ elements in any orthogonal polygon under each of the considered visibility models. Finally, we describe an $O(n^4)$ time algorithm for finding a minimum witness set in orthogonal polygons under rectangular and staircase visibility models.

2 Preliminaries

Let P be a simple polygon. A vertex of P is *convex* if its interior angle is less than π and is *reflex* if its interior angle is greater than π.

Consider two points p and q in P. Under the *line-segment* visibility model, p and q are mutually visible if the line segment $\overline{pq}$ is contained in P. Under the *rectangular* visibility model, p and q are mutually visible if the smallest isothetic rectangle containing p and q, denoted by $R(p, q)$, is contained in P. Under the *staircase* visibility model, p and q are mutually visible if P contains a monotone isothetic polygonal path joining p and q. Finally, under the *k-periscope* visibility model, for $k \in \mathbb{N} \setminus \{0\}$, p and q are mutually visible if P contains an isothetic polygonal path joining p and q with at most k bends.

The following definitions are common to all the visibility models described above. The *kernel* of P, denoted by $\mathrm{K}(P)$, is the set of points in P from which every point in P is visible. Let p be a point in P. The *visibility polygon* of p, denoted by $\mathrm{VP}(p)$, is the set of points of P that are visible to p. The *visibility kernel* of $\mathrm{VP}(p)$, for short the *visibility kernel of* p, denoted by $\mathrm{VK}(p)$, is the set of points from which each point of $\mathrm{VP}(p)$ is visible.

Recall that a witness set is defined as a set of points in a given polygon P, such that if any set of points G guards W, then P is also guarded by G. We say that a point p *witnesses* another point q if guarding p guarantees that q is also guarded.

The following auxiliary results were proved for line-segment visibility. Nevertheless, they also hold for rectangular, staircase and k-periscope visibility models. This is because they rely on properties of visibility polygons that are valid in all the visibility models mentioned above.

Theorem 1 *[3, Theorem 1] Let P be a simple polygon and let W be a point set in P. Then W is a witness set for P if and only if the union of the visibility kernels of the elements of W covers P completely.*

Lemma 2 *[3, Lemma 1] Let P be a polygon and let p and q be points in P. Then p witnesses q if and only if q lies in $\mathrm{VK}(p)$.*

Lemma 3 *[3, Lemma 2] Let P be a simple polygon. A point p in P witnesses a point q in P if and only if $\mathrm{VP}(p) \subset \mathrm{VP}(q)$.*

Lemma 4 *[3, Lemma 3] Let P be a simple polygon, and let p, q and r be points in P. If p witnesses q and q witnesses r, then p witnesses r.*

We now give two definitions of directed graphs that we use in the next section. Two nodes u and v in a directed graph are said to be mutually adjacent if there is an arc from u to v and there is an arc from v to u. A *clique* in a directed graph is a set of pairwise mutually adjacent nodes of the directed graph.

The *pixelation* of P is the partition of P obtained by extending a horizontal and a vertical line inward at every reflex vertex until each line hits the boundary. The regions obtained from this partition are known as *pixels*. We denote as Ψ the set of pixels obtained from the pixelation of an orthogonal polygon P. Note that, in general, Ψ may have a quadratic amount of elements.

3 Witnessing orthogonal polygons

It is known that, under line-segment visibility, if there exists a finite witness set W, then the elements of a minimal witness set W are always be placed on the boundary of the polygon [3]. For orthogonal polygons under rectangular, staircase or k-periscope visibility that is not always the case, even though we can always find a finite witness set for an orthogonal polygon under these three visibility models (as proven below in Lemma 7). In Figure 1 we show an orthogonal polygon that is not witnessed even if we place a witness at each point of its boundary for each of the considered visibility models.

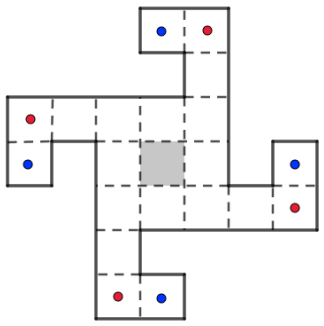


Figure 1: An orthogonal polygon P such that its boundary is not a witness set. The red points guard the boundary of P under rectangular visibility. The blue points guard the boundary of P under 1-periscope and staircase visibility. In both cases the gray region remains unguarded. Hence, there has to be a witness in the interior of P. To attain the same effect for k-periscope visibility we only need to bend $k-1$ times each extremity of P.

Now, we prove that it is always possible to find a finite witness set for an orthogonal polygon under rectangular, staircase and k-periscope visibility. Consider the set of pixels Ψ obtained from the pixelation of an orthogonal polygon P. We next show that any two points in a pixel of Ψ have the same visibility kernel.

Lemma 5 *Let X be a pixel obtained from the pixelation of P and let $p, q \in X$ be two distinct points. Then $\mathrm{VK}(p) = \mathrm{VK}(q)$, and $X \subset \mathrm{VK}(p)$ under rectangular, staircase and k-periscope visibility.*

Proof. Let H be the maximal rectangle contained in P whose top edge contains the top edge of X and whose bottom edge contains the bottom edge of X. Similarly, let V be the maximal rectangle contained in P whose left edge contains the left edge of X and whose right edge contains the right edge of X. Note that the left and right edges of H and the top and bottom edges of V are contained in edges of P.

Note that if $\mathrm{VP}(p) = \mathrm{VP}(q)$, then $\mathrm{VK}(p) = \mathrm{VK}(q)$. Therefore we prove that $\mathrm{VP}(p) = \mathrm{VP}(q)$. Consider a point $r \in P$ seen by p. Now we prove that r is also visible to q. Note that if r is contained in H or V then r is trivially visible from q under any of the three visibility models. Therefore, we suppose that r is not in H nor V.

First, we consider rectangular visibility, see Figure 2a. As r is seen by p, $R(p, r)$ is contained in P. Observe that the horizontal edges of $R(p, r)$ and $R(q, r)$ incident to p and q, respectively, are contained in H. Similarly, the vertical edges of $R(p, r)$ and $R(q, r)$ incident to p and q, respectively, are contained in V. As the symmetric difference of $R(p, r)$ and $R(q, r)$ is contained in $V \cup H$ for any $r \in P$ visible to p, $R(q, r)$ is contained in P. Hence, q sees r.

Now consider staircase visibility, see Figure 2b. Since p sees r, P contains a monotone isothetic polygonal path $T = t_0, t_1, \ldots, t_k$ joining p and r. Let t_i be the line segment of T with an endpoint in $H \cup V$ and the other one outside, and let ℓ be the straight line containing t_i. Let s_1 be the line segment orthogonal to ℓ joining q and ℓ and let $s = s_1 \cap \ell$. Let s_2 be the line segment joining s and $t_i \cap t_{i+1}$. Note that either t_i contains s_2 or s_2 contains t_i. In any case, the isothetic polygonal path $s_1, s_2, t_{i+1}, t_{i+2}, \ldots, t_k$ is monotone and is contained in P. Hence, q sees r.

Finally, consider k-periscope visibility, see Figure 2c. Since p sees r, P contains an isothetic polygonal path $T = t_0, t_1, \ldots, t_k$ joining p and r with at most k bends. Let t_i be the first line segment of T with an endpoint in X and the other one outside of X, and let $t' = t_{i+1} \cap t_{i+2}$. Let s be the intersection point of the line through t_{i+1} and the line through q parallel to t_i. Note that s is either contained in H or V. Thus, the polygonal path $\overline{qs}, \overline{st}', t_{i+2}, t_{i+3}, \ldots, t_k$ joining q and r is completely contained in P and has at most k bends. Hence, q sees r.

As $\mathrm{VP}(p)$ for any $p \in X$ is contained in the visibility polygon of any other point in X, $X \subset \mathrm{VK}(p)$. □

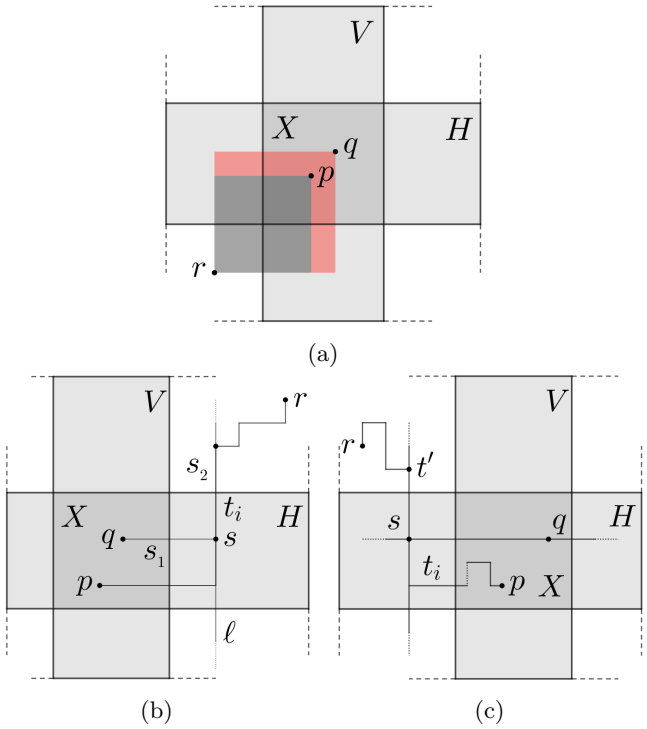


Figure 2: Illustration of the proof for Lemma 5. Given two points p and q in the same pixel X, they have the same visibility polygon under: (a) rectangular visibility, (b) staircase visibility and (c) k-periscope visibility.

The following corollary is a direct consequence of the previous lemma.

Corollary 6 *Let p be a point in an orthogonal polygon P. Then, $\mathrm{VK}(p)$ is the union of a set of pixels of the pixelation of P under rectangular, staircase and k-periscope visibility.*

Lemma 5 allows us to give the following definitions. We define the *visibility kernel* of a pixel a, denoted as $\mathrm{VK}(a)$, as the visibility kernel of any point in a. We say that a pixel a witnesses a pixel b if any point in a contains any point in b in its visibility kernel. Note that, by Lemma 4, if the pixel a witnesses the pixel b, then a witnesses the region of P witnessed by b.

Lemma 7 *Let P be an orthogonal polygon with n vertices. There is always a finite witness set W for P under rectangular, staircase and k-periscope visibility. Furthermore, W has $O(n^2)$ elements.*

Proof. Let Ψ be the set of pixels obtained from the pixelation of P. By Lemma 5, the visibility kernel of

any point p in P contains the pixel of P containing it. By Theorem 1, a subset W of P is a witness set if the union of the visibility kernels of the elements of W is P. Therefore, a set of points containing a point in each pixel of Ψ is a witness set for P. Since $|\Psi| \in O(n^2)$, we can always find a finite witness set with $O(n^2)$ elements for an orthogonal polygon under the considered visibility models. □

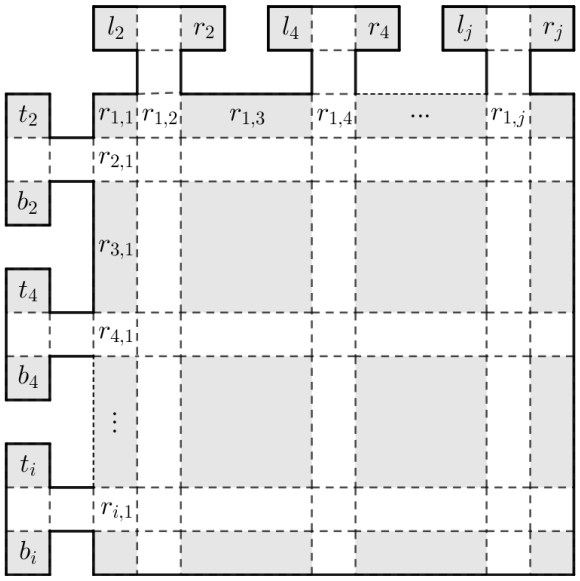


Figure 3: A family of orthogonal polygons that needs a quadratic number of witnesses under staircase visibility.

Theorem 8 *There are orthogonal polygons with n vertices for which any witness set has cardinality $\Omega(n^2)$ under staircase visibility.*

Proof. Let P be an orthogonal polygon consisting of a rectangle R with m vertically oriented T-shaped orthogonal polygons attached to the interior of its left edge and m horizontally oriented T-shaped orthogonal polygons attached to the interior of its top edge. We illustrate this construction in Figure 3.

Consider the pixelation of P. Observe that R is subdivided in a grid with i rows and j columns of pixels, with $i = j = 2m + 1$. We denote as $r_{i,j}$ the pixel at the i-th row and the j-th column of R.

Consider the pixels of the T-shaped subpolygons which are shown shaded in Figure 3. We label these pixels as follows. If T is attached to the left edge of R at the i-th row of the pixelation we label the top pixel of T as t_i and the bottom pixel of T as b_i. If T is attached to the top edge of R at the j-th column of the pixelation we label the left pixel of T as l_j and the right pixel of T as r_j.

Consider a pixel $r_{i,j}$ with both i and j odd, shown in gray in Figure 3. Then, the set consisting of a guard in each pixel b_k for $k < i$, t_k for $k > i$, r_k for $k < j$ and l_k for $k > j$ guards each pixel in P except $r_{i,j}$. Therefore, we need to place a witness in each of the $r_{i,j}$ with both i and j odd. Note that there are $(m + 1)^2$ such pixels in R.

Since P has $n = 16m + 4$ vertices and there are $(m + 1)^2$ pixels in R in which we need to place a witness, it follows that P needs $\Omega(n^2)$ witnesses under staircase visibility.

□

It follows from Lemmas 5 and 7 that any minimal witness set contains at most one point in each pixel of P. For the sake of simplicity, we will henceforth say that a set of pixels L is a witness set if a set containing a point in each pixel of L is a witness set.

The following remarks follow from Theorem 1 and Lemmas 2, 3 and 4:

- If a pixel a is not contained in the visibility kernel of any other pixel in P, then a must be included in the witness set W.
- If a pixel a is contained in the visibility kernel of the pixel b but b is not contained in the visibility kernel of a, then a cannot be included in a minimum witness set.
- If two or more different pixels contain each other on their respective visibility kernels, then only one of them can be included in a minimal witness set.

3.1 An algorithm for finding a minimum witness set

In order to find the pixels contained in a minimum witness set, we first obtain the set of pixels Ψ from the pixelation of P. Then, we construct a directed graph H, which we call the *kernel graph* of P, in such a way that there is a bijection between the set of nodes of H and Ψ. After that, we compute the visibility kernel K of the pixel represented by each node $u \in H$. Finally, we add to H the arc from u to v if K contains the pixel represented by $v \in H$, with $u \neq v$. For the sake of simplicity, we say that a node u in H witnesses another node v if H contains the arc (u, v).

To compute the visibility kernel of a point p in an orthogonal polygon under rectangular and staircase visibility, we first compute $\mathrm{VP}(p)$ and then we compute $\mathrm{K}(\mathrm{VP}(p))$, the kernel of $\mathrm{VP}(p)$. It is straightforward to see that, under rectangular visibility, the visibility region $\mathrm{VP}(p)$ of a point is also an orthogonal polygon. For orthogonal polygons without holes, and under staircase visibility, Gewaly [5] proved that the visibility region of a point is also an orthogonal polygon. Therefore, we can use one of the existing algorithms for computing the kernel of an orthogonal polygon under rectangular or staircase visibility.

Now we prove that the polygon obtained in this manner is indeed the visibility kernel of the point p. In orthogonal polygons with holes under staircase visibility that is not always the case as shown in Figure 4.

Proposition 9 *Let P be an orthogonal polygon (possibly with holes). Let p be a point in P. Then the polygon obtained by computing the kernel of* $\mathrm{VP}(p)$ *is equal to* $\mathrm{VK}(p)$ *under rectangular visibility.*

Proof. It is straightforward to see that $\mathrm{K}(\mathrm{VP}(p))$ is contained in $\mathrm{VK}(p)$. Therefore we only prove that $\mathrm{K}(\mathrm{VP}(p))$ contains $\mathrm{VK}(p)$. Suppose that there exists a point $q \in \mathrm{VK}(p)$ which is not contained in the polygon obtained by computing the kernel of $\mathrm{VP}(p)$. Thus, there exists a point $r \in \mathrm{VP}(p)$ such that $R(q,r)$ is contained in P but not in $\mathrm{VP}(p)$. As p sees both q and r, then both $R(p,q)$ and $R(p,r)$ are contained in P. Therefore, $R(p,q) \cup R(p,r) \cup R(q,r) \subset P$. This implies that for any point $l \in R(q,r)$ we have that $R(p,l) \subset R(p,q) \cup R(p,r) \cup R(q,r) \subset P$. Therefore, $l \in \mathrm{VP}(p)$, which implies that $R(q,r) \subset \mathrm{VP}(p)$, a contradiction. Hence, q is contained in the polygon obtained by computing the kernel of $\mathrm{VP}(p)$. □

Proposition 10 *Let P be an orthogonal polygon without holes. Let p be a point in P. Then the polygon obtained by computing the kernel of* $\mathrm{VP}(p)$ *is equal to* $\mathrm{VK}(p)$ *under staircase visibility.*

Proof. It is straightforward to see that $\mathrm{K}(\mathrm{VP}(p))$ is contained in $\mathrm{VK}(p)$. Therefore we only prove that $\mathrm{K}(\mathrm{VP}(p))$ contains $\mathrm{VK}(p)$. Suppose there exists a point $q \in \mathrm{VK}(p)$ which is not contained in the polygon obtained by computing the kernel of $\mathrm{VP}(p)$. Thus, there exists a point $r \in \mathrm{VP}(p)$ such that T, the monotone isothetic polygonal path joining q and r, is contained in P but not in $\mathrm{VP}(p)$. As p sees both q and r, there exist two monotone isothetic polygonal paths contained in P, the first one T' joining p and q, and the second one T'' joining p and r. Since P has no holes, the region R bounded by T, T' and T'' is contained in P. Thus, we can always find a monotone isothetic polygonal path M joining p to any point of T, such that M is contained in R. Therefore, p sees every point in T which implies that T is contained in $\mathrm{VP}(p)$, a contradiction. Hence, q is contained in the polygon obtained by computing the kernel of $\mathrm{VP}(p)$. □

Now we show how to find a minimum witness set once we have constructed the kernel graph H of P. Observe that, by Lemma 4, for any clique C in H, any node of C witnesses all the elements of C.

We say that a node u of H is a *source node* if for each arc of the form (v,u) for any other node v, H contains the arc (u,v).

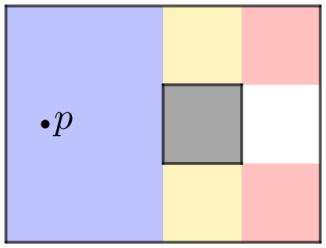


Figure 4: An orthogonal polygon with a hole (shown in gray) and an interior point p. Under staircase or 1-periscope visibility the following holds. The visibility polygon of p is the union of the blue, yellow and red regions. The visibility kernel of p is the union of the blue and red regions. The kernel of the visibility polygon of p is the blue region.

Theorem 11 *Let P be an orthogonal polygon. Let H be the kernel graph of P. Let C be a set containing one node in each maximal clique of source nodes in H. Then any set containing exactly one point in the pixel represented by each node of C is a minimum witness set for P.*

Proof. Let u be a node of H. If u is a source node then it can only be witnessed by a node contained in a clique containing u. Note that, since witnessing is transitive (Lemma 4), each node in H is contained in at most one maximal clique. Therefore, we need one witness per maximal clique of source nodes, placed in any of the pixels associated to the nodes of the clique.

Now suppose that u is not a source node. As witnessing is transitive, there exists an arc from a source node to u. Otherwise, u would be a source node. Therefore, it is not necessary to place a witness in a pixel corresponding to a non-source node in H.

Hence, the witness set composed by a pixel for each maximal clique of source nodes in H is a minimum witness set for P. □

In order to report a minimum witness set, we do a traversal of H as follows. If the node $u \in H$ is not a source node, we remove it from H. If u is a source node, we add the pixel represented by u to the witness set W and remove u as well as its neighborhood from H. Note that in this manner we add to the witness set at most one pixel for each maximal clique of source nodes in H.

Now we analyze the running time of the proposed solution. Obtaining the pixelation takes $O(n^2)$ time, since we need to report $O(n^2)$ regions. The time required to create the directed graph H depends on the subroutines used to compute the visibility kernel of each pixel.

In their book [4], Fink and Wood give an $O(n \log n)$ time algorithm to obtain $\mathrm{VP}(p)$ from a point p in an orthogonal polygon under rectangular visibility. In [11], Schuierer and Wood give an $O(n)$ time algorithm to

obtain the kernel of an orthogonal polygon under rectangular visibility. Note that, by Corollary 6, VK(p) is a set of pixels under rectangular visibility. Note also that VK(p) is a rectangle. Therefore, finding the coordinates enclosing the set of pixels in VK(p) takes $O(1)$ time. Since the visibility kernel of a point may have $O(n^2)$ pixels, H may have $O(n^4)$ arcs. Therefore, constructing H takes $O(n^4)$ time under rectangular visibility.

In [5], Gewali gives an $O(n)$ time algorithm to obtain VP(p) from a point p in an orthogonal polygon without holes under staircase visibility. In the same paper, he gives an algorithm to obtain the kernel of a point in $O(n)$ time for orthogonal polygons without holes under staircase visibility. Once we have computed the visibility kernel of a point, it is not difficult to see that we can find all the pixels it contains in $O(n^2)$ time. Since we only do this once for each pixel, this step takes $O(n^4)$ time. Therefore, constructing H takes $O(n^4)$ time under staircase visibility.

In the last step of the algorithm we do a traversal of H to report the obtained minimum witness set. Since we process each node of H just once, this step takes $O(n^2)$ time. Therefore, our procedure takes $O(n^4)$ overall time under rectangular and staircase visibility. For the case of k-periscope visibility, we can achieve the same running time under the assumption that we can obtain the visibility kernel of a point in $O(n^2)$ time. However, efficiently calculating the visibility kernel under k-periscope visibility is, to the best of our knowledge, an open problem.

4 Conclusions

In this paper we studied the Witness Problem on orthogonal polygons under three models of orthogonal visibility. We proved that there are orthogonal polygons that are not witnessed by their boundary under rectangular, staircase and k-periscope visibility. Next proved that all orthogonal polygons admit a finite witness set under these three visibility models. We achieved this by using the so called pixelation of an orthogonal polygon, in which any two points in the same pixel turned out to have the same visibility polygon. We also proved that, under staircase visibility, some orthogonal polygons require a quadratic number of witnesses. As the main result, we gave an $O(n^4)$ time algorithm for computing a minimum witness set for orthogonal polygons under the rectangular and staircase visibility models. This algorithm makes use of the pixelation of a polygon, and relies on an algorithm for computing the visibility kernel of a point under each visibility model.

References

[1] T. Biedl and S. Mehrabi. On r-guarding thin orthogonal polygons. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 64, pages 17:1–17:13. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

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

[3] K.-Y. Chwa, B.-C. Jo, C. Knauer, E. Moet, R. Van Oostrum, and C.-S. Shin. Guarding art galleries by guarding witnesses. *International Journal of Computational Geometry & Applications*, 16(02n03):205–226, 2006.

[4] E. Fink and D. Wood. *Restricted-orientation convexity*. Springer Science & Business Media, 2012.

[5] L. P. Gewali. Recognizing s-star polygons. *Pattern recognition*, 28(7):1019–1032, 1995.

[6] L. P. Gewali and S. Ntafos. Covering grids and orthogonal polygons with periscope guards. *Computational Geometry*, 2(6):309–334, 1993.

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

[8] R. Motwani, A. Raghunathan, and H. Saran. Covering orthogonal polygons with star polygons: The perfect graph approach. *Journal of computer and system sciences*, 40(1):19–48, 1990.

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

[10] D. Schuchardt and H.-D. Hecker. Two np-hard art-gallery problems for ortho-polygons. *Mathematical Logic Quarterly*, 41(2):261–267, 1995.

[11] S. Schuierer and D. Wood. Staircase visibility and computation of kernels. *Algorithmica*, 14(1):1–26, 1995.

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

[13] J. Urrutia. Art gallery and illumination problems. In *Handbook of computational geometry*, pages 973–1027. Elsevier, 2000.

[14] C. Worman and J. M. Keil. Polygon decomposition and the orthogonal art gallery problem. *International Journal of Computational Geometry & Applications*, 17(02):105–138, 2007.

Red-Blue-Partitioned MST, TSP, and Matching

Matthew P. Johnson*

Abstract

Arkin et al. [2] recently introduced *partitioned pairs* network optimization problems: Given n pairs of points in a metric space, the task is to color one point from each pair red and the other blue, and then to compute two separate *network structures* or disjoint (node-covering) subgraphs of a specified sort, one on the graph induced by the red points and the other on the blue points. Three structures have been investigated by [2]—*spanning trees*, *traveling salesperson tours*, and *perfect matchings*—and the three objectives to optimize for when computing such pairs of structures: *min-sum*, *min-max*, and *bottleneck*. We provide improved approximation guarantees and/or strengthened hardness results for these nine NP-hard problem settings.

1 Introduction

We consider the class of *partitioned pairs* network optimization problems recently introduced by Arkin et al. [2]. Given a complete metric-weighed graph G whose vertex set consists of n pairs $\{p_1, q_1\}, ..., \{p_n, q_n\}$ (with n even), the task is to color one node from each pair red and the other blue, and then to compute two *network structures* or disjoint (node-covering) subgraphs of a specified sort, one on the graph induced by the blue nodes and the other on the red nodes. One motivation is robustness: if the pairs represent n different types of resources needed to build the desired network structure, with two available instances p_i, q_i of each type i, then solving the problem means computing two separate independent instances of the desired structure, one of which can be used as a backup if the other fails.

The structures that have been investigated are *spanning trees*, *traveling salesperson*, and *perfect matchings*. A solution consists of a disjoint pair of subgraphs covering all nodes, i.e., two (partial) matchings, two trees, or two cycles, and there are different potential ways of evaluating the cost of the pair. The optimization objectives that have been considered are: 1) minimize the sum of the two structures' costs (*min-sum*), 2) minimize the maximum of the two structures' costs (*min-max*), and 3) minimize the weight of the heaviest edge used in either of the structures (*bottleneck*).

Contributions. We provide a variety of results for these nine problem settings (all of which turn out to be NP-hard; see Table 1), including algorithms with improved approximation guarantees and/or stronger hardness results for each. In particular, we provide tighter analyses of the min-sum and min-max approximation factors (along with problem instances matching these factors) of Arkin et al. [2]'s 2-MST approximation algorithm, which is equivalent to that of Algorithm 1 below. We also show that a simple extension of this algorithm (see Algorithm 2 below) provides improved min-sum and min-max approximation guarantees for 2-TSP. See the full version of the paper for omitted proofs.

Related work. The primary antecedent of this work is Arkin et al. [2] (see also references therein), which introduced the class of 2-partitioned network optimization problems. Earlier related problem settings include optimizing a path visiting at most one point from each pair [8], generalized MST [15, 17, 3], generalized TSP [3], constrained forest problems [9], adding conflict constraints to MST [18, 12, 6] and to perfect matching [16, 6], and balanced partition of MSTs [1].

2 2-MST

2.1 Min-sum/min-max 2-MST: algorithm

In this section we give a simple algorithm (see Algorithm 1) that provides an approximation guarantee for 2-MST under both the min-sum and min-max objectives. The key lemma that the approximation guarantee relies on proves a property about the result of partitioning a metric-edge-weighted spanning tree into a 2-component spanning forest.

Initially we show that *for any 2-coloring* $V_1 \cup V_2 = V$ of the graph, the sum of the costs of MSTs on V_1 and V_2 will be at most 3 times the cost of the spanning tree on V. Then we modify the argument to improve the combined cost of the two trees slightly, reducing it by the weight of their single heaviest edge in the following key lemma.

Theorem 1 *Let V^C be a set of points lying within a metric space. Let T^C be an MST on V. Let $V_1^C \cup V_2^C = V^C$ be any 2-coloring of V^C, and let T_1^C and T_2^C be MSTs of V_1^C and V_2^C, respectively. Let $e_C = \{v_L^C, v_R^C\}$ be a heaviest edge in T, with weight w_C. Let T_L, T_R be the trees (on nodes V_L^C, V_R^C, respectively) obtained by*

*Lehman College and The Graduate Center, CUNY

Table 1: Summary of results. $R, B \subseteq E$ denote the red and blue solutions, respectively. UB values indicate the approximation factors we obtain, *all for general metric spaces*; LB values indicate hardness of approximation lower bounds, *all (except min-sum and min-max TSP) for the special case of metric weights* $\{1, 2\}$. Best prior bounds (all due to [2]) are also shown, where $\rho_{\text{St}} \leq 2$ denotes the underlying metric space's Steiner ratio (conjectured to be $\frac{2}{\sqrt{3}} \approx 1.1547$ in Euc. 2D [11]), and ρ_{tsp} denotes TSP's best achievable approximation factor in the underlying metric space (currently $\rho_{\text{tsp}} = 1.5$ in general [4]).

		min-sum $c(R) + c(B)$	min-max $\max\{c(R), c(B)\}$	bottleneck $\max\{w_e : e \in B \cup R\}$
MST	**our UB**:	**3**	**4**	–
	[2]'s UB:	$(3\rho_{\text{St}})$	$(4\rho_{\text{St}})$	(9)
	our LB:	**NP-h**	**NP-h**	**2**
	[2]'s LB:	(-)	(NP-h in metric)	(-)
TSP	**our UB**:	**4**	**4**	–
	[2]'s UB:	$(3\rho_{\text{tsp}})$	$(6\rho_{\text{tsp}})$	(18)
	our LB:	$\mathbf{123/122 \approx 1.00819}$ ***with metric weights*** $\{\mathbf{.5}, \mathbf{1}, \mathbf{1.5}, \mathbf{2}\}$		**2**
	[2]'s LB:	(-)	(-)	(-)
matching	**our UB**:	–	–	–
	[2]'s UB:	(2)	(3)	(3)
	our LB:	$\mathbf{\frac{8305}{8304} \approx 1.00012}$	$\mathbf{\frac{8305}{8304} \approx 1.00012}$	**2**
	[2]'s LB:	(NP-h in metric)	(weakly NP-h in 2D Euc.)	(-)

Algorithm 1 Min-sum/min-max 2-MST approx

1: $T \leftarrow$ an MST on the $2n$ nodes
2: $\{T_L, T_R\} \leftarrow$ result of deleting a max-weight edge $e_\times$ from T
3: **for** each node pair $(p_i, q_i) \in V_L \times V_R$ **do** color p_i blue and q_i red
4: **for** each other node pair (p_i, q_i) **do** assign p_i, q_i arbitrary distinct colors
5: **for** $c \in \{b, r\}$: $T_c \leftarrow$ a minimum-weight tree spanning the color-c nodes
6: **return** $\{T_b, T_r\}$

deleting e_C from T^C, where $v_L^C \in T_L^C$ and $v_R^C \in T_R^C$. Then we have:

(a) $c(T_1^C) + c(T_2^C) \leq 3c(T^C) - w_C$, *and*

(b) $\max\{c(T_1^C), c(T_2^C)\} \leq 2c(T^C) - w_C$.

Observation 1 *There exist families of instances showing that bounds (a) and (b) of Theorem 1 are (simultaneously) tight.*

Then we analyze Algorithm 1, which forms trees T_L, T_R by deleting a max-weight edge $e_\times$ (of weight $w_\times$) from an MST T computed on the $2n$ nodes, and then colors all "lone" nodes appearing without their partners in T_L blue and all lone nodes in T_R red, and assigns arbitrary distinct colors to all other node pairs.

Theorem 2 *Algorithm 1 provides a 3-approximation for min-sum 2-MST.*

The proof analyzes three cases, depending on whether one, both, or neither T_L, T_R contains a pair, the first two cases of which imply that OPT must cross between T_L and T_R at least once or twice, respectively. The challenge is that $c(OPT)$ is lower-bounded by $c(T_L) + c(T_R)$ but not by $c(T) = c(T_L) + w_\times + c(T_R)$. We bound ALG by carefully applying Theorem 1 to *both* T_L *and* T_R, and we obtain a bound on $c(OPT)$ include $w_\times$ or $2w_\times$, permitting the two bounds to be compared, by subtracting max-weight edges from one or both sides.

This immediately implies that the same algorithm provides 6-approximation for min-max 2-MST, but we perform a tighter analysis.

Theorem 3 *Algorithm 1 provides a 4-approximation for min-max 2-MST.*

Extending Observation 1, we obtain:

Observation 2 *There exist families of instances showing that the 2-MST min-sum and min-max approximation ratios are both tight.*

2.2 Min-sum/min-max/bottleneck: hardness

We provide a reduction inspired by the reduction of [7] from Three-Dimensional Matching to the problem of partitioning a bipartite graph into two connected components, each containing exactly half the vertices.

In our reduction, however, we reduce the traditional 3-SAT problem.

Given the 3-SAT formula, we construct the following graph (see Fig. 1). For each clause, create a path of length p. For each variable x_i, we create create two nodes, x_i and $\bar{x}_i$. We also create a path of length p_b called b and a path of length p_r called r. From each x_i or $\bar{x}_i$, we draw an edge to the final nodes of the paths

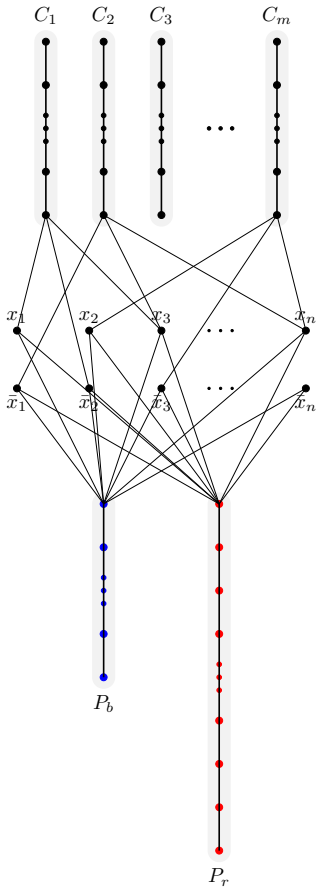


Figure 1: Spanning tree reduction.

corresponding the clauses that the literal appears in. Finally, from each x_i *and* $\bar{x}_i$, we draw edges to the final nodes paths b and r. All the edges defined have 1; all non-defined edges have weight 2. (In all cases when we refer to the "final" node of one of these $m+2$ paths, we mean the node with degree > 2.)

The path lengths are defined as follows: $p_r = (n+1) \cdot n^3 + n + n + 1, p_b = n^3 + n + 1, p = n^3 + 1$.

Then the total number of nodes in the graph constructed is: $|V| = n \cdot p + p_b + p_r + m, = 2 \cdot (n_r + m)$.

Finally, we must specify the $\{p_i, q_i\}$ pair relationships of these nodes. Each pair $\{x_i, \bar{x}_i\}$ is a $\{p_i, q_i\}$ pair. All p_r nodes of path p_r are p_i s. All p_b nodes of path p and all p nodes of path corresponding to an element are q_i nodes. Observe that results in an equal number of p_i and q_i nodes since $p_b + n \cdot p = p_r$.

Lemma 4 *The formula is satisfiable iff the constructed graph admits a 2-MST solution using only weight-1 edges.*

Thus we conclude:

Theorem 5 *In the special case of metric graphs with weights 1 and 2, min-sum and min-max, 2-MST are both (strongly) NP-Complete, and bottleneck 2-MST is NP-hard to approximate with factor better than 2.*

Algorithm 2 Min-sum/min-max 2-TSP approx

Identical to Alg, 1, except with lines 5,6 replaced by:

5: $C \leftarrow$ a TSP tour, computed from T by edge-doubling
6: **for** $c \in \{b, r\}$: $C_c \leftarrow$ a tour of the color-c nodes, computed by shortcutting C
7: **return** $\{C_b, C_r\}$

3 2-TSP

3.1 Min-sum/min-max/bottleneck 2-TSP: hardness

Clearly the min-sum and min-max objectives for 2-TSP are at least as hard to approximate as ordinary TSP in the same metric space (e.g., hard to approximate with factor better than 123/122 [13], even with edge weights $\{.5, 1, 1.5, 2\}$): to reduce TSP to either of these, simply introduce a co-located pair $\{p_v, q_v\}$ for each node v in the TSP instance. Similarly, the same reduction implies that the bottleneck objective for 2-TSP is at least as hard to approximate as ordinary bottleneck TSP in the same metric space (e.g., hard to approximate with factor better than 2, even with edge weights $\{1, 2\}$).

3.2 Min-sum/min-max 2-TSP: algorithm

Now we adapt Algorithm 1 above to obtain a 4-approximation algorithm for min-sum and min-max 2-TSP (see Algorithm 2).

The proof again analyzes three cases, depending on whether one, both, or neither T_L, T_R contains a pair. Unlike with 2-MST, 2-TSP's $c(OPT)$ *is* lower-bounded by $c(T)$ in the first two cases, and so we can compare it to the simple upper bound on $c(ALG)$ of $4c(T)$.

Theorem 6 *Algorithm 2 is a 4-approximation algorithm for min-sum 2-TSP.*

Theorem 7 *Algorithm 2 is a 4-approximation algorithm for min-max 2-TSP.*

Observation 3 *There exist families of instances showing that the 2-TSP min-sum and min-max approximation factors are both tight.*

4 2-Matching

4.1 Preliminaries

In the case of perfect matching we require that the number of pairs n be even. It will be convenient to re-express the 2-Matching problem as an equivalent problem concerning cycle covers.

We begin with some observations about the nature of feasible solutions in this setting. By definition, two nodes p_i, q_i from the same pair can never be matched

because they must receive different colors. Each must then be matched with a node of the same color, and each of *those* nodes's partners must receive the opposite color and be matched with a node of that color, and so on, in a consistent fashion. One way to make this consistency requirement concrete is the following alternative description. First, for each pair $\{p_i, q_i\}$, draw a length-2 path (of unit-weight edges) between them, separated by a dummy node d_i, and in the resulting $3n$-node graph G' consider instead the task of finding a 2-factor, i.e., a node-disjoint cycle cover, of minimum cost. In particular, consider seeking a cycle cover that uses only unit-weight edges, which would have cost $3n$.

Definition 1 *Say that a 2-matching or cycle cover is* feasible *if it uses only unit-weight edges. We call a non-dummy node of G' (i.e., a node from G) a* real node*; similarly, we call an edge between a dummy node and a real node G' a* dummy edge *and a path $p_i d_i q_i$ a* dummy path*; we call an edge between two real nodes a* real edge*.*

Observe that any feasible 2-matching in G will induce a 2-factor of G': imagine G' drawn in a "tripartite" style, with the red nodes in the left column, the blue nodes in the right column, and the dummy nodes in the center column. Then each path $p_i - d_i - q_i$ forms a "cross-edge" (going either left or right), each red edge appears in the left column, and each blue edge appears in the right column. Each non-dummy node is matched with one other node in the 2-matching, so if we combine the edges of the paths $p_i - d_i - q_i$ to those of the matching, then in the graph induced by these edges, each of the $3n$ nodes will have degree 2. This implies the edge set is a 2-factor. Note that the cost of the 2-factor differs by a known amount ($2n$, because each dummy nodes two edges are unit-weight)) from the (min-sum) cost of the corresponding 2-matching.

The problem of finding a minimum-cost 2-factor is known to be polynomial-time solvable by reduction to bipartite matching (folklore). Unfortunately, a 2-factor of G' will not necessarily induce a valid 2-matching on G. In G' as defined, the additional property needed (somewhat analogously to bipartite graphs having no odd cycles) is for *each cycle's size to be a multiple of 6*, which we will call a $C_{6\times}$-cover.

Definition 2 *Let a $C_{6\times}$-cover for a given graph be a 2-factor, i.e., a node-disjoint collection of subgraphs covering all nodes, where each subgrraph is a member of $\{C_6, C_{12}, C_{18}, ...\}$.*

Lemma 8 *Any feasible $C_{6\times}$-cover for G' will induce a feasible 2-matching for G.*

Unfortunately, unlike the problem of deciding whether a graph admits a feasible cycle cover, deciding whether it admits a $C_{6\times}$-cover is NP-Complete [10]. This fact does not immediately imply the hardness of the 2-Matching problems, however, because G' is not an arbitrary graph. We can characterize it as follows. It contains $3n$ nodes consisting of n triples $\{p_i, d_i, q_i\}$, where each d_i is degree 2, with neighbors p_i, q_i.

4.2 Bottleneck 2-Matching: hardness

To prove hardness, we give a reduction inspired by Papadimitriou's reduction [5] from 3-SAT to the problem of deciding whether a graph can be partitioned into a node-disjoint collection of cycles, each of size *at least 6*.

We reduce from Monotone 1-in-3 SAT (which has no negated literals) to the problem of deciding whether G' admits a (feasible, i.e., using unit-weight edges only) $C_{6\times}$-cover. Recall that edge weights in G' are 1 or 2, and that each dummy node's two edges are weight-1. Given the boolean formula, we proceed as follows.

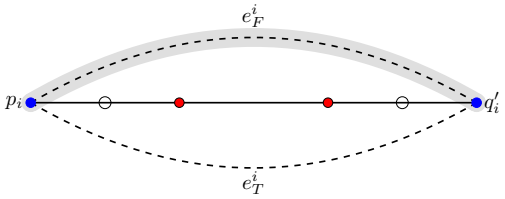
For each variable x_i, we create a gadget as shown in Fig. 2a. It consists of a 6-path $(p_i, d_i, q_i, p'_i, d'_i, q'_i)$, whose nodes form two triples $\{p_i, d_i, q_i\}, \{p'_i, d'_i, q'_i\}$, plus an edge (p_i, q'_i) labeled e^i_T and a *pseudoedge* labeled e^i_F. There will be exactly two feasible ways to cover the nodes of this gadget in a $C_{6\times}$-cover, with the cycle including e^T_i, corresponding to x_i being true, and the one including e^F_i, corresponding to false.

For each clause C_j, we create a gadget as shown in Fig. 2b. It consists of two copies of K_4, where each node u^j_ℓ in one K_r is connected by a 2-path and dummy node to a corresponding node v^j_ℓ in the other. Three *pseudoedges* connecting a distinguished node u^j_0 to the other three nodes of the same K_4 are labeled f^j_1, f^j_2, f^j_3. If a feasible $C_{6\times}$-cover, one of these edges will be on and the other two off, corresponding to a satisfied 1-in-3 SAT clause.

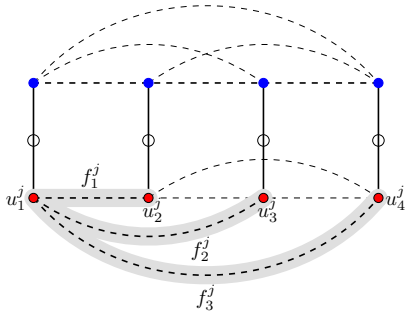
Definition 3 *A* pseudoedge *is an edge, or the result of attaching a connection gadget to a pseudoedge.*

Finally, to implement the appearance of a variable in a clause, we use the gadget shown in Fig. 2c, which will appear in sequence. Applying a connection gadget to pseudoedges e^i_F and f^j_ℓ does the following:

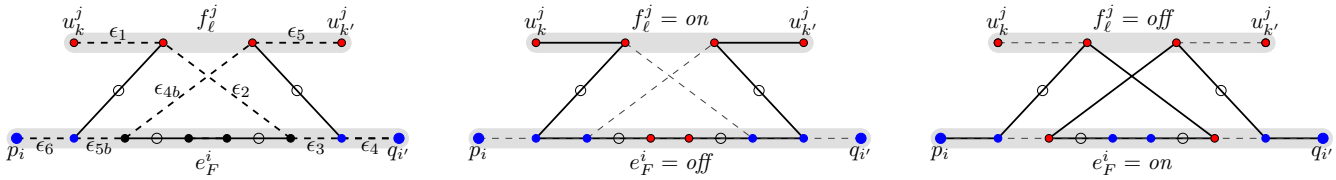
1. the last (rightmost) edge of f^j_ℓ is split into a 9-path path via the creation of 8 new nodes (compare e^i_F in Figs. 2a, 2c(left), and 2d);
2. f^j_ℓ's edge is replaced with two new edges (labeled ϵ_1, ϵ_5 in Fig. 2c) incident to two new nodes (compare f^j_ℓ in Figs. 2b and 2c(left));
3. f^j_ℓ's first new node is connected to e^i_F's first and seventh new nodes, by a 2-path and an edge, respectively (see Fig. 2c(left)); and
4. f^j_ℓ's second new node is connected to e^i_F's second and eighth new nodes, by an edge and a 2-path, respectively (see Fig. 2c(left)).



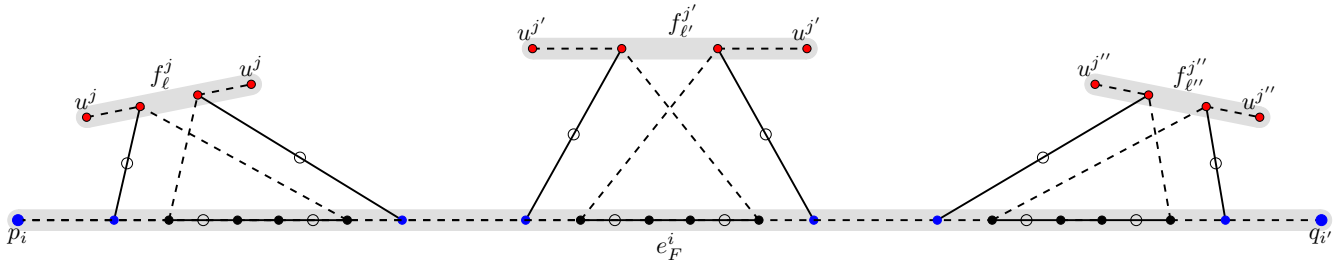
(a) Variable gadget for x_i. Any feasible $C_{6\times}$-cover must include pseudoedge e^i_F xor edge e^i_T.



(b) Clause gadget for C_j. Any feasible $C_{6\times}$-cover must include *exactly one* of the three distinguished pseudoedges f_1, f_2, f_3 (plus one of the unlabeled dashed edges from the bottom and two of the top).



(c) Connection gadget (left fig.) for an appearance (negated iff $v = F$) of variable x_i in clause C_j. The lower shaded path is a more detailed view of one of the x_i gadget's pseudoedge e^i_F (see (a)); the upper shaded path is a more detailed view of one of the C_j gadget's three distinguished pseudoedges f^j_1, f^j_2, f^j_3 (see (b)). We show (see Lemma 8) that there are only two possible feasible $C_{6\times}$-covers of the gadget, one in which f^j_ℓ is *on* and e^i_F is *off*, meaning this connection represents C_j's unique true literal (middle fig.), and one in which f^j_ℓ is *off* and e^i_F is *on*, meaning it represents one of C_j's two false literals (right fig.).



(d) The example e^i_F shown here is a more detailed view of pseudoedge e^i_F in variable x_i's gadget (see (a)). An e^i_F can have multiple connections (*in this example*, three), corresponding to appearances of x_i's in different clauses (in this example, an x_i literal appears as the ℓth literal in clause C_j, and so on; typically j, j', j'' will *all be distinct*); an f^j_ℓ has only one connection, since it indicates what literal the ℓth literal in clause C_j is. Subscripts of u nodes are omitted for clarity.

Figure 2: Gadgets used in 2-Matching's hardness proof. Real nodes are shown filled in, dummy nodes unshaded. Edges that must be used in any feasible solution are shown solid, other edges dashed. e^i_F and f^j_1, f^j_2, f^j_3 are pseudoedges, i.e., schematic representations of paths that connections attach to.

For each variable x_i appearing (in some position $k \in [3]$) within a clause C_j, we draw a connection gadget between x_i's e^i_F and C_j's f^j_k. First observe the following, which can be verified by inspection:

Fact 1 *If all pseudoedges e^i_F and f^j_k were simply edges, then a $C_{6\times}$-cover would induce one of two legal states within any variable X_i's gadget, with exactly one of e^i_F, e^i_T on, and one of three legal states within any clause C_j's gadget, with exactly one of f^j_1, f^j_2, f^j_3 on.*

Now we show that any $C_{6\times}$-cover will induce one of two canonical states on each connection gadget (see Fig. 2c middle and right), each pseudoedge, and each variable gadget.

Lemma 9 *Within any pseudoedge pair (e^i_F, f^j_k) connected by a connection gadget, a feasible $C_{6\times}$-cover induces one of only two legal states: one with the first and last edges (labeled ϵ_1 and ϵ_5, respectively, in Fig. 2c(left)) within f^j_k on ("f^j_k is on"), and the other with with the first and last edges (labeled ϵ_6 and ϵ_4, respectively, in Fig. 2c(left)) within e^i_F on ("e^i_F is on").*

This immediately implies:

Corollary 1 *A feasible $C_{6\times}$-cover induces one of two canonical states within each variable gadget and one of three canonical states within each cause gadget.*

In a solution where the clause's edge f_k^j is on, this forces $e_{i_k}^F$ to be off, and hence $e_{i_k}^T$ to be on; similarly, it forces clause C_j's other two distinguished pseudoedges to be off, and hence the variables connected to those edges to be false. (The clause gadget's other edges can be freely used or not, as needed to form a feasible $C_{6\times}$-cover.) Finally, observe that the final constructed graph G' indeed satisfies the required structure for corresponding to an equivalent instance G of the 2-Matching problem: every dummy node has exactly two neighbors (both real), and every real node has exactly one dummy neighbor.

From the arguments above, we conclude that G' admits an all-unit weight $C_{6\times}$-cover iff G admits an all-unit weight 2-matching iff the underlying boolean formula is satisfiable. Thus we conclude:

Theorem 10 *In the special case of metric graphs with weights 1 and 2, bottleneck 2-Matching is NP-hard to approximate with factor better than 2 (and min-sum and min-max 2-Matching are both (strongly) NP-Complete).*

4.3 Min-sum/min-max 2-Matching: hardness

By reduction from a special case of Max 1-in-3 SAT, we can obtain a hardness of approximation result for the min-sum and min-max objectives. Let Max 1-in-3 SAT-5 denote Max 1-in-3 SAT under the restriction that each variable appears in at most 5 clauses.

Lampis has shown (implicitly in [14][1]) the following:

Lemma 11 *There exists a family of* Max 1-in-3 SAT-5 *instances with $15m$ clauses and $8.4m$ variables, each appearing in at most 5 clauses, for which, for any $\epsilon > 0$, it is NP-hard to decide whether the minimum number of unsatisfiable clauses is at most ϵm or at least $(0.5 - \epsilon)m$.*

For concreteness, let Min Not-1-in-3 SAT-5 indicate the optimization problem of minimizing the number of unsatisfied clauses in a 1-in-3 SAT-5 formula.

Now we argue that the same construction used above provides an approximation-preserving reduction from Min Not-1-in-3 SAT-5.

Corollary 2 *Min-sum and min-max 2-Matching are both, in the special case of metric graphs with weights 1 and 2, NP-hard to approximate with factor better than $8305/8304 \approx 1.00012$.*

[1] Karpinksi et al. [13] provide a similar construction yielding a stronger hardness of approximation lower bound for Metric TSP, but adapting that construction to our present problem actually leads to a slightly weaker lower bound.

Acknowledgements. This work was supported in part by NSF award INSPIRE-1547205, and by the Sloan Foundation via a CUNY Junior Faculty Research Award. We thank Ali Assapour, Ou Liu, and Elahe Vahdani for useful discussions.

References

[1] M. Andersson, J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Balanced partition of minimum spanning trees. *International Journal of Computational Geometry & Applications*, 13(04):303–316, 2003.

[2] E. M. Arkin, A. Banik, P. Carmi, G. Citovsky, S. Jia, M. J. Katz, T. Mayer, and J. S. B. Mitchell. Network optimization on partitioned pairs of points. In *ISAAC*, pages 6:1–6:12, 2017.

[3] B. Bhattacharya, A. Ćustić, A. Rafiey, A. Rafiey, and V. Sokol. Approximation algorithms for generalized MST and TSP in grid clusters. In *COCOA*, pages 110–125. 2015.

[4] N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. Technical Report 88, Management Sciences Research Group, Carnegie-Mellon University, Pittsburgh, PA, 1976.

[5] G. Cornuejols and W. Pulleyblank. A matching problem with side conditions. *Discrete Mathematics*, 29(2):135–159, 1980.

[6] A. Darmann, U. Pferschy, J. Schauer, and G. J. Woeginger. Paths, trees and matchings under disjunctive constraints. *Discrete Applied Mathematics*, 159(16):1726–1735, 2011.

[7] M. E. Dyer and A. M. Frieze. On the complexity of partitioning graphs into connected subgraphs. *Discrete Applied Mathematics*, 10(2):139–153, 1985.

[8] H. N. Gabow, S. N. Maheshwari, and L. J. Osterweil. On two problems in the generation of program test paths. *IEEE Transactions on Software Engineering*, (3):227–231, 1976.

[9] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

[10] P. Hell and D. G. Kirkpatrick. Packings by cliques and by finite families of graphs. *Discrete Mathematics*, 49(1):45–59, 1984.

[11] A. O. Ivanov and A. A. Tuzhilin. The Steiner ratio Gilbert–Pollak conjecture is still open. *Algorithmica*, 62(1-2):630–632, 2012.

[12] M. M. Kanté, C. Laforest, and B. Momege. Trees in graphs with conflict edges or forbidden transitions. In *TAMC*, pages 343–354. Springer, 2013.

[13] M. Karpinski, M. Lampis, and R. Schmied. New inapproximability bounds for TSP. *Journal of Computer and System Sciences*, 81(8):1665–1677, 2015.

[14] M. Lampis. Improved inapproximability for TSP. In *APPROX/RANDOM*, pages 243–253. Springer, 2012.

[15] Y.-S. Myung, C.-H. Lee, and D.-W. Tcha. On the generalized minimum spanning tree problem. *Networks*, 26(4):231–241, 1995.

[16] T. Öncan, R. Zhang, and A. P. Punnen. The minimum cost perfect matching problem with conflict pair constraints. *Computers & Operations Research*, 40(4):920–930, 2013.

[17] P. C. Pop. New models of the generalized minimum spanning tree problem. *Journal of Mathematical Modelling and Algorithms*, 3(2):153–166, 2004.

[18] R. Zhang, S. N. Kabadi, and A. P. Punnen. The minimum spanning tree problem with conflict constraints and its variations. *Discrete Optimization*, 8(2):191–205, 2011.

Optimal Solutions for a Geometric Knapsack Problem using Integer Programming*

Rafael G. Cano† Cid C. de Souza† Pedro J. de Rezende†

Abstract

The objective of this paper is to present an experimental study of the Geometric Knapsack Problem (GKP) with the goal of obtaining provably optimal solutions. We introduce an Integer Linear Programming model for the GKP and apply it to hundreds of instances of two classes: one comprised of uniformly generated points with randomly assigned values; and another composed of convex layered points with value distribution biased towards concentrating negative-valued points on the innermost layers. Trial tests were used to guide the choice of input parameters so as to avoid generating trivial instances. Our experiments show that the layered class is significantly harder to be solved to optimality, in practice, since even instances with as few as 35 points could not be solved within 5 minutes of CPU time.

1 Introduction

Geometric knapsack problems (GKP) are extensions of the classic knapsack problem to a geometric setting. In the classic version, we are given a set of items with specified *weights* and *values*, together with a knapsack of limited capacity. The objective is to select a subset of items whose combined weight does not exceed the capacity of the knapsack and whose total value is maximum.

The geometric variants typically consist of the so-called *fence enclosure problems* [1]. Here, we restrict our study to a two-dimensional version in which items are points in the plane. Consider a set $P = \{p_1, p_2, \ldots, p_n\}$ of n distinct points. With each point $p_i \in P$ there is an associated real value v_i, unrestricted in sign. The "knapsack" (also called *fence*) consists of a simple polygon, and the selected items are the points that it encloses. Unlike the classic variant, items do not have an explicit weight. However, there is a cost associated with the total length of the fence. The objective is to maximize the *net profit* given by the total value of the enclosed points minus the cost of the fence. An example is shown in Figure 1.

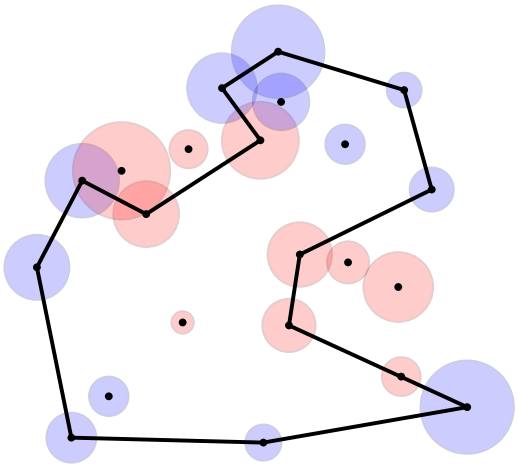


Figure 1: An instance of the GKP with an optimal fence. Positive and negative point values are represented by blue and red circles, resp., with radii proportional to the magnitude of the values.

Formally, we say that a simple polygon φ (a fence) *strictly encloses* all the points in P that lie in the interior of the region bounded by φ. On the other hand, among the points that lie *on* the polygon boundary, φ also *encloses* those of non-negative value. The reason for this apparent asymmetry is that, if we only take into account the points that are strictly enclosed by the fence, then an exact solution, such as the one shown in Figure 1, would not be attainable. Nonetheless, our extended definition of enclosure allows us to compute a polygon that realizes the supremum of the net profit function. Moreover, it is always possible to slightly alter the given polygon in such a way that it strictly encloses all positive-valued points lying on its boundary. This modification would increase the length of the fence by some $\varepsilon > 0$, but the resulting additional cost could be made as small as desired.

We denote by P_φ the set of all points enclosed by φ. Let L_φ be the total Euclidean perimeter of φ, and $c \geq 0$ be a construction cost per unit of length of the fence. The GKP requires a fence φ to be built, which maximizes the net profit $\left(\sum_{p_i \in P_\varphi} v_i\right) - c \cdot L_\varphi$.

*This work was supported by grants from Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) #304727/2014-8, #309627/2017-6, Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) #2014/12236-1, #2018/11100-0, and Fundo de Apoio ao Ensino, à Pesquisa e Extensão (FAEPEX).

†Institute of Computing, University of Campinas, Campinas – Brazil, {rgcano,cid,rezende}@ic.unicamp.br

1.1 Related Work

The GKP was proposed by Arkin et al. [1], who introduced its many variants. In their work, they allow the set of items to consist of either points, line segments or simple polygons. They also consider versions in which there is an upper bound on the total length of fence available. For point items, they show that the problem is NP-hard if values are unrestricted in sign (which is precisely the variant that we address here). However, if all values are non-negative, they describe two exact $O(n^3)$ algorithms. Moreover, if there is limited length of fence to use, the problem of maximizing the total value of the enclosed points is NP-hard even with non-negative values. For polygonal items, they present different polynomial-time algorithms, depending on whether the fence is allowed to cross the objects.

Our version of the GKP is also related to the following problem. Given two sets of points R (the "red points") and B (the "blue points"), we say that a *separating polygon* for R and B is a simple polygon φ such that all points of R are in the interior or on the boundary of φ, and all points of B are in the exterior or on the boundary of φ (or vice-versa). The Red-Blue Separation Problem (RBSP) consists of finding the minimum-perimeter separating polygon for the given sets of points. This problem was shown to be NP-hard by Eades and Rappaport [5]. Approximation algorithms were proposed by Mata and Mitchell [6] and Arora and Chang [2].

Reinbacher et al. [8] study an enclosure problem in the context of Geographic Information Systems (GIS). They address the problem of computing boundaries to imprecise, verbally-defined regions, such as "Northern Portugal" or "British Midlands". In order to estimate the boundary, they first extract a set of known locations that are generally considered to be inside or outside the desired region. Then, they compute a separating polygon that encloses the inside points and obeys certain geographic criteria.

Finally, our work also benefits from the theory developed for the Traveling Salesman Problem (TSP). In particular, we use some results presented by Balas [3] for the Prize Collecting TSP. In this version, a salesman travels between pairs of cities and collects an amount of prize money at each city that he visits. Contrary to the classic variant, he can choose not to visit a city at the cost of a penalty. The goal is to visit enough cities so as to collect a minimum specified amount of prize money while minimizing travel and penalty costs.

1.2 Our Contribution

In this work, we present a formulation of the GKP as an Integer Linear Program (ILP). This is, to the best of our knowledge, the first exact algorithm for this problem. We use the ILP to obtain provably optimal solutions for instances with up to 40 points. Based on a series of experiments, we also devised a class of instances that are particularly challenging to be solved in practice.

The remainder of the text is organized as follows. Section 2 discusses some properties of optimal solutions. Section 3 presents our ILP. Section 4 describes the instances used in our tests and reports the main results of our experimental evaluation. Some final remarks are provided in Section 5.

2 Structure of Optimal Solutions

Given an instance of the GKP, consider an optimal fence $\varphi*$ that encloses the set of points $P_{\varphi*}$. Clearly, $\varphi*$ must be the minimum-perimeter polygon that separates $P_{\varphi*}$ and $P \setminus P_{\varphi*}$. Thus, properties that apply to optimal solutions of the RBSP also apply to the GKP. In particular, for both problems, all vertices of the constructed polygon must be points of the given set. This is illustrated in Figure 2, where points of P are shown as solid disks. Since vertices r and s are not points of P, a polygon of smaller perimeter can be obtained using the chords $\overline{r'r''}$ and $\overline{s's''}$, for the triangles (r', r, r'') and (s', s'', s) do not contain any points of P.

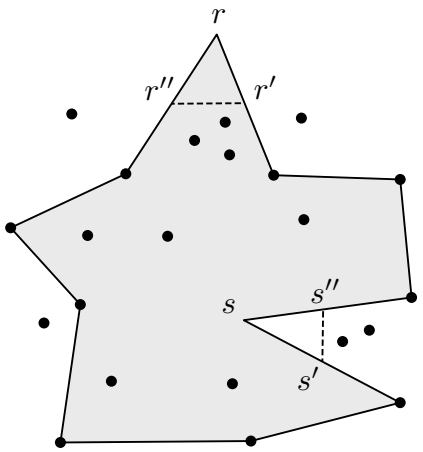


Figure 2: A suboptimal fence with two vertices r and s that are not points of P.

Moreover, it is easy to obtain a stronger result for the GKP. Suppose a convex vertex r is a point of P with negative value. By definition, r is not enclosed by the fence. Thus, the previous perimeter-reducing construction would still maintain the same set of enclosed points and the original fence would not be optimal. Similarly, if $s \in P$ is a reflex vertex of positive value, by the aforementioned construction, a fence of smaller perimeter could be built. Therefore, all convex (reflex) vertices of an optimal fence must be points of P of positive (negative) values.

3 Integer Programming Model

Let $G = (P, A)$ be the complete directed graph whose node set is the set of input points P and $A = \{(p_i, p_j) : p_i, p_j \in P, i \neq j\}$. Given three points $p_i, p_j, p_k \in P$, let $\Delta(p_i, p_j, p_k)$ denote their orientation, i.e., the sign of the signed area of the triangle p_i, p_j, p_k. To simplify our exposition, we assume that the points of P are in general position. Hence, $\Delta(p_i, p_j, p_k) \neq 0$ for any three points $p_i, p_j, p_k \in P$.

In our formulation, we represent the fence as a single directed cycle in G. For each arc $(p_i, p_j) \in A$ we define a binary variable x_{ij} that takes value 1 if and only if this arc is part of the fence. Also, for each point p_i we define a binary variable y_i that indicates whether p_i is enclosed by the fence. Two auxiliary expressions will be used to simplify the description of the constraints. First, note that a point p_i is a vertex of the fence whenever there is an arc incident to p_i. Thus, we define

$$B(p_i) = \sum_{(p_i,p_j)\in A} x_{ij}. \tag{1}$$

Since the solution must consist of a single directed cycle, we include a set of constraints that limit the number of outgoing and incoming arcs on each point to at most one. Thus, $B(p_i)$ may take one of two values: 1 if p_i is on the fence, and 0 otherwise.

In order to model the objective function, it is also necessary to determine which points are enclosed by a fence. Ultimately, we must solve the point location problem w.r.t. an arbitrary simple polygon built from (a subset of) the input points. We apply an approach similar to the following parity checking ray-crossing algorithm. Denote by ℓ_i the horizontal half-line that extends from a point p_i to the positive direction of the x-axis. Given a simple polygon φ, p_i is interior to φ iff ℓ_i crosses (the boundary of) φ an odd number of times.

However, parity checking in a linear model is not a simple task. Nonetheless, arc directions are useful to circumvent this issue. First, we adopt the convention that the cycle representing the fence must be counterclockwise-oriented (we will later show how to enforce this in the model). Now, given a point p_i not on the border of the fence, we trace ℓ_i starting at p_i, as shown in Figure 3. Each time we cross an arc (p_j, p_k), we inspect the value of $\Delta(p_i, p_j, p_k)$. If it is positive, we are moving from the inside to the outside of the fence, otherwise we are moving from the outside to the inside. Let n_i^{out} and n_i^{in} be the number of times we move to the outside and to the inside of the fence, respectively. Clearly, p_i is outside the fence iff $n_i^{\text{out}} = n_i^{\text{in}}$.

Let $R_i \subseteq A$ be the set of arcs crossed by ℓ_i (excluding the intersections at p_i itself, if any). We further define $R_i^+ = \{(p_j, p_k) \in R_i : \Delta(p_i, p_j, p_k) > 0\}$ and $R_i^- = \{(p_j, p_k) \in R_i : \Delta(p_i, p_j, p_k) < 0\}$. From the previous observation, we may write

$$n_i^{\text{out}} = \sum_{(p_j,p_k)\in R_i^+} x_{jk} \qquad \text{and} \qquad n_i^{\text{in}} = \sum_{(p_j,p_k)\in R_i^-} x_{jk}.$$

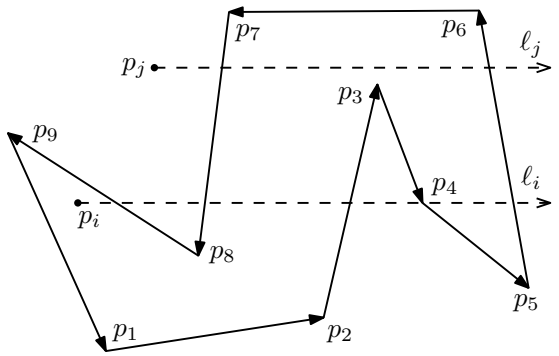


Figure 3: Illustration of the ray-crossing algorithm used to determine which points are enclosed by the fence.

As an example, for the point p_i in Figure 3, all variables in the first summation have value 0, except for $x_{8,9}$, $x_{2,3}$ and $x_{5,6}$, so $n_i^{\text{out}} = 3$. In the second summation, the only variables with value 1 are $x_{7,8}$ and $x_{3,4}$, so $n_i^{\text{in}} = 2$. Note that ℓ_i intersects both (p_3, p_4) and (p_4, p_5) at p_4. Whenever the half-line intersects the fence at a vertex p_k, we only consider that it crosses those arcs whose other endpoint is above p_k. Thus $x_{4,5}$ is not included in the summation. Finally, we define

$$I(p_i) = n_i^{\text{out}} - n_i^{\text{in}}. \tag{2}$$

From the previous discussion, it follows that for any point p_i not on the border of the fence, $I(p_i)$ has value 1 when p_i is inside the fence and 0 otherwise. It should be noted that these values may differ for points lying on the fence (e.g., in Figure 3, $I(p_3) = 1$ and $I(p_5) = 0$). Even in this case $I(p_i)$ can only take values 0 or 1. As we argue in the next section, these cases will not be of importance for our formulation.

It remains for us to show how to enforce a counterclockwise orientation on the constructed cycle. In order to do this, we make use of the observations in Section 2. Since positive-valued points cannot be reflex vertices of an optimal fence, we write constraints to enforce this property. Let $p_i, p_j, p_k \in P$ be three distinct points, with $v_j > 0$ (recall that v_j denotes the value associated with point p_j). If $\Delta(p_i, p_j, p_k) < 0$, then p_j becomes a reflex vertex whenever arcs (p_i, p_j) and (p_j, p_k) are used together in a fence; thus we may write $x_{ij} + x_{jk} \leq 1$. Analogously, if $v_j < 0$ and $\Delta(p_i, p_j, p_k) > 0$, the previous inequality also prevents p_j from becoming a convex vertex. In general, this inequality is valid whenever $v_j \cdot \Delta(p_i, p_j, p_k) < 0$ and it guarantees that all cycles will lead to counterclockwise-oriented polygons.

Given an arc $(p_i, p_j) \in A$, let d_{ij} denote the Euclidean distance between p_i and p_j. Also, given a set of points

$S \subset P$, let $\delta(S)$ denote the set of all arcs of G directed from a point in S to a point in $P \setminus S$, i.e., $\delta(S) = \{(p_i, p_j) \in A : p_i \in S \text{ and } p_j \in P \setminus S\}$.

The following ILP is a formulation of the GKP. The objective function to be maximized is

$$\sum_{p_i \in P} v_i \cdot y_i - \sum_{(p_i,p_j) \in A} c \cdot d_{ij} \cdot x_{ij} \tag{3}$$

subject to the following constraints:

$$\sum_{(p_i,p_j) \in A} x_{ij} \leq 1 \qquad \forall\, p_i \in P \tag{4}$$

$$\sum_{(p_i,p_j) \in A} x_{ij} = \sum_{(p_j,p_i) \in A} x_{ji} \qquad \forall\, p_i \in P \tag{5}$$

$$x_{ij} + x_{ji} \leq 1 \qquad \forall\, p_i, p_j \in P \tag{6}$$

$$x_{ij} + x_{jk} \leq 1 \qquad \forall\, p_i, p_j, p_k \in P : \; v_j \cdot \Delta(p_i, p_j, p_k) < 0 \tag{7}$$

$$y_i \leq I(p_i) + B(p_i) \qquad \forall\, p_i \in P : v_i > 0 \tag{8}$$

$$y_i \geq I(p_i) - B(p_i) \qquad \forall\, p_i \in P : v_i < 0 \tag{9}$$

$$\sum_{(p_i,p_j) \in \delta(S)} x_{ij} \geq B(p_k) + B(p_\ell) - 1 \qquad \forall\, S \subset P,\; 2 \leq |S| \leq n-2,\; p_k \in S, p_\ell \in P \setminus S. \tag{10}$$

The objective function (3) computes the sum of the values of all enclosed points minus the sum of the costs of all arcs in the cycle that represents the fence. Constraints (4) limit the number of outgoing arcs of each node to at most one, and constraints (5) ensure that whenever an arc enters a node, another arc must leave it. Constraints (6) prevent two opposite arcs from being chosen. Constraints (7) guarantee that positive-valued points cannot become reflex vertices of the fence, and negative-valued points cannot become convex vertices.

Constraints (8) and (9) enforce the desired meaning of the y variables. If p_i has positive value, then, due to the maximization of the objective function, y_i will be set to 1, except if some constraint forbids it. Hence, we must only force it to be 0 when p_i is strictly outside the fence. This is done by constraints (8). Similarly, if p_i has negative value, y_i will be set to 0 since we are maximizing (3), unless it violates some constraint. Thus, we must only force it to be 1 when p_i is strictly inside the fence. This is accomplished by constraints (9).

Finally, we must guarantee that a single cycle will be constructed by the model. This is done by constraints (10). These constraints are well-known in the literature of the TSP. The classic TSP requires that a cycle be constructed using all nodes of the input graph (i.e., it must be a Hamiltonian cycle). However, in our case, points can be left out, so we use inequalities that were originally studied by Balas [3] for the Prize Collecting TSP. Those inequalities state that, given a subset S of the input points, if points $p_k \in S$ and $p_\ell \in P \setminus S$ are nodes on a constructed cycle, then there must be at least one arc from S to $P \setminus S$. Although there is an exponential number of them, we used a well-known procedure to separate them in polynomial time using a max-flow min-cut algorithm (see, e.g., Padberg and Rinaldi [7]).

As a final remark, we mentioned earlier that $I(p_i)$ might take value 0 or 1 if p_i is on the fence itself. Note that this variation does not affect the correctness of our model, since constraints (8) and (9) are trivially satisfied when $B(p_i) > 0$.

4 Experiments

We now present our experimental study of the GKP, which had two main goals. Firstly, we wanted to understand what features make an instance particularly hard to solve in practice. This led us to create a challenging set of benchmark instances for exact methods. Secondly, we wished to evaluate the performance of the proposed ILP and to determine which instance sizes can be solved in a reasonable amount of time. More details are given in the next sections.

4.1 Instances

In order to create instances for the GKP, we must decide on two major factors: the spatial distribution of the points and the assignment of their associated values. In this work, we limit ourselves to uniformly distributed points in a square. The magnitudes of point values are also drawn from a uniform distribution and are restricted to a predefined range. We do, however, vary the way in which the sign of each value is chosen, giving rise to two classes of instances.

In the first class, each value is given an equal probability of receiving a positive or a negative sign. We refer to these instances as *uniform instances*. Uniform instances exhibit no particular structure and are useful as a baseline to evaluate the performance of our ILP.

For the second class, we attempt to create more challenging instances based on the following idea. Since the objective function seeks to maximize the net profit, an optimal fence will naturally enclose as many positive-valued points as possible, while avoiding the enclosure of negative ones. Thus, an instance becomes more difficult when negative-valued points are hard to avoid. So, we assign positive values to the outermost points, and gradually increase the probability that a point receives a negative value as we move inwards through the point set.

More formally, to create instances for the second class, we start by computing the convex layers of the set of points. Given a point p_i, denote by d the depth of its layer and by D the depth of the innermost layer. The

probability that p_i receives a negative value is set to $\sqrt{d/D}$ (we consider that the depth of the first layer – the convex hull – is zero). Here, we apply the square root so as to obtain a higher number of negative-valued points, otherwise, the instance loses some of its complexity, as we determined through some preliminary experiments. We refer to instances of this class as *layered instances*.

We conclude this section by providing further details on the creation of our instances. The square from which the n points are drawn has side $100\sqrt{n}$. The magnitude of the values is selected in the range $[50, 150]$. The lower bound of 50 is used because values with very small magnitudes rarely have any significant impact on optimal solutions, so the associated points often become irrelevant. The complete set of instances can be found on our web page [4].

4.2 Computational Results

We executed the experiments on an Intel Xeon E5-2603 1.60GHz CPU with 32GB of RAM. Integer programs were solved with a branch and cut algorithm using CPLEX 12.8 in deterministic mode using a single thread. Our code was written in C++ and compiled with g++ 5.4.0 with optimization flag `-O3`.

Initially, we ran some preliminary experiments to observe the behavior of each class of instances with different values of the cost c per unit length of fence. Note that if c is too high, the optimal fence will enclose a single point of maximum value, and the instance becomes trivial. Similarly, if c is too low, the cost of the fence becomes negligible. In this case, the optimal fence will enclose all (and only) positive-valued points, thus, the problem reduces to the RBSP. To avoid both scenarios, we chose intermediate values for c. For each class, we ran tests with three values that were found to be the most suitable according to the results of our preliminary experiments: $\{0.4, 0.6, 0.8\}$ and $\{0.3, 0.4, 0.5\}$ for uniform and layered instances, respectively.

We created instances of five different sizes: $\{20, 25, 30, 35, 40\}$. For each size, we created 10 instances of each class. Thus, in total, we have 50 uniform and 50 layered instances. We, then, ran our ILP on each one of them with the three specified cost values, yielding a total of 300 test problems. The results are summarized in Tables 1 and 2. For each size n and cost c, we report the minimum (tmin) and maximum (tmax) running times of the 10 instances. We also show the average running time (tavg) together with the standard deviation. We imposed a time limit of 5 minutes for each run, and the last column shows the number of instances solved to optimality within this time bound.

The results for uniform instances (Table 1) indicate that this class does not pose difficulties to our ILP, which was able to find optimal solutions for all instances. Although the average running time is always below one minute, there is a very large deviation, which is, in most cases, larger than the average itself. In several cases, the solver took a long time to find a good feasible solution to the ILP, which led to the processing of a high number of nodes in the branch and bound tree and, consequently, high running times.

c	n	tmin	tmax	tavg	#opt
0.4	20	0.1	0.6	0.3 ± 0.2	10
	25	0.2	11.8	2.2 ± 3.6	10
	30	0.2	16.6	5.4 ± 6.2	10
	35	0.5	57.7	10.8 ± 17.6	10
	40	1.6	201.5	52.6 ± 68.8	10
0.6	20	0.1	1.7	0.3 ± 0.5	10
	25	0.1	6.0	1.0 ± 1.8	10
	30	0.2	10.2	2.7 ± 3.1	10
	35	0.4	43.9	10.0 ± 13.5	10
	40	1.4	157.3	39.3 ± 47.9	10
0.8	20	0.1	0.3	0.1 ± 0.1	10
	25	0.1	1.2	0.3 ± 0.3	10
	30	0.1	1.9	0.8 ± 0.6	10
	35	0.4	54.3	11.0 ± 16.4	10
	40	1.0	178.9	50.7 ± 62.5	10

Table 1: Summary of the results for uniform instances. Times are given in seconds.

c	n	tmin	tmax	tavg	#opt
0.3	20	0.2	1.2	0.6 ± 0.4	10
	25	0.3	52.4	13.5 ± 16.9	10
	30	6.3	231.7	91.5 ± 72.6	10
	35	0.6	300.0	200.0 ± 128.7	5
	40	59.4	300.0	251.0 ± 89.9	3
0.4	20	0.2	3.0	0.9 ± 0.9	10
	25	0.4	63.0	16.0 ± 21.8	10
	30	3.1	120.6	34.4 ± 39.2	10
	35	0.7	300.0	153.1 ± 116.7	8
	40	14.9	300.0	235.0 ± 106.3	4
0.5	20	0.1	2.4	1.1 ± 0.9	10
	25	0.3	38.3	11.0 ± 12.8	10
	30	0.3	237.2	38.4 ± 72.5	10
	35	1.2	300.0	119.4 ± 126.1	9
	40	2.5	300.0	204.9 ± 128.1	4

Table 2: Summary of the results for layered instances. Times are given in seconds.

As for layered instances, the results in Table 2 show that this is, as desired, a much harder class. We were not able to find optimal solutions for several instances with 35 and 40 points. The average running times are also higher in all cases. Therefore, we believe this is a challenging set of instances to serve as a benchmark for the GKP.

In our last experiment, we examined the behavior of individual instances for a wide range of fence cost values.

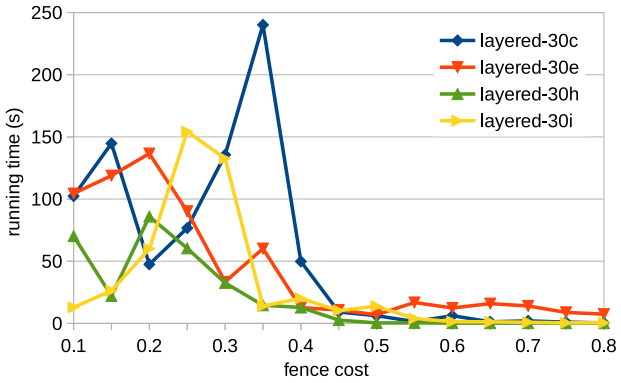


Figure 4: Running time for each value of c for four layered instances with 30 points each.

We selected four layered instances and ran our ILP on them with values of c that range from 0.10 to 0.80 in increments of 0.05. The running times are depicted in Figure 4.

The graph shows that for low values of c, the instances are harder to solve. As we mentioned earlier, when c is too low, the problem reduces to the RBSP, which is also NP-hard. The high variance in the running times is due to CPLEX's branch and bound process, which is, to a certain extent, unpredictable. If the solver cannot find a good solution quickly, a lot of branching is required and the computation time increases. However, for high values of c, the problem becomes easier. This happens because the cost of several edges becomes prohibitive and the solver can easily prune some nodes of the branch and bound tree.

We observed this behavior for all test instances. Although the specific value of c varies among instances, there is always some threshold for which they become trivial. This is illustrated in Figure 5, which shows four optimal solutions for the instance layered-30c, obtained with four different values of c. The rightmost figure shows the shortest optimal non-degenerate fence for that instance. If c is further increased, the optimal solution becomes a degenerate polygon enclosing a single point.

5 Conclusion

We addressed an NP-hard variant of the Geometric Knapsack Problem and proposed an ILP formulation for it. This is, to the best of our knowledge, the first exact method to solve this variant. We also devised two classes of instances to evaluate our ILP. One of them, namely, layered instances, proved to be quite challenging for our formulation, and we propose it as a benchmark for this problem. As for future work, we believe the development of effective heuristics could improve the running times of our ILP, since in several cases, CPLEX struggled to find feasible solutions. Finally, the study of known optimal solutions could also lead to significant speedups, especially if one can develop preprocessing procedures that take advantage of the geometric properties of each instance.

References

[1] E. M. Arkin, S. Khuller, and J. S. Mitchell. Geometric knapsack problems. *Algorithmica*, 10(5):399–427, 1993.

[2] S. Arora and K. Chang. Approximation schemes for degree-restricted MST and red–blue separation problems. *Algorithmica*, 40(3):189–210, 2004.

[3] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[4] R. G. Cano, C. C. de Souza, and P. J. de Rezende. Geometric knapsack – instances and experimental results, 2018. `www.ic.unicamp.br/~cid/Problem-instances/Geometric-Knapsack`.

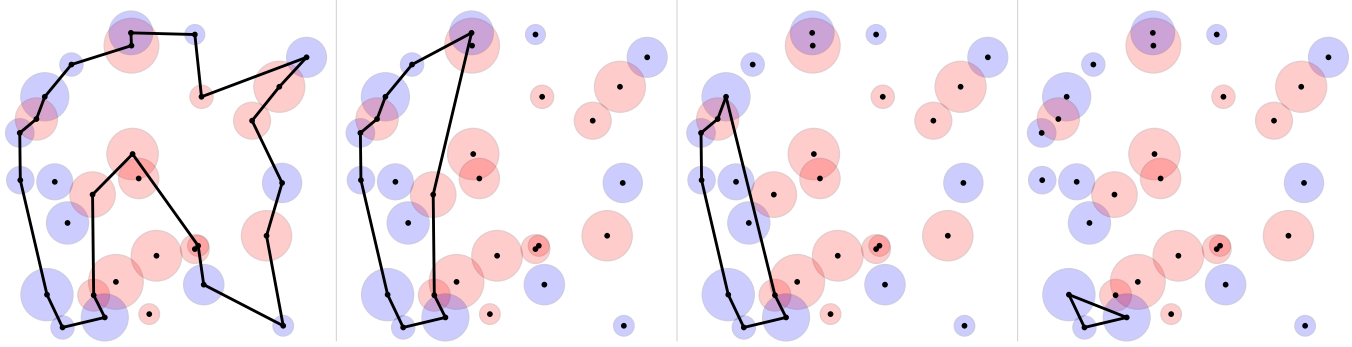


Figure 5: Optimal fences for the instance layered-30c with fence costs of 0.25, 0.40, 0.60 and 0.75, from left to right. Positive and negative point values are represented by blue and red circles, resp., with radii proportional to the magnitude of the values.

[5] P. Eades and D. Rappaport. The complexity of computing minimum separating polygons. *Pattern Recognition Letters*, 14(9):715–718, 1993.

[6] C. S. Mata and J. S. Mitchell. Approximation algorithms for geometric tour and network design problems. In *Proc. of the Eleventh Annual Symposium on Computational Geometry*, SCG '95, pages 360–369. ACM, 1995.

[7] M. Padberg and G. Rinaldi. Facet identification for the symmetric traveling salesman polytope. *Mathematical Programming*, 47(1):219–257, 1990.

[8] I. Reinbacher, M. Benkert, M. van Kreveld, J. S. Mitchell, J. Snoeyink, and A. Wolff. Delineating boundaries for imprecise regions. *Algorithmica*, 50(3):386–414, 2008.

Approximate Data Depth Revisited

David Bremner * Rasoul Shahsavarifar *

Abstract

Halfspace depth and β-skeleton depth are two types of depth functions in nonparametric data analysis. The halfspace depth of a query point $q \in \mathbb{R}^d$ with respect to $S \subset \mathbb{R}^d$ is the minimum portion of the elements of S which are contained in a halfspace which passes through q. For $\beta \geq 1$, the β-skeleton depth of q with respect to S is defined to be the total number of *β-skeleton influence regions* that contain q, where each of these influence regions is the intersection of two hyperballs obtained from a pair of points in S. The β-skeleton depth introduces a family of depth functions that contain *spherical depth* and *lens depth* if $\beta = 1$ and $\beta = 2$, respectively. The main results of this paper include approximating the planar halfspace depth and β-skeleton depth using two different approximation methods. First, the halfspace depth is approximated by the β-skeleton depth values. For this method, two dissimilarity measures based on the concepts of *fitting function* and *Hamming distance* are defined to train the halfspace depth function by the β-skeleton depth values obtained from a given data set. The goodness of this approximation is measured by a function of error values. Secondly, computing the planar β-skeleton depth is reduced to a combination of some range counting problems. Using existing results on range counting approximations, the planar β-skeleton depth of a query point is approximated in $O(n\ poly(1/\varepsilon, \log n))$, $\beta \geq 1$. Regarding the β-skeleton depth functions, it is also proved that this family of depth functions converge when $\beta \to \infty$. Finally, some experimental results are provided to support the proposed method of approximation and convergence of β-skeleton depth functions.

1 Introduction

Data depth is a method to generalize the concept of rank in the univariate data analysis to the multivariate case. Data depth measures the centrality of a data point with respect to a dataset, and it gives a center-outward ordering of data points. In other words, applying a data depth on a dataset generates a partial ordered set (poset) of the data points. A poset is a set together with a partial ordering relation which is reflexive, antisymmetric and transitive. Over the last decades various notions of data depth such as *halfspace depth* (Hotelling [15, 28]; Tukey [30]), *simplicial depth* (Liu [19]) *Oja depth* (Oja [21]), *regression depth* (Rousseeuw and Hubert [22]), and others have been introduced in the area of non-parametric multivariate data analysis. These depth functions are different in application, definition, and complexity of computation. Among the different notions of data depth, we focus on halfspace depth and a recently defined data depth named β-skeleton depth (Yang and Modarres [32]).

In 1975, Tukey generalized the definition of univariate median and defined the halfspace median as a point in which the halfspace depth is maximized, where the halfspace depth is a multivariate measure of centrality of data points. Halfspace depth is also known as Tukey depth or location depth. In general, the halfspace depth of a query point q with respect to a given data set S is the smallest portion of data points that are contained in a closed halfspace through q [6, 30]. The halfspace depth function has various properties such as vanishing at infinity, affine invariance, and decreasing along rays. These properties are proved in [10]. Many different algorithms for the computation of halfspace depth in lower dimensions have been developed elsewhere [6, 7, 8, 24]. The bivariate and trivariate case of halfspace depth can be computed exactly in $O(n \log n)$ and $O(n^2 \log n)$ time [23, 29], respectively. However, computing the halfspace depth of a query point with respect to a data set of size n in dimension d is an NP-hard problem if both n and d are part of the input [16]. Due to the hardness of the problem, designing efficient algorithms to compute (or approximate) the halfspace depth of a point remains an interesting task in the research area of data depth [5, 9]. Some results on ε-approximation of halfspace depth can be found in [1, 3, 14].

In 2017, Yang and Modarres introduced a family of depth functions called β-skeleton depth, indexed by a single parameter $\beta \geq 1$. The β-skeleton depth of a query point $q \in \mathbb{R}^d$ with respect to a given data set S is defined as the portion of β-skeleton influence regions that contain q. The influence regions of β-skeleton depth are the multidimensional generalization of lunes in the definition of the *β-skeleton graph* [17]. A notable characteristic of the β-skeleton depth is related to its

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

time complexity that grows linearly in the dimension d whereas no polynomial algorithms (in the dimension) in higher dimensions are known for most other data depths. To the best of our knowledge, the current best algorithm for computing the β-skeleton depth in higher dimension d is the straightforward algorithm which takes $\Theta(dn^2)$. The authors, in their previous work [26], improved this bound for the planar β-skeleton depth. They developed an $O(n^{3/2+\epsilon})$ algorithm for all values of $\beta \geq 1$, and a $\Theta(n \log n)$ algorithm for the special case of $\beta = 1$. Spherical depth (Elmore, Hettmansperger, and Xuan [11]) and lens depth (Liu and Modarres [20]) can be obtained from β-skeleton depth by considering $\beta = 1$ and $\beta = 2$, respectively. It is known that the β-skeleton depth function is monotonic, maximized at the center, and vanishing at infinity. The β-skeleton depth function is also orthogonally (affinely) invariant if the Euclidean (Mahalanobis) distance is used to construct the β-skeleton influence regions [11, 20, 31, 32].

The concept of data depth is widely studied by statisticians and computational geometers. Some directions that have been considered by researchers include defining new depth functions, improving the complexity of computations, computing both exact and approximate depth values, and computing depth functions in higher dimensions. Two surveys by Aloupis [2] and Small [28] can be referred to as overviews of data depth from a computational geometer's and a statistician's point of view, respectively.

In this paper, different methods are presented to approximate the halfspace and β-skeleton depth functions. Computing the β-skeleton depth is reduced to a combination of range counting problems. Using different range counting approximations in [4, 13, 27], the planar β-skeleton depth ($\beta \geq 1$) of a given point is approximated in $O(n\ poly(1/\varepsilon, \log n))$ query time. Furthermore, we propose a technique to approximate the halfspace depth using the β-skeleton depth. In this method, two dissimilarity measures based on the concepts of *fitting function* and *Hamming distance* are defined to train the halfspace depth function by the β-skeleton depth values obtaining from a given data set. The goodness of approximation can be measured by the sum of square of error values. We also show that β-skeleton depth functions converge when $\beta \to \infty$. Finally, some experimental results are provided regarding our proposed method of approximation.

2 Halfspace Depth

Definition: The halfspace depth of a query point $q \in \mathbb{R}^d$ with respect to a given data set $S = \{x_1, ..., x_n\} \subseteq \mathbb{R}^d$ is defined as the minimum portion of points of S contained in any closed halfspace that has q on its boundary. Using the notation of $HD(q; S)$, the above definition can be presented by (1).

$$HD(q; S) = \frac{2}{n} \min\{|S \cap H| : H \in \mathbb{H}\}, \qquad (1)$$

where $2/n$ is the normalization factor[1], $\mathbb{H}$ is the class of all closed halfspaces in $\mathbb{R}^d$ that pass through q, and $|S \cap H|$ denotes the number of points within the intersection of S and H. As illustrated in Figure 1, $HD(q_1; S) = 6/13$ and $HD(q_2; S) = 0$, where S is a given set of points in the plane and q_1, q_2 are two query points not in S.

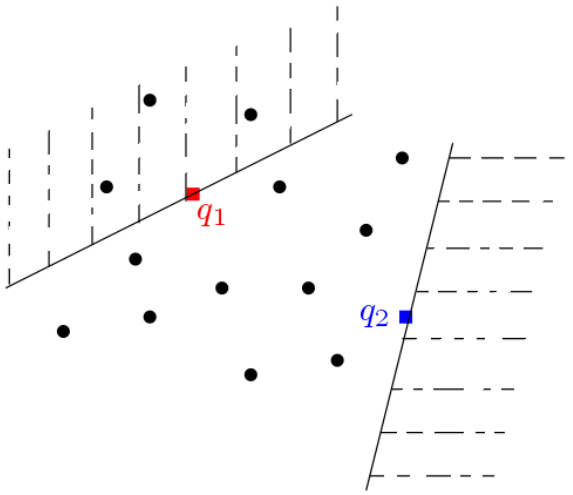


Figure 1: Two examples of halfspace depth in the plane

3 β-skeleton Depth

Definition: For $1 \leq \beta \leq \infty$, the β-skeleton influence region of x_i and x_j ($S_\beta(x_i, x_j)$) is defined as follows:

$$S_\beta(x_i, x_j) = B(c_i, r) \cap B(c_j, r), \qquad (2)$$

where $r = \frac{\beta}{2}\|x_i - x_j\|$, $c_i = \frac{\beta}{2}x_i + (1 - \frac{\beta}{2})x_j$, and $c_j = (1 - \frac{\beta}{2})x_i + \frac{\beta}{2}x_j$.

In the case of $\beta = \infty$, the β-skeleton influence region is well defined, and it is a slab defined by two halfspaces. Figure 2 shows the β-skeleton influence regions for different values of β.

Definition: Let $S = \{x_1, ..., x_n\}$ be a set of points in $\mathbb{R}^d$. For the parameter $1 \leq \beta \leq \infty$, the β-skeleton depth of a query point $q \in \mathbb{R}^d$ with respect to S, is defined as the proportion of the β-skeleton influence regions $S_\beta(x_i, x_j), 1 \leq i < j \leq n$ that contain q. Using the

[1]Instead of the normalization factor $1/n$ which is common in literature, we use the normalization factor $2/n$ in order to let the depth of 1 to be achievable.

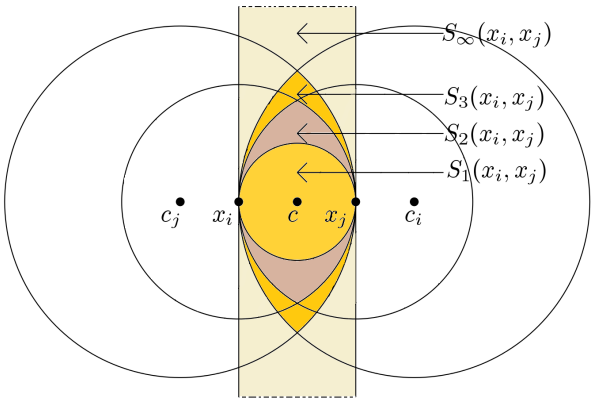


Figure 2: The β-skeleton influence regions defined by x_i and x_j for β=1, 2, 3, and $\beta = \infty$, where $c = \frac{x_i+x_j}{2}$, $c_i = \frac{3}{2}x_i + (1-\frac{3}{2})x_j$, and $c_j = (1-\frac{3}{2})x_i + \frac{3}{2}x_j$

indicator function I, this definition can be represented by Equation (3).

$$SkD_\beta(q;S) = \frac{1}{\binom{n}{2}} \sum_{1\le i<j\le n} I(q \in S_\beta(x_i, x_j)) \qquad (3)$$

It can be verified that $q \in S_\beta(x_i, x_j)$ is equivalent to the inequality of $\frac{\beta}{2}\|x_i - x_j\| \ge \max\{\|q-c_i\|, \|q-c_j\|\}$. The straightforward algorithm for computing the β-skeleton depth of $q \in \mathbb{R}^d$ takes $\Theta(dn^2)$ time because the above inequality should be checked for all $1 \le i, j \le n$.

4 Dissimilarity Measures

In this section, two different types of dissimilarity measures for depth functions are introduced.

4.1 Fitting Functions and Dissimilarity Measures

To determine the dissimilarity between two vectors $U = (u_1, ..., u_n)$ and $V = (v_1, ..., v_n)$, the idea of fitting functions can be applied. Considering the goodness measures of fitting functions, assume that f is the best function fitted to U and V which means that $u_i = f(v_i) \pm \delta_i$. Let $\xi_i = u_i - \overline{U}$, where $\overline{U}$ is the average of u_i $(1 \le i \le n)$. We define the dissimilarity measure between U and V $(d_E(U, V))$ to be a function of δ_i and ξ_i as follows:

$$d_E(U, V) = 1 - r^2, \qquad (4)$$

where r^2 is the coefficient of determination [25] which is defined by:

$$r^2 = \frac{\sum_{i=1}^{n}(\xi_i^2 - \delta_i^2)}{\sum_{i=1}^{n}\xi_i^2}. \qquad (5)$$

Since $r^2 \in [0, 1]$, $d_E(U, V) \in [0, 1]$. A smaller value of $d_E(U, V)$ represents more similarity between U and V.

Definition: Let $S = \{x_1, ..., x_n\}$ be a finite set. It is said that $P = (S, \preceq)$ is a partially ordered set (poset) if $\preceq$ is a partial order relation on S, that is, for all $x_i, x_j, x_t \in S$: (a) $x_i \preceq x_i$; (b) $x_i \preceq x_j$ and $x_j \preceq x_t$ implies that $x_i \preceq x_t$; (c) $x_i \preceq x_j$ and $x_j \preceq x_i$ implies that $x_i \equiv_p x_j$, where $\equiv_p$ is the corresponding equivalency relation.
Poset $P = (S, \preceq)$ is called a chain if any two elements of S are comparable, i.e., given $x_i, x_j \in S$, either $x_i \preceq x_j$ or $x_j \preceq x_i$. If there is no comparable pair among the elements of S, the corresponding post is an anti chain. Figure 3 illustrates different posets with the same elements.

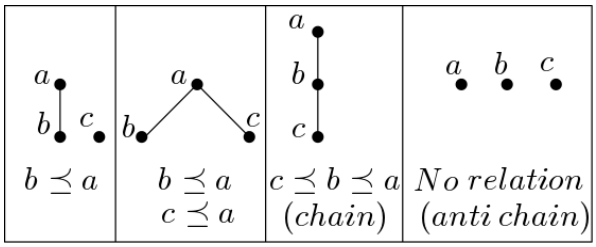


Figure 3: Different posets and relations among their elements

4.2 Dissimilarity Measures Between two Posets

The idea of defining the following distance comes from the proposed structural dissimilarity measure between posets in [12]. Let $\mathbb{P} = \{P_t = (S, \preceq_t) | t \in \mathbb{N}\}$ be a finite set of posets, where $S = \{x_1, ..., x_n\}$. For $P_k \in \mathbb{P}$ we define a matrix $M^k_{n\times n}$ by:

$$M^k_{ij} = \begin{cases} 1 & x_i \preceq_k x_j \\ 0 & \text{otherwise.} \end{cases}$$

We use the notation of $d_c(P_f, P_g)$ to define a dissimilarity between two posets $P_f, P_g \in \mathbb{P}$ as follows:

$$d_c(P_f, P_g) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}|M^f_{ij} - M^g_{ij}|}{n^2 - n} \qquad (6)$$

It can be verified that $d_c(P_f, P_g) \in [0, 1]$, where the closer value to 1 means the less similarity between P_f and P_g. This measure of similarity is a metric on $\mathbb{P}$ because for all $P_f, P_g, P_h \in \mathbb{P}$,

- $d_c(P_f, P_g) \ge 0$
- $d_c(P_f, P_g) = 0 \Leftrightarrow P_f = P_g$

- $d_c(P_f, P_g) = d_c(P_g, P_f)$
- $d_c(P_f, P_h) \leq d_c(P_f, P_g) + d_c(P_g, P_h)$.

Proving these properties is straightforward. The proof of last property which is less trivial can be found in Appendix (see Lemma 3).

5 Approximation of Halfspace Depth

We propose a method, which is different from other approximation methods in the literatures, to approximate the halfspace depth. Motivated by statistical applications and machine learning techniques, we train the halfspace depth function using the values of another depth function. Among all depth functions, the $\beta-$skeleton depth is chosen to approximate the halfspace depth because it is easy to compute and its time complexity, i.e. $O(dn^2)$, grows linearly in higher dimension d.

5.1 Approximation of Halfspace Depth and Fitting Function

Suppose that $S = \{x_1, ..., x_n\}$ is a set of data points. By choosing some subsets of S as training samples, we consider the problem of learning the halfspace depth function using the $\beta-$skeleton depth values. Finally, by applying the cross validation techniques in machine learning, the best function f can be obtained such that $HD(x_i; S) = f(SkD_\beta(x_i; S)) \pm \delta_i$. The function f can be considered as an approximation function for halfspace depth, where the value of $d_E(HD, SkD_\beta)$ that can be computed using Equation (4) is the error of approximation.

5.2 Approximation of Halfspace Depth and Poset Dissimilarity

In some applications, the structural ranking among the elements of S is more important than the depth value of single points. Let $S = \{x_1, ..., x_n\}$ be a set of points and D be a depth function. Applying D on x_i with respect to S generates a poset. In fact, $P_D = (D(x_i; S), \leq)$ is a chain because for every $x_i, x_j \in S$, the values of $D(x_i; S)$ and $D(x_j; S)$ are comparable. For halfspace depth and β-skeleton depth, their dissimilarity measure of rankings can be obtained by Equation (6) as follows:

$$d_c(HD, SkD_\beta) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} |M_{ij}^{HD} - M_{ij}^{SkD_\beta}|}{n^2 - n}.$$

The smaller value of $d_c(HD, SkD_\beta)$, the more similarity between HD and SkD_β in ordering the elements of S.

In sections 5.1 and 5.2, instead of β-skeleton, any other depth function can be considered to approximate halfspace depth. Considering any other depth function, we can compute the goodness of approximation using dissimilarity measures d_E and d_c.

Conjecture 1 *For two depth functions D_1 and D_2, the small value of $d_c(D_1, D_2)$ implies the small value of $d_E(D_1, D_2)$ and vice versa.*

6 Approximation of β-skeleton Depth

The convergence of β-skeleton depth functions when $\beta \to \infty$ is investigated in this section. Furthermore, the problem of planar β-skeleton depth is reduced to a combination of disk and halfspace range counting problems. This reduction is applied to approximate the planar β-skeleton depth using range counting approximations.

6.1 Convergence of β-skeleton Depth Functions

The following theorem helps understand the definition of SkD_∞ in Equation (3).

Theorem 1 *If $\beta \to \infty$, all β-skeleton depth functions converge to SkD_∞. In other words, for data set $S = \{x_1, ..., x_n\}$ and query point q,*

$$\lim_{\beta\to\infty} \frac{SkD_\beta(q; S)}{SkD_{(\beta+1)}(q; S)} = 1. \tag{7}$$

Proof. Referring to (3), the definition of β-skeleton depth, it is enough to prove that if $\beta \to \infty$,

$$\forall x_i, x_j \in S;\ S_\beta(x_i, x_j) = S_{(\beta+1)}(x_i, x_j). \tag{8}$$

It is proved that $S_\beta(x_i, x_j) \subseteq S_{(\beta+1)}(x_i, x_j)$, and $u_\beta = u_{(\beta+1)}$ if $\beta \to \infty$ (see Lemma 4 and 5 in Appendix), where u_β is a fixed intersection point of $B(c_i, r)$ and $B(c_j, r)$ (see figure 4). Since the influence regions of β-skeleton depth functions are closed and convex, the proof is complete if

$$\lim_{\beta\to\infty} \frac{A(S_{\beta ij})}{A(S_{(\beta+1)ij})} = 1, \tag{9}$$

where $A(S_{\beta ij})$ is the area of $S_\beta(x_i, x_j)$. It can be verified that $A(S_{\beta ij})$ is equal to $\theta r^2 - da$, where $l = d(x_i, x_j)$, $a = (l/2)\sqrt{(2\beta - 1)}$, $r = \beta l/2$, $\theta = \cos^{-1}((\beta - 1)/\beta)$, and $d = (\beta - 1)l$. See Figure 4.

$$\frac{A(S_{\beta ij})}{A(S_{(\beta+1)ij})} = \frac{\beta^2 \cos^{-1}[(\beta - 1)/\beta] - (\beta - 1)\sqrt{2\beta - 1}}{(\beta + 1)^2 \cos^{-1}[\beta/(\beta + 1)] - \beta\sqrt{2\beta + 1}}$$

Equation (9) is proved because

$$\lim_{\beta\to\infty} \cos^{-1}\left(\frac{\beta - 1}{\beta}\right) = \lim_{\beta\to\infty} \cos^{-1}\left(\frac{\beta}{\beta + 1}\right) = 0.$$

□

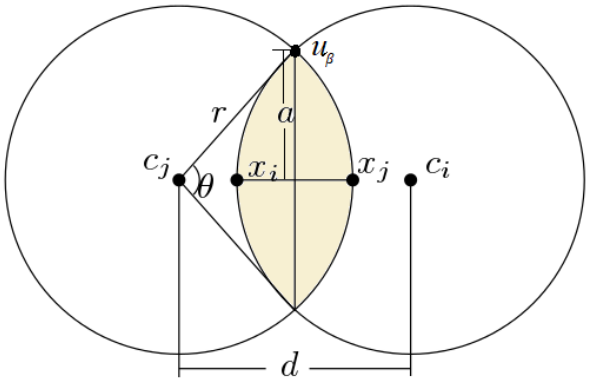


Figure 4: An illustration of d, a, r, l, and θ.

6.2 $\varepsilon-$approximation of Planar β-skeleton Depth

In this section, we prove that for $S = \{x_1, ..., x_n\} \subset \mathbb{R}^2$ and $q \in \mathbb{R}^2$, computing $SkD_\beta(q; S)$ is equivalent to at most $3n$ semialgebraic range counting problems. This result can be applied to approximate the planar β-skeleton depth in $O(n\ poly(1/\varepsilon, \log n))$ query time with $O(n\ poly(1/\varepsilon, \log n))$ storage and the same bound for preprocessing time.

Given a set $S = \{x_1, ..., x_n\} \subset \mathbb{R}^2$, a semialgebraic range τ with constant description complexity, and a parameter $\varepsilon > 0$, S can be preprocessed into a data structure that helps to efficiently compute an approximated count η_τ which satisfies the (ε, τ)-*approximation* given by:

$$(1 - \varepsilon)|S \cap \tau| \leq \eta_\tau \leq (1 + \varepsilon)|S \cap \tau|.$$

Considering S and ε as introduced above, a short list of recent results for approximation of some semialgebraic range counting problems in $\mathbb{R}^2$ is as follows. To obtain an ε-approximation of β-skeleton depth, we use these results as black boxes.

- **Halfspace:** With $O(npoly(1/\varepsilon, \log n))$ preprocessing time, one can construct a data structure of size $O(npoly(1/\varepsilon, \log n))$ such that for a given halfspace $\hbar$, it outputs a number $\eta_\hbar$ that satisfies the $(\varepsilon, \hbar)$-approximation with query time $O(poly(1/\varepsilon, \log n))$. For points in $\mathbb{R}^3$, the same results can be obtained [13].
- **Disk:** By standard lifting of planar points to the paraboloid in $\mathbb{R}^3$, one can reduce a disk range query in the plane to a halfspace range query in three dimensions [13].
- **Circular cap:** A circular cap is the larger part of a circle cut by a line. In near linear time, S can be preprocessed into a linear size data structure that returns the emptiness result of any circular cap query in $O(poly \log n)$. With $O(\varepsilon^{-2}T(n) \log n)$ preprocessing time and $O(\varepsilon^{-2}S(n) \log n)$ storage, one can construct a data structure such that for a given circular cap ρ, it returns in $O(\varepsilon^{-2}Q(n) \log n)$ time, a number η_ρ that satisfies the (ε, ρ)-approximation. Both of $T(n)$ and $S(n)$ are near linear and $Q(n)$ is the time for an emptiness query of ρ [27].

Definition: For an arbitrary non-zero point $a \in \mathbb{R}^2$ and parameter $\beta \geq 1$, $\ell(p)$ is a line that is perpendicular to $\overrightarrow{a}$ at the point $p = p(a, \beta) = (\beta - 1)a/\beta$. This line forms two halfspaces $H_o(p)$ and $H_a(p)$. The one that includes the origin is $H_o(p)$ and the other one that includes a is $H_a(p)$.

Definition: For a disk $B(c, r)$ with center $c = c(a, \beta) = \beta a/(2(\beta - 1))$ and radius $r = \|c\|$, $B_o(c, r)$ is the intersection of $H_o(p)$ and $B(c, r)$, and $B_a(c, r)$ is the intersection of $H_a(p)$ and $B(c, r)$, where $\beta > 1$ and a is an arbitrary non-zero point in $\mathbb{R}^2$.

Figure 5 is an illustration of these definitions for different values of parameter β.

Theorem 2 *For arbitrary non-zero points a, b in $\mathbb{R}^2$ and parameter $\beta > 1$, $b \in H_o(p) \setminus \{intB_o(c, r)\}$ if and only if the origin $O = (0, 0)$ is contained in $S_\beta(a, b)$, where $c = \beta a/(2(\beta - 1))$, $r = \|c\|$, and $p = (\beta - 1)a/\beta$.*

Proof. First, we show that $B_o(c, r)$ is a well-defined set meaning that $\ell(p)$ intersects $B(c, r)$. We compute $d(c, \ell(p))$, the distance of c from $\ell(p)$, and prove that this value is not greater than r. It can be verified that $d(c, \ell(p)) = d(c, p)$. Let $k = \beta/(2(\beta - 1))$; the following calculations complete this part of the proof.

$$\begin{aligned} d(c, p) &= d\left(\frac{\beta a}{2(\beta - 1)}, \frac{(\beta - 1)a}{\beta}\right) = d\left(ka, \frac{1}{2k}a\right) \\ &= \left(k - \frac{1}{2k}\right)\sqrt{({a_x}^2 + {a_y}^2)} = \left(\frac{2k^2 - 1}{2k}\right)\|a\| \\ &\leq \frac{2k^2}{2k}\|a\| = k\|a\| = r \end{aligned}$$

We recall the definition of β-influence region given by $S_\beta(a, b) = B(c_a, \frac{\beta}{2}\|a - b\|) \cap B(c_b, \frac{\beta}{2}\|a - b\|)$, where $c_a = \frac{\beta}{2}a + (1 - \frac{\beta}{2})b$ and $c_b = \frac{\beta}{2}b + (1 - \frac{\beta}{2})a$. Using this definition, the following equivalencies can be derived from $O \in S_\beta(a, b)$.

$$O \in S_\beta(a, b) \Leftrightarrow \frac{\beta\|a - b\|}{2} \geq max\{\|c_a\|, \|c_b\|\} \Leftrightarrow$$

$$\beta\|a - b\| \geq max\{\|\beta(a - b) + 2b\|, \|\beta(b - a) + 2a\|\} \Leftrightarrow$$

$$\beta^2\|a - b\|^2 \geq max\{\|\beta(a - b) + 2b\|^2, \|\beta(b - a) + 2a\|^2\}$$

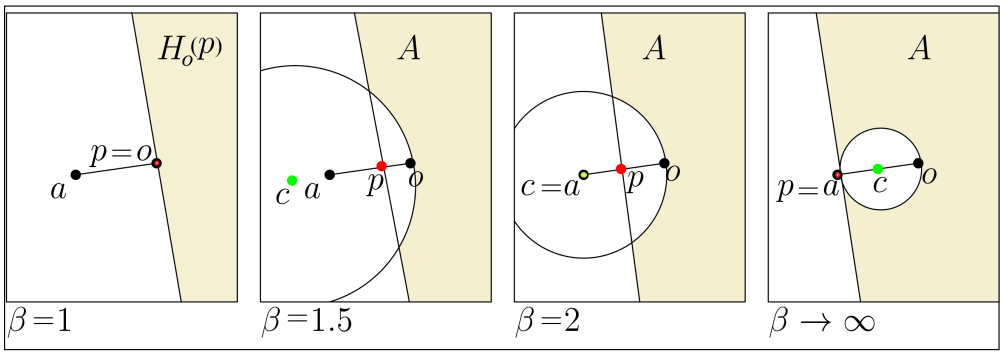


Figure 5: The $H_o(p)$ and $B(c,r)$ defined by $a \in \mathbb{R}^2$ for $\beta = 1,\ 1.5,\ 2,$ and $\beta \to \infty$, where $A = H_o(p) \setminus \{intB_o(c,r)\}$

$$\Leftrightarrow 0 \geq max\{b^2(1-\beta) + \beta \overrightarrow{a}.\overrightarrow{b}, a^2(1-\beta) + \beta \overrightarrow{a}.\overrightarrow{b}\}.$$

By solving these inequalities for $(\beta-1)/\beta$ which is equal to $1/2k$, we have:

$$\frac{1}{2k} \geq max\left\{\frac{\overrightarrow{a}.\overrightarrow{b}}{\|a\|^2}, \frac{\overrightarrow{a}.\overrightarrow{b}}{\|b\|^2}\right\}. \tag{10}$$

For a fixed point a, the inequalities in Equation (10) determine one halfspace and one disk given by (11) and (12), respectively.

$$\frac{1}{2k} \geq \frac{\overrightarrow{a}.\overrightarrow{b}}{\|a\|^2} \Leftrightarrow \overrightarrow{a}.\overrightarrow{b} \leq \frac{1}{2k}\|a\|^2. \tag{11}$$

$$\begin{aligned} &\frac{1}{2k} \geq \frac{\overrightarrow{a}.\overrightarrow{b}}{\|b\|^2} \Leftrightarrow b^2 - 2k\overrightarrow{a}.\overrightarrow{b} \geq 0 \\ &\Leftrightarrow b^2 - 2k\overrightarrow{a}.\overrightarrow{b} + k^2a^2 \geq k^2a^2 \\ &\Leftrightarrow (b-ka)^2 \geq (k\|a\|)^2. \end{aligned} \tag{12}$$

The proof is complete because for a point a, the set of all points b contained in the feasible region defined by Equations (11) and (12) is equal to $H_o(p) \setminus \{intB_o(c,r)\}$. □

Note 1: It can be verified that for the value of $\beta = 2+\sqrt{2}$, the given halfspace in (11) passes through the center of the given disk in (12). See Lemma 6 in Appendix.

Note 2: The range counting approximations work after translating the points of S by $-q$.

For $S = \{x_1, ..., x_n\} \subset \mathbb{R}^2$ and $x_i \in S$, let $\hbar(x_i)$ be the given halfspace by (11), and $B(x_i)$ be the given disk by (12). From Theorem 2, it can be deduced that we need to do

- n halfspace range counting approximations to approximate $SkD_1(q;S)$ because

$$SkD_1(q;S) = \frac{1}{2}\sum_{i=1}^{n}|\hbar(x_i)|.$$

- n halfspace and n disk range counting approximations to approximate $SkD_\infty(q;S)$ because

$$SkD_\infty(q;S) = \frac{1}{2}\sum_{i=1}^{n}(|\hbar(x_i)| - |B(x_i)|).$$

- n halfspace, n disk, and n circular cap range counting approximations to approximate $SkD_\beta(q;S)$, for $1 < \beta < 2+\sqrt{2}$, because

$$SkD_\beta(q;S) = \frac{1}{2}\sum_{i=1}^{n}(|\hbar(x_i)| - |B(x_i)| + |\rho(x_i)|),$$

where $\rho(x_i)$ is the circular cap obtained from $b(x_i)$ cut by $\hbar(x_i)$.

- n halfspace and n circular cap range counting approximations to approximate $SkD_\beta(q;S)$, for $2+\sqrt{2} \leq \beta < \infty$, because

$$SkD_\beta(q;S) = \frac{1}{2}\sum_{i=1}^{n}(|\hbar(x_i)| - |\rho(x_i)|).$$

7 Experimental Results

In this section some experimental results are provided to support Section 5 and Theorem 1. We compute the planar halfspace depth and planar β-skeleton depth of $q \in Q$ with respect to S for different values of β, where Q and S are sets of randomly generated points (with double precision floating point coordinates) of sizes 1000 and 2500, respectively. The results of our experiments are summarized in Table 1 and its corresponding figures provided in Appendix. The columns $2-4$ of Table 1 show that the halfspace depth can be approximated by a quadratic function of the β-skeleton depth with a relatively small value of $d_E(SkD, HD) \cong 0.003$. In particular, $HD \cong 4.3(SkD_\beta)^2 - 2.8(SkD_\beta) + 0.47$ if $\beta \to \infty$. Columns $5-7$ include the experimental results to support Theorem 1.

Acknowledgment: The authors would like to thank Joseph O'Rourke and other attendees of CCCG2017 for inspiring discussions regarding the relationships between depth functions and posets.

References

[1] Peyman Afshani and Timothy M Chan. On approximate range counting and depth. *Discrete & Computational Geometry*, 42(1):3–21, 2009.

[2] Greg Aloupis. Geometric measures of data depth. *DIMACS series in discrete mathematics and theoretical computer science*, 72:147, 2006.

[3] Bagchi, Amitabha and Chaudhary, Amitabh and Eppstein, David and Goodrich, Michael T. Deterministic sampling and range counting in geometric data streams. *ACM Transactions on Algorithms (TALG)*, 3(2), 2007.

[4] Boris Aronov and Sariel Har-Peled. On approximating the depth and related problems. *SIAM Journal on Computing*, 38(3):899–921, 2008.

[5] Boris Aronov and Micha Sharir. Approximate halfspace range counting. *SIAM Journal on Computing*, 39(7):2704–2725, 2010.

[6] David Bremner, Dan Chen, John Iacono, Stefan Langerman, and Pat Morin. Output-sensitive algorithms for tukey depth and related problems. *Statistics and Computing*, 18(3):259–266, 2008.

[7] David Bremner, Komei Fukuda, and Vera Rosta. Primal-dual algorithms for data depth. *DIMACS series in discrete mathematics and theoretical computer science*, 72:147, 2006.

[8] Timothy M Chan. An optimal randomized algorithm for maximum tukey depth. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 430–436. Society for Industrial and Applied Mathematics, 2004.

[9] Dan Chen, Pat Morin, and Uli Wagner. Absolute approximation of tukey depth: Theory and experiments. *Computational Geometry*, 46(5):566–573, 2013.

[10] David L Donoho and Miriam Gasko. Breakdown properties of location estimates based on halfspace depth and projected outlyingness. *The Annals of Statistics*, pages 1803–1827, 1992.

[11] Ryan T Elmore, Thomas P Hettmansperger, and Fengjuan Xuan. Spherical data depth and a multivariate median. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 72:87, 2006.

[12] Marco Fattore, Rosanna Grassi, and Alberto Arcagni. Measuring structural dissimilarity between finite partial orders. In *Multi-indicator Systems and Modelling in Partial Order*, pages 69–84. Springer, 2014.

[13] Sariel Har-Peled. *Geometric approximation algorithms*. Number 173. American Mathematical Soc., 2011.

[14] Sariel Har-Peled and Micha Sharir. Relative (p, ε)-approximations in geometry. *Discrete & Computational Geometry*, 45(3):462–496, 2011.

[15] Harold Hotelling. Stability in competition. In *The Collected Economics Articles of Harold Hotelling*, pages 50–63. Springer, 1990.

[16] David S. Johnson and Franco P Preparata. The densest hemisphere problem. *Theoretical Computer Science*, 6(1):93–107, 1978.

[17] David G Kirkpatrick and John D Radke. A framework for computational morphology. In *Machine Intelligence and Pattern Recognition*, volume 2, pages 217–248. Elsevier, 1985.

[18] Regina Y Liu. *Data depth: robust multivariate analysis, computational geometry, and applications*, volume 72. American Mathematical Soc., 2006.

[19] Regina Y Liu et al. On a notion of data depth based on random simplices. *The Annals of Statistics*, 18(1):405–414, 1990.

[20] Zhenyu Liu and Reza Modarres. Lens data depth and median. *Journal of Nonparametric Statistics*, 23(4):1063–1074, 2011.

[21] Hannu Oja. Descriptive statistics for multivariate distributions. *Statistics & Probability Letters*, 1(6):327–332, 1983.

[22] Peter J Rousseeuw and Mia Hubert. Regression depth. *Journal of the American Statistical Association*, 94(446):388–402, 1999.

[23] Peter J Rousseeuw and Ida Ruts. Algorithm as 307: Bivariate location depth. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 45(4):516–526, 1996.

[24] Peter J Rousseeuw and Anja Struyf. Computing location depth and regression depth in higher dimensions. *Statistics and Computing*, 8(3):193–203, 1998.

[25] Douglas S Shafer and Z Zhang. Beginning statistics. *Phylis-Barnidge publisher*, 304, 2012.

[26] Rasoul Shahsavarifar and David Bremner. Computing the planar β-skeleton depth. *arXiv preprint arXiv:1803.05970*, 2018.

[27] Hayim Shaul. *Range Searching: Emptiness, Reporting, and Approximate Counting*. University of Tel-Aviv, 2011.

[28] Christopher G Small. A survey of multidimensional medians. *International Statistical Review/Revue Internationale de Statistique*, pages 263–277, 1990.

[29] Anja J Struyf and Peter J Rousseeuw. Halfspace depth and regression depth characterize the empirical distribution. *Journal of Multivariate Analysis*, 69(1):135–153, 1999.

[30] John W Tukey. Mathematics and the picturing of data. In *Proceedings of the international congress of mathematicians*, volume 2, pages 523–531, 1975.

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

[32] Mengta Yang and Reza Modarres. β-skeleton depth functions and medians. *Communications in Statistics-Theory and Methods*, pages 1–17, 2017.

Appendix

Lemma 3 *For posets A, B, C in $\mathbb{P} = \{P_t = (S, \preceq_t) | t \in \mathbb{N}\}$, $d_c(A,B) \leq d_c(A,C) + d_c(C,B)$, where*

$$d_c(A,B) = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} |M_{ij}^A - M_{ij}^B|}{n^2 - n}$$

and

$$M_{ij}^k = \begin{cases} 1 & x_i \preceq_k x_j \\ 0 & otherwise. \end{cases}$$

Proof.

$$\begin{aligned} d_c(P_f, P_h) &= \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} |M_{ij}^f - M_{ij}^h|}{n^2 - n} \\ &= \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} |(M_{ij}^f - M_{ij}^g) + (M_{ij}^g - M_{ij}^h)|}{n^2 - n} \\ &\leq \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} (|M_{ij}^f - M_{ij}^g| + |M_{ij}^g - M_{ij}^h|)}{n^2 - n} \\ &= \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} |M_{ij}^f - M_{ij}^g|}{n^2 - n} + \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} |M_{ij}^g - M_{ij}^h|}{n^2 - n} \\ &= d_c(P_f, P_g) + d_c(P_g, P_h). \end{aligned}$$

□

Lemma 4 *For $\beta' > \beta \geq 1$ and $a, b \in \mathbb{R}^2$, $S_\beta(a,b) \subseteq S_{\beta'}(a,b)$, where $S_\beta(a,b)$ is the intersection of two disks $B(C_{ab\beta}, R_{ab\beta})$ and $B(C_{ba\beta}, R_{ba\beta})$, $C_{ab\beta} = (\beta/2)(a-b) + b$, and $R_{ab\beta} = (\beta/2)d(a,b)$.*

Proof. To prove that $B(C_{ab\beta}, R_{ab\beta}) \cap B(C_{ba\beta}, R_{ba\beta})$ is a subset of $B(C_{ab\beta'}, R_{ab\beta'}) \cap B(C_{ba\beta'}, R_{ba\beta'})$, it is enough to prove $B(C_{ab\beta}, R_{ab\beta}) \subseteq B(C_{ab\beta'}, R_{ab\beta'})$ and $B(C_{ba\beta}, R_{ba\beta}) \subseteq B(C_{ba\beta'}, R_{ba\beta'})$. We only prove the first one, and the second one can be proved similarly. Suppose that $\beta < \beta' = \beta + \varepsilon; \varepsilon > 0$. It is trivial to check that two disks $B(C_{ab\beta}, R_{ab\beta})$ and $B(C_{ab\beta'}, R_{ab\beta'})$ meet at b. See Figure 6. Let $t \neq b$ be an extreme point of $B(C_{ab\beta}, R_{ab\beta})$. This means that

$$\begin{aligned} d(t, C_{ab\beta}) = R_{ab\beta} &\Leftrightarrow d(t, \frac{\beta(a-b)}{2} + b) = \frac{\beta(d(a,b))}{2} \\ &\Leftrightarrow \left| \frac{\beta(a-b)}{2} + (b-t) \right|^2 = \left(\frac{\beta(a-b)}{2} \right)^2 \\ &\Leftrightarrow |b-t|^2 - \beta(b-a)\cdot(b-t) = 0 \end{aligned}$$

This means $(b-a)\cdot(b-t) \geq 0$. Hence,

$$\begin{aligned} &|b-t|^2 - \beta(b-a)\cdot(b-t) = 0 \Leftrightarrow \\ &|b-t|^2 - (\beta+\varepsilon)(b-a)\cdot(b-t) < 0 \Leftrightarrow \\ &|b-t|^2 - \beta'(b-a)\cdot(b-t) < 0 \Leftrightarrow \\ &d(t, C_{ab\beta'}) - R_{ab\beta'} < 0 \end{aligned}$$

The last inequality means that t is an interior point of $B(C_{ab\beta'}, R_{ab\beta'})$. □

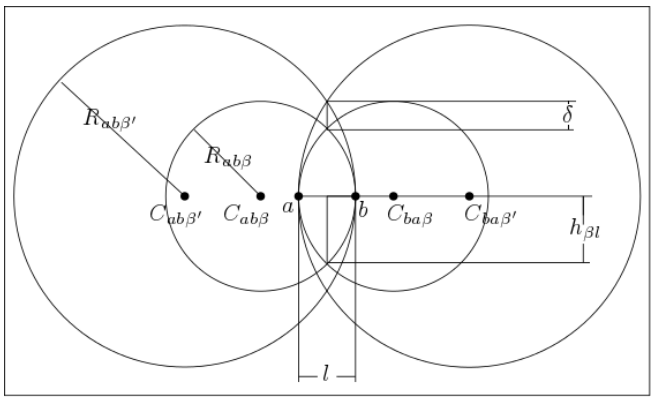


Figure 6: $S_\beta(a,b)$ and $S_{\beta'}(a,b)$.

Lemma 5 *For $a, b \in \mathbb{R}^2$,*

$$\lim_{\beta\to\infty} \delta = \lim_{\beta\to\infty} (h_{(\beta+1)l} - h_{\beta l}) = 0, \tag{13}$$

where $h_{\beta l} = (l/2)\sqrt{(2\beta - 1)}$. See Figure 6.

Proof. *Instead of proving* (13)*, we prove its equivalent form as follows:*

$$\lim_{\beta\to\infty} \frac{h_{(\beta+1)l}}{h_{\beta l}} = \lim_{\beta\to\infty} \frac{(l/2)\sqrt{(2\beta+1)}}{(l/2)\sqrt{(2\beta-1)}} = \lim_{\beta\to\infty} \frac{\sqrt{(2\beta+1)}}{\sqrt{(2\beta-1)}} = 1.$$

□

Lemma 6 *For $k = \beta/(2(\beta-1))$ and $a, b \in \mathbb{R}^2$ (a is fixed and b is arbitrary), halfspace $\overrightarrow{a}.\overrightarrow{b} \leq (1/2k)\|a\|^2$ passes through the center of disk $(b-ka)^2 \geq (k\|a\|)^2$ if $\beta = 2 + \sqrt{2}$.*

Proof. It is enough to substitute b in the given halfspace with ka which is the center of the given disk.

$$\begin{aligned} &\overrightarrow{a}.\overrightarrow{(ka)} \leq \frac{1}{2k}\|a\|^2 \Rightarrow k\|a\|^2 \leq \frac{1}{2k}\|a\|^2 \Rightarrow 2k^2 \leq 1 \Rightarrow \\ &2(\frac{\beta}{2(\beta-1)})^2 \leq 1 \Rightarrow -\beta^2 + 4\beta - 2 \leq 0 \Rightarrow \beta = 2 \pm \sqrt{2} \end{aligned}$$

Since $\beta \geq 1$, the $\beta = 2 + \sqrt{2}$ is valid. □

Table 1: Summary of experimental results

x	$HD = f(x)$	$d_E(x, HD)$	Figure	$f(x, SkD_\infty)$	$d_E(x, SkD_\infty)$	Figure
SkD_1	$3.103x^2 - 0.013x + 0.013$	0.0016	7	$0.942x + 0.288$	0.007	12
SkD_2	$2.71x^2 - 0.48x + 0.04$	0.0019	8	$0.858x + 0.23$	0.002	13
SkD_3	$2.92x^2 - 0.82x + 0.08$	0.0021	9	$0.86x + 0.192$	0.001	14
SkD_{1000}	$4.26x^2 - 2.76x + 0.47$	0.0031	10	$0.999x + 0.002$	0.0	15
SkD_{10000}	$4.27x^2 - 2.78x + 0.47$	0.0031	11	$1.0x + 0.0$	0.0	16

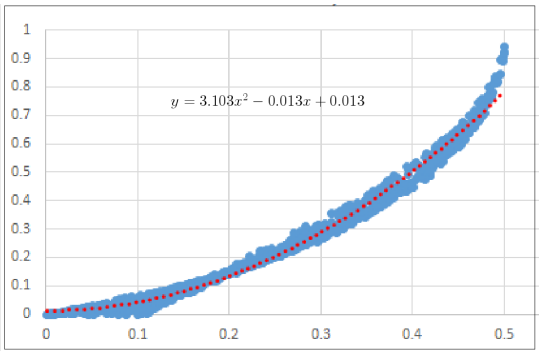


Figure 7: *HD* versus SkD_1.

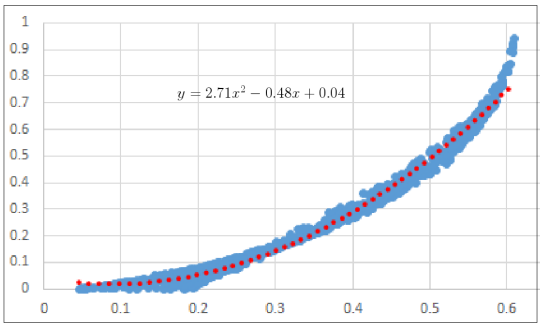


Figure 8: *HD* versus SkD_2.

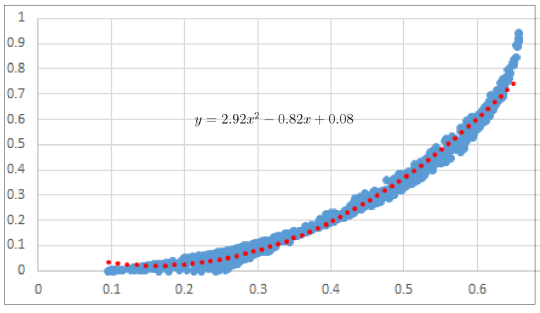


Figure 9: *HD* versus SkD_3.

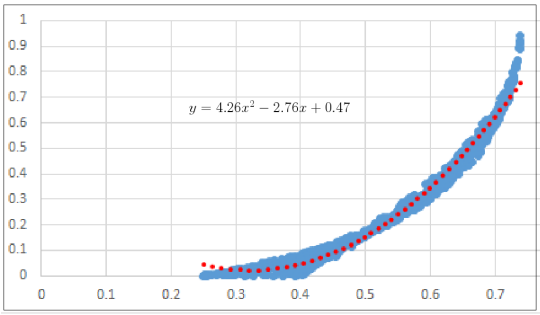


Figure 10: *HD* versus SkD_{1000}.

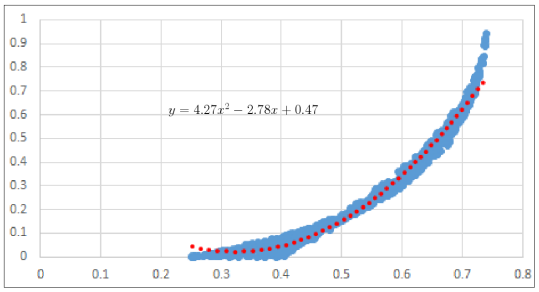


Figure 11: *HD* versus SkD_{10000}.

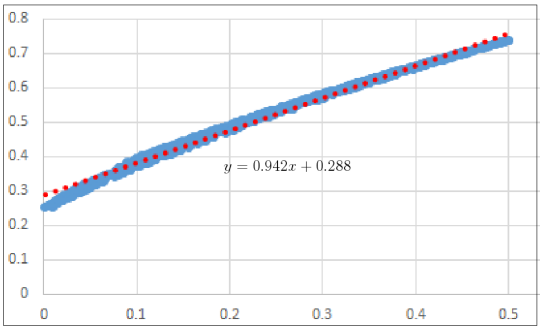


Figure 12: SkD_∞ versus SkD_1.

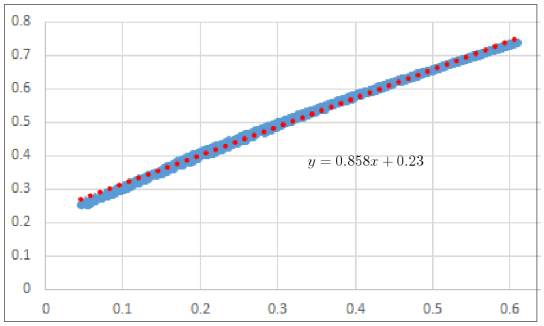


Figure 13: SkD_∞ versus SkD_2.

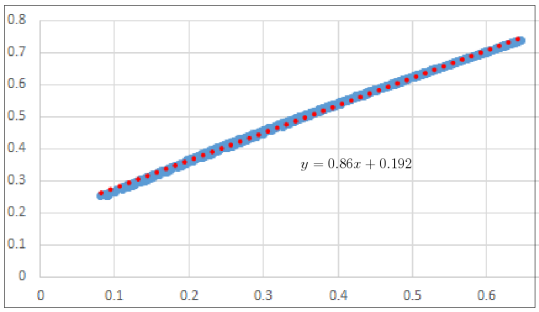


Figure 14: SkD_∞ versus SkD_3.

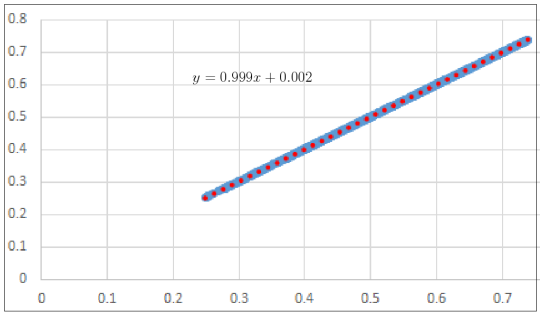


Figure 15: SkD_∞ versus SkD_{1000}.

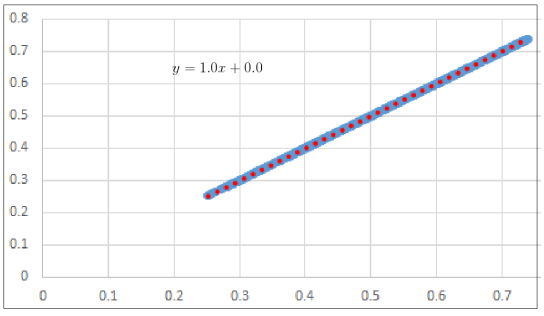


Figure 16: SkD_∞ versus SkD_{10000}.

Approximate range closest-pair search

Jie Xue* Yuan Li† Ravi Janardan ‡

Abstract

The range closest-pair (RCP) problem, as a range-search version of the classical closest-pair problem, aims to store a dataset of points in some data structure such that whenever a query range Q is given, the closest-pair inside Q can be reported efficiently. This paper studies an approximate version of the RCP problem in which the answer pair is allowed to be "approximately" contained in the query range. A general reduction from the approximate RCP problem to the range-minimum and range-reporting problems is given, which works for a general class of query spaces. The reduction is applied to obtain efficient approximate RCP data structures for disk queries in $\mathbb{R}^2$ and ball queries in higher dimensions. Finally, the paper also shows that for orthogonal queries, the approximate RCP problem is (asymptotically) at least as hard as the orthogonal range-minimum problem.

1 Introduction

The closest-pair problem, as one of the most fundamental problems in Computational Geomtry, finds many real-world applications in similarity search and collision detection, etc. In some scenarios, instead of finding the global closest-pair, users are interested in computing the closest-pair inside some specified query range. This results in the so-called *range closest-pair* (RCP) problem, which aims to store a dataset of n points in some data structure such that whenever a query range Q is given, the closest-pair inside Q can be reported efficiently. The RCP problem has been the subject of some recent papers [1, 4, 5, 6, 7, 9].

Unlike most traditional range-search problems, the RCP problem is *non-decomposable*. That is, even if a query range Q can be written as $Q = Q_1 \cup Q_2$, the answer for Q cannot be obtained efficiently from the answers for Q_1 and Q_2. Due to this non-decomposability, many traditional range-search techniques are inapplicable to the RCP problem, which makes the problem quite challenging. Even for very simple query types in $\mathbb{R}^2$ (e.g., quadrants, strips, etc.), the RCP problem is nontrivial. In higher dimensions, it is even not clear how to build efficient data structures for answering RCP queries.

When handling such a difficult range-search problem, approximation could be helpful. In this paper, we study an approximate version of the RCP problem, where the approximation is defined with respect to the query ranges. Specifically, we allow the returned point-pair to be *approximately* (instead of strictly) contained in the query range Q in the sense that one point of the pair can be slightly outside Q but still within a small expansion of Q. For example, consider the disk query in the plane. Given a query disk Q and an approximation factor $\varepsilon > 0$ (which is part of the query), the data structure should return a pair (a, b) of points in the dataset which satisfies the following conditions:
(i) the distance between a and b is at most the distance of the closest-pair in Q.
(ii) $a \in Q$ and $b \in (1+\varepsilon)Q$, where $(1+\varepsilon)Q$ is the disk obtained by expanding Q by a factor $1+\varepsilon$.
Such an approximation can be useful in many real-world applications where the underlying data and/or query is not known precisely anyway. We are interested in how to build efficient data structures for this kind of approximate RCP search.

1.1 Related work

The RCP problem has received attention in recent years [1, 4, 5, 6, 7, 9]. The problem was for the first time introduced in the work [6]. The papers [4, 5, 7] mainly studied the RCP problem in $\mathbb{R}^2$ for orthogonal queries, while the paper [1] considered halfplane queries. Very recently, the previous results were all improved in [9]. In the table below, we summarize the best known bounds for various query types in $\mathbb{R}^2$ (Space refers to the space cost of the data structure and Qtime refers to the query time). In higher dimensions, the RCP problem is quite open. To our best knowledge, even in $\mathbb{R}^3$, no RCP data structure with guaranteed worst-case performance is known currently.

1.2 Our contributions

As mentioned before, in this paper, we study the problem of building efficient data structures for approximate RCP search. Throughout the paper, the query ranges under consideration are always convex bodies (i.e., convex compact subsets) in $\mathbb{R}^d$. Let $\mathcal{Q}$ be a collection of

*University of Minnesota, Twin Cities, xuexx193@umn.edu
†Facebook Inc., lydxlx@fb.com
‡University of Minnesota, Twin Cities, janardan@umn.edu

Query	Source	Space	Qtime
Quadrant	[9]	$O(n)$	$O(\log n)$
Strip	[9]	$O(n \log n)$	$O(\log n)$
Rectangle	[9]	$O(n \log^2 n)$	$O(\log^2 n)$
Halfplane	[9]	$O(n)$	$O(\log n)$

Table 1: Summary of the best known bounds for the RCP problem in $\mathbb{R}^2$.

convex bodies in $\mathbb{R}^d$, called the *query space*. An *approximate* $\mathcal{Q}$-RCP data structure built on a dataset S in $\mathbb{R}^d$ can return, for a specified query (Q, ε) where $Q \in \mathcal{Q}$ is the query range and $\varepsilon > 0$ is the (user-specified) approximation factor, a pair $\phi = (a, b)$ of points in S such that **(i)** $\|b - a\|_2$ is at most the distance of the closest-pair in $S \cap Q$ and **(ii)** $a \in Q$, $b \in (1 + \varepsilon)Q$, where $(1 + \varepsilon)Q$ is the $(1 + \varepsilon)$-expansion of Q (see Section 2 for a precise definition).

Our main contribution is a general reduction from the approximate RCP problem to the range-reporting and range-minimum problems for the same query space (Theorem 1). Our reduction works for any query space $\mathcal{Q}$ (consisting of convex bodies) whose elements have width-diameter ratio lower-bounded by a positive constant (this ratio will be explained Section 2). As concrete applications of the reduction, we obtain efficient approximate RCP data structures for disk queries in $\mathbb{R}^2$ (Corollary 3) and ball queries in higher dimensions (Corollary 4). These query types have not been considered in previous work. Finally, we give a hardness result which shows that, for orthogonal queries, the approximate RCP problem is (asymptotically) at least as hard as the orthogonal range-minimum problem (Theorem 5).

The rest of the paper is organized as follows. Section 2 presents some preliminaries. (We suggest the reader reads this section carefully before moving on.) The general reduction is given in Section 3, while its applications are given in Section 4. In Section 5, we present the hardness result.

2 Preliminaries

Point-pairs and closest-pair. For a pair $\phi = (a, b)$ of points in $\mathbb{R}^d$, the *length* of ϕ, denoted by $|\phi|$, is referred to the distance between a and b, i.e., $|\phi| = \|a-b\|_2$. The *closest-pair* (in a set of points) is the pair of (distinct) points with minimum length. For a point-set S, we denote by $\kappa(S)$ the *closest-pair distance* of S, i.e., the length of the closest-pair in S.

Slabs. A *slab* in $\mathbb{R}^d$ is a closed region bounded by two distinct parallel hyperplanes in $\mathbb{R}^d$. The *thickness* of a slab L, denoted by $\text{thk}(L)$, is the distance between its two bounding hyperplanes. Note that for any slab in $\mathbb{R}^d$, we can always write the equations of its two bounding hyperplanes as $\sum_{i=1}^d a_i x_i + b = -1$ and $\sum_{i=1}^d a_i x_i + b = 1$ for some $a_1, \ldots, a_d, b \in \mathbb{R}$.

Diameter, width, and directional width. Let X be a convex body in $\mathbb{R}^d$, and $\mathcal{L}_X$ be the collection of all minimal (with respect to the partial order of "$\subseteq$") slabs enclosing X. For a unit vector $\mathbf{u}$ in $\mathbb{R}^d$, the *directional width* of X in the direction $\mathbf{u}$, denoted by $\text{wid}_{\mathbf{u}}(X)$, is defined as

$$\text{wid}_{\mathbf{u}}(X) = \sup_{x \in X} \langle \mathbf{u}, x \rangle - \inf_{x \in X} \langle \mathbf{u}, x \rangle,$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product. Equivalently, $\text{wid}_{\mathbf{u}}(X)$ is the thickness of the slab $L \in \mathcal{L}_X$ whose two bounding hyperplanes are perpendicular to $\mathbf{u}$. The *diameter* $\text{diam}(X)$ of X is defined as $\text{diam}(X) = \sup_{\mathbf{u}}(\text{wid}_{\mathbf{u}}(X))$ for $\mathbf{u}$ taken over all unit vectors in $\mathbb{R}^d$, while the *width* $\text{wid}(X)$ of X is defined as $\text{wid}(X) = \inf_{\mathbf{u}}(\text{wid}_{\mathbf{u}}(X))$. Equivalently, we can also define the diameter and width as $\text{diam}(X) = \sup_{L \in \mathcal{L}_X}(\text{thk}(L))$ and $\text{wid}(X) = \inf_{L \in \mathcal{L}_X}(\text{thk}(L))$. The *width-diameter ratio* of X, denoted by $\gamma(X)$, is defined as $\gamma(X) = \text{wid}(X)/\text{diam}(X)$. If $\mathcal{X}$ is a collection of convex bodies in $\mathbb{R}^d$, we define the *width-diameter ratio* of $\mathcal{X}$ as $\gamma(\mathcal{X}) = \inf_{X \in \mathcal{X}} \gamma(X)$.

Expansion of a convex body. In order to introduce our result, we need to formally define what we mean by "expanding" a convex body. Let X be a convex body in $\mathbb{R}^d$. If X is a ball, then expanding X (by a factor $\delta \geq 1$) can be simply defined as scaling X with respect to the center of X by a factor of δ. This definition can be naturally generalized to a general convex body as follows. For a slab L bounded by two hyperplanes $\sum_{i=1}^d a_i x_i + b = -1$ and $\sum_{i=1}^d a_i x_i + b = 1$, we define the δ-*expansion* of L (for $\delta \geq 1$), denoted by δL, as the slab bounded by the two hyperplanes $\sum_{i=1}^d a_i x_i + b = -\delta$ and $\sum_{i=1}^d a_i x_i + b = \delta$. Let $\mathcal{L}_X$ be the collection of all minimal (with respect to the partial order of "$\subseteq$") slabs enclosing X. Then we define the δ-*expansion* of X, denoted by δX, as $\delta X = \bigcap_{L \in \mathcal{L}_X} \delta L$. Under this definition, the 1-expansion of X is X itself (since $X = \bigcap_{L \in \mathcal{L}_X} L$). Furthermore, as one can easily verify, $\text{wid}_{\mathbf{u}}(\delta X) = \delta \text{wid}_{\mathbf{u}}(X)$ for any unit vector $\mathbf{u}$.

3 A general reduction

Let $\mathcal{Q}$ be a collection of convex bodies in $\mathbb{R}^d$. Recall that an *approximate* $\mathcal{Q}$-RCP data structure built on a dataset S in $\mathbb{R}^d$ can return, for a specified query (Q, ε) where $Q \in \mathcal{Q}$ is the query range and $\varepsilon > 0$ is the approximation factor, a pair $\phi = (a, b)$ of points in S such that **(i)** $\|b-a\|_2 \leq \kappa(S \cap Q)$ and **(ii)** $a \in Q$, $b \in (1+\varepsilon)Q$. Note that here ε is specified in the query and needs not to be known beforehand.

Our main result is a reduction from the approximate RCP problem to the range-reporting and range-

minimum problems for the same query space. This reduction works for any query space $\mathcal{Q}$ (consisting of convex bodies) satisfying $\gamma(\mathcal{Q}) > 0$.

Theorem 1 *Let $\mathcal{Q}$ be a fixed collection of convex bodies in $\mathbb{R}^d$ satisfying $\gamma(\mathcal{Q}) > 0$. Given a range-minimum data structure $\mathcal{D}_1$ and a range-reporting data structure $\mathcal{D}_2$ for query space $\mathcal{Q}$, one can build an approximate $\mathcal{Q}$-RCP data structure $\mathcal{D}$ such that*

- *If the space of $\mathcal{D}_1$ is $s_1(n)$ and the space of $\mathcal{D}_2$ is $s_2(n)$, then the space of $\mathcal{D}$ is $O(s_1(n) + s_2(n))$.*
- *If the query time of $\mathcal{D}_1$ is $q_1(n)$ and the query time of $\mathcal{D}_2$ is $q_2(n, k)$ where k is the number of points to be reported, then the query time of $\mathcal{D}$ is $O(q_1(n)+q_2(n, \varepsilon^{-d})+\varepsilon^{-d}\log(1/\varepsilon))$ where ε is the parameter specified in the query.*
- *If the preprocessing time of $\mathcal{D}_1$ is $p_1(n)$ and the preprocessing time of $\mathcal{D}_2$ is $p_2(n)$, then the preprocessing time of $\mathcal{D}$ is $O(p_1(n) + p_2(n) + n \log n)$.*

The rest of this section is dedicated to proving the above result. To this end, we first describe the construction of the desired data structure in Theorem 1, and then analyze its space, query time, and preprocessing time. Let S be the given dataset in $\mathbb{R}^d$ of size n. We want to build an approximate $\mathcal{Q}$-RCP data structure $\mathcal{D}$ on S, given the range-minimum data structure $\mathcal{D}_1$ and the range-reporting data structure $\mathcal{D}_2$.

Data structure. For a point $a \in S$, let $\mathsf{nn}(a) \in S$ denote the nearest neighbor of a in $S\backslash\{a\}$. We associate the information of $\mathsf{nn}(a)$ with the point a for all $a \in S$. Define a weight function $w : S \to \mathbb{R}$ as $w(a) = \|\mathsf{nn}(a) - a\|_2$. This gives us a weighted dataset $\mathcal{S} = (S, w)$. We build on $\mathcal{S}$ the range-minimum data structure $\mathcal{D}_1$. Also, we build on S the range-reporting data structure $\mathcal{D}_2$. Then our approximate $\mathcal{Q}$-RCP data structure $\mathcal{D}$ (built on S) simply consists of $\mathcal{D}_1$ and $\mathcal{D}_2$.

Query algorithm. Let (Q, ε) be a query, where $Q \in \mathcal{Q}$ is the query range and $\varepsilon > 0$ is the approximation factor. Our query algorithm consists of two phases. In the first phase, we use the range-minimum data structure $\mathcal{D}_1$ to find the point $a^* \in S \cap Q$ with the minimum weight. If $w(a^*) \le \varepsilon\mathrm{wid}(Q)$, then we report the pair $(a^*, \mathsf{nn}(a^*))$ and terminate the query process. Otherwise, we proceed to the second phase. In the second phase, we use the range-reporting data structure $\mathcal{D}_2$ to report all the points in $S \cap Q$. Then we simply run the standard divide-and-conquer closest-pair algorithm on $S \cap Q$ to find the closest-pair and report it as the answer.

Correctness. Let ϕ be the answer returned by our query algorithm. If ϕ is reported in the second phase, then it is in fact the closest-pair in $S \cap Q$ and hence our algorithm is clearly correct (as $|\phi| = \kappa(S \cap Q)$ and both points of ϕ are contained in Q). Suppose ϕ is reported in the first phase. Then $\phi = (a^*, \mathsf{nn}(a^*))$ where a^* is the point in $S \cap Q$ with the minimum weight. Assume (s, t) is the closest-pair in $S \cap Q$. We see $\|\mathsf{nn}(a^*) - a^*\|_2 = w(a^*) \le w(s) = \|\mathsf{nn}(s) - s\|_2 \le \|t - s\|_2 = \kappa(S \cap Q)$. Next, we show that $a^* \in Q$ and $\mathsf{nn}(a^*) \in (1 + \varepsilon)Q$, whence the correctness of our algorithm is verified. We have $a^* \in Q$ by the definition of a^*. To see $\mathsf{nn}(a^*) \in (1 + \varepsilon)Q$, let L be any minimal (with respect to the partial order of "$\subseteq$") slab enclosing Q. The thickness $\mathrm{thk}(L)$ of L is at least $\mathrm{wid}(Q)$. Furthermore, we have $\mathrm{dist}(\mathsf{nn}(a^*), L) \le \mathrm{dist}(\mathsf{nn}(a^*), Q) \le \|\mathsf{nn}(a^*) - a^*\|_2 \le \varepsilon\mathrm{wid}(Q) \le \varepsilon\mathrm{thk}(L)$, where $\mathrm{dist}(\mathsf{nn}(a^*), L)$ (resp., $\mathrm{dist}(\mathsf{nn}(a^*), Q)$) denotes the minimum distance between $\mathsf{nn}(a^*)$ and a point in L (resp., Q), which is zero when $\mathsf{nn}(a^*) \in L$ (resp., $\mathsf{nn}(a^*) \in Q$). Hence, we have $\mathsf{nn}(a^*) \in (1 + \varepsilon)L$, which implies $\mathsf{nn}(a^*) \in (1 + \varepsilon)Q$ (as the slab L is arbitrarily chosen). See Figure 1 for an intuitive illustration. Since $\|\mathsf{nn}(a^*)-a^*\|_2 \le \kappa(S\cap Q)$ and $a^* \in Q$, $\mathsf{nn}(a^*) \in (1+\varepsilon)Q$, our algorithm is correct.

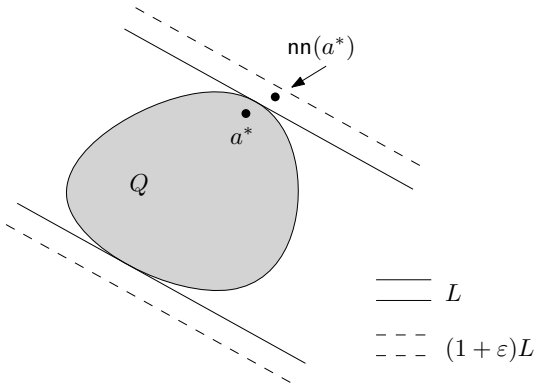


Figure 1: $\mathsf{nn}(a^*) \in (1 + \varepsilon)L$ because $\|\mathsf{nn}(a^*) - a^*\|_2 \le \varepsilon\mathrm{wid}(Q) \le \varepsilon\mathrm{thk}(L)$.

Analysis. We now show that the space, query time, and preprocessing time of our data structure $\mathcal{D}$ satisfy the requirements in Theorem 1. The space of $\mathcal{D}$ is clearly $O(s_1(n) + s_2(n))$, as it just consists of $\mathcal{D}_1$ and $\mathcal{D}_2$. To analyze the query time of $\mathcal{D}$, we observe that there are not too many points reported in the second phase.

Lemma 2 *The number of the points reported in the second phase is bounded by $O(\varepsilon^{-d})$.*

Proof. Recall that in the query algorithm, we proceed to the second phase only if $w(a^*) > \varepsilon\mathrm{wid}(Q)$. Since a^* is the point in $S\cap Q$ with the minimum weight, we have $w(a) > \varepsilon\mathrm{wid}(Q)$ for all $a \in S \cap Q$, i.e., $\|\mathsf{nn}(a) - a\|_2 > \varepsilon\mathrm{wid}(Q)$ for all $a \in S \cap Q$. It follows that $\|b - a\|_2 > \varepsilon\mathrm{wid}(Q)$ for any $a, b \in S \cap Q$ unless $a = b$. Now we can show $|S\cap Q| = O(\varepsilon^{-d})$ using the Pigeonhole Principle. Indeed, $\mathrm{diam}(Q) = \mathrm{wid}(Q)/\gamma(Q) \le \mathrm{wid}(Q)/\gamma(\mathcal{Q})$, hence $\|b - a\|_2 \le \mathrm{diam}(Q) \le \mathrm{wid}(Q)/\gamma(\mathcal{Q})$ for any $a, b \in S\cap Q$. So there exists a hyper-cube of side-length

$\mathrm{wid}(Q)/\gamma(\mathcal{Q})$ that contains $S \cap Q$. Because the pairwise distances of the points in $S \cap Q$ are greater than $\varepsilon \mathrm{wid}(Q)$, we have $|S \cap Q| = O((\gamma(\mathcal{Q}) \cdot \varepsilon)^{-d}) = O(\varepsilon^{-d})$ by the Pigeonhole Principle. (The Pigeonhole Principle implies that a set of points in a hyper-cube with side-length α whose pairwise distances are greater than β has size $O((\alpha/\beta)^d)$.) □

The time cost of the first phase is $O(q_1(n))$. By Lemma 2, the time cost for range reporting in the second phase is $O(q_2(n, \varepsilon^{-d}))$. The standard closest-pair algorithm on m points runs in $O(m \log m)$ time, therefore to compute the closest-pair among the reported points takes $O(\varepsilon^{-d} \log(1/\varepsilon))$ time. Thus, the total query time of $\mathcal{D}$ is $O(q_1(n) + q_2(n, \varepsilon^{-d}) + \varepsilon^{-d} \log(1/\varepsilon))$. Finally, we analyze the preprocessing time of $\mathcal{D}$. To build $\mathcal{D}$, we need to compute $\mathsf{nn}(a)$ for all $a \in S$. This can be done using the well-known all-nearest-neighbor algorithm [3], which takes $O(n \log n)$ time. After all $\mathsf{nn}(a)$ are computed, we build $\mathcal{D}_1$ and $\mathcal{D}_2$ directly. Thus, the overall preprocessing time is $O(p_1(n) + p_2(n) + n \log n)$. This completes the proof of Theorem 1.

Discussion. We now briefly discuss which kinds of concrete query spaces our reduction is applicable to. The condition in Theorem 1 for the query space $\mathcal{Q}$ is $\gamma(\mathcal{Q}) > 0$. If $\mathcal{Q}$ is the collection of all balls in $\mathbb{R}^d$ (e.g., all disks in $\mathbb{R}^2$), then $\gamma(\mathcal{Q}) = 1$, and our reduction is applicable; we will discuss this in detail in the next section. More generally, let C be a convex body with nonempty interior in $\mathbb{R}^d$ called *base shape* (note that $\gamma(C) > 0$ in this case). Define $\mathcal{Q}_C$ as the collection of all convex bodies that can be obtained by applying rotation, isotropic scaling, and translation on the base shape C. Then $\gamma(\mathcal{Q}_C) = \gamma(C) > 0$, and our reduction is applicable. We remark that our reduction is inapplicable to the axis-parallel box query, since $\gamma(\mathcal{B}) = 0$ where $\mathcal{B}$ is the collection of all axis-parallel boxes in $\mathbb{R}^d$ (indeed, the width-diameter ratio of a box can be arbitrarily small). However, if we consider a sub-collection $\mathcal{B}_\eta \subseteq \mathcal{B}$ consisting of the boxes in which the ratio of the length of the shortest edge to the length of the longest edge is at least η (where $\eta > 0$), then $\gamma(\mathcal{B}_\eta) \geq \eta/\sqrt{d} > 0$, and our reduction is applicable.

4 Applications

In this section, we apply our general reduction to some specific query spaces to build efficient approximate RCP data structures.

First, we consider the disk query. Let $\mathcal{O}$ be the collection of all disks in $\mathbb{R}^2$. Clearly $\gamma(\mathcal{O}) = 1$ and thus the reduction in Section 3 applies to the query space $\mathcal{O}$. Therefore, to build an approximate $\mathcal{O}$-RCP data structure, it suffices to have the disk range-minimum and range-reporting data structures. It is well-known that the disk range-minimum (resp., range-reporting) problem can be reduced (via lifting) to the halfspace range-minimum (resp., range-reporting) problem in $\mathbb{R}^3$. Halfspace range-reporting in $\mathbb{R}^3$ can be solved optimally (i.e., with $O(n)$ space, $O(\log n + k)$ query time, and $O(n \log n)$ preprocessing time), using the data structure given in [2]. Note that this data structure can also be used to answer halfspace range-emptiness queries in $O(\log n)$ time (i.e., decide whether a given halfspace contains no points in the dataset). By taking advantage of this range-emptiness data structure, we can easily build in $O(n \log^2 n)$ time a halfspace range-minimum data structure in $\mathbb{R}^3$ with $O(n \log n)$ space and $O(\log^2 n)$ query time; we defer the details to Section 4.1. As such, Theorem 1 implies the following corollary.

Corollary 3 *There exists an approximate $\mathcal{O}$-RCP data structure with $O(n \log n)$ space and $O(\log^2 n + \varepsilon^{-2} \log(1/\varepsilon))$ query time, which can be built in $O(n \log^2 n)$ time.*

More generally, we consider the ball query in $\mathbb{R}^d$. Let $\mathcal{O}_d$ be the collection of all disks in $\mathbb{R}^d$ where $d \geq 3$. Again, we have $\gamma(\mathcal{O}_d) = 1$ and thus the reduction in Section 3 works. Similar to the disk case, the range-minimum (resp., range-reporting) problem for query space $\mathcal{O}_d$ can be reduced to the halfspace range-minimum (resp., range-reporting) problem in $\mathbb{R}^{d+1}$. By reducing the halfspace range-minimum problem to the halfspace range-emptiness problem, we can obtain a halfspace range-minimum data structure in $\mathbb{R}^{d+1}$ with $O(n^{\lceil d/2 \rceil})$ space and $O(\log^2 n)$ query time; we defer the details to Section 4.1. The halfspace range-reporting in $\mathbb{R}^{d+1}$ can be solved with $O(n^{\lceil d/2 \rceil} \log^c n)$ space and $O(\log n + k)$ query time [8], where c is a sufficiently large constant. Therefore, Theorem 1 implies the following corollary.

Corollary 4 *There exists an approximate $\mathcal{O}_d$-RCP data structure with $O(n^{\lceil d/2 \rceil} \log^c n)$ space and $O(\log^2 n + \varepsilon^{-d} \log(1/\varepsilon))$ query time.*

While our reduction can be applied to obtain efficient approximate RCP data structures for disk and ball queries, it is unfortunately inapplicable to orthogonal queries (i.e., axis-parallel box queries). In Section 5, we will consider the approximate RCP problem for orthogonal queries and show that it is (asymptotically) at least as hard as the orthogonal range-minimum problem.

4.1 Halfspace range-minimum data structures

We show how to solve the halfspace range-minimum problem via halfspace range-emptiness queries. Suppose there is a halfspace range-emptiness data structure $\mathcal{D}_0$ in $\mathbb{R}^d$, whose space is $s_0(n)$, query time is $q_0(n)$, and preprocessing time is $p_0(n)$. We build a halfspace range-minimum data structure $\mathcal{D}$ in $\mathbb{R}^d$ as follows.

Let $\mathcal{S} = (S, w)$ be a weighted dataset in $\mathbb{R}^d$. Assume $S = \{a_1, \ldots, a_n\}$ where $w(a_1) \leq \cdots \leq w(a_n)$. The data structure $\mathcal{D}$ built on $\mathcal{S}$, denoted by $\mathcal{D}(\mathcal{S})$, is constructed recursively. If $n = 1$, then $\mathcal{D}(\mathcal{S})$ is the trivial data structure. Otherwise, let $S_1 = \{a_1, \ldots, a_{n/2}\}$ and $S_2 = \{a_{n/2+1}, \ldots, a_n\}$. We recursively build $\mathcal{D}(\mathcal{S}_1)$ and $\mathcal{D}(\mathcal{S}_2)$, where $\mathcal{S}_i = (S_i, w_{|S_i})$ ($w_{|S_i}$ denotes the restriction of the weight function w to S_i). Furthermore, we build the halfspace range-emptiness data structure $\mathcal{D}_0(S_1)$. Then $\mathcal{D}(\mathcal{S})$ is the combination of $\mathcal{D}(\mathcal{S}_1)$, $\mathcal{D}(\mathcal{S}_2)$, and $\mathcal{D}_0(S_1)$. If we write the space of $\mathcal{D}$ as $s(n)$ and the preprocessing time (excluding the time for sorting the points by their weights) of $\mathcal{D}$ as $p(n)$, we have the recurrences $s(n) = 2s(n/2) + s_0(n/2)$ and $p(n) = 2p(n/2) + p_0(n/2)$.

To answer a halfspace range-minimum query H using $\mathcal{D}(\mathcal{S})$, we first query $\mathcal{D}_0(S_1)$ to see whether $S_1 \cap H$ is empty. If $S_1 \cap H$ is nonempty, then the answer should be some point in $\mathcal{S}_1$, and thus we can recursively query $\mathcal{D}(\mathcal{S}_1)$ to find it. If $S_1 \cap H$ is empty, the answer should be in $\mathcal{S}_2$, and we can query $\mathcal{D}(\mathcal{S}_2)$ to find it. If we write the query time of $\mathcal{D}$ as $q(n)$, we have the recurrence $q(n) = q(n/2) + q_0(n/2)$.

In $\mathbb{R}^3$, the optimal halfspace range-reporting data structure [2] gives us a halfspace range-emptiness data structure with $s_0(n) = O(n)$ space, $q_0(n) = O(\log n)$ query time, and $p_0(n) = O(n \log n)$ preprocessing time. Thus the above recurrences solve to $s(n) = O(n \log n)$, $q(n) = O(\log^2 n)$, and $p(n) = O(n \log^2 n)$.

In $\mathbb{R}^{d+1}$ for $d \geq 3$, there exists a halfspace range-emptiness data structure with $s_0(n) = O(n^{\lceil d/2 \rceil})$ space and $q_0(n) = O(\log n)$ query time [8]. Thus the above recurrences solve to $s(n) = O(n^{\lceil d/2 \rceil})$ and $q(n) = O(\log^2 n)$.

5 Hardness result for orthogonal queries

In this section, we show that, for orthogonal queries, the approximate RCP data structure is (asymptotically) at least as hard as the range-minimum problem. We use $\mathcal{B}$ to denote the collection of all axis-parallel boxes in $\mathbb{R}^d$. An orthogonal range-minimum (resp., RCP) data structure in $\mathbb{R}^d$ refers to a range-minimum (resp., RCP) data structure for query space $\mathcal{B}$.

Theorem 5 *Given an approximate orthogonal RCP data structure $\mathcal{D}_0$ in $\mathbb{R}^d$, one can build an orthogonal range-minimum data structure $\mathcal{D}$ in $\mathcal{R}^d$ such that*
- *If the space of $\mathcal{D}_0$ is $s(n)$, then the space of $\mathcal{D}$ is $O(s(2n) + n)$.*
- *If the query time of $\mathcal{D}_0$ is $q(n, \varepsilon)$, then the query time of $\mathcal{D}$ is $O(q(2n, 1) + \log n)$.*
- *If the preprocessing time of $\mathcal{D}_0$ is $p(n)$, then the preprocessing time of $\mathcal{D}$ is $O(p(2n) + n \log n)$.*

Proof. Let $\mathcal{S} = (S, w)$ be a weighted dataset in $\mathbb{R}^d$ of size n. We show how to build the desired orthogonal range-minimum data structure $\mathcal{D}$ on $\mathcal{S}$, with $\mathcal{D}_0$ in hand. Suppose $S = \{a_1, \ldots, a_n\}$. For convenience, assume $a_1, \ldots, a_n$ have distinct coordinates in each dimension and have distinct weights. We keep d sorted lists $\Gamma_1, \ldots, \Gamma_d$ of $\{a_1, \ldots, a_n\}$, where Γ_j is sorted by the j-th coordinates of the points. For $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, d\}$, we write $t_{i,j}$ as the *rank* of the j-th coordinate of a_i in S, i.e., $t_{i,j} = k$ if the index of a_i in Γ_j is k. Set $\alpha = 1/(100\sqrt{d})$. Define for each $i \in \{1, \ldots, n\}$ a hypercube $C_i = \prod_{j=1}^{d} [t_{i,j}, t_{i,j} + \alpha]$. We now choose $2n$ points $b_1, \ldots, b_n, b'_1, \ldots, b'_n$ such that **(i)** $b_i, b'_i \in C_i$ for all $i \in \{1, \ldots, n\}$ and **(ii)** $\|b_i - b'_i\|_2 < \|b_j - b'_j\|_2$ if $w(a_i) < w(a_j)$. See Figure 2 for an intuitive illustration. Our construction above guarantees that $\|b_i - b'_i\|_2 \leq 1/100$ for all $i \in \{1, \ldots, n\}$. Furthermore, for distinct $i, j \in \{1, \ldots, n\}$, the distance between a point in C_i and a point in C_j is at least 0.5. We let $S' = \{b_1, \ldots, b_n, b'_1, \ldots, b'_n\}$, and build the orthogonal RCP data structure $\mathcal{D}_0$ on S'. Then the desired data structure $\mathcal{D}$ is just $\mathcal{D}_0$ and the sorted lists $\Gamma_1, \ldots, \Gamma_d$.

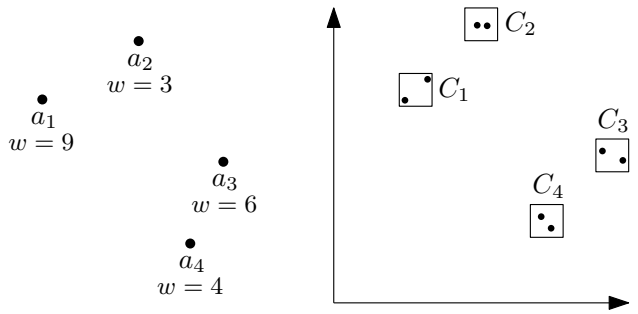


Figure 2: Illustration of the hyper-cubes $C_1, \ldots, C_n$. The two points contained in each C_i are b_i and b'_i.

To answer an orthogonal range-minimum query $Q = \prod_{j=1}^{d} [p_j, q_j]$ on $\mathcal{S}$ using $\mathcal{D}$, we first create another box Q' as follows. For all $j \in \{1, \ldots, d\}$, let L_j be the slab bounded by the two hyperplanes $x_j = p_j$ and $x_j = q_j$, and define $u_j = \min\{t_{i,j} : a_i \in L_j\}$ and $v_j = \max\{t_{i,j} : a_i \in L_j\}$. Then we set $Q' = \prod_{j=1}^{d} [u_j, v_j + \alpha]$, and query the data structure $\mathcal{D}_0$ with (Q', ε) for $\varepsilon = 1$. (Actually, any $\varepsilon > 0$ works here.) Let ϕ be the pair returned by $\mathcal{D}_0$. Assume that a_k is the point in $S \cap Q$ with the minimum weight, i.e., the answer to the range-minimum query Q. We claim that $\phi = (b_k, b'_k)$. By the construction of Q', for all $i \in \{1, \ldots, n\}$, $b_i, b'_i \in Q'$ if $a_i \in Q$ and $b_i, b'_i \notin Q'$ otherwise. Therefore, according to the locations of $b_1, \ldots, b_n, b'_1, \ldots, b'_n$, we observe that **(i)** the closest-pair in $S' \cap Q'$ is (b_k, b'_k) and **(ii)** the distance between a point in $S' \cap Q'$ and a point in $S' \setminus (S' \cap Q')$ is at least 0.5 (as the two points must be contained in different C_i's). Note that $\|b_k - b'_k\|_2 \leq \sqrt{d}\alpha = 1/100$. It follows that $\|b_k - b'_k\|_2 < \|f - g\|_2$ for any distinct points $f \in S' \cap Q'$ and $g \in S'$. Since $|\phi| \leq \|b_k - b'_k\|_2$ and one point of ϕ must be contained in Q', we have $\phi = (b_k, b'_k)$. As such,

with ϕ in hand, we can know a_k and answer the query Q.

Now we analyze the performance of $\mathcal{D}$. The space of $\mathcal{D}$ is clearly $O(s(2n)+n)$, as it consists of $\mathcal{D}_0$ and the sorted lists $\Gamma_1, \ldots, \Gamma_d$. The query time of $\mathcal{D}$ consists of the time for constructing Q' and querying $\mathcal{D}_0$. To construct Q', it suffices to compute $u_1, \ldots, u_d, v_1, \ldots, v_d$, which can be done in $O(\log n)$ time using binary search in $\Gamma_1, \ldots, \Gamma_d$. Querying $\mathcal{D}_0$ takes $O(q(2n, 1))$ time. Thus the query time of $\mathcal{D}$ is $O(q(2n, 1) + \log n)$. The preprocessing of $\mathcal{D}$ can be done in $O(p(2n) + n \log n)$ time. Indeed, we use $O(n \log n)$ time to create the sorted lists $\Gamma_1, \ldots, \Gamma_d$. With the lists in hand, we can compute $t_{i,j}$ and S' in linear time. Finally, constructing $\mathcal{D}_0$ takes $O(p(2n))$ time. □

Theorem 5 implies that for orthogonal query ranges, the range minimum problem is (asymptotically) no harder than the approximate RCP problem considered here (or equivalently, the approximate RCP problem is asymptotically at least as hard as the range minimum problem), assuming that $s(n)$ is $\Omega(n)$, $p(n)$ is $\Omega(n \log n)$, and the part of $q(n, \varepsilon)$ that depends on n is $\Omega(\log n)$.

6 Conclusion and future work

In this paper, we studied an approximate version of the RCP problem in which one point of the answer pair is allowed to be slightly outside the query range. We gave a general reduction from the approximate RCP problem to the range-minimum and range-reporting problems, which works for any query space consisting of convex bodies whose width-diameter ratio is lower bounded by a positive constant. By applying our reduction, we obtained efficient approximate RCP data structures for disk and ball queries. Finally, we showed that the approximate RCP problem for orthogonal queries is (asymptotically) at least as hard as the orthogonal range-minimum problem.

Next, we raise an open question for future work. The approximation used in this paper is with respect to the query range. Perhaps, the most natural approximation model is with respect to the quality of the answer. That is, for a specified query range Q, we want to report a $(1 + \varepsilon)$-approximate closest-pair in Q, i.e., a pair of points (strictly contained in Q) whose distance is at most $(1+\varepsilon)\cdot\kappa(S\cap Q)$. How to design efficient RCP data structures for this approximation model is an interesting direction for future study.

References

[1] M. A. Abam, P. Carmi, M. Farshi, and M. Smid. On the power of the semi-separated pair decomposition. In *Workshop on Algorithms and Data Structures*, pages 1–12. Springer, 2009.

[2] P. Afshani and T. M. Chan. Optimal halfspace range reporting in three dimensions. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 180–186. Society for Industrial and Applied Mathematics, 2009.

[3] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM (JACM)*, 42(1):67–90, 1995.

[4] P. Gupta. Range-aggregate query problems involving geometric aggregation operations. *Nordic Journal of Computing*, 13(4):294–308, 2006.

[5] P. Gupta, R. Janardan, Y. Kumar, and M. Smid. Data structures for range-aggregate extent queries. *Computational Geometry: Theory and Applications*, 2(47):329–347, 2014.

[6] J. Shan, D. Zhang, and B. Salzberg. On spatial-range closest-pair query. In *International Symposium on Spatial and Temporal Databases*, pages 252–269. Springer, 2003.

[7] R. Sharathkumar and P. Gupta. Range-aggregate proximity queries. *Technical Report IIIT/TR/2007/80. IIIT Hyderabad, Telangana*, 500032, 2007.

[8] C. D. Toth, J. O'Rourke, and J. E. Goodman. *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 2017.

[9] J. Xue, Y. Li, S. Rahul, and R. Janardan. New bounds for range closest-pair problems. In *Proceedings of the 34th Symposium on Computational Geometry*. Schloss Dagstuhl-Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing, 2018.

Time-Dependent Shortest Path Queries Among Growing Discs

Anil Maheshwari* Arash Nouri* Jörg-Rüdiger Sack*

Abstract

The determination of time-dependent collision-free shortest paths has received a fair amount of attention. Here, we study the problem of computing a time-dependent shortest path among growing discs which has been previously studied for the instance where the departure times are fixed. We address a more general setting: For two given points s and d, we wish to determine the function $\mathcal{A}(t)$ which is the minimum arrival time at d for any departure time t at s. We present a $(1+\epsilon)$-approximation algorithm for computing $\mathcal{A}(t)$.

As part of preprocessing, we execute $O(\frac{1}{\epsilon}\log(\frac{\mathcal{V}_r}{\mathcal{V}_c}))$ shortest path computations for fixed departure times, where $\mathcal{V}_r$ is the maximum speed of the robot and $\mathcal{V}_c$ is the minimum growth rate of the discs. For any query departure time $t \geq 0$ from s, we can approximate the minimum arrival time at the destination in $O(\log(\frac{1}{\epsilon}) + \log\log(\frac{\mathcal{V}_r}{\mathcal{V}_c}))$ time, within a factor of $1+\epsilon$ of optimal. Since we treat the shortest path computations as black-box functions, for different settings of growing discs, we can plug-in different shortest path algorithms. Thus, the exact time complexity of our algorithm is determined by the running time of the shortest path computations.

1 Introduction

An algorithmic challenge in robotics arises when a point object (modeling a robot, person or vehicle) is operating among moving entities or obstacles, e.g., settings in which a point object needs to avoid encountering individuals who are moving from known locations, with known speeds, but in unknown directions. This uncertainty can be modeled by discs, whose radii grow over time. Therefore, the task of computing a shortest path avoiding these individuals reduces to computing shortest path among growing discs. This particular motivation arose, e.g., in video games [1].

Given are a set of growing discs $\mathscr{C} = \{C_1, ..., C_n\}$ (the obstacles), a point robot R with maximum speed $\mathcal{V}_r$, a source point s and a destination point d, located on the plane. The radii of the discs are growing with the same constant speed $V \in (0, \mathcal{V}_r)$. The *shortest path among growing discs* (SPGD) problem is to find a shortest path from s to d, such that the robot leaves the source immediately (i.e., at time $t = 0$) and does not intersect the interior of the discs, to reach d as quickly as possible. Yi [2] showed that this problem can be solved in $O(n^2 \log n)$ time.

In this paper, we study the time-dependent version of the SPGD problem, where the departure time is a variable. The objective is to find the minimum arrival time function $\mathcal{A}(t)$, defined as the earliest time when the robot can arrive at destination d such that: (1) R leaves s at time t, (2) R does not intersect the interior of the discs after the departure. We refer to this problem as the *time-dependent shortest path among growing discs* (TDSP) problem.

Related results. The SPGD problem has been studied in different settings. Overmars et al. studied this problem in the setting where the discs are growing with equal constant speed. They presented an $O(n^3 \log n)$ time algorithm, where n is the number of discs. This result was improved by Yi [2] who showed that the shortest path among the same-speed growing discs can be found in $O(n^2 \log n)$ time. Yi also presented an $O(n^3 \log n)$ time algorithm for the case where the discs are growing with different speeds. Nouri and Sack [3] studied a general version of this problem where the speeds are polynomial functions of time.

Time-dependent shortest path problems have been studied in network settings (see [4, 5, 6]). A *network* is a graph $G = (V, E)$ with edge set E and node set V. Each edge $e \in E$ is assigned with a real valued weight. Given a source node $s \in V$ and a destination node $d \in V$, a shortest path from s to d is a path in G, where the sum of the weights of its constituent edges is minimized. However, in many applications, the weight of the edges are dynamically changing over time. In such situations, the total weight of a path depends on the departure time at its source. The problem of computing shortest paths from s to d for all possible departure times at s is known as the time-dependent shortest path problem. The general shortest path problem on time-dependent networks has been proven to be NP-Hard [7]. However, there are several approximation algorithms (see [8, 5, 6]), which are of interest in real-world applications [9].

Contribution. We say an approximation function $\overline{\mathcal{A}}(t, \epsilon)$ is a $(1+\epsilon)$-approximation for function $\mathcal{A}(t)$ if $\mathcal{A}(t) \leq \overline{\mathcal{A}}(t, \epsilon) \leq (1+\epsilon)\mathcal{A}(t)$ for all positive values of t. Here, our contribution is to compute a $(1+\epsilon)$-approximation for the minimum arrival time function $\mathcal{A}(t)$. The preprocessing step of our algorithm exe-

*School of Computer Science, Carleton University, {anil,arash,sack}@scs.carleton.ca

cutes $O(\frac{1}{\epsilon}\log(\frac{\mathcal{V}_r}{\mathcal{V}_c}))$ time-minimal path computations for fixed departure times, where $\mathcal{V}_r$ is the maximum speed of the robot and $\mathcal{V}_c$ is the minimum growth rate of the discs. Then, for a given query departure time $t \geq 0$ from s, we can report the minimum arrival time at the destination in $O(\log(\frac{1}{\epsilon}) + \log\log(\frac{\mathcal{V}_r}{\mathcal{V}_c}))$ time, within a factor of $1+\epsilon$ of optimal. We first start with a simple version of the problem, where all discs are growing with speed $\mathcal{V}_c$. In this setting, each time-minimal path computation runs in $O(n^2 \log n)$ time [2]. Our algorithm runs the shortest path computations as a black-box and its time complexity is determined by the number of such calls. This enables us to extend our algorithm for different settings of growing discs [2, 10], by plugging-in appropriate shortest path algorithms.

In Section 3, we establish several properties of the arrival time function. These properties allow us to work with arrival time functions instead of a more indirect approach of using the travel time, which has been utilized in previous work. We show that $\mathcal{A}(t)$ is the lower envelope of a set of curves called the arrival time functions. The algorithm's output size is denoted by $F_{\mathcal{A}}$ and counts the number of pieces (i.e. the sub-arcs) on the lower envelope needed to represent the function $\mathcal{A}(t)$. In Section 3.1, we establish a lower bound for $F_{\mathcal{A}}$. This lower bound, along with the complexity of computing the lower envelope, provides the motivation to study the approximation algorithm in the first place.

In Section 4, we define the *reverse shortest path problem*, in which we are to find a path from the destination to the source. Existing algorithms for the time dependent shortest paths utilize the reverse shortest path computations. These computations were done by a reversal of Dijkstra's algorithm [11, 5]. Here, we need to generalize the existing shortest path computations for growing discs [2] to shortest path computations for shrinking discs.

2 Preliminaries

2.1 Time-minimal paths among growing discs

A *robot-path* is a path in the plane that connects the source point s to the destination point d. The time at which the robot departs s is called the *departure time* and the time it arrives at d is the *arrival time*. We call a path λ *valid* (or *collision-free*) if it does not intersect any of the discs in $\mathscr{C}$; otherwise, λ is *invalid*. For a fixed departure time, a **time-minimal path** is a valid robot-path, where the arrival time is minimized over all valid paths.

It is proven that on any time-minimal path, the point robot always moves with maximal velocity of $\mathcal{V}_r$ [1]. It is shown in [3, 12] that any time-minimal path from s to d is solely composed of two types of alternating sub-paths: (1) **tangent paths:** straight line paths that are tangent to pairs of discs, and (2) **spiral paths:** logarithmic spiral paths that each lies on a boundary of the growing disc. We describe these two paths in the following.

For any pair of discs C_i and C_j, we define four *tangent paths* corresponding to their tangent lines: right-right, right-left, left-right and left-left tangents, denoted by ℓ_{ij}^{rr}, ℓ_{ij}^{rl}, ℓ_{ij}^{rl} and ℓ_{ij}^{ll} (see Figure 4). Each tangent path $\overrightarrow{\ell_{ij}^{rr}}(\tau) = \overline{pq}$ represents a straight line robot-path from a point p on the boundary of C_i to a point q on the boundary of C_j. The two points p and q are called *Steiner points*. A *spiral path* $\overrightarrow{\sigma} = \overrightarrow{\widehat{pq}}$, represents the trace of the robot's move, over time, from p to q, where $p, q \in \partial C_i$ are two "consecutive" Steiner points (see Figure 4 (a)). Note that the length of these paths are changing over time. If the robot leaves p at time τ, then it arrives at q at time $\hat{\tau}$, where $\tau < \hat{\tau}$. Observe that there exist $O(n^2)$ (moving) tangent/spiral paths and $O(n^2)$ (moving) Steiner points.

To simplify our exposition, we let the two points s and d be two discs with zero radii and zero velocities and add them to $\mathscr{C}$. Let $\mathcal{E}$ be the set of all spiral and tangent paths. Let $\mathcal{S}$ be the set of all Steiner points. We construct a directed *adjacency graph* $G = (V_s, E_s)$ as follows. With each Steiner point $v \in \mathcal{S}$ we associate a unique vertex, $\dot{v}$, in V_s. Then, with each path $\overrightarrow{uv} \in \mathcal{E}$ we associate a unique edge $\overrightarrow{\dot{u}\dot{v}}$ in E_s. The weight of each edge in G is the length of its corresponding tangent or spiral path, which is a function of time. Since each edge in E_s is associated with a path in $\mathcal{E}$, therefore, each path in the graph G is associated with a sequence of paths in $\mathcal{E}$.

Yi [2] showed that by running Dijkstra's algorithm on the adjacency graph, the SPGD problem can be solved in $O(n^2 \log n)$ time. The main steps of this algorithm are as follows:

(*i*) Identify the Steiner points, tangent paths and spiral paths (i.e., the sets $\mathcal{S}$ and $\mathcal{E}$).

(*ii*) Construct the adjacency graph $G = (V_s, E_s)$.

(*iii*) Run Dijkstra's algorithm to find a time-minimal path between $s, d \in V_s$.

In our approximation algorithm, we use the above algorithm as a black-box. The input to this algorithm is a set of growing discs and a departure time τ. The output is a shortest path between s and d in the adjacency graph, denoted by $\pi(s, d, \tau)$.

2.2 Minimum time-dependent arrival time

Let $\mathscr{P}$ be the set of all paths between s and d, in the adjacency graph G. Let $\pi \in \mathscr{P}$ and τ be a departure time at s. Recall that there exists a unique geometric path corresponding to the pair (π, τ). For instance, Figure 5 shows the geometric paths corresponding to a

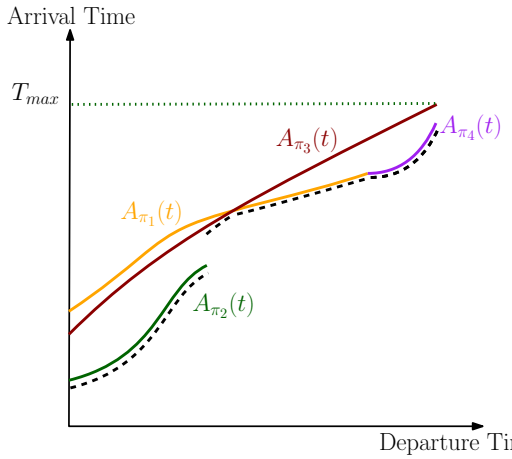


Figure 1: This figure represents the arrival curves corresponding to the time-minimal paths in Figure 5. The dotted curve (the lower envelope) represents the minimum arrival time function $\mathcal{A}(t)$.

set of paths $\{\pi_1, \pi_2, \pi_3, \pi_4\}$ at three different departure times.

For a given path $\pi \in \mathscr{P}$, let the *arrival time function* $\mathcal{A}_\pi(t)$ represents the arrival time of the robot-path corresponding to π when the departure time is t. Let $\mathscr{A} = \{\mathcal{A}_\pi(t) | \pi \in \mathscr{P}\}$ be the arrival time function corresponding to the paths in $\mathscr{P}$. The lower envelope of the set $\mathscr{A}$, denoted by $\mathcal{A}(t)$, is defined as the point-wise minimum of all the arrival functions in $\mathscr{A}$. The lower envelope is formally defined as:

$$\mathcal{A}(t) = \min_{\pi \in \mathscr{P}} \overline{A_\pi}(t)$$

where $\overline{A_\pi}(t) = \mathcal{A}_\pi(t)$ if π is a valid path and $\overline{A_\pi}(t) = \infty$, otherwise. We call $\mathcal{A}(t)$ the *minimum arrival time function*, which is found via finding the lower envelope of the arrival functions in $\mathscr{A}$. We also refer to the minimum arrival time functions as arrival curves.

Figure 1 depicts the arrival functions corresponding to the paths in Figure 5. $\mathcal{A}(t)$ is found as the lower envelope of the arrival functions $\{\mathcal{A}_{\pi_1}(t), \mathcal{A}_{\pi_2}(t), \mathcal{A}_{\pi_3}(t), \mathcal{A}_{\pi_4}(t)\}$. Observe that after a certain time, the destination point will be contained in some disc. We denote this time by T_{max}. Since there exists no valid path from s to d after T_{max}, for any $t > T_{max}$ we have $\mathcal{A}(t) = \infty$. Observe that the lower envelope is composed of sub-arcs of arrival curves in $\mathscr{A}$. A *sub-arc* is a maximal connected piece of an arrival curve on the lower envelope.

3 Properties of $\mathcal{A}(t)$

In this section, we define some properties of the minimum arrival time function. First, we show that $\mathcal{A}(t)$ is an increasing function. If the robot is allowed to move with any speed lower than $\mathcal{V}_r$, then the below lemma is

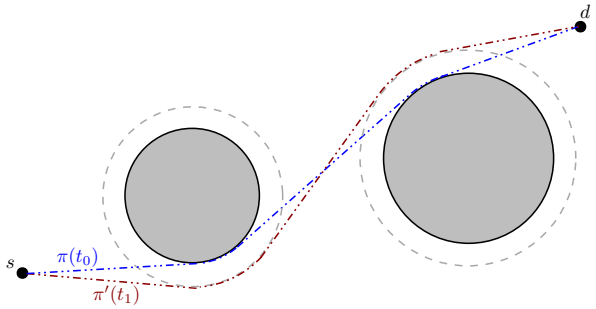


Figure 2: This figure illustrates two time-minimal paths for two departure times t_0 and t_1, where $t_1 > t_0$. For the sake of simplicity, we assumed that the speed of the robot is considerably higher than the growth rates of the discs ($V \ll \mathcal{V}_r$).

straightforward (the later it departs from the source the later it arrives at the destination). However, we assumed that the robot moves with maximum speed at all time and only uses tangent and/or spiral paths. Thus, the following proof is necessary.

Lemma 1 *$\mathcal{A}(t)$ is an increasing function.*

Proof. Let $\pi(t_0)$ and $\pi'(t_1)$ be any two time-minimal robot-paths between s and d, corresponding to two departure times t_0 and t_1, where $t_0 < t_1$ (see Figure 2). We prove that $\mathcal{A}_\pi(t_0) < \mathcal{A}_{\pi'}(t_1)$ and consequently $\mathcal{A}(t_0) < \mathcal{A}(t_1)$.

Let $T_{\pi'}(t_1) = \mathcal{A}_{\pi'}(t_1) - t_1$ be the travel time for the path $\pi'(t_1)$. Similarly, let $T_\pi(t_0) = \mathcal{A}_\pi(t_0) - t_0$. By contradiction, we assume $\mathcal{A}_{\pi'}(t_1) \leq \mathcal{A}_\pi(t_0)$. Then:

$$T_{\pi'}(t_1) + t_1 \leq T_\pi(t_0) + t_0$$
$$\overset{t_0 \leq t_1}{\Rightarrow} T_{\pi'}(t_1) < T_\pi(t_0)$$

Therefore, $\pi(t_0)$ is a longer robot-path than $\pi'(t_1)$. Since the discs are continuously growing, the free space is shrinking simultaneously. Thus, if $\pi'(t_1)$ is a valid robot-path, it is also valid for any departure time before t_1. As illustrated in Figure 2, $\pi'(t_1)$ is a valid path when the robot leaves s at time t_0.

Since $|\pi'(t_1)| < |\pi(t_0)|$ and $\pi'(t_1)$ is a valid robot-path when the robot departs s at time t_0, this contradicts the fact that $\mathcal{A}_{\pi'}(t_1) \leq \mathcal{A}_\pi(t_0)$. □

Define $|\overline{sd}|$ as the Euclidean distance between the source and the destination. Recall that $\mathcal{V}_r$ is the maximum speed of the robot, and $\mathcal{V}_c$ is the minimum growth rate among the discs in $\mathscr{C}$.

Let $\overrightarrow{sd}$ be a tangent path from s to d. For a given departure time t, if $\overrightarrow{sd}(t)$ is not obstructed by any disc, then the calculation of $\mathcal{A}(t)$ is straightforward. Thus, we are interested in computing $\mathcal{A}(t)$ when $\overrightarrow{sd}(t)$ is invalid,

i.e., $\overrightarrow{sd}(t)$ is intersected by a disc for any departure time t. With the above assumption, we have the following lemma which states the upper and the lower bound on $\mathcal{A}(t)$.

Lemma 2 *Let τ be a departure time, where $\mathcal{A}(\tau)$ is defined (i.e., $\mathcal{A}(\tau) < \infty$). Then,*

(i) $\mathcal{A}(\tau) \geq \frac{|\overline{sd}|}{\mathcal{V}_r}$

(ii) $\mathcal{A}(\tau) \leq \frac{|\overline{sd}|}{\mathcal{V}_c}$

Proof. (i) Let λ_1 be a valid time-minimal robot-path from s to d with departure time 0. It is observed that the length of λ_1 is greater or equal to $|\overline{sd}|$. Thus, $\frac{|\overline{sd}|}{\mathcal{V}_r} \leq \mathcal{A}(0)$. By Lemma 1, we have $\mathcal{A}(0) \leq \mathcal{A}(\tau)$. Therefore, $\frac{|\overline{sd}|}{\mathcal{V}_r} \leq \mathcal{A}(\tau)$.

(ii) Let λ_2 be the straight line (invalid) robot-path from s to d, which is obstructed by the disc $C \in \mathscr{C}$. Let the robot depart s at time τ and move along the path λ_2 with speed $\mathcal{V}_r$, until it arrives at the boundary of C at point q. Note that the robot arrives at q at time $\tau + \frac{|\overline{sq}|}{\mathcal{V}_r}$. Let x be the shortest Euclidean distance between d and the boundary of C at time $\tau + \frac{|\overline{sq}|}{\mathcal{V}_r}$. If C encloses d at time T, then the robot must arrive at the destination at or before T. Thus, we obtain $T \leq \tau + \frac{|\overline{sq}|}{\mathcal{V}_r} + \frac{x}{V}$. Because $x \leq |\overline{qd}|$ and $V < \mathcal{V}_c$, we have:

$$T \leq \tau + \frac{|\overline{sq}|}{\mathcal{V}_r} + \frac{|\overline{qd}|}{\mathcal{V}_c}$$

Since the above inequality is true for all departure times (including $\tau = 0$), we must have:

$$T \leq \frac{|\overline{sq}|}{\mathcal{V}_c} + \frac{|\overline{qd}|}{\mathcal{V}_c} = \frac{|\overline{sd}|}{\mathcal{V}_c}$$
$$\Rightarrow \mathcal{A}(\tau) \leq \frac{|\overline{sd}|}{\mathcal{V}_c}$$

□

3.1 The output size

The *output size* of the time-dependent shortest path, denoted by $F_{\mathcal{A}}$, is defined as the number of sub-arcs in the lower-envelope needed to represent the minimum arrival time function $\mathcal{A}(t)$. In Lemma 3, we provide an example for which $F_{\mathcal{A}}$ is $\Theta(n^2)$. Therefore, the number of sub-arcs in the lower envelope is lower bounded by $\Omega(n^2)$. This lower bound, along with the complexity of calculating the sub-arcs, inspired us to develop an approximation algorithm for this problem.

Lemma 3 *$F_{\mathcal{A}}$ is lower bounded by $\Omega(n^2)$.*

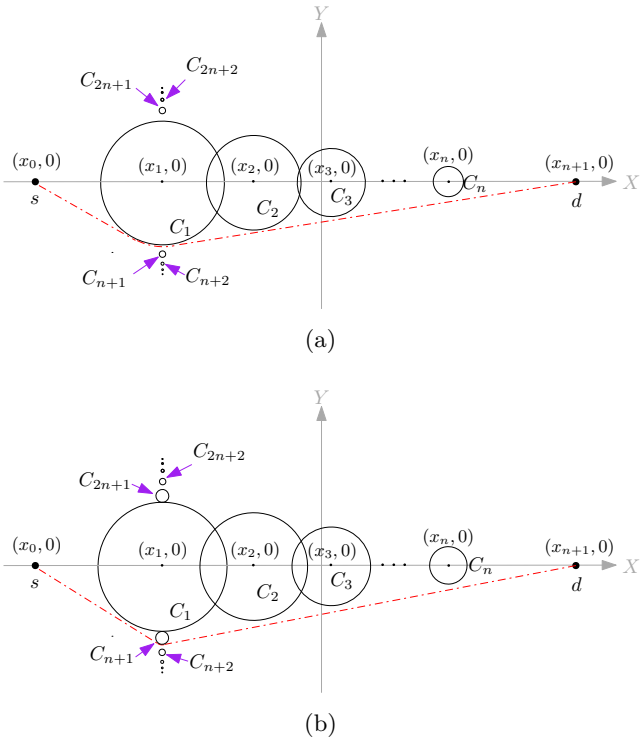


Figure 3: This figure illustrates an example where there are $\Theta(n^2)$ unique time-minimal paths for different departure times. For the ease of demonstration, we assumed $V \ll \mathcal{V}_r$.

Proof. We prove this lemma by giving an example where $F_{\mathcal{A}}$ is $\Theta(n^2)$. In this example, which is illustrated in Figure 3, the source is located at $(x_0, 0)$ and the destination is located at $(x_{n+1}, 0)$, where $x_0 \ll x_{n+1}$. A set of growing discs $\mathscr{C}_1 = \{C_1, ..., C_n\}$ are sorted along the x-axis, such that $C_i \in \mathscr{C}_1$ is centered at point $(x_i, 0)$, where $x_{i-1} < x_i < x_{i+1}$ and $x_0 \ll x_i \ll x_{n+1}$. We also define a set of "disjoint" growing discs $\mathscr{C}_2 = \{C_{n+1}, ..., C_{2n}\}$, such that $C_i \in \mathscr{C}_2$ is centered at point (x_1, y_i), where $y_{i+1} < y_i < 0$. Similarly, we define a set of "disjoint" growing discs $\mathscr{C}_3 = \{C_{2n+1}, ..., C_{3n}\}$, such that $C_i \in \mathscr{C}_3$ is centered at point $(x_1, -y_i)$. For the sake of simplicity, we assume that $V \ll \mathcal{V}_r$, where V is the speed of the discs and $\mathcal{V}_r$ is the maximum speed of the robot.

Let $\mathcal{T}_1 = \{\tau_1, ..., \tau_n\}$ be a set of departure times, sorted in increasing order. Denote by $\{\lambda_{\tau_1}, ..., \lambda_{\tau_n}\}$ the set of time-minimal robot-paths corresponding to the departure times in $\mathcal{T}_1$. We choose the initial radius of disc C_1 large enough (with respect to the other discs in $\mathscr{C}_1$) so that the robot-path λ_{τ_1} is tangent to disc C_1 and does not touch the other discs in $\mathscr{C}_1$ (see Figure 3 (a)). Similarly, we choose the initial radius of disc C_2 such that λ_{τ_2} is only tangent to C_1 and C_2. We repeat the same procedure for all the departure times in $\mathcal{T}_1$. Consequently, each robot-path λ_{τ_i} is tangent all the discs in $\{C_1, C_2, ..., C_i\}$.

Observe that we can choose an appropriate value for y_{n+1} such that C_1 and C_{n+1} intersect at time τ_1', where for a small value of α we have $\tau_n < \tau_1' < \tau_n + \alpha$. Let $\lambda_{\tau_1'}$ be the time-minimal robot-path corresponding to the departure time τ_1'. As illustrated in Figure 3 (b), observe that $\lambda_{\tau_1'}$ is tangent to disc C_{n+1}. By an argument similar to the above paragraph, we can define a set $\mathcal{T}_2 = \{\tau_1', ..., \tau_n'\}$, where the corresponding time-minimal paths of the departure times $\tau_i' \in \mathcal{T}_2$ are tangent to all the discs in $\{C_{n+1}, C_2, ..., C_i\}$. We repeat the above procedure for all the discs in $\mathscr{C}_2$. Let $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2 \cup ... \cup \mathcal{T}_n$.

Let $\pi(s, d, \tau_i)$ and $\pi(s, d, \tau_j)$ be a pair of shortest paths in the adjacency graph G, corresponding to the departure times $\tau_i, \tau_j \in \mathcal{T}_1$, where $i \neq j$. Since λ_{τ_i} and λ_{τ_j} are tangent to different sets of discs, we have $\pi(s, d, \tau_i) \neq \pi(s, d, \tau_j)$. Thus, observe that $\mathcal{A}(t)$ consists of $\Theta(|\mathcal{T}|) = \Theta(n^2)$ sub-arcs. □

4 Approximating $\mathcal{A}(t)$

4.1 The reverse shortest path

Let us define a function $\mathcal{A}^{-1} : [0, T_{max}] \to [0, T_{max}]$ where $\mathcal{A}^{-1}(t)$ is the latest departure time at s, when the robot arrives at d at time t. In this section, we describe our method for computing the function $\mathcal{A}^{-1}(t)$ for a set of "fixed" values of t. We generalize the time-minimal path algorithm presented in [2] to the case where the discs are shrinking.

A growing disc C_i is defined by a pair $(O_i, R_i(t))$, where O_i is the center and $R_i(t)$ is the radius of C_i at time t. Let $\partial C_i(t)$ denote the boundary of the disc C_i at time $t \in [0, T_{max}]$. We now define a *shrinking disc* $\hat{C}_i$ by a pair $(O_i, \hat{R}_i(t))$, such that $\hat{R}_i(t) = R_i(T_{max} - t)$. Note the following properies:

- $\hat{C}_i$ and C_i are centered at the same point.
- $\partial C_i(0) = \partial \hat{C}_i(T_{max})$.
- $\partial C_i(t) = \partial \hat{C}_i(T_{max} - t)$.

Let $\hat{\mathscr{C}} = \{\hat{C}_1, ..., \hat{C}_n\}$ be a set of shrinking discs. We begin with the following observation.

Observation 1 *For any two given times τ and $\hat{\tau}$, where $0 \leq \tau < \hat{\tau} \leq T_{max}$, $C_i(\tau)$ and $C_j(\hat{\tau})$ have the same tangent lines as $\hat{C}_i(T_{max} - \tau)$ and $\hat{C}_j(T_{max} - \hat{\tau})$.*

Let $\ell_{ij}^{rl}(\tau) = \overline{pq}$ be a tangent line where $p \in \partial C_i(\tau)$ and $q \in \partial C_j(\hat{\tau})$. Similarly, let $\hat{\ell}_{ji}^{lr}(T_{max} - \hat{\tau}) = \overline{qp}$ be a tangent line where $q \in \partial \hat{C}_j(T_{max} - \hat{\tau})$ and $p \in \partial \hat{C}_i(T_{max} - \tau)$. By Observation 1, $\ell_{ij}^{rl}(\tau)$ is equivalent to $\hat{\ell}_{ji}^{lr}(T_{max} - \hat{\tau})$. Thus, the two tangent paths $\overrightarrow{\hat{\ell}_{ji}^{lr}}(T_{max} - \hat{\tau})$ and $\overrightarrow{\ell_{ij}^{rl}}(\tau)$ are the same line segments, but with opposite directions. We call $\overrightarrow{\hat{\ell}_{ji}^{lr}}(T_{max} - \hat{\tau})$ the *reverse tangent path* of $\overrightarrow{\ell_{ij}^{rl}}(\tau)$. Similarly, for a spiral path $\overrightarrow{\sigma}(\tau)$ there exists a *reverse spiral path* $\overrightarrow{\hat{\sigma}}(T_{max} - \hat{\tau})$. Moreover, we can extend this definition to a robot-path: let λ be a valid robot-path from s to d, where the departure time is τ_s and the arrival time is τ_d. Then, there exists a *reverse robot-path* $\hat{\lambda}$ from d to s whose departure time is $T_{max} - \tau_d$ and arrival time is $T_{max} - \tau_s$.

Recall that in Step (ii) of the SPGD algorithm (see Section 2.1), the adjacency graph G is constructed using the identified tangents and spiral paths in Step (i). Similarly, we construct the *reverse adjacency graph* $\hat{G}$ using the reverse tangents and spiral paths.

Lemma 4 *Let $\pi(u, v, \tau_u)$ be a valid path from vertex u to vertex v in G, where the departure time is τ_u and the arrival time is τ_v. Then, there exists a valid path $\hat{\pi}(v, u, T_{max} - \tau_v)$ in $\hat{G}$ whose arrival time is $T_{max} - \tau_u$.*

Proof. Since $\pi(u, v, \tau_u)$ is a valid path, there exists a robot-path λ with departure time τ_u and arrival time τ_v. By definition, there exists a reverse robot-path of λ, denoted by $\hat{\lambda}$, whose departure time and arrival time are $T_{max} - \tau_v$ and $T_{max} - \tau_u$, respectively. Thus, there exists a valid path $\hat{\pi}(v, u, T_{max} - \tau_v)$ in $\hat{G}$. □

By the above lemma, for any path π from s to d in G, there exists a path $\hat{\pi}$ from d to s in $\hat{G}$ of the same length as π. Thus, in order to find a shortest path form s to d, we can find a shortest path in $\hat{G}$ and reverse its direction. Therefore, similar to the SPGD algorithm in Section 2.1, a reverse shortest path can be found by running Dijkstra's algorithm in $\hat{G}$. We summarize the steps of the above algorithm as follows.

(i) Identify the reverse tangents and spiral paths.

(ii) Construct the reverse adjacency graph $\hat{G}$.

(iii) Run Dijkstra's algorithm on $\hat{G}$ to find a time-minimal path.

We call the above algorithm the *reverse shortest path among growing discs* (RSPGD). Using this algorithm, for any given time t, a time-minimal path can be found which arrives at destination at time t.

Corollary 1 *For a given arrival time t at d, $\mathcal{A}^{-1}(t)$ can be computed by running the RSPGD algorithm.*

We should remark that finding a shortest path from d to s in $\hat{G}$ does not always yield a time-minimal path among shrinking discs. For example, consider the case where the robot stops and waits for some discs to shrink to a certain size, until they open a previously blocked path. Then, the robot starts moving with maximum velocity towards the destination along the recently opened path. This contradicts our assumption that the robot always moves with the maximum speed. Thus, a shortest path in $\hat{G}$ does not guarantee a time-minimal robot-path among shrinking discs.

4.2 Approximation Algorithm

In this section, we present a $(1+\epsilon)$-approximation algorithm for computing the minimum arrival time function $\mathcal{A}(t)$. To obtain an approximation for $\mathcal{A}(t)$, our algorithm (Algorithm 1) computes a set of arrival time values $A = \{a_1, a_2, ..., a_m\}$, such that, for $2 \leq i \leq m$, $\frac{a_i}{a_{i-1}} = 1+\epsilon$ (i.e., arrival times are spaced within a factor of $1+\epsilon$ from each other). Since $\mathcal{A}(t)$ is an increasing function (refer to Lemma 1), for each valid arrival time, there exists a unique departure time. This is a key distinction to some of the previously standard variants of the time-dependent shortest path problems. For each $a_i \in A$, the algorithm runs the RSPGD algorithm to find its corresponding departure time b_i. Let us denote the departure time values by a set $B = \{b_1, b_2, ..., b_m\}$. Each departure time in B is referred to as a *sampled time*.

Algorithm 1 Computing B

1: $B = \emptyset$, $A = \emptyset$, $i = 0$, $a_0 = \mathcal{A}(0)$
2: ▷ *$\mathcal{A}(0)$ is calculated by running the SPGD algorithm [2]*
3: **while** $a_i < T_{max}$ **do**
4: $a_{i+1} = (1+\epsilon)a_i$
5: $b_i = \mathcal{A}^{-1}(a_{i+1})$
6: $B := B \cup \{b_i\}$
7: $A := A \cup \{a_{i+1}\}$
8: $i = i + 1$
9: **return** B and A

Lemma 5 *Algorithm 1 runs $O(\frac{1}{\epsilon}\log(\frac{\mathcal{V}_r}{\mathcal{V}_c}))$ time-minimal path computations.*

Proof. We first estimate the number of sampled times in B. Let $B = \{b_1, b_2, ..., b_m\}$. By definition we have

$$\frac{\mathcal{A}(b_m)}{\mathcal{A}(b_1)} = \frac{a_m}{a_1} = (1+\epsilon)^{m-1}$$

By Lemma 2, for any $1 \leq i \leq k$ we have $\mathcal{A}(b_i) \leq \frac{|\overline{sd}|}{\mathcal{V}_r}$ and $\mathcal{A}(b_i) \geq \frac{|\overline{sd}|}{\mathcal{V}_c}$. So, we obtain

$$(1+\epsilon)^{m-1} \leq \frac{\mathcal{V}_r}{\mathcal{V}_c}$$

For $\epsilon \in (0,1)$, we observe that $\frac{\epsilon}{2} < \log(1+\epsilon)$. Thus,

$$(m-1)\frac{\epsilon}{2} \leq \log\left(\frac{\mathcal{V}_r}{\mathcal{V}_c}\right)$$

Therefore, there are $O(\frac{1}{\epsilon}\log(\frac{\mathcal{V}_r}{\mathcal{V}_c}))$ sampled times in B. For each sampled time, the algorithm runs an instance of the reverse shortest path algorithm in Line 5. Thus, the total number of time-minimal path computations is $O(\frac{1}{\epsilon}\log(\frac{\mathcal{V}_r}{\mathcal{V}_c}))$. □

Using the sampled times reported by Algorithm 1, we now define a step function $\overline{\mathcal{A}} : [0, T_{max}] \times (0,1) \to A$ such that for $t \in [b_i, b_{i+1})$, we have $\overline{\mathcal{A}}(t, \epsilon) = a_{i+1}$.

Lemma 6 *For any real constant value of $\epsilon \in (0,1)$, function $\overline{\mathcal{A}}$ is a $(1+\epsilon)$-approximation for the arrival time function $\mathcal{A}$.*

Proof. The proof is deferred to the Appendix. □

Theorem 7 *The minimum arrival time function can be approximated by executing $O(\frac{1}{\epsilon}\log(\frac{\mathcal{V}_r}{\mathcal{V}_c}))$ time-minimal path computations.*

Proof. This is a direct result of Lemmas 5 and 6. □

Since the time-minimal path algorithm for fixed departure times runs in $O(n^2\log(n))$ time [2], the time complexity of our preprocessing algorithm is $O(\frac{n^2}{\epsilon}\log(\frac{\mathcal{V}_r}{\mathcal{V}_c})\log(n))$. For a given query departure time $t \geq 0$, we can report the approximated value of the minimum arrival time (i.e., $\overline{\mathcal{A}}(t, \epsilon)$) using a binary search in B. In Lemma 5, we proved that the size of the set B is $O(\frac{1}{\epsilon}\log(\frac{\mathcal{V}_r}{\mathcal{V}_c}))$. Thus, the query time of our algorithm is $O(\log(\frac{1}{\epsilon}) + \log\log(\frac{\mathcal{V}_r}{\mathcal{V}_c}))$.

5 Conclusions

In this paper, we studied the time-dependent minimum arrival time problem among growing discs. We presented a $(1+\epsilon)$-approximation to compute the minimum arrival time function. Our algorithm runs shortest path algorithms as a black-box and its time complexity is determined by the number of such calls. Therefore, for different problem settings, we can plug-in different shortest path algorithms. For example, Nouri and Sack [3] studied a variant of the SPGD problem where the growth rates of the discs are given as polynomial functions of degree β. In this algorithm a query time-minimal path can be found in $O(n^2\log(\beta n))$ time. By plugging-in this algorithm, our preprocessing step executes $O(\frac{1}{\epsilon}\log(\frac{\mathcal{V}_r}{\mathcal{V}_c}))$ shortest path computations, running in $O(\frac{n^2}{\epsilon}\log(\frac{\mathcal{V}_r}{\mathcal{V}_c})\log(\beta n))$ time.

In order to compute the output size of the minimum arrival time function $\mathcal{A}(t)$, one would need to determine the number of sub-arcs in the lower envelope, denoted by $F_{\mathcal{A}}$. We presented a lower bound on $F_{\mathcal{A}}$ and we leave as an open problem to establish an upper bound.

Another interesting open problem is to approximate the minimum arrival time function when the query involves the departure time at s, as well as s and d as part of the input.

References

[1] Jur van den Berg and Mark Overmars. Planning the shortest safe path amidst unpredictably mov-

ing obstacles. *Algorithmic Foundation of Robotics*, pages 103–118, 2008.

[2] Jiehua Yi. *A Ubiquitous GIS: Framework, Services and Algorithms Development*. PhD thesis, Ottawa, Carleton University, Ont., Canada, 2009.

[3] Arash Nouri and Jorg-Rudiger Sack. Query shortest paths amidst growing discs. *CoRR*, arXiv : abs/1804.01181, 2018.

[4] Frank Dehne, Masoud T. Omran, and Jörg-Rüdiger Sack. Shortest paths in time-dependent FIFO networks. *Algorithmica*, 62(1):416–435, 2012.

[5] Masoud Omran and Jörg-Rüdiger Sack. Improved approximation for time-dependent shortest paths. *Computing and Combinatorics: 20th International Conference, COCOON 2014*, pages 453–464, 2014.

[6] Luca Foschini, John Hershberger, and Subhash Suri. On the complexity of time-dependent shortest paths. *Algorithmica*, 68(4):1075–1097, 2014.

[7] Ariel Orda and Raphael Rom. Minimum weight paths in time-dependent networks. *Networks*, 21(3):295–319, 1991.

[8] Frank Dehne, Masoud T. Omran, and Jörg-Rüdiger Sack. Shortest paths in time-dependent FIFO networks using edge load forecasts. In *Proceedings of the Second International Workshop on Computational Transportation Science*, IWCTS '09, pages 1–6, New York, NY, USA, 2009. ACM.

[9] Danny Z. Chen. Developing algorithms and software for geometric path planning problems. *ACM Comput. Surv.*, 28(4es), December 1996.

[10] A. Nouri and Jörg-Rüdiger Sack. Query Shortest Paths Amidst Growing Discs. *Preprint submitted to SWAT 2018*, 2018.

[11] Carlos F. Daganzo. Reversibility of the time-dependent shortest path problem. *Transportation Research Part B: Methodological*, 36(7):665 – 668, 2002.

[12] J. van den Berg. Path planning in dynamic environments. *Ph.D. Thesis, Utrecht University, Utrecht, The Netherlands*, 2007.

Appendix

Lemma 6 *For any real constant value of $\epsilon \in (0,1)$, function $\overline{\mathcal{A}}$ is a $(1+\epsilon)$-approximation for the arrival time function $\mathcal{A}$.*

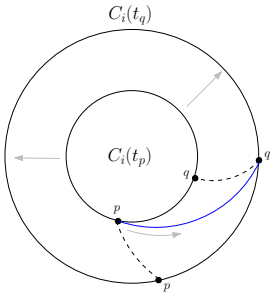
Proof. By definition, for any $t \in [b_i, b_{i+1})$ we have $\overline{\mathcal{A}}(t,\epsilon) = a_{i+1}$. Referring to the fact that $\mathcal{A}$ is an increasing function, for any $t \in [b_i, b_{i+1})$ we obtain $\mathcal{A}(b_i) \leq \mathcal{A}(t) < \mathcal{A}(b_{i+1})$. Thus,

$$\frac{\mathcal{A}(b_{i+1})}{\mathcal{A}(b_{i+1})} < \frac{\overline{\mathcal{A}}(t,\epsilon)}{\mathcal{A}(t)} \leq \frac{\mathcal{A}(b_{i+1})}{\mathcal{A}(b_i)}$$

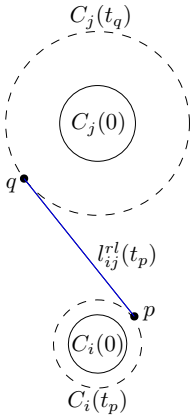
Since we have $\mathcal{A}(b_{i+1}) = a_{i+1}$ and $\mathcal{A}(b_i) = a_i$:

$$1 < \frac{\overline{\mathcal{A}}(t,\epsilon)}{\mathcal{A}(t)} \leq \frac{a_{i+1}}{a_i} = 1+\epsilon$$

□



(a)



(b)

Figure 4: Two robot-paths are illustrated: (a) spiral path, (b) a right-left tangent path. Note that t_p is the departure time and t_q is the arrival time where $t_q > t_p$.

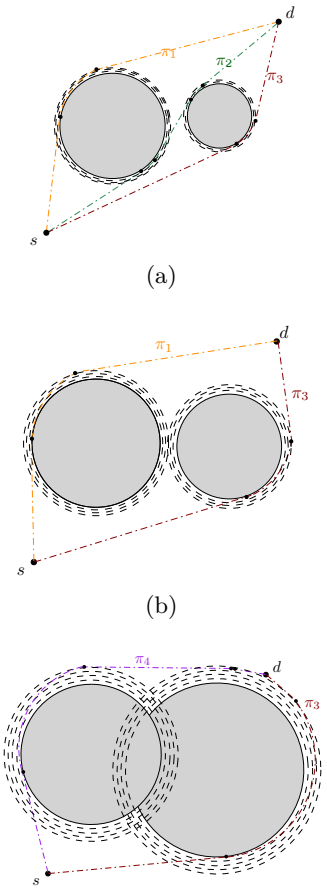


Figure 5: This figure illustrates valid paths between s and d for three different departure times. (a) At this time, three paths π_1, π_2 and π_3 are valid. (b) Path π_2 is obstructed and becomes invalid. Consequently, path π_3 is the time-minimal path. (c) Path π_1 is obstructed and π_4 is the time-minimal path.

Trajectory Planning for an Articulated Probe

Ovidiu Daescu* Kyle Fox* Ka Yaw Teo*

Abstract

We present a geometric-combinatorial algorithm for computing a feasible solution for a new trajectory planning problem involving a simple articulated probe. The probe is modeled as two line segments ab and bc, with a joint at the common point b, where bc is of fixed length r and ab is of arbitrarily large (infinite) length. Initially, ab and bc are collinear. Given a set of obstacles represented as n line segments, and a target point t, the probe is to first be inserted in straight line, followed possibly by a rotation of bc, so that in the final configuration c coincides with t, all while avoiding intersections with the line segments (obstacles). We prove that a feasible probe trajectory can be determined in $O(n^2 \log n)$ time using $O(n \log n)$ space (in fact, our algorithm finds a set of "extremal" feasible configurations). In the process, we address and solve some other interesting problems, such as circular sector emptiness queries and a special case of circular arc ray shooting queries for line segments in the plane.

1 Introduction

Consider the following *trajectory (or motion) planning problem*. An articulated needle-like probe is modeled in $\Re^2$ as two line segments, ab and bc, joined at point b. Line segment bc may rotate at point b. The length of line segment ab can be arbitrarily large (infinitely long), while line segment bc has a fixed length r (e.g., unit length).

A two-dimensional workspace is defined as the region bounded by a circle S, which contains a set of n disjoint line segment obstacles P (see Figure 1). Let t be a point in the free space (i.e., inside S and outside the obstacles).

In the beginning, the probe assumes a straight configuration, that is, line segments ab and bc are collinear, with $b \in ac$. We call this an *unarticulated* configuration. Starting from outside S, the unarticulated probe, represented by straight line segment abc, may be inserted into S as long as no obstacle is intersected by abc. After the insertion, line segment bc may be rotated at point b up to $\pi/2$ radians in either direction, provided that line segment bc does not collide with any obstacle. If a rotation is performed, then we have an *articulated* configuration of the probe.

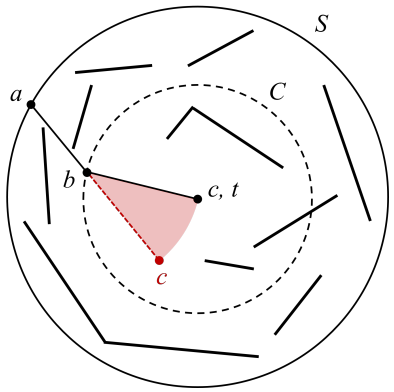


Figure 1: Trajectory planning for an articulated probe. After a straight insertion of line segment abc, in order to reach point t in the midst of obstacles, line segment bc may be required to rotate from its intermediate position (red dashed line) to the final position (black solid line).

A *feasible probe trajectory* consists of an initial insertion of straight line segment abc, possibly followed by a rotation of line segment bc at point b, such that point c ends at the target point t, while avoiding obstacles in the process of insertion and rotation.

The objective of the problem is to determine a feasible probe trajectory, if one exists. As far as the authors are aware, no previous geometric-combinatorial algorithm for this problem is known. Possible extensions of the problem include reporting the space of all feasible probe trajectories and finding feasible probe trajectories of *maximum clearance*, although these extensions are beyond the scope of this extended abstract.

Because bc may only rotate up to $\pi/2$ radians, it is an easy observation that for any feasible probe trajectory, point b is the first intersection of segment ab with a circle C of radius r centered at point t. As illustrated in Figure 1, segment bc may rotate about point b, and the area swept by segment bc is a sector of a circle (a portion of a disk enclosed by two radii and an arc), with the center located on C, radius r, and the endpoint of one of its two bounding radii located at point t. For conciseness, the center of the circle on which a circular sector is based is referred to herein as the *center of the circular sector*.

*Department of Computer Science, University of Texas at Dallas, Richardson, TX, USA. {daescu, kyle.fox, ka.teo}@utdallas.edu

Related work

The motion of a linkage – that is, a sequence of fixed-length edges connected consecutively through joints – has been formerly studied from various perspectives, ranging from basic properties and questions (e.g., reachability, reconfiguration, and locked decision) with strong geometric and topological aspects [4, 9] to application-driven robotic arm modeling and motion planning problems [10].

In contrast to those previous studies, our paper is concerned with finding a collision-free path of motion for a two-bar linkage constrained to an ordered sequence of motions – namely, a straight insertion (of the linkage) followed by a rotation (at the joint). Furthermore, one of the links is considered to be unbounded in length.

Motivation

The problem setting described in the current study has practical relevance in the field of robotics, particularly in minimally invasive robotic surgery, where the plane of insertion for a surgical probe can be defined based on various medical imaging techniques. In minimally invasive surgical approaches, a small incision is made, and the surgical operation is performed by using specialized tools inserted through the incision.

Most conventional surgical devices are straight, rigid, or flexible. A simple articulated probe such as one defined herein could be useful in minimally invasive surgery for reaching previously unattainable targets by circumventing critical structures, and for reaching multiple targets from a single insertion site while minimizing healthy tissue damage.

Results and contributions

We describe an algorithm that finds a feasible probe trajectory in $O(n^2 \log n)$ time using $O(n \log n)$ space. In fact, our algorithm finds a set of so called "extremal" feasible probe configurations. In such a configuration, one or two obstacle endpoints are tangent to the probe (see Lemma 1 below).

In the process of describing our solution, we solve some special cases of a number of fundamental problems of theoretical interest in computational geometry, such as circular sector intersection and circular sector emptiness queries. In particular, we present a data structure of near-linear size with logarithmic query time for solving a special instance of the circular arc intersection query problem (i.e., for a query circular arc with a fixed radius r and fixed endpoint t).

Our algorithm for articulated probe trajectory planning can be extended for polygonal obstacles, where we can exploit output sensitive algorithms with respect to the number of polygons and the complexity of the visibility (to infinity) from a given point.

2 Main observation

Our algorithm crucially depends upon the following observation. It immediately implies that it suffices to test a finite number of probe trajectories for feasibility. We refer to these trajectories as *extremal*.

Lemma 1 *There exists a feasible probe trajectory such that the probe assumes either I: an **unarticulated** final configuration (i.e., it is a straight line segment abc with $c = t$) that intersects an obstacle endpoint, or II: an **articulated** final configuration (i.e., line segments ab and bc are not colinear and $c = t$) that intersects an obstacle endpoint outside C and another obstacle endpoint inside or outside C.*

Proof. The existence of feasible probe trajectories for case I and II can be proven using simple perturbation arguments. The full proof of Lemma 1 is given in the Appendix. □

3 Solution approach

Based on the observation stated in Lemma 1, the set of extremal feasible probe trajectories can be obtained using the following approach. For the purpose of analysis and clarity, the line segments of P are divided into those lying inside C and those lying outside C. Since a line segment may intersect C at most two times, a line segment may be partitioned by C into at most three line segments. Let P_{in} (resp. P_{out}) be the set of line segments lying inside (resp. outside) C. In addition, let V, V_{in}, and V_{out} denote the set of endpoints of the line segments of P, P_{in}, and P_{out}, respectively. Let $n_{in} = |V_{in}|$, $n_{out} = |V_{out}|$. We have $n_{in} + n_{out} = O(n)$.

Case I. Feasible unarticulated probe trajectory

We compute the set R of $O(n)$ rays, each of which originates at point t, passes through a vertex of V, and does not intersect any line segment of P. Each ray $\gamma \in R$ represents a feasible unarticulated probe trajectory, and the set R of rays can be obtained by computing the visibility polygon from point t in $O(n \log n)$ time [1, 6, 15].[1]

Lemma 2 *The set of extremal feasible unarticulated probe trajectories can be determined in $O(n \log n)$ time.*

[1] For our case of disjoint line segments or even polygonal obstacles, we could use the *visibility complex* to compute R. The visibility complex is a two-dimensional subdivision in which each cell corresponds to a collection of rays with the same visibility properties [12]. For a simple scene of polygonal obstacles with a total of n vertices, the visibility complex can be computed in $O(n \log n + m)$ time using $O(m)$ space, where m is the size of the visibility complex (or the corresponding visibility graph) [13]. In the worst case, $m = O(n^2)$. After the visibility complex is built, one can compute the view from a point in time $O(k \log n)$, where k is the size of the computed view (from which the "visible" tangents or rays can be reported).

Case II. Feasible articulated probe trajectory

For ease of exposition, the two subcases of Case II, depending on whether an articulated final configuration intersects 1) an obstacle endpoint outside C and an obstacle endpoint inside C, or 2) two obstacle endpoints outside C, are considered separately.

Subcase 1. In order to find a feasible probe trajectory with an articulated final configuration that intersects an obstacle endpoint outside C and an endpoint inside C, we first determine a feasible articulated *final* configuration in the following manner.

We compute the set R_{in} of rays, each of which originates at point t, passes through an endpoint of V_{in}, and does not intersect any line segment of P_{in}. By using the same algorithm for computing the visibility polygon of t used in Case I, R_{in} can be obtained in $O(n_{in} \log n_{in})$ time (alternatively, we can actually extract R_{in} directly from the visibility polygon constructed for Case I in $O(n)$ additional time).

For each ray $\gamma_{in} \in R_{in}$, we i) find the intersection point b of γ_{in} and C in $O(1)$ time, and ii) compute the set R_{out} of rays, each of which originates at point b, passes through an endpoint of V_{out}, and does not intersect any line segment of P_{out}. This can be done in $O(n_{out} \log n_{out})$ time (we could get rid of the $O(\log n)$ factor with some care, using duality, while treating all such b points at once).

A pair of rays $\gamma_{in} \in R_{in}$ and $\gamma_{out} \in R_{out}$ intersecting at a point b defines a feasible final configuration of an articulated probe trajectory, that intersects an endpoint outside C and an endpoint inside C. Given that the number of rays in R_{in} is bounded by $O(n_{in})$, the worst-case running time for finding the final configuration pairs of rays γ_{in} and γ_{out}, intersecting at a point b, is in the order of $O(n_{in} n_{out} \log n_{out} + n_{in} \log n_{in})$.

Subcase 2. In order to find a feasible probe trajectory with an articulated final configuration that intersects two endpoints outside C, we determine a feasible intermediate configuration (i.e., the probe configuration after inserting straight line segment abc into S and before rotating line segment bc) using the following procedure.

For each endpoint $v \in V_{out}$, we compute the set R of rays, each of which has the following properties: it originates at endpoint v, passes through an endpoint of $V_{out} \setminus \{v\}$, does not intersect any line segment of P, and its *reversal* intersects C and goes at least a distance r beyond the intersection point with C without intersecting any line segment of P. Again, R can be obtained in $O(n \log n)$ time by computing the visibility polygon of v. For each ray $\gamma \in R$, in $O(1)$ time, we find the first intersection point b of C with the reversal of γ.

A ray $\gamma \in R$ whose reversal intersects C at a point b and satisfies the obstacle free restriction above represents a feasible intermediate configuration of an articulated probe trajectory that intersects two vertices outside C. Since $|V_{out}| = n_{out}$, the worst-case running time for finding such a ray γ is $O(n_{out} n \log n)$.[2]

An articulated probe trajectory with a feasible final or intermediate configuration is feasible if and only if the area swept by segment bc after the initial insertion (i.e., a circular sector) is not intersected by any obstacle. Thus, the remainder of Case II entails a circular sector intersection problem, detailed in the next section.

4 Circular sector intersection queries

The general line segment circular sector intersection query problem can be formally stated as follows.

Given a set P of n line segments, preprocess it so that, for a query circular sector σ, one can efficiently determine whether σ intersects P.

For our purposes, it suffices to solve a special case of this problem where the radius of the circular sector is fixed to r and one endpoint of the circular arc of the sector is fixed at t. The intersection of a line segment and a circular sector can only occur as some combination of the three basic scenarios depicted in Figure 2.

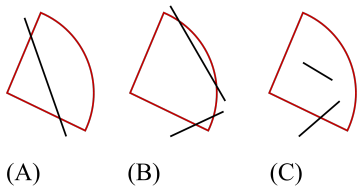


Figure 2: Basic scenarios of a line segment intersecting with a circular sector. (A) The segment intersects both radii of the sector. (B) The segment intersects the sector's circular arc at least once. (C) At least one segment endpoint lies inside the sector.

Recall that a feasible final or intermediate configuration for an articulated probe trajectory has been found in the previous section. Thus, one of the radii of the query circular sector is surely not intersected by any line segment of P. Therefore, the basic scenario in Figure 2A can be eliminated from consideration.

Hence, our case of the circular sector intersection problem reduces to the following two problems:

1. Circular arc intersection query – for detecting the basic scenario in Figure 2B.
2. Circular sector emptiness query – for detecting the basic scenario in Figure 2C.

[2] As before, we can handle subcases 1 and 2 using the visibility complex approach.

Circular arc intersection queries. Consider the following circular arc intersection problem.

Problem 1 *Given a set P of n line segments, preprocess it so that, for a query circular arc γ that originates at a fixed point t and has a fixed radius r, one can efficiently determine if γ intersects P.*

Notice that since a query circular arc γ originates from a fixed point t and has a fixed radius r, the center of γ is always located on a circle C of radius r centered at point t.

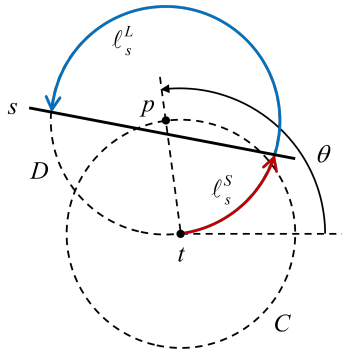


Figure 3: Counter-clockwise circular arcs ℓ_s^S and ℓ_s^L emanating from point t.

Let D be a circle of radius r centered at any point $p \in C$ (Figure 3). Note that circle D passes through the center t of circle C. We let θ be the angle of tp relative to the x-axis; D is uniquely determined by θ since we know p lies on C. We will consider only query arcs that emanate counter-clockwise from t. The other case can be handled symmetrically.

Fix a line segment s, and let h_s be its supporting line. We will define two partial functions $\ell_s^S, \ell_s^L : [0, 2\pi) \to \mathbb{R}_{\geq 0}$. Let $\theta \in [0, 2\pi)$ and let D be the circle for that θ as defined above. If D intersects h_s and the first intersection lies on s, let $\ell_s^S(\theta)$ be the length of the counter-clockwise arc from t to that first intersection. Otherwise, $\ell_s^S(\theta)$ is undefined. Similarly, if D has a second intersection with h_s on s, let $\ell_s^L(\theta)$ be the length of the arc to that intersection. We easily observe the following properties of ℓ_s^S (Figure 4). The same statements apply to ℓ_s^L as well.

Property 1 *Function ℓ_s^S is defined over at most two maximal contiguous subsets of $[0, 2\pi)$.*

Property 2 *Given two segments s_i, s_j, we have $\ell_{s_i}^S(\theta) = \ell_{s_j}^S(\theta)$ for at most one value of θ. Specifically, it is the value of θ for which D's shorter counter-clockwise arc ends at the intersection of s_i and s_j. Because s_i and s_j can only intersect at their endpoints, $\ell_{s_i}^S(\theta) = \ell_{s_j}^S(\theta)$ only at the endpoints of the maximal contiguous subsets of $[0, 2\pi)$ for which $\ell_{s_i}^S$ and $\ell_{s_j}^S$ are defined.*

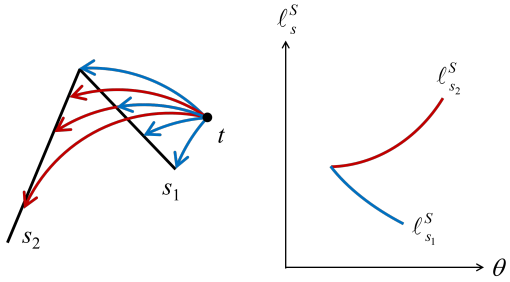


Figure 4: Illustration of the properties of function ℓ_s^S. A given line segment s induces a partially-defined piecewise continuous curve l_s^S. Two curves $\ell_{s_i}^S$ and $\ell_{s_j}^S$ intersect (at most once) if and only if their corresponding line segments s_i and s_j intersect (e.g., s_1 and s_2).

The lower envelope of n segments in the plane has complexity bounded by the third order Davenport-Schinzel sequence, which is $O(n\alpha(n))$, where $\alpha(n)$ is the inverse of the Ackermann function [14]. The lower envelope can be found by a worst-case optimal divide-and-conquer algorithm running in $O(n \log n)$ time [7]. Let V^S be the lower envelope of the curves $\ell_{s_i}^S$ for all given line segments $s_i \in P$. Given the properties of each $\ell_{s_i}^S$ (in particular that the curves intersect at endpoints), the size of lower envelope V^S is actually bounded by the second order Davenport-Schinzel sequence, which is $O(n)$, and we can again compute it in $O(n \log n)$ time. We define and compute V^L similarly for the curves $\ell_{s_i}^L$. In order to determine whether a query circular arc γ intersects P, the angle θ of center p of circular query arc γ from point t is looked up in V^S and V^L by using two binary searches that take $O(\log n)$ time. If the length of γ is less than $\ell_{s_i}^S(\theta)$ and $\ell_{s_i}^L(\theta)$ for all s_i, then γ does not intersect any line segment of P. Otherwise, it intersects the segment which lies on a lower envelope at θ. Thus, we obtain the following result, which can be easily shown to be worst case optimal (see [15]).

Lemma 3 *A set P of n non-crossing line segments can be preprocessed in $O(n \log n)$ time into a data structure of size $O(n)$ so that, for a query circular arc γ that originates at a fixed point t and has a fixed radius r, one can determine whether γ intersects P in $O(\log n)$ time.*

Circular sector emptiness queries. Our special case of the circular sector emptiness problem can be stated as follows.

Problem 2 *Given a set P of n points in the plane preprocess it so that, for a query circular sector σ of fixed radius r whose arc contains a fixed point t, one can efficiently determine whether σ contains any point of P.*

Circular sector σ can be partitioned into i) a triangle $\triangle bct$ and ii) a circular segment bounded by arc ct and

the chord connecting the endpoints of the arc. Notice that circular sector σ is void of P if and only if both the triangle and circular segment are void of P. Thus, Problem 2 can be reduced to the following two subproblems – 1) restricted triangular emptiness query and 2) restricted circular segment emptiness query.

Consider the restricted triangular emptiness problem stated below.

Subproblem 1 *Given a set P of n points in the plane, preprocess it so that, for a query triangle $\triangle$ with a vertex incident on a fixed point t, one can efficiently determine whether $\triangle$ contains any point of P.*

As proposed by Benbernou et al. [2], Subproblem 1 can be solved as follows. The points of P can be at first sorted around point t in counter-clockwise order. Consider a wedge formed by two rays emanating from point t. Let i and j be the first and last points, respectively, within the wedge in counter-clockwise order. Points i and j can be determined for any given wedge in $O(\log n)$ time. Based on this observation, with $O(n \log n)$ preprocessing space and time, a restricted triangular emptiness query can be answered in $O(\log n)$ time. Daescu et al. [5] also used a similar idea to build a data structure for halfplane farthest-point queries.

The result for Subproblem 1 is summarized in the following lemma.

Lemma 4 *A set P of n points in the plane and a fixed point t can be preprocessed in $O(n \log n)$ time into a data structure of size $O(n \log n)$ so that, for a query triangle $\triangle$ with a vertex incident on t, one can determine whether $\triangle$ contains any point of P in $O(\log n)$ time.*

The restricted circular segment emptiness problem is given as follows.

Subproblem 2 *Given a set P of n points in the plane, preprocess it so that for a query circular segment s, bounded by a circular arc originating from a fixed point t and the chord connecting the endpoints of the arc, one can efficiently determine if s contains any point of P.*

Let s_{ct} be a query circular segment bounded by circular arc ct (of a circle C of radius r) and the chord connecting points c and t. In order to determine if s_{ct} contains any point of P, we begin by finding its corresponding "enclosing" circular segment (or semi-circle) s_{pt} as illustrated in Figure 5. s_{pt} is a circular segment bounded by arc pt and the chord connecting points p and t. Given circular arc ct emanating from point t and running counter-clockwise, s_{pt} can be determined by extending the arc until it intersects with a circle D of radius $2r$ centered at point t. The case of clockwise circular segments can be handled symmetrically.

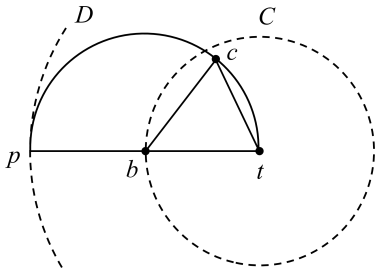


Figure 5: Circular segment s_{ct} and its "enclosing" circular segment s_{pt}.

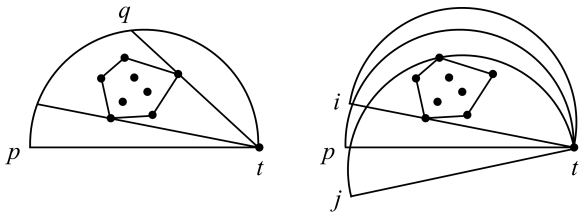


Figure 6: Circular segment s_{pt} and its corresponding event interval indicated by $[i, j]$.

Let $P_{pt} \subseteq P$ be the set of points in s_{pt}, and $CH(P_{pt})$ be the convex hull of P_{pt}. As shown in Figure 6, at most two tangent lines on the convex hull pass through point t. Let q be the intersection point between arc pt and the first of the two tangents in counter-clockwise order. If point c is located on arc qt, then s_{ct} is empty of P.

We now describe a preprocessing procedure based on the earlier observations. At first, observe that, as s_{pt} rotates about point t counter-clockwise, a point of P may enter and leave s_{pt}. Each of these point-entering and -leaving events can be determined in $O(1)$ time by computing the intersections between the boundary of s_{pt} and each point of P. Since a point of P can enter and leave s_{pt} at most once, the total number of point-entering and -leaving events is bounded by $2n$. These events can be sorted in counter-clockwise order in $O(n \log n)$ time.

Let s_{it} and s_{jt} be the circular segments associated with any two consecutive events in sorted order, where i and j are the endpoints of the bounding arcs (emanating from point t) for s_{it} and s_{jt}, respectively (see Figure 6, right). Notice that, the set of points of P in s_{pt} remains constant within this event interval. For each of these event intervals, the set of points of interest, their convex hull, and ultimately point q can be determined by using a dynamic convex hull data structure [3, 8], which requires $O(n)$ space, $O(n \log n)$ preprocessing time, $O(\log n)$ time per update operation, and $O(\log n)$ time for tangent queries. A simple $O(\log n)$ query-time data structure of linear size can then be built

to store point q for each event interval.

Thus, given a query circular segment s_{ct}, point p can be computed in $O(1)$ time, followed by a look-up of the event interval $[i, j]$ that contains p and its associated point q in $O(\log n)$ time. If the endpoint c of s_{ct} is located within arc qt, then s_{ct} does not contain any point of P.

Lemma 5 *For a fixed point t, a set P of n points in the plane can be preprocessed in $O(n \log n)$ time into a data structure of size $O(n)$ so that, given a query circular segment s, bounded by a circular arc originating from t and the chord connecting the endpoints of the arc, one can determine whether s contains any point of P in $O(\log n)$ time.*

Altogether, the following result is obtained for Problem 2.

Lemma 6 *For a positive number r and a fixed point t, a set P of n points in the plane can be preprocessed in $O(n \log n)$ time into a data structure of size $O(n \log n)$ so that, given a query circular sector σ with radius r and an endpoint of its arc located at t, one can determine whether σ contains any point of P in $O(\log n)$ time.*

Remark. We can solve the more general circular sector emptiness problem without a fixed radius or point t on the arc using a multilevel data structure similar to one by Matoušek [11] for counting points in the intersection of halfspaces (see also [5] for a similar approach on a related problem). Specifically, the first level is constructed for halfplane range queries to select the points of P lying on the σ side of the line supporting bc, and the second level is for halfplane range queries on the resulting points to select those lying on the σ side of bt. Thus, these two levels are used to find the points inside the wedge centered at b. Each subset of P on the second level is further preprocessed for nearest neighbor queries by computing its Voronoi diagram and augmenting it for point location. At query time, we can locate b in this data structure in logarithmic time. If the closest point is within distance r of b, then the circular sector is not empty of P. By following the strategy outlined in the first half of [11, Theorem 6.2], we can create a tradeoff between space and time usage by our data structure.

Lemma 7 *A set P of n points can be preprocessed into a data structure of size $O(m)$ in $O(m \log n)$ time so that, for a query circular sector σ of radius r centered at point p, one can determine whether σ is void of P in $O(n/m^{1/2} \log^{5/2} n)$ time for any $n^{1+\epsilon} \leq m \leq n^2$.*

Finishing up. According to Lemmas 3 and 6, the result for our case of the circular sector intersection problem can be stated as follows.

Lemma 8 *For a positive value r and a fixed point t, a set of P of n line segments can be preprocessed in $O(n \log n)$ time into a data structure of size $O(n \log n)$ so that, given a query circular sector σ with radius r whose circular arc has an endpoint at t, one can determine whether σ intersects P in $O(\log n)$ time.*

Let n_s be the number of endpoints of V within distance $2r$ from point t. Then, in Case II, given that $O(n_{in} n_{out} + n_{out}^2)$ queries are to be processed in the worst case and we only need to worry about endpoints lying sufficiently close to t, the following result is obtained.

Lemma 9 *A feasible articulated probe trajectory can be determined in $O((n_{in} n_{out} + n_{out}^2) \log n_s)$ time using $O(n_s \log n_s + n)$ space.*

Given that the space/time complexity of Case II (Lemma 9) is dominant over that of Case I (Lemma 2), the solution approach proposed herein for finding a feasible probe trajectory leads to the following theorem.

Theorem 10 *A feasible probe trajectory can be determined in $O((n_{in} n_{out} + n_{out}^2) \log n_s)$ time using $O(n_s \log n_s + n)$ space.*

Recall that $n_{in}, n_{out}, n_s \leq n$. Thus, the space usage and running time are bounded by $O(n \log n)$ and $O(n^2 \log n)$, respectively.

5 Conclusion

We presented an efficient geometric-combinatorial algorithm for a novel trajectory planning problem involving a simple articulated probe. Specifically, we can determine a feasible probe trajectory in $O(n^2 \log n)$ time. Our algorithm reduced to special cases of the circular sector intersection problem, for which we provided solutions.

A number of open problems remain: (1) Our algorithm works by enumerating over a set of possible "extremal" solutions. Can it be sped up, possibly by skipping some of these solutions? (2) Can the algorithm be extended to find a representation of all feasible probe trajectories? (3) Can the space usage of the special circular sector queries be reduced to $O(n)$? We conjecture that our result would then be optimal. (4) Can we find an efficient general data structure for circular arc ray shooting queries among (disjoint or intersecting) line segments?

Acknowledgements. The authors would like to thank Pankaj K. Agarwal and Carola Wenk for helpful discussions and pointers to related literature.

References

[1] E. Arkin and J. Mitchell. An optimal visibility algorithm for a simple polygon with star-shaped holes. Technical report, Cornell University Operations Research and Industrial Engineering, 1987.

[2] N. M. Benbernou, M. Ishaque, and D. L. Souvaine. Data structures for restricted triangular range searching. In *20th Annual Canadian Conference on Computational Geometry*, pages 15–18. Citeseer, 2008.

[3] G. S. Brodal and R. Jacob. Dynamic planar convex hull. In *Foundations of Computer Science, 2002. Proceedings. The 43rd Annual IEEE Symposium on*, pages 617–626. IEEE, 2002.

[4] R. Connelly and E. D. Demaine. Geometry and topology of polygonal linkages. *Handbook of discrete and computational geometry*, pages 233–256, 2017.

[5] O. Daescu, N. Mi, C. Shin, and A. Wolff. Farthest-point queries with geometric and combinatorial constraints. *Computational Geometry*, 33(3):174–185, 2006.

[6] P. J. Heffernan and J. S. Mitchell. An optimal algorithm for computing visibility in the plane. *SIAM Journal on Computing*, 24(1):184–201, 1995.

[7] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. *Information Processing Letters*, 33(4):169–174, 1989.

[8] J. Hershberger and S. Suri. Off-line maintenance of planar configurations. *Journal of Algorithms*, 21(3):453–475, 1996.

[9] J. Hopcroft, D. Joseph, and S. Whitesides. Movement problems for 2-dimensional linkages. *SIAM Journal on Computing*, 13(3):610–629, 1984.

[10] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[11] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete & Computational Geometry*, 10(2):157–182, 1993.

[12] M. Pocchiola and G. Vegter. The visibility complex. *International Journal of Computational Geometry & Applications*, 6(03):279–308, 1996.

[13] S. Rivière. Topologically sweeping the visibility complex of polygonal scenes. In *Proceedings of the Eleventh Annual Symposium on Computational Geometry*, pages 436–437. ACM, 1995.

[14] M. Sharir and P. K. Agarwal. *Davenport-Schinzel sequences and their geometric applications*. Cambridge university press, 1995.

[15] S. Suri and J. O'Rourke. Worst-case optimal algorithms for constructing visibility polygons with holes. In *Proceedings of the Second Annual Symposium on Computational Geometry*, pages 14–23. ACM, 1986.

Appendix

For case I, suppose that a feasible probe trajectory T exists, such that the final pose of the probe is unarticulated and point c coincides with point t. In other words, t has unobstructed vision to some points on the bounding circle S. Let T' be the trajectory resulting from rotating T about point t in clockwise direction until T intersects an obstacle endpoint v. It is apparent that T' is also a feasible trajectory, and its articulation point b' is the intersection of segment vt and circle C.

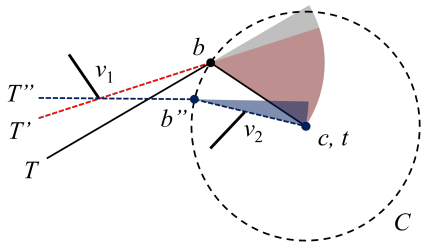


Figure 7: Case II of Lemma 1. T'' represents a feasible articulated probe trajectory such that the final configuration of the probe intersects an obstacle endpoint v_1 outside C and an obstacle endpoint v_2 inside C.

For case II, assume that a feasible trajectory T exists such that the final pose of the probe is articulated (i.e., line segments ab and bc are not collinear) and point c coincides with point t (Figure 7). Suppose probe trajectory T rotates segment bc clockwise around b to reach point t; the other case uses symmetric arguments. Let T' be the trajectory resulting from rotating line segment ab of T about point b in clockwise direction until line segment ab intersects an obstacle endpoint v_1 outside C. Given that the area swept by line segment bc of T' is within that of T (indicated by the shaded circular sectors in Figure 7), T' is also a feasible trajectory.

Now, let T'' be the trajectory obtained by rotating line segment bc of T' about point t in counter-clockwise direction while simultaneously rotating ab around v_1 in the clockwise direction until either abc becomes a line segment or either ab or bc intersects some obstacle endpoint v_2. Note that as T' changes into T'' point b of T' slides on circle C in counter-clockwise direction into a new position b''. If abc becomes a line segment, we have achieved case I of the lemma. We now assume otherwise.

Observe that every point of the circular sector centered at b'' lies on the t side of the line through v_1 and b. They also lie on the b side of the line through b'' and t. Therefore, these points either lie in the circular sector of radius r centered at t with arc endpoints at b and b'', or they lie in the wedge emanating from the circular sector centered at b. We know the sector centered at t is empty, because it was swept while constructing T''. We now argue the remaining points of the sector centered at b'' not only lie in the wedge from b, but they actually lie in the circular sector centered at b. Because T' is a feasible probe trajectory, the sector at b and therefore the whole sector at b'' is empty as well, and T'' is a feasible probe trajectory.

Indeed, let x be a point of the sector centered at b'' that

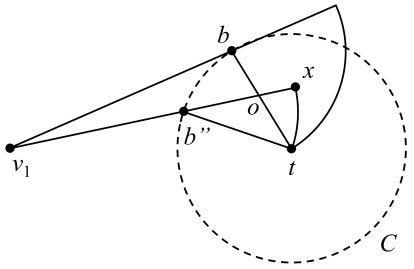


Figure 8: Point x lies inside the circular sector centered at b.

lies in the wedge at b. Let o be the intersection of the line segments bt and $b''x$ (Figure 8). By the triangle inequality,

$$\begin{aligned} |bx| &\leq |bo| + |ox| \\ &= |bt| - |ot| + |b''x| - |b''o| \\ &\leq |bt| + |b''x| - |b''t| \\ &= |b''x| \\ &\leq r \end{aligned}$$

If v_2 is inside circle C, then point b'' is the intersection between circle C and a ray emanating from point t through v_2. Otherwise, both v_1 and v_2 lie on the line segment ab''.

Distance-Two Colorings of Barnette Graphs

Tomás Feder* Pavol Hell† Carlos Subi‡

Abstract

Barnette identified two interesting classes of cubic polyhedral graphs for which he conjectured the existence of a Hamiltonian cycle. Goodey proved the conjecture for the intersection of the two classes. We examine these classes from the point of view of distance-two colorings. A distance-two r-coloring of a graph G is an assignment of r colors to the vertices of G so that any two vertices at distance at most two have different colors. Note that a cubic graph needs at least four colors. The distance-two four-coloring problem for cubic planar graphs is known to be NP-complete. We claim the problem remains NP-complete for tri-connected bipartite cubic planar graphs, which we call type-one Barnette graphs, since they are the first class identified by Barnette. By contrast, we claim the problem is polynomial for cubic plane graphs with face sizes $3, 4, 5$, or 6, which we call type-two Barnette graphs, because of their relation to Barnette's second conjecture. We call Goodey graphs those type-two Barnette graphs all of whose faces have size 4 or 6. We fully describe all Goodey graphs that admit a distance-two four-coloring, and characterize the remaining type-two Barnette graphs that admit a distance-two four-coloring according to their face size.

For quartic plane graphs, the analogue of type-two Barnette graphs are graphs with face sizes 3 or 4. For this class, the distance-two four-coloring problem is also polynomial; in fact, we can again fully describe all colorable instances – there are exactly two such graphs.

1 Introduction

Tait conjectured in 1884 [22] that all cubic polyhedral graphs, i.e., all tri-connected cubic planar graphs, have a Hamiltonian cycle; this was disproved by Tutte in 1946 [24], and the study of Hamiltonian cubic planar graphs has been a very active area of research ever since, see for instance [1, 11, 17, 19]. Barnette formulated two conjectures that have been at the centre of much of the effort: (1) that *bipartite* tri-connected cubic planar graphs are Hamiltonian (the case of Tait's conjecture where all face sizes are even) [4], and (2) that tri-connected cubic planar graphs with all face sizes $3, 4, 5$ or 6 are Hamiltonian, cf. [3, 20]. Goodey [12, 13] proved that the conjectures hold on the intersection of the two classes, i.e., that tri-connected cubic planar graphs with all face sizes 4 or 6 are Hamiltonian. When all faces have sizes 5 or 6, this was a longstanding open problem, especially since these graphs (tri-connected cubic planar graphs with all face sizes 5 or 6) are the popular *fullerene graphs* [8]. The second conjecture has now been affirmatively resolved in full [18]. For the first conjecture, two of the present authors have shown in [10] that if the conjecture is false, then the Hamiltonicity problem for tri-connected cubic planar graphs is NP-complete. In view of these results and conjectures, in this paper we call bipartite tri-connected cubic planar graphs *type-one Barnette graphs*; we call cubic plane graphs with all face sizes $3, 4, 5$ or 6 *type-two Barnette graphs*; and finally we call cubic plane graphs with all face sizes 4 or 6 *Goodey graphs*. Note that it would be more logical, and historically accurate, to assume tri-connectivity also for type-two Barnette graphs and for Goodey graphs. However, we prove our positive results without needing tri-connectivity, and hence we do not assume it.

Cubic planar graphs have been also of interest from the point of view of colorings [6, 15]. In particular, they are interesting for distance-two colourings. Let G be a graph with degrees at most d. A *distance-two r-coloring* of G is an assignment of colors from $[r] = \{1, 2, \ldots, r\}$ to the vertices of G such that if a vertex v has degree $d' \leq d$ then the $d' + 1$ colors of v and of all the neighbors of v are all distinct. (Thus a distance-two coloring of G is a classical coloring of G^2.) Clearly a graph with maximum degree d needs at least $d + 1$ colors in any distance-two coloring, since a vertex of degree d and its d neighbours must all receive distinct colors. It was conjectured by Wegner [25] that a planar graph with maximum degree d has a distance-two r-colouring where $r = 7$ for $d = 3$, $r = d + 5$ for $d = 4, 5, 6, 7$, and $r = \lfloor 3d/2 \rfloor + 1$ for all larger d. The case $d = 3$ has been settled in the positive by Hartke, Jahanbekam and Thomas [14], cf. also [23].

For cubic planar graphs in general it was conjectured in [14] that if a cubic planar graph is tri-connected, or has no faces of size five, then it has a distance-two six-coloring. We propose a weaker version of the second case of the conjecture, namely, we conjecture that *a bipartite cubic planar graph can be distance-two six-colored.* We prove this in one special case (Theorem 5), which of course also confirms the conjecture of Hartke, Jahanbekam and Thomas for that case. Heggerness and

*Palo Alto, tomas@theory.stanford.edu
†School of Computing Science, SFU, pavol@sfu.ca
‡Los Altos Hills, carlos.subi@hotmail.com

Telle [16] have shown that the problem of distance-two four-coloring cubic planar graphs is NP-complete. On the other hand, Borodin and Ivanova [5] have shown that subcubic planar graphs of girth at least 22 can be distance-two four-colored. In fact, there has been much attention focused on the relation of distance-two colorings and the girth, especially in the planar context [5, 15].

Our results focus on distance-two colorings of cubic planar graphs, with particular attention on Barnette graphs, of both types. We prove that a cubic plane graph with all face sizes divisible by four can always be distance-two four-colored, and a give a simple condition for when a bi-connected cubic plane graph with all face sizes divisible by three can be distance-two four-colored using only three colors per face. It turns out that the distance-two four-coloring problem for type-one Barnette graphs is NP-complete, while for type-two Barnette graphs it is not only polynomial, but the positive instances can be explicitly described. They include one infinite family of Goodey graphs (cubic plane graphs with all faces of size 4 or 6), and all type-two Barnette graphs which have all faces of size 3 or 6. Interestingly, there is an analogous result for quartic (four-regular) graphs: all quartic planar graphs with faces of only sizes 3 or 4 that have a distance-two five coloring can be explicitly described; there are only two such graphs.

Note that we use the term "plane graph" when the actual embedding is used, e.g., when discussing the faces; on the other hand, when the embedding is unique, as in tri-connected graphs, we stick with writing "planar".

The proofs omitted here can be found in [9].

2 Relations to edge-colorings and face-colorings

Distance-two colorings have a natural connection to edge-colorings.

Theorem 1 *Let G be a graph with degrees at most d that admits a distance-two $(d+1)$-coloring, with d odd. Then G admit an edge-coloring with d colors.*

Proof. The even complete graph K_{d+1} can be edge-colored with d colors by the Walecki construction [2]. We fix one such coloring c, and then consider a distance-two $(d+1)$-coloring of G. If an edge uv of G has colors ab at its endpoints, we color uv in G with the color $c(ab)$. It is easy to see that this yields an edge-coloring of G with d colors. □

We call the resulting edge-coloring of G the *derived edge-coloring* of the original distance-two coloring.

In this paper, we mostly focus on the case $d = 3$ (the *subcubic* case). Thus we use the edge-coloring of K_4 by colors red, blue, green. This corresponds to the unique partition of K_4 into perfect matchings. Note that for every vertex v of K_4 and every edge-color i, there is a unique other vertex u of K_4 adjacent to v in edge-color i. Thus if we have the derived edge-coloring we can efficiently recover the original distance-two coloring. In the subcubic case, in turns out to be sufficient to have just one color class of the edge-coloring of G.

Theorem 2 *Let G be a subcubic graph, and let R be a set of red edges in G. The question of whether there exists a distance-two four-coloring of G for which the derived edge-coloring has R as one of the three color classes can be solved by a polynomial time algorithm. If the answer is positive, the algorithm will identify such a distance-two coloring.*

There is also a relation to face-colorings. It is a folklore fact that the faces of any bipartite cubic plane graph G can be three-colored [21]. This three-face-coloring induces a three-edge-coloring of G by coloring each edge by the color not used on its two incident faces. (It is easy to see that this is in fact an edge-coloring, i.e., that incident edges have distict colors.) We call an edge-coloring that arises this way from some face-coloring of G a *special three-edge-coloring* of G. We first ask when is a special three-edge-coloring of G the derived edge-coloring of a distance-two four-coloring of G.

Theorem 3 *A special three-edge-coloring of G is the derived edge-coloring of some distance-two four-coloring of G if and only if the size of each face is a multiple of 4.*

Proof. The edges around a face f alternate in colors, and the vertices of f can be colored consistently with this alternation if and only if the size of f is a multiple of 4. This proves the "only if" part. For the "if" part, suppose all faces have size multiple of 4. If there is an inconsistency, it will appear along a cycle C in G. If there is only one face inside C, there is no inconsistency. Otherwise we can join some two vertices of C by a path P inside C, and the two sides of P inside C give two regions that are inside two cycles C', C''. The consistency of C then follows from the consistency of each of C', C'' by induction on the number of faces inside the cycle. □

Corollary 4 *Let G be a cubic plane graph in which the size of each face is a multiple of four. Then G can be distance-two four-colored.*

We now prove a special case of the conjecture stated in the introduction, that all bipartite cubic plane graphs can be distance-two six-colored. Recall that the faces of any bipartite cubic plane graph can be three-colored.

Theorem 5 *Suppose the faces of a bipartite cubic plane graph G are three-colored red, blue and green, so that the*

red faces are of arbitrary even size, while the size of each blue and green face is a multiple of 4. *Then* G *can be distance-two six-colored.*

Proof. Let G' be the multigraph obtained from G by shrinking each of the red faces. Clearly G' is planar, and since the sizes of blue and green faces in G' are half of what they were in G, they will be even, so G' is also bipartite. Let us label the two sides of the bipartition as A and B respectively. Now consider the special three-edge coloring of G associated with the face coloring of G. Each red edge in this special edge-coloring joins a vertex of A with a vertex of B; we orient all red edges from A to B. Now traversing each red edge in G in the indicated orientation either has a blue face on the left and green face on the right, or a green face on the left and blue face on the right. In the former case we call the edge *class one* in the latter case we call it *class two*. Each vertex of G is incident with exactly one red edge; the vertex inherits the class of its red edge. The vertices around each red face in G are alternatingly in class 1 and class 2. We assign colors $1, 2, 3$ to vertices of class one and colors $4, 5, 6$ to vertices of class two. It remains to decide how to choose from the three colors available for each vertex. A vertex adjacent to red edges in class i has only three vertices within distance two in the same class, namely the vertex across the red edge, and the two vertices at distance two along the red face in either direction. Therefore distance-two coloring for class i corresponds to three-coloring a cubic graph. Since neither class can yield a K_4, such a three-coloring exists by Brooks' theorem [7]. This yields a distance-two six-coloring of G. □

3 Distance-two four-coloring of type-one Barnette graphs is NP-complete

We now state our main intractability result.

Theorem 6 *The distance-two four-coloring problem for tri-connected bipartite cubic planar graphs is NP-complete.*

In this note we only derive the following weaker version of our claim. (See [9] for the proof of the entire claim.)

Theorem 7 *The distance-two four-coloring problem for bipartite planar subcubic graphs is NP-complete.*

Proof. Consider the graph H in Figure 1.

We will reduce the problem of H-coloring planar graphs to the distance-two four-coloring problem for bipartite planar subcubic graphs. In the *H-coloring problem* we are given a planar graph G and and the question is whether we can color the vertices of G with colors that are vertices of H so that adjacent vertices of G obtain adjacent colors. This can be done if and only if G is three-colorable, since the graph H both contains a triangle and is three-colorable itself. (Thus any three-coloring of G is an H-coloring of G, and any H-coloring of G composed with a three-coloring of H is a three-coloring of G.) It is known that the three-coloring problem for planar graphs is NP-complete, hence so is the H-coloring problem.

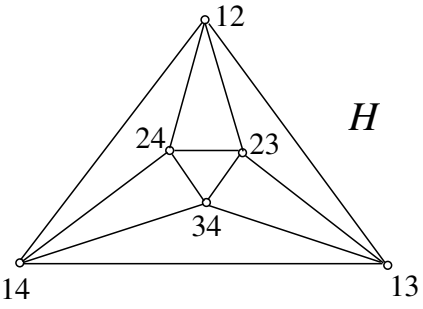


Figure 1: The graph H for the proof of Theorem 7

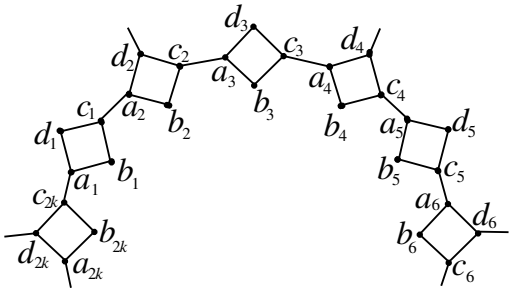


Figure 2: The ring gadget

Thus suppose G is an instance of the H-coloring problem. We form a new graph G' obtained from G by replacing each vertex v of G by a *ring* gadget depicted in Figure 2. If v has degree k, the ring gadget has $2k$ squares. A *link* in the ring is a square $a_i b_i c_i d_i a_i$ followed by the edge $c_i a_{i+1}$. A link is *even* if i is even, and *odd* otherwise. Every even link in the ring will be used for a connection to the rest of the graph G', thus vertex v has k available links. For each edge vw of G we add a new vertex f_{vw} that is adjacent to a vertex d_s in one available link of the ring for v and a vertex d'_t in one available link of the ring for w. (We use primed letters for the corresponding vertices in the ring of w to distinguish them from those in the ring of v.) The actual choice of (the even) subscripts s, t does not matter, as long as each available link is only used once. The resulting graph is clearly subcubic and planar. It is also bipartite, since

we can bipartition all its vertices into one independent set A consisting of all the vertices $a_i, c_i, b_{i+1}, di + 1$ with odd i in all the rings, and another independent set B consisting of the vertices $a_i, c_i, b_{i+1}, di + 1$ with even i in all the rings. Moreover, we place all vertices f_{vw} into the set A. Note that in any distance-two four-coloring of the ring, each link must have four different colors for vertices a_i, b_i, c_i, d_i, and the same color for a_i and a_{i+1}. Thus all a_i have the same color and all c_i have the same color. The pair of colors in b_i, d_i is also the same for all i; we will call it the *characteristic pair of the ring for* v. For any pair ij of colors from $1, 2, 3, 4$, there is a distance-two coloring of the ring that has the characteristic pair ij.

One can prove that G is H-colorable if and only if G' is distance-two four-colorable. □

We remark that (with some additional effort) we can prove that the problem is still NP-complete for the class of tri-connected bipartite cubic planar graphs with no faces of sizes larger than 44.

4 Distance-two four-coloring of Goodey graphs

Recall that Goodey graphs are type-two Barnette graph with all faces of size 4 and 6 [12, 13]. In other words, a *Goodey graph* is a cubic plane graph with all faces having size 4 or 6. By Euler's formula, a Goodey graph has exactly six square faces, while the number of hexagonal faces is arbitrary.

A *cyclic prism* is the graph consisting of two disjoint even cycles $a_1a_2 \cdots a_{2k}a_1$ and $b_1b_2 \cdots b_{2k}b_1, k \geq 2$, with the additional edges a_ib_i, $1 \leq i \leq 2k$. It is easy to see that cyclic prisms have either no distance-two four-coloring (if k is odd), or a unique distance-two four-coloring (if $k \geq 2$ is even). Only the cyclic prisms with $k = 2, 3$ are Goodey graphs, and thus from Goodey cyclic graphs only the cube (the case of $k = 2$) has a distance-two coloring, which is moreover unique.

In fact, all Goodey graphs that admit distance-two four-coloring can be constructed from the cube as follows. The Goodey graph C_0 is the cube, i.e., the cyclic prism with $k = 2$. The Goodey graph C_1 is depicted in Figure 4. It is obtained from the cube by separating the six square faces and joining them together by a pattern of hexagons, with three hexagons meeting at a vertex tying together the three faces that used to meet in one vertex. The higher numbered Goodey graphs are obtained by making the connecting pattern of hexagons higher and higher. The next Goodey graph C_2 has two hexagons between any two of the six squares, with a central hexagon in the centre of any three of the squares, the following Goodey graph C_3 has three hexagons between any two of the squares and three hexagons in the middle of any three of the squares, and so on. Thus in general we replace every vertex of the cube by a triangular pattern of hexagons whose borders are replacing the edges of the cube. We illustrate the vertex replacement graphs in Figure 3, without giving a formal description. The entire Goodey graph C_1 is depicted in Figure 4.

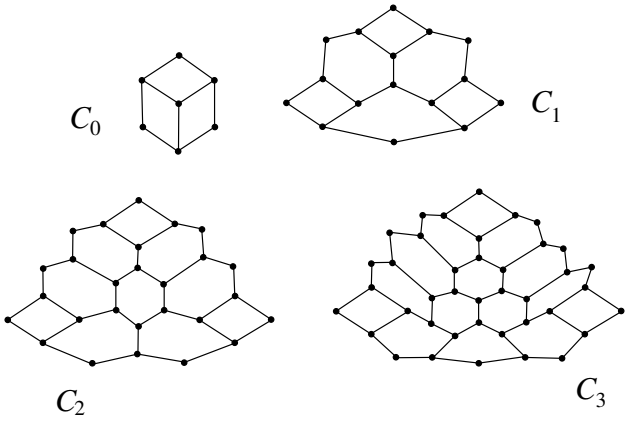


Figure 3: The vertex replacements for Goodey graphs C_0, C_1, C_2, and C_3

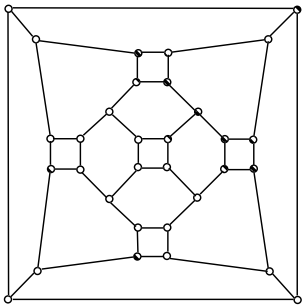


Figure 4: The Goody graph C_1

We have the following results.

Theorem 8 *The Goodey graphs $C_k, k \geq 0$, have a unique distance-two four-coloring, up to permutation of colors.*

Theorem 9 *The Goodey graphs $C_k, k \geq 0$, are the only bipartite cubic planar graphs having a distance-two four-coloring.*

We can therefore conclude the following.

Corollary 10 *The distance-two four-coloring problem for Goodey graphs is solvable in polynomial time.*

Recognizing whether an input Goodey graph is some C_k can be achieved in polynomial time; in the same time bound G can actually be distance-two four-colored.

5 Distance-two four-coloring of type-two Barnette graphs is polynomial

We now return to general type-two Barnette graphs, i.e., cubic plane graphs with face sizes $3, 4, 5$, or 6. As a first step, we analyze when a general cubic plane graph admits a distance-two four-coloring which has three colors on the vertices of every face of G.

Theorem 11 *A cubic plane graph G has a distance-two four-coloring with three colors per face if and only if*

1. *all faces in G have size which is a multiple of 3,*
2. *G is bi-connected, and*
3. *if two faces share more than one edge, the relative positions of the shared edges must be congruent modulo 3 in the two faces.*

The last condition means the following: if faces F_1, F_2 meet in edges e, e' and there are n_1 edges between e and e' in (some traversal of) F_1, and n_2 edges between e and e' in (some traversal of) F_2, then $n_1 \equiv n_2 \mod 3$.

Proof. Suppose G has a distance-two four-coloring with three colors in each face. The unique way to distance-two color a cycle with colors $1, 2, 3$ is by repeating them in some order $(123)^*$ along one of the two traversals of the cycle. Therefore the length is a multiple of 3 so (1) holds. Moreover, there can be no bridge in G as that would imply a face that self-intersects and is traversed in opposite directions along any traversal of that face, disagreeing with the order $(123)^*$ in one of them; thus (2) also holds. Finally, (3) holds because the common edges must have the same colors in both faces.

Suppose the conditions hold, and consider the dual G^D of G. (Note that each face of G^D is a triangle.) We find a distance-two coloring of G as follows. Let F be a face in G; according to conditions (1-2), its vertices can be distance-two colored with three colors. That takes care of the vertex F in G^D. Using condition (3), we can extend the coloring of G to any face F' adjacent to F in G^D. Note that we can use the fourth colour, 4, on the two vertices adjacent in F' to the two vertices of a common edge. In this way, we can propagate the distance-two coloring of G along the adjacencies in G^D. If this produces a distance-two coloring of all vertices of G, we are done. Thus it remains to show there is no inconsistency in the propagation. If there is an inconsistency, it will appear along a cycle C in G^D. If there is only one face inside of C, then C is a triangle corresponding to a vertex of G, and there is no inconsistency. Otherwise we can join some two vertices of C by a path P inside C, and the two sides of P inside C give two regions that are inside two cycles C', C''. The consistency of C then follows from the consistency of each of C', C'' by induction on the number of faces inside the cycle. □

It turns out that conditions (1 - 3) are automatically satisfied for cubic plane graphs with faces of sizes 3 or 6.

Corollary 12 *Type-two Barnette graphs with faces of sizes 3 or 6 are distance-two four-colorable.*

Proof. Such a graph must be bi-connected, i.e., cannot have a bridge, since no triangle or hexagon can self-intersect. Moreover, only two hexagons can have two common edges, and it is easy to check that they must indeed be in relative positions congruent modulo 3 on the two faces. (Since all vertices must have degree three.) Thus the result follows from Theorem 11. □

Theorem 13 *Let G be type-two Barnette graph. Then G is distance-two four-colorable if and only if it is one of the graphs $C_k, k \geq 0$, or all faces of G have sizes 3 or 6.*

Proof. If there are faces of size both 3 and 4 (and possibly size 6), then there must be (by Euler's formula) two triangles and three squares, and as in the proof of Theorem 9, the squares must be joined by chains of hexagons, which is not possible with just three squares.

If there is a face of size 5, then there is no distance-two four-coloring since all five vertices of that face would need different colors. □

6 Distance-two coloring of quartic graphs

A *quartic graph* is a regular graph with all vertices of degree four. Thus any distance-two coloring of a quartic graph requires at least five colors. A *four-graph* is a plane quartic graph whose faces have sizes 3 or 4. The argument to view these as analogues of type-two Barnette graphs is as follows. For cubic plane Euler's formula limits the numbers of faces that are triangles, squares, and pentagons, but does not limit the number of hexagon faces. Similarly, for plane quartic graphs, Euler's formula implies that such a graph must have 8 triangle faces, but places no limits on the number of square faces.

We say that two faces are *adjacent* if they share an edge.

Lemma 14 *If a four-graph can be distance-two five-colored, then every square face must be adjacent to a triangle face. Thus G can have at most 24 square faces.*

Proof. We view the numbers $1, 2, 3, 4$ modulo 4, and number 5 is separate. Let $u_1u_2u_3u_4$ be a square face that has no adjacent triangle face. (This is depicted in Figure 5 as the square in the middle.) Color u_i by i. Let

the adjacent square faces be $u_iu_{i+1}w_{i+1}v_i$. One of v_i, w_i must be colored 5 and the other one $i+2$. Then either all v_i or all w_i are colored 5, say all w_i are colored 5, and all v_i are colored $i+1$. Then $v_iu_iw_i$ cannot be a triangle face, or w_i, w_{i+1} would be both colored 5 at distance two. Therefore $t_iv_iu_iw_i$ must be a square face. (In the figure, this is indicated by the corner vertices being marked by smaller circles; these must exist to avoid a triangle face.) This means that the original square is surrounded by eight square faces for $u_1u_2u_3u_4$, and t_i must have color $i+3$, since u_i, v_{i+3}, v_i, w_i have colors $i, i+1, i+2, 5$.

But then there cannot be a triangle face $x_iv_iw_{i+1}$, since x_i is within distance two of u_i, u_{i+1}, v_i, t_i, w_{i+1} of colors $i, i+1, i+2, i+3, 5$, so each of the adjacent square faces $u_iu_{i+1}w_{i+1}v_i$ for $u_1u_2u_3u_4$ has adjacent square faces as well. This process of moving to adjacent square faces eventually reaches all faces as square faces, contrary to the fact that there are 8 triangle faces. □

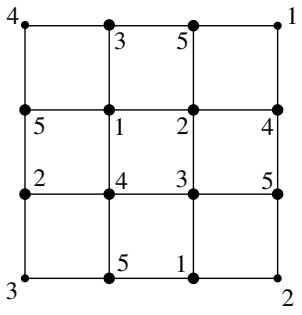


Figure 5: One square without adjacent triangles implies all faces must be squares

It follows that there are only finitely many distance-two five-colorable four-graphs.

Corollary 15 *The distance-two five-coloring problem for four-graphs is polynomial.*

In fact, we can fully describe all four-graphs that are distance-two five-colorable. Consider the four-graphs G_0, G_1 given in Figure 6. The graph G_0 has 8 triangle faces and 4 square faces, the graph G_1 has 8 triangle faces and 24 square faces. Note that G_0 is obtained from the cube by inserting two vertices of degree four in two opposite square faces. Similarly, G_1 is obtained from the cube by replacing each vertex with a triangle and inserting into each face of the cube a suitably connected degree four vertex. (In both figures, these inserted vertices are indicated by smaller size circles.)

Theorem 16 *The only four-graphs G that can be distance-two five-colored are G_0, G_1. These two graphs can be so colored uniquely up to permutation of colors.*

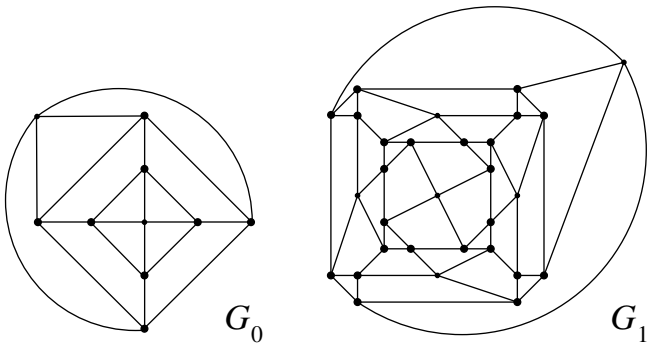


Figure 6: The only four-graphs that admit a distance-two five-coloring

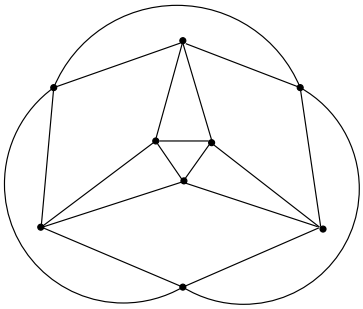


Figure 7: A four-graph requiring nine colors in any distance-two coloring

We close with a few remarks and open problems.

Wegner's conjecture [25] that any planar graph with maximum degree $d = 3$ can be distance-two seven-colored has been proved in [14, 23]. That bound is actually achieved by a type-two Barnette graph, namely the graph obtained from K_4 by subdividing three incident edges. Thus the bound of 7 cannot be lowered even for type-two Barnette graphs.

Wegner's conjecture for $d = 4$ claims that any planar graph with maximum degree four can be distance-two nine-colored. The four-graph in Figure 7 actually requires nine colors in any distance-two coloring. Thus if Wegner's conjecture for $d = 4$ is true, the bound of 9 cannot be lowered, even in the special case of four-graphs. It would be interesting to prove Wegner's conjecture for four-graphs, i.e., to prove that *any four-graph can be distance-two nine-colored.*

Finally, we've conjectured that any bipartite cubic planar graph can be distance-two six-colored (a special case of a conjecture of Hartke, Jahanbekam and Thomas [14]). The hexagonal prism (a cyclic prism with $k = 3$, which is a Goodey graph), actually requires six colors. Hence if our conjecture is true, the bound of 6 cannot be lowered even for Goodey graphs. It would be interesting to prove our conjecture for Goodey graphs, i.e., to prove that *any Goodey graph can be distance-two six-colored.*

References

[1] R.E.L. Aldred, S. Bau, D.A. Holton, and B.D. MacKay. Non-hamiltonian 3-connected cubic planar graphs. *SIAM J. Discrete Math.* 13:25–32, 2000.

[2] B. Alspach. The wonderful Walecki construction. *Bull. Inst. Combin. Appl.* 52:7–20, 2008.

[3] D. Barnette. On generating planar graphs. *Discrete Math.* 7:199–208, 1974.

[4] D. Barnette. Conjecture 5. *Recent Progress in Combinatorics* (Ed. W.T. Tutte), Academic Press, New York 343, 1969.

[5] O.V. Borodin and A.O. Ivanova. 2-distance 4-colorability of planar subcubic graphs with girth at least 22. *Discussiones Math. Graph Theory* 32:141–151, 2012.

[6] O.V. Borodin. Colorings of plane graphs: a survey, *Discrete Math.* 313:517–539, 2013.

[7] R.L. Brooks, On coloring the nodes of a network. *Proc. Cambridge Philos. Soc.* 37:194–197, 1941.

[8] R. Erman, F. Kardoš, J. Miskuf. Long cycles in fullerene graphs. *J. Math. Chemistry* 46:1103–1111, 2009.

[9] T. Feder, P. Hell, and C. Subi. Distance-two colorings of Barnette graphs *arXiv*:1807.01410 [cs.CG]

[10] T. Feder and C. Subi. On Barnette's conjecture. *Electronic Colloquium on Computational Complexity (ECCC)* TR06-015, 2006.

[11] M.R. Garey, D.S. Johnson, and R.E. Tarjan. The planar Hamiltonian circuit problem is NP-complete. *SIAM J. Comput.* 5:704–714, 1976.

[12] P.R. Goodey. Hamiltonian circuits in polytopes with even sided faces. *Israel J. Math.* 22:52–56, 1975.

[13] P.R. Goodey. A class of Hamiltonian polytopes. *J. Graph Theory* 1:181–185, 1977.

[14] S.G. Hartke, S. Jahanbekam, and B. Thomas. The chromatic number of the square of subcubic planar graphs. *arXiv*:1604.06504.

[15] F. Havet. Choosability of the square of planar subcubic graphs with large girth. *Discrete Math.* 309:3353–3563, 2009.

[16] P. Heggernes and J.A. Telle. Partitioning graphs into generalized dominating sets. *Nordic J. Computing* 5:128–143, 1998.

[17] D.A. Holton, B.D. McKay. The smallest non-Hamiltonian 3-connected cubic planar graphs have 38 vertices. *J. Combin. Theory B* 45:305–319, 1988.

[18] F. Kardoš. A computer-assisted proof of Barnette-Goodey conjecture: Not only fullerene graphs are Hamiltonian. *arXiv math*:1409.2440

[19] X. Lu. A note on 3-connected cubic planar graphs. *Discrete Math.* 310:2054–2058, 2010.

[20] J. Malkevitch. Polytopal graphs. *in Selected Topics in Graph Theory* (L. W. Beineke and R. J. Wilson eds.) 3:169–188, 1998.

[21] O. Ore. The four colour problem. *Academic Press*, 1967.

[22] P.G. Tait. Listing's topologie. *Philosophical Magazine*, 5th Series 17:30-46, 1884. Reprinted in Scientific Papers, Vol. II, pp. 85–98.

[23] C. Thomassen. The square of a planar cubic graph is 7-colorable. manuscript 2006.

[24] W.T. Tutte. On hamiltonian circuits. *J. Lond. Math. Soc.* 21:98–101, 1946.

[25] G. Wegner. Graphs with given diameter and a coloring problem. Technical Report, University of Dortmund, Germany, 1977.

Emanation Graph: A New t-Spanner*

Bardia Hamedmohseni† Zahed Rahmati‡ Debajyoti Mondal§

Abstract

We introduce a new *t-spanner*, called *emanation graph* M_k, based on the idea of shooting rays out of each vertex at specific angles, determined by k, the *grade* of the emanation graph. Emanation graphs of grade one coincide with the competition mesh, which was studied by Mondal and Nachmanson [18] in the context of network visualization. They proved that the spanning ratio of such a graph is bounded by $(2+\sqrt{2}) \approx 3.41$.

In this paper, we prove an improved $\sqrt{10} \approx 3.162$ upper bound on the spanning ratio of emanation graphs of grade one, which in fact improves the previous result. We also prove that the spanning ratio of the emanation graphs of grade k is at least $\frac{2+\sin(\frac{\pi}{2^k})}{1+\cos(\frac{\pi}{2^k})}$, for sufficiently large n.

1 Introduction

Let G be a geometric graph embedded in the Euclidean plane, and let u and v be a pair of vertices in G. Let $d_G(u,v)$ and $d_E(u,v)$ be the minimum graph distance (i.e., shortest path distance) and Euclidean distance between u and v, respectively. The *spanning ratio* of G is $\max_{\{u,v\}\in G} \frac{d_G(u,v)}{d_E(u,v)}$, i.e., the maximum ratio between $d_G(u,v)$ and $d_E(u,v)$ over all pairs of vertices $\{u,v\}$ in G. Graph G is called a *t-spanner* of the complete geometric graph, if for every pair of vertices $\{u,v\}$ in G, the distance $d_G(u,v)$ is at most t times of their Euclidean distance $d_E(u,v)$.

The *t-spanners* are commonly used in computational geometry. They also find applications in wireless network routing [9] and in network visualizations [18, 19]. A rich body of research is devoted towards the construction of t-spanners, and there has also been significant efforts to find tight spanning ratios for different classes of geometric graphs.

In this paper, we examine *plane geometric spanners*, i.e., no two edges in the spanner cross except at their common endpoints. A natural question in this context is as follows: Given a set of points P of n points in the plane, can we compute a planar spanner $G = (V, E)$ of P with small size, degree and spanning ratio? We allow the spanner to have Steiner points, i.e., $P \subseteq V$, thus V may contain vertices that do not correspond to any point of P. Note that keeping the degree, size and spanning ratio of the spanners small are often motivated by application areas, and appeared in the literature [9, 8]. Note that we do not require the paths between a pair of Steiner points to be bounded.

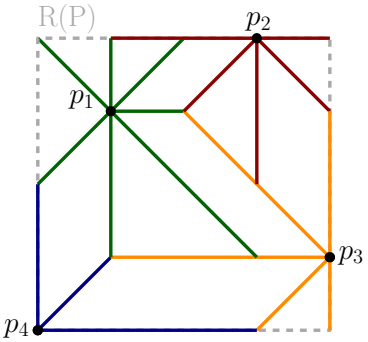


Figure 1: The emanation graph of grade two, for four points in the Euclidean plane.

In this paper, we introduce a new type of *t-spanners*, called the *emanation graph*. Given a set P of n points in a bounding box $R(P)$, and an integer $k > 0$, the *emanation graph M_k of grade k* is constructed by emanating, from each point $p_i \in P$, 2^{k+1} rays with equal angular distances of $\frac{\pi}{2^k}$, and equal constant speed. Each ray stops as soon as it hits another ray of larger length, or $R(P)$. If two parallel rays collide, then they both stop and if two or more non-parallel rays of equal length collide, then arbitrarily one of them continues, and the other rays stop. The vertices formed by the collision of rays are considered as Steiner points. Figure 1 depicts M_2 for four points in the plane.

In the following we briefly review the literature related to the planar spanners (both with and without Steiner points).

1.1 Background

Delaunay graphs are one of the most studied plane geometric spanners. Chew [10] showed that the L_1-metric Delaunay graph is a $\sqrt{10}$-spanner. There have been sev-

*Work of D. Mondal is supported in part by NSERC.

†Department of Mathematics and Computer Science, Amirkabir University of Technology, Tehran, Iran hamedmohseni@aut.ac.ir

‡Department of Mathematics and Computer Science, Amirkabir University of Technology, Tehran, Iran zrahmati@aut.ac.ir

§Department of Computer Science, University of Saskatchewan, Saskatoon, Canada dmondal@cs.usask.ca

eral attempts to find tight spanning ratio for Delaunay triangulations (L_2-metric Delaunay graphs) [16, 12, 6]. The currently best known upper and lower bound on the spanning ratio of the Delaunay triangulation is 1.998 [20] and 1.5932 [21], respectively.

Another popular class of plane geometric spanner is *half-Θ_6 graphs*, which is formed by partitioning the space around each vertex into six cones of equal angle, and then connecting the vertex to the bisector nearest neighbor in the first, third and fifth cones (for some fixed clockwise ordering of the cones); the bisector nearest neighbor in a cone means the neighbor with the smallest projection on the bisector of the cone. The half-Θ_6 graphs are 2-spanners [10].

While both the Delaunay triangulations and half-Θ_6 graphs have linear number of edges and small spanning ratio, they may have vertices with *unbounded* degree. Bose et al. [7] showed that plane t-spanners of bounded degree exist (for some constant t). A significant amount of research followed this result, which examines the construction of bounded degree plane spanners with low spanning ratio. Some of the best known spanning ratios for spanners with maximum degree $4, 6$ and 8 are 20 [15], 6 [4] and 4.414 [8], respectively.

Although there exist point sets that do not admit a planar spanner of spanning ratio less than 1.43 [13], by allowing $O(n)$ Steiner points, one can obtain a spanning ratio of $(1+\epsilon)$-spanners, for any $\epsilon > 0$. Arikati et al. [1] showed that one can construct a plane geometric $(1+\epsilon)$-spanner with $O(n/\epsilon^4)$ Steiner points. Bose and Smid [9] asked whether the dependence on ϵ can be improved.

Recently, Dehkordi et al. [11] proved that any set of n points admits a 'planar angle-monotone graph of width 90°' with $O(n)$ Steiner points. Since an angle monotone graphs of width α is a $\frac{1}{cos(\alpha/2)}$-spanner [3], this implies the existence of a $\sqrt{2}$-spanner with $O(n)$ Steiner points, which may contain vertices of *unbounded* degree. See [17] for more details on the construction of angle-monotone graphs with Steiner points.

Mondal and Nachmanson introduced a class of geometric graphs (with Steiner points), called *competition mesh*, and used those graphs to implement a large network visualization system (GraphMaps [18]). They proved the competition mesh is a $(2+\sqrt{2})$-spanner. A competition mesh is exactly the emanation graph of grade one, and hence their result implies an uppper bound of $(2+\sqrt{2})$ on the spanning ratio of M_1. Mondal and Nachmanson [18] noticed that the competition mesh can be viewed as a variation of a motorcycle graphs [14]. This also holds for the emanation graphs.

Instead of choosing three cones in the half-Θ_6 graphs, one can connect a vertex to the bisector nearest neighbors in all the six cones, which gives rise to the full-Θ_6 graphs. The concept has also been extended to full-Θ_r graphs [5], where the space around the vertices

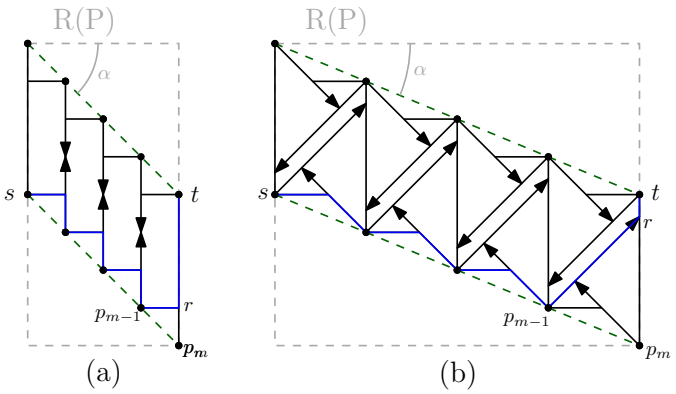


Figure 2: Illustration for lower bound proof.

are partitioned into r cones of equal angle $\theta = 2\pi/r$. Similarly, there exist Yao-graphs Y_r, where the nearest neighbor in a cone is chosen based on the Euclidean distance [2]. However, all these generalizations yield *non-planar* spanners.

1.2 Contributions

We introduce a class of *plane* geometric spanners, called emanation graphs, which generalizes the competition mesh [18]. We prove a $\sqrt{10}$ upper bound on the spanning ratio of emanation graphs of grade one, which improves the previously known upper bound of $(2+\sqrt{2})$. We also prove that the spanning ratio of every emanation graph with r rays, where $r = 4q+2$ and $q \geq 1$, is at most $\frac{1}{\sin(\pi/r)\sin(\pi/2r)}$. In contrast, we prove the spanning ratio of the emanation graphs of grade k to be at least $\frac{2+\sin(\frac{\pi}{2^k})}{1+\cos(\frac{\pi}{2^k})}$ (for sufficiently large n). Note that Mondal and Nachmanson [18] proposed several heuristics to simplify the emanation graphs (e.g., deleting the segments that do not lie on the shortest paths), which can also be applied to emanation graphs of higher grade. However, we do not consider any such simplification methods in this paper.

2 Lower Bounds

In this section, we prove the lower bounds on the spanning ratio of the emanation graphs.

Theorem 1 *There exists an emanation graph M_k of n vertices with spanning ratio $\frac{2+\sin(\frac{\pi}{2^k})}{1+\cos(\frac{\pi}{2^k})}$, for sufficiently large n.*

Proof. We refer the reader to Figures 2(a) and (b), which depict the case when $k = 1$ and $k = 2$, respectively. We construct a set of n points inside a bounding box R as follows. Imagine two parallel guidelines with an angle of $\alpha = \frac{\pi}{2^{k+1}}$, as shown in green dashed

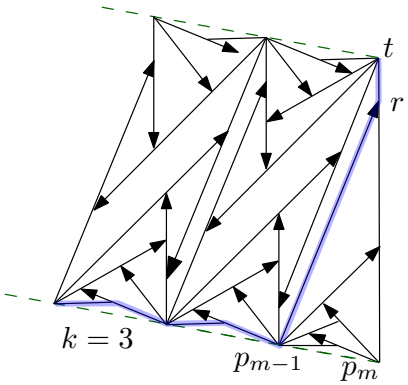


Figure 3: Illustration for the case when $k = 3$.

lines. Specifically, the two points s and t, which will achieve the lower bound, are lying along the horizontal axis. One of the two guidelines starts at s and the other guideline starts at t. As shown in the figure, the top-left corner of the bounding box R is determined by the intersection of the vertical line through s and the guideline that starts at t. The bottom-right corner of R is determined by the intersection of the vertical line through t and the guideline that starts at s.

We may assume that the number of vertices is even (if the number of vertices is odd, then we place one vertex on the bottom-left corner of R). We distribute $n/2$ points $p_1, p_2, \ldots, p_{n/2}(= p_m)$ on the guideline incident to s. We place the points ensuring that the segments $p_i p_{i+1}$ all have the same length. We place the rest of the points symmetrically on the other guideline.

We now define a canonical path $\mathcal{L}$ that starts at s and ends at t, as follows. The path $\mathcal{L}$ visits all the points $s(= p_1), \ldots, p_{m-1}$, by following the rays closest to the guidelines but staying above the guideline. The path continues from p_{m-1} by following the ray that reach closest to t. Assume that the ray intersects R at point r. Then the path continues the vertical segment rt to reach t. Figures 2(a) and (b) depict the path $\mathcal{L}$ in blue. Figures 3 illustrates the scenario when $k = 3$. We will later prove in Lemma 2 that the canonical path $\mathcal{L}$ is a shortest path between s and t.

For any two points a and b, we denote the horizontal and vertical distances between them by $|ab|_x$ and $|ab|_y$, respectively. By $\overline{ab}$, we denote the straight line segment connecting a and b. Note that s and t are horizontally aligned, and p_m is vertically aligned with t. Assume that $d_E(p_i, p_{i+1}) = 2$, and $d_E(p_1, p_m) = 2(m-1)$, where $m = n/2$. Therefore, $|p_m t|_y = 2(m-1)\sin\alpha$, and $|s p_m|_x = 2(m-1)\cos\alpha$ which is equal to $d_E(s,t)$.

Since the triangle $\triangle p_{m-1} p_m r$ is isosceles, the length of $\overline{p_{m-1} r}$ is equal to that of $\overline{p_m r}$. The length of the subpath $p_1, \ldots, p_{m-1}$ of $\mathcal{L}$ is $2(m-2)/\cos\alpha$, so the graph distance $d_G(s,t)$ between s and t is

$$\frac{2(m-2)}{\cos\alpha} + |p_m t|_y = \frac{2(m-2)}{\cos\alpha} + 2(m-1)\sin\alpha.$$

Lemma 2 proves that the shortest path between s and t is $\mathcal{L}$. Thus the spanning ratio would be

$$\begin{aligned}\frac{d_G(s,t)}{d_E(s,t)} &= \frac{\frac{2(m-2)}{\cos\alpha} + 2(m-1)\sin\alpha}{2(m-1)\cos\alpha} \\ &= \frac{2(m-2) + 2(m-1)\cos\alpha\sin\alpha}{2(m-1)\cos^2\alpha}\end{aligned}$$

For a sufficiently large m, the proof obtains:

$$\begin{aligned}\lim_{m\to\infty}\frac{d_G(s,t)}{d_E(s,t)} &= \frac{2 + 2\cos\alpha\sin\alpha}{2\cos^2\alpha} \\ &= \frac{2 + \sin(2\alpha)}{1 + \cos(2\alpha)} \\ &= \frac{2 + \sin(\frac{\pi}{2^k})}{1 + \cos(\frac{\pi}{2^k})}.\end{aligned}$$

□

Theorem 1 concludes that the lower bound of spanning ratio for the emanation graph of grade $k = 1$ is 3, and for a graph of grade $k = 2$ is approximately 1.58.

Lemma 2 *The selected path in Theorem 1 is a shortest path between s and t.*

Proof. (Sketch) We use the construction described in Theorem 1 (e.g., see Figure 4) to show that any path from s to t is at least as large as the canonical path $\mathcal{L}$ (marked in blue). First observe that it suffices to restrict our attention to x-monotone paths. One can categorize the candidate monotone paths in two groups: (I) Paths that have the same length as $\mathcal{L}$, such paths can be formed by replacing segments of $\mathcal{L}$ by their symmetric counterparts, two of these counterparts are highlighted in yellow. (II) Paths with segments that do not belong to (I), two of such segments are highlighted in red. We only need to show that the paths in (II) can not be shorter than that of $\mathcal{L}$.

By the symmetric structure of the graph, it is straightforward to observe that the paths in (II) can gradually be transformed into the canonical path $\mathcal{L}$ without changing the length. For example, the yellow path from p_j to t can be replaced by the blue path from p_j to t. Appendix includes the formal details. □

3 Upper Bounds

In this section we give the upper bounds on the spanning ratio of the emanation graphs.

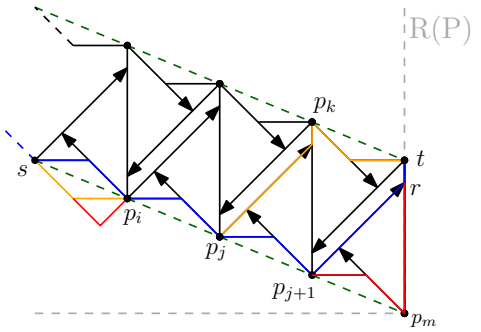


Figure 4: Illustration for proof of Lemma 2.

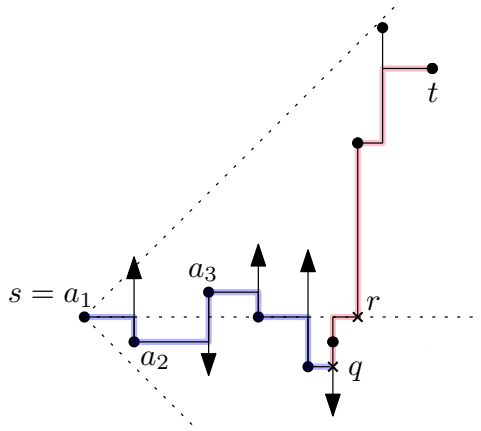


Figure 5: Illustration for proof of Theorem 3.

3.1 Emanation Graphs of Grade One

Theorem 3 *The spanning ratio of every emanation graph of grade one is at most* $\sqrt{10} \approx 3.162$.

Proof. Let s and t be a pair of vertices in the emanation graph. Consider four cones around s, where the cones are determined by two lines passing through s with slopes $+1$ and -1, respectively, as illustrated in Figure 5. Without loss of generality assume that t lies in the rightward cone C of s.

We now construct an x-monotone path P_x, which lies entirely in cone C, as follows: The path starts at s and for each original vertex, the path follows its rightward segment ℓ. If a rightward segment is stopped by another segment ℓ', then the path follows ℓ' to the original vertex that created ℓ'. Figure 5 illustrates a subpath $s(= a_1), \ldots, q$ of P_x in blue. For any subpath $a_i, \ldots, a_j$ on P_x, we will use the notation $Y_{a_i a_j}$ (resp., $X_{a_i a_j}$) to refer to the sum of the lengths of all the vertical (resp., horizontal) segments in $a_i, \ldots, a_j$.

By construction of P_x and the definition of the emanation graph, the length of any horizontal segment on P_x is at least as large as the subsequent vertical segment. Hence for every subpath $a_i, \ldots, a_j$ in P_x, which starts with a horizontal segment, we will have $X_{a_i a_j} \geq Y_{a_i a_j}$.

Without loss of generality assume that t lies on or above P_x. We now construct another path P_y following the same construction as that of P_x, but following the upward segments. Note that t is now in the region bounded by the paths P_x and P_y. We now construct an $(-x-y)$-monotone path P_t starting at t. P_t starts at t and follows the leftward segment. If the last segment ℓ of P_t is stopped by a horizontal (resp., vertical) segment ℓ', then we follow ℓ' towards the leftward (resp., downward) direction.

Note that P_t now either intersects P_x or P_y. Assume first that P_t intersects P_x at point q (see Figure 5). Let ℓ_h be the horizontal line through s. Assume that t lies above and q lies below ℓ_h (Note that the other cases would give rise to a smaller spanning ratio). Let r be the intersection point of P_t with ℓ_h. Thus the sum of the length of subpath of P_s from s to q and the subpath of P_t from q to t is as follows:

$$\begin{aligned}
|sq|_x + Y_{sq} + |qt|_x + |qt|_y &= (|sq|_x + |qt|_x) + Y_{sq} + |qt|_y \\
&= |st|_x + Y_{sq} + |qt|_y \\
&= |st|_x + Y_{sq} + |qr|_y + |rt|_y \\
&\leq 2|st|_x + Y_{sq} + |rt|_y \\
&\leq 2|st|_x + |st|_x + |rt|_y \\
&= 3|st|_x + |rt|_y \\
&= 3|st|_x + |st|_y
\end{aligned}$$

Therefore, the spanning ratio is: $f = \frac{(3|st|_x + |st|_y)}{\sqrt{(|st|_x)^2 + (|st|_y)^2}}$. To find and upper bound we need to maximize f, therefore we expect $|st|_x = 3|st|_y$, thus $f \leq \sqrt{10} \approx 3.162$.

Assume now that P_t intersects P_y at point q. But this case would be the same as when P_t intersects P_x with t lying on the upward cone of s. However, applying the same analysis, we again get an upper bound of $(3|st|_x + |st|_y)$ on the length of the path $s, \ldots, q, \ldots, t$, and hence an upper bound of 3.162. □

3.2 Generalization

Note that instead of grades, emanation graphs can also be defined with any set of r rays emanating from each vertex, where the rays create r cones of equal angle $\theta = 2\pi/r$. In this section, we prove a general upper bound on the spanning ratio of emanation graphs with r rays, where $r = 4q + 2$, where $q \geq 1$. We first describe the concept of angle-monotone paths.

A polygonal path is an *angle-monotone path of width* γ if the angles of any two edges in the path differ by at most γ (Figure 7(a)). Every angle-monotone path of width γ is a $(\frac{1}{\cos(\gamma/2)})$-spanner [3]. A geometric graph in the plane is angle-monotone of width γ if every pair of vertices is connected by an angle-monotone path of width γ. Hence these graphs are also $(\frac{1}{\cos(\gamma/2)})$-

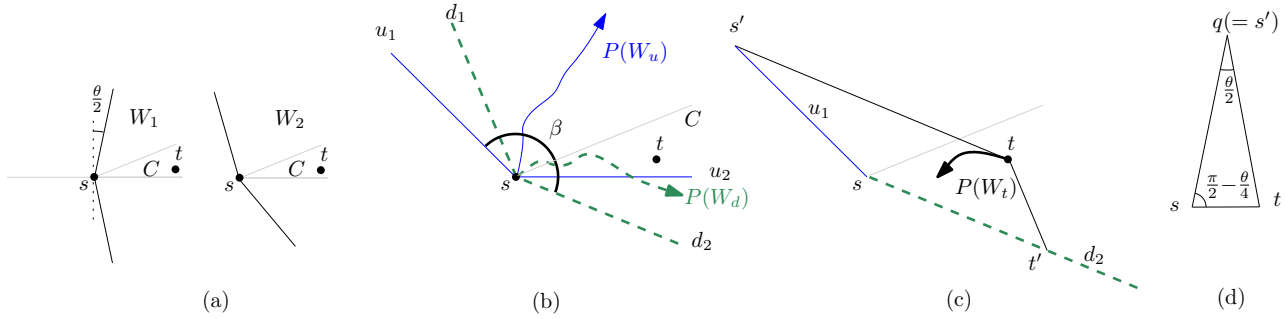


Figure 6: Illustration for the upper bound on the spanning ratio.

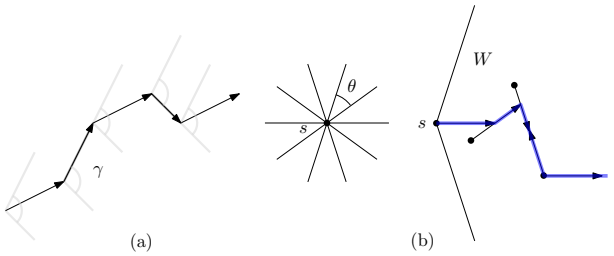


Figure 7: (a) An angle-monotone path of width γ. (b) Illustration for $P(W)$, where $r = 10$.

spanners. In the following we will prove that every emanation graph with r rays is an angle-monotone graph of width $\frac{1}{\sin(\pi/r)\sin(\pi/2r)}$.

Let M be an emanation graph with r rays, and let s and t to be a pair of vertices in G. Since we assumed that $r = 4q+2$, we may assume that there two horizontal rays around s, but *no vertical rays*. Let C be the cone incident to the rightward ray of s (lying above the ray), and without loss of generality assume that t lies in C (Figure 6(a)).

Let W be a wedge with angle $(\pi - \theta)$ such that the rightward ray of s is the bisector b of W (Figure 7(b)). By $P(W)$ we denote a path that starts following the ray parallel to b and continues as follows: If a segment stops the last segment of the current path, then we follow the ray towards the direction which is monotone with respect to b. If we reach an original vertex, then we continue to follow the ray parallel to the bisector. Note that $P(W)$ is an angle monotone path of width $(\pi - \theta)$ and lies entirely inside W.

We now define wedges $W_1, W_2, \ldots$ around s, where W_1 coincides with W and the subsequent cones are obtained by rotating W counter clockwise by an angle of θ (Figure 6(a)). Let W_u and W_d be two wedges, each of angle $(\pi - \theta)$ and contains t. Furthermore, $P(W_u)$ contains t or lies above t, and similarly, $P(W_d)$ contains t or lies below t. Let u_1 and u_2 be sides of W_u that lie above and below t, respectively. Similarly, d_1 and d_2 be the sides of W_d that lie above and below t, respectively.

Case 1: We first consider the case when W_u and W_d exist and choose W_u and W_d such that they minimize the angle β between u_1 and d_2 (Figure 6(b)). Note that $\beta \leq \pi$. Otherwise, by construction, β has to be at least $(\pi + \theta)$, and hence the wedge W' determined by d_1 and u_2 will be at least $(\pi - \theta)$. In this case, we can improve the choice of W_u and W_d further by replacing one of them using W'.

Let W_t be a wedge of angle $(\pi - \theta)$ with apex at t forming a quadrangle $ss'tt'$, as illustrated in Figure 6(c). In fact, we will choose W_t such that $\min\{\angle ss't, \angle st't\}$ is maximized. Note that $P(W_t)$ must intersect either $P(W_u)$ or $P(W_d)$ at some point q. We now use the path $P' = (s, \ldots, q, \ldots, t)$ to compute an upper bound on the spanning ratio. Since $P(W_u)$ and $P(W_d)$ are angle monotone paths of width $(\pi - \theta)$, the length of P' is at most $\frac{d_E(s,q)+d_E(q,t)}{\cos(\pi/2-\theta/2)}$. This term is maximized when $\angle ss't$ is the smallest, i.e., when $\angle ss't = (\theta/2)$, and $d_E(s,q) = d_E(q,t)$ (see Figure 6(d)). In this case, $d_E(s,q)+d_E(q,t) = 2 \cdot d_E(s,q) = \frac{d_E(s,t)}{\sin(\theta/4)}$. Consequently, the length of P' is at most

$$\frac{d_E(s,q) + d_E(q,t)}{\cos(\pi/2 - \theta/2)} = \frac{d_E(s,t)}{\sin(\theta/2) \cdot \sin(\theta/4)}$$

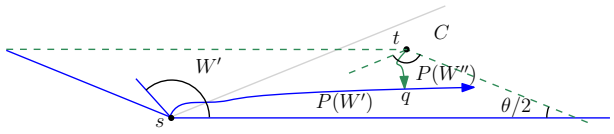


Figure 8: Illustration for Case 2.

Case 2: The remaining case is when W_u and W_d do not exist. Without loss of generality assume that for every wedge W (with apex at s) of angle $(\pi - \theta)$ containing t, the path $P(W)$ lying below t. In this scenario, let W' be the wedge that contains C with one side determined by the rightward ray of s. We then consider a downward wedge W'' (with apex at t) of angle $(\pi - \theta)$, as illustrated in Figure 8. Let q be the intersection point of the paths $P(W)$ and $P(W'')$. Since $P(W)$ and $P(W'')$ are angle

monotone paths of width $(\pi - \theta)$, the spanning ratio in this case can be bounded to $(\frac{1}{\sin(\theta/2)\sin(\theta/4)})$ using the same analysis as in Case 1. The following theorem summarizes the result of this section.

Theorem 4 *The spanning ratio of every emanation graph with r rays, where $r = 4q + 2$ and $q \geq 1$, is at most $\frac{1}{\sin(\pi/r)\sin(\pi/2r)}$.*

4 Open Questions

For emanation graphs with 6 rays, Theorem 4 gives us an upper bound of 7.72, which is larger than the upper bound we obtained for the emanation graphs with four rays (i.e., M_1 has an upper bound of 3.16). This raises an interesting question of whether we can prove the lower bound on the spanning ratio of emanation graphs of grade 2 to be larger than 3.16. Note that such a scenario where increasing the number of cones increases the spanning ratio can be found in the context of Θ_r-graphs [5].

It would be interesting to find max-degree-4 planar geometric spanners with at most $4n$ Steiner points and a spanning ratio better than $\sqrt{10}$. Note that these bounds ($4n$ Steiner points, and spanning ratio $\sqrt{10}$) are currently achieved by emanation graphs of grade one (equivalently, by the competition mesh [18]).

References

[1] S. R. Arikati, D. Z. Chen, L. P. Chew, G. Das, M. H. M. Smid, and C. D. Zaroliagis. Planar spanners and approximate shortest path queries among obstacles in the plane. In *Algorithms - ESA '96, Fourth Annual European Symposium, Barcelona, Spain, September 25-27, 1996, Proceedings*, pages 514–528, 1996.

[2] L. Barba, P. Bose, M. Damian, R. Fagerberg, W. L. Keng, J. O'Rourke, A. van Renssen, P. Taslakian, S. Verdonschot, and G. Xia. New and improved spanning ratios for yao graphs. *Journal of Computational Geometry (JoCG)*, 6(2):19–53, 2015.

[3] N. Bonichon, P. Bose, P. Carmi, I. Kostitsyna, A. Lubiw, and S. Verdonschot. Gabriel triangulations and angle-monotone graphs: Local routing and recognition. In *Proceedings of the 24th International Symposium on Graph Drawing and Network Visualization (GD)*, pages 519–531, 2016.

[4] N. Bonichon, C. Gavoille, N. Hanusse, and L. Perkovic. Plane spanners of maximum degree six. In *Proceedings of the 37th International Colloquium, on Automata, Languages and Programming (ICALP)*, pages 19–30, 2010.

[5] P. Bose, J. D. Carufel, P. Morin, A. van Renssen, and S. Verdonschot. Towards tight bounds on theta-graphs: More is not always better. *Theor. Comput. Sci.*, 616:70–93, 2016.

[6] P. Bose, L. Devroye, M. Löffler, J. Snoeyink, and V. Verma. Almost all delaunay triangulations have stretch factor greater than pi/2. *Comput. Geom.*, 44(2):121–127, 2011.

[7] P. Bose, J. Gudmundsson, and M. H. M. Smid. Constructing plane spanners of bounded degree and low weight. *Algorithmica*, 42(3-4):249–264, 2005.

[8] P. Bose, D. Hill, and M. H. M. Smid. Improved spanning ratio for low degree plane spanners. *Algorithmica*, 80(3):935–976, 2018.

[9] P. Bose and M. H. M. Smid. On plane geometric spanners: A survey and open problems. *Comput. Geom.*, 46(7):818–830, 2013.

[10] L. P. Chew. There is a planar graph almost as good as the complete graph. In *Proceedings of the Third Annual Symposium on Computational Geometry (SoCG)*, pages 169–177, 1986.

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

[12] D. P. Dobkin, S. J. Friedman, and K. J. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete & Computational Geometry*, 5:399–407, 1990.

[13] A. Dumitrescu and A. Ghosh. Lower bounds on the dilation of plane spanners. *Int. J. Comput. Geometry Appl.*, 26(2):89–110, 2016.

[14] D. Eppstein, M. T. Goodrich, E. Kim, and R. Tamstorf. Motorcycle graphs: Canonical quad mesh partitioning. *Comput. Graph. Forum*, 27(5):1477–1486, 2008.

[15] I. A. Kanj, L. Perkovic, and D. Türkoglu. Degree four plane spanners: Simpler and better. *Journal of Computational Geometry (JoCG)*, 8(2):3–31, 2017.

[16] J. M. Keil and C. A. Gutwin. Classes of graphs which approximate the complete euclidean graph. *Discrete & Computational Geometry*, 7:13–28, 1992.

[17] A. Lubiw and D. Mondal. Angle-monotone graphs: Construction and local routing. *CoRR*, abs/1801.06290, 2018.

[18] D. Mondal and L. Nachmanson. A new approach to GraphMaps, a system browsing large graphs as interactive maps. In *Proceedings of the 13th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP)*, pages 108–119, 2018.

[19] L. Nachmanson, R. Prutkin, B. Lee, N. H. Riche, A. E. Holroyd, and X. Chen. GraphMaps: Browsing large graphs as interactive maps. In *Proceedings of the 23rd International Symposium on Graph Drawing and Network Visualization (GD)*, pages 3–15, 2015.

[20] G. Xia. The stretch factor of the delaunay triangulation is less than 1.998. *SIAM J. Comput.*, 42(4):1620–1659, 2013.

[21] G. Xia and L. Zhang. Toward the tight bound of the stretch factor of delaunay triangulations. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry (CCCG)*, pages 175–180, 2011.

Appendix

Proof of Lemma 2

Proof. We use the construction described in Theorem 1 (e.g., see Figure 4) to show that any path from s to t is at least as large as the canonical path $\mathcal{L}$ (marked in blue). First observe that it suffices to restrict our attention to x-monotone paths. One can categorize the candidate monotone paths in two groups: (I) Paths that have the same length as $\mathcal{L}$, such paths can be formed by replacing segments of $\mathcal{L}$ by their symmetric counterparts, two of these counterparts are highlighted in yellow. (II) Paths with segments that do not belong to (I), two of such segments are highlighted in red. We only need to show that the paths in (II) can not be shorter than that of $\mathcal{L}$.

By the symmetric structure of the graph, it is straightforward to observe that the paths in (II) can gradually be transformed into the canonical path $\mathcal{L}$ without changing the length. For example, the yellow path from p_j to t can be replaced by the blue path from p_j to t. Here, we describe a proof by induction. In fact, we prove a stronger claim, i.e., $\mathcal{L}$ is a shortest path and for any original vertex q on the bottom guideline, a shortest path between s and q can be computed by following the rays closest to the guideline.

A formal way to see this is to apply an induction on the number of vertices. The claim is straightforward to verify when the emanation graph has only four vertices (e.g., consider the emanation graph determined by the rightmost four vertices in Figure 4). Assume now that the claim holds for the emanation graph of $2q$ vertices, where $4 \leq 2q < n(= 2q + 2)$, and consider the case when the graph has n vertices. Any x-monotone shortest path P of type (II) from s to t must pass through an original vertex other than s and t. If it passes through a vertex p_j on the bottom guideline, then the claim follows by induction. Specifically, we can choose p_j to be the source, and then the subpath p_j to t of P can be replaced by a subpath of $\mathcal{L}$ by induction. Note that the path from s to p_j can also be replaced by a subpath of $\mathcal{L}$ by induction. Thus $\mathcal{L}$ must be a shortest path between s to t.

On the other hand, if P passes through some vertex p_k on the top guideline, then we can swap the role of s and t to prove the existence of a path symmetric to $\mathcal{L}$ using the analysis used in the previous case. □

Uniform 2D-Monotone Minimum Spanning Graphs*

Konstantinos Mastakas†

Abstract

A geometric graph G is $xy-$*monotone* if each pair of vertices of G is connected by a $xy-$monotone path. We study the problem of producing the $xy-$monotone spanning geometric graph of a point set P that (i) has the minimum cost, where the cost of a geometric graph is the sum of the Euclidean lengths of its edges, and (ii) has the least number of edges, in the cases that the Cartesian System xy is specified or freely selected. Building upon previous results, we easily obtain that the two solutions coincide when the Cartesian System is specified and are both equal to the rectangle of influence graph of P. The *rectangle of influence graph* of P is the geometric graph with vertex set P such that two points $p, q \in P$ are adjacent if and only if the rectangle with corners p and q does not include any other point of P. When the Cartesian System can be freely chosen, we note that the two solutions do not necessarily coincide, however we show that they can both be obtained in $O(|P|^3)$ time. We also give a simple 2−approximation algorithm for the problem of computing the spanning geometric graph of a $k-$rooted point set P, in which each root is connected to all the other points (including the other roots) of P by $y-$monotone paths, that has the minimum cost.

1 Introduction

A sequence of points in the Euclidean plane $q_0, q_1, \ldots, q_t$ is called $y-$*monotone* if the sequence of their y coordinates, i.e. $y(q_0), y(q_1), \ldots, y(q_t)$, is either decreasing or increasing, with $y(p)$ denoting the y coordinate of the point p. A geometric path $Q = (q_0, q_1, \ldots, q_t)$ is called $y-$*monotone* if the sequence of its vertices, i.e. the sequence $q_0, q_1, \ldots, q_t$, is $y-$monotone. If Q is $y'-$monotone for some axis y' then Q is called *monotone*. Let $G = (P, E)$ be a geometric graph. If each $p, q \in P$ are connected by a $y-$monotone path then G is called $y-$*monotone*. If G is $y'-$monotone for some axis y' then G is called *uniform monotone* (following the terminology of [22]). Uniform monotone graphs were called $1-$*monotone graphs* by Angelini [3]. If each $p, q \in P$ are connected by a monotone path, where the direction of monotonicity might differ for different pairs of vertices, then G is called *monotone*. Monotone graphs were introduced by Angelini et al. [4]. Drawing an (abstract) graph as a monotone (geometric) graph has been a topic of research [3, 4, 5, 13, 24].

The *Monotone Minimum Spanning Graph problem*, i.e. the problem of constructing the monotone spanning geometric graph of a given point set that has the minimum cost, where the cost of a geometric graph is the sum of the Euclidean lengths of its edges, was recently introduced (but not solved) in [22] and it remains an open problem whether it is NP-hard. Since the more general (without the requirement of monotonicity) *Euclidean Minimum Spanning Tree problem* can be solved in $\Theta(|P| \log |P|)$ time [27], this constitutes a great differentiation that is induced by the addition of the property of monotonicity.

A point set P is $k-$*rooted* if there exist k points $r_1, r_2, \ldots, r_k \in P$ distinguished from the other points of P which are called the *roots* of P. A geometric graph $G = (P, E)$ is called $k-$*rooted* if P is $k-$rooted and its roots are the roots of P. A $k-$rooted geometric graph G is $k-$*rooted* $y-$*monotone* if each root $r \in P$ and each point $p \in P \setminus \{r\}$ are connected by $y-$monotone paths. Similarly, G is $k-$*rooted uniform monotone* (following the terminology of [22]) if it is $k-$rooted $y'-$monotone for some axis y'. For simplicity, we may also denote point sets or geometric graphs that are $1-$rooted simply as rooted. A polygon that is $2-$rooted $y-$monotone, in which its roots are its lowest and highest vertices, can be triangulated in linear time [11]. Lee and Preparata [16] preprocessed a subdivision S of the plane such that the region in which a query point belongs can be found quickly, by (i) extending the geometric graph bounding S to a $2-$rooted $y-$monotone planar geometric graph in which the roots are the highest and lowest vertices of S, and (ii) constructing a set of appropriate $y-$monotone paths from the lowest to the highest vertex of S. Additionally, Lee and Preparata [16] noted that a $2-$rooted planar geometric graph, where all vertices have different y coordinates, in which the roots are the highest and lowest vertices of the graph is $2-$rooted $y-$monotone if and only if each non-root vertex has both a neighbor above it and a neighbor below it. Furthermore, a rooted geometric graph $G = (P, E)$, where all vertices have different y coordinates, with a

*This research was financially supported by the Special Account for Research Grants of the National Technical University of Athens.

†School of Applied Mathematical and Physical Sciences, National Technical University of Athens, Athens, Greece, kmast@math.ntua.gr

(single) root r that is not the highest or lowest point of P is rooted $y-$monotone if and only if each non-root vertex p has a neighbor q such that $y(q)$ is between $y(r)$ (inclusive) and $y(p)$ [22]. Additionally, rooted uniform monotone graphs can be efficiently recognized [22]. The *$k-$rooted $y-$monotone (uniform monotone) minimum spanning graph* (following the terminology of [22]) of a $k-$rooted point set P is the $k-$rooted $y-$monotone (uniform monotone) spanning graph of P that has the minimum cost. The rooted $y-$monotone (uniform monotone) minimum spanning graph[1] of a rooted point set P can be produced in $O(|P| \cdot \log^2 |P|)$ (resp., $O(|P|^2 \cdot \log |P|)$) time [22]. The problem of drawing a rooted tree as a rooted $y-$monotone minimum spanning graph is studied in [20]. The ($|P|-$rooted) $y-$monotone minimum spanning graph of a point set P is the geometric path that traverses all the points of P by moving north, from the lowest point to the highest point of P [22]. Regarding the problem of producing the $k-$rooted $y-$monotone minimum spanning graph of a $k-$rooted point set P, with $1 < k < |P|$, it is an open problem, posed in [22], whether it is NP-hard.

The restricted fathers tree problem was introduced in [12] and is related to the rooted y-monotone minimum spanning graph problem constrained to rooted point sets P in which the y coordinate of the root is zero and the y coordinates of the other points of P are all negative (or all positive). The input of the *restricted fathers tree problem* is a complete graph with root where each edge has a cost and each vertex has a value and the goal is to output the spanning tree in which the path from the root to each vertex decreases in value that has the minimum cost. The restricted fathers tree problem is greedily solvable [12, Corollary 2.6].

A geometric path $Q = (q_0, q_1, \ldots, q_t)$ is *$xy-$monotone* if the sequence of its vertices is both $x-$monotone, i.e. the sequence $x(q_0), x(q_1), \ldots, x(q_t)$, is monotone, and $y-$monotone. Q is *2D-monotone* (following the terminology of [22]) if it is $x'y'-$monotone for some orthogonal axes x', y'. A geometric graph $G = (P, E)$ is *2D-monotone* (following the terminology of [22]) if each pair of points of P is connected by a 2D-monotone path. 2D-monotone paths/graphs were called *angle-monotone paths/graphs* by Bonichon et al. [8]. Bonichon et al. [8] showed that deciding if a geometric graph $G = (P, E)$ is 2D-monotone can be done in $O(|P| \cdot |E|^2)$ time. Triangulations with no obtuse internal angles are 2D-monotone graphs [10, 19]. There exist point sets for which any 2D-monotone spanning graph is not planar [8]. The problem of constructing 2D-monotone graphs with asymptotically less than quadratic edges was studied by Lubiw and Mondal [18]. It is an open problem, posed in [22], whether the 2D-monotone spanning graph of a point set P that has the minimum cost can be efficiently computed.

The *(rooted) $xy-$monotone* and *(rooted) uniform 2D-monotone* (using the terminology of [22]) graphs are defined similar to the (rooted) $y-$monotone and (rooted) uniform monotone graphs. Deciding if a rooted geometric graph $G = (P, E)$ is rooted $xy-$monotone (uniform 2D-monotone) can be done in $O(|E|)$ (resp., $O(|E| \cdot \log |P|)$) time [22]. Additionally, the rooted $xy-$monotone (uniform 2D-monotone) spanning graph of a rooted point set P that has the minimum cost[2] can be computed in $O(|P| \cdot \log^3 |P|)$ (resp., $O(|P|^2 \log |P|)$) time [22]. We focus on the production of the *$xy-$monotone minimum spanning graph ($xy-MMSG$)* of a point set P, i.e. the xy-monotone spanning graph of P that has the minimum cost, and the production of the *uniform $2D-$monotone minimum spanning graph ($2D-UMMSG$)* of a point set P, i.e. the uniform $2D-$monotone spanning graph of P that has the minimum cost. We also study the corresponding problems regarding the production of the spanning graphs with the least number of edges, i.e. the production of the xy-monotone spanning graph with the least number of edges and the production of the uniform $2D-$monotone spanning graph with the least number of edges.

A curve C is *increasing-chord* [15, 26] if for each p_1, p_2, p_3, p_4 traversed in this order along it, the length of the line segment $\overline{p_1p_4}$ is greater than or equal to the length of $\overline{p_2p_3}$. Alamdari et al. [1] introduced *increasing-chord graphs* which are the geometric graphs for which each two vertices are connected by an increasing-chord path. Increasing-chord graphs are widely studied [1, 6, 10, 21, 23]. The problem of producing increasing-chord spanning graphs (where Steiner points may be added) of a point set P was studied in [1, 10, 21]. The approach employed in [1, 10, 21], was to connect the points of P by 2D-monotone paths since as noted by Alamdari et al. [1] 2D-monotone paths are also increasing-chord paths.

Let P be a point set and let $p, q \in P$ then p and q are *rectangularly visible* if the rectangle with corners p and q does not include any other point of P. Furthermore, the *rectangle of influence graph* of P is the geometric graph spanning P such that $\overline{pq}$ is an edge of the graph if and only if p and q are rectangularly visible. Alon et al. [2] denoted rectangularly visible points as *separated points* and the rectangle of influence graph as the *separation graph*. Computing the rectangle of influence graph $G = (P, E)$ of P can be done in $O(|P| \cdot \log |P| + |E|)$ time [25]. There exist point sets P for which the number of edges of their rectangle of influence graph is $\Omega(|P|^2)$ [2]. The rectangle of influence graph does not remain the same if the Cartesian System is rotated [14, Proposition 3].

[1] In [22] it is shown that it is actually a tree.

[2] In [22] it is shown that it is actually a tree, denoted as the *rooted $xy-$monotone (uniform 2D-monotone) minimum spanning tree* in [22] and abbreviated as the *rooted $xy-MMST$* (resp., *rooted $2D-UMMST$*) in [22].

Drawing an abstract graph as a rectangle of influence graph has been studied [17].

Our Contribution. Building upon previous results, we easily obtain that given a point set P the $xy-$MMSG of P is equal to the $xy-$monotone spanning graph of P that has the least number of edges and are both equal to the rectangle of influence graph of P. We note that given a point set P the $2D-$UMMSG of P does not necessarily coincide with the uniform 2D-monotone spanning graph of P that has the least number of edges. We also show that both the $2D-$UMMSG of P and the uniform 2D-monotone spanning graph of P that has the least number of edges can be produced in $O(|P|^3)$ time. Additionally, we give a simple $2-$approximation algorithm for the problem of producing the $k-$rooted $y-$monotone minimum spanning graph of a $k-$rooted point set.

2 Preliminaries

2.1 $xy-$Monotone Minimum Spanning Graphs

Angelini [3] noted the following Fact regarding $y-$monotone graphs.

Fact 1 (Angelini [3]) *Let $G = (P, E)$ be a $y-$monotone graph where no two points of P have the same y coordinate and let $p, q \in P$ such that for each $r \in P \setminus \{p, q\}$ the sequence p, r, q is not $y-$monotone. Then, p and q are adjacent in G.*

Fact 1 is easily extended in the context of $xy-$monotone graphs. More specifically, let $G = (P, E)$ be a xy-monotone graph and $p, q \in P$ such that for each $r \in P \setminus \{p, q\}$ the sequence of points p, r, q is not $xy-$monotone, then p and q are adjacent in G. Alon et al. [2] noted that the points p, q of a point set P are rectangularly visible if and only if for each $r \in P \setminus \{p, q\}$ the sequence of points p, r, q is not $xy-$monotone. Hence, the rectangle of influence graph of P is a subgraph of G.

Liotta et al. [17, Lemma 2.1] showed that the rectangle of influence graph of a point set is a $xy-$monotone graph[3].

From the previous two sentences, regarding the rectangle of influence graph, we obtain the following Corollary.

Corollary 1 *Let P be a point set. The $xy-$MMSG of P and the $xy-$monotone spanning graph of P that has the least number of edges coincide and they are both equal to the rectangle of influence graph of P.*

[3]Technically speaking, Liotta et al. [17] showed that the rectangle of influence graph of a point set is a graph such that each two vertices are connected by a path lying inside the rectangle defined by these vertices but upon careful reading the path that is obtained in their proof is $xy-$monotone.

We recall that the rectangle of influence graph $G = (P, E)$ of P can be produced in $O(|P| \cdot \log |P| + |E|)$ time [25] which is optimal [25] and that there exist point sets P for which the rectangle of influence graph has size $\Omega(|P|^2)$ [2] as well as point sets for which it has linear size [2].

2.2 Rooted Uniform 2D-Monotone Graphs

Mastakas and Symvonis [22] studied the problem of recognizing rooted uniform 2D-monotone graphs. They initially noted the following Fact.

Fact 2 (Observation 8 in [22]) *Let G be a geometric graph $G = (P, E)$ with root r. If one rotates a Cartesian System $x'y'$, then G may become rooted $x'y'-$monotone while previously it was not, or vice versa, only when the y' axis becomes (or leaves the position where it previously was) parallel or orthogonal to*

1. *a line passing through r and a point $p \in P \setminus \{r\}$.*
2. *an edge $\overline{pq} \in E$, where $p, q \neq r$.*

Based on Fact 2, Mastakas and Symvonis [22] gave a rotational sweep algorithm denoted as the *rooted uniform 2D-monotone recognition algorithm* in [22].

Fact 3 ([22]) *The rooted uniform 2D-monotone recognition algorithm*

i) computes, in $O(|E| \cdot \log |P|)$ time, a set of sufficient Cartesian Systems, of size $O(|E|)$, which are associated with (1) lines passing through r and a point $p \in P \setminus \{r\}$ and (2) edges $\overline{pq} \in E$, where $p, q \neq r$.

ii) tests, in $O(|E|)$ total time[4], if G is rooted $x'y'-$monotone for some Cartesian System $x'y'$ in the previously computed set of sufficient Cartesian Systems.

Fact 4 (Theorem 1 in [22]) *Let P be a rooted point set then the rooted $y-$monotone minimum spanning graph of P can be obtained in $O(|P| \cdot \log^2 |P|)$ time.*

3 The 2D-UMMSG Problem

We now deal with the construction of the $2D-$UMMSG and the uniform $2D-$monotone spanning graph with

[4]Technically speaking in [22] it is shown that the remaining steps, i.e. the steps after the computation of the sufficient Cartesian Systems, of the rooted uniform 2D-monotone recognition algorithm take $O(|E| \cdot \log |P|)$ total time. Internally in the rooted uniform 2D-monotone recognition algorithm given in [22], for each $p \in P \setminus \{r\}$ it is stored the set of adjacent points to p that are in the rectangle w.r.t. the Cartesian System $x'y'$ with corners p and r, which is denoted as $A(p, x', y')$ in [22]. Furthermore, it is stored the set of points $p \in P \setminus \{r\}$ for which $|A(p, x', y')| > 0$ which is denoted as $B(x', y')$ in [22]. However, only the cardinalities of these sets are necessary [22, Lemma 9], hence if instead of the sets $A(p, x', y'), p \in P$ and $B(x', y')$ their cardinalities are stored, the remaining steps of the rooted uniform 2D-monotone recognition algorithm take $O(|E|)$ total time.

the least number of edges. We initially show that the $2D-$UMMSG of a point set P can be obtained in $O(|P|^3)$ time. For this, we employ a rotational sweep technique. Our approach regarding the construction of the $2D-$UMMSG is similar to the approach employed for the calculation of the rooted uniform $2D-$monotone spanning graph that has the minimum cost in [22]. We assume that no three points of P are collinear and no two line segments $\overline{pq}$ and $\overline{p'q'}$, $p, p', q, q' \in P$, are parallel or orthogonal.

Let P be a point set and p be a point of P. Let $RV(p, x', y')$ denote the subset of points of P that are rectangularly visible from p w.r.t. the Cartesian System $x'y'$. See for example, Figure 1(a).

Proposition 2 *If we rotate a Cartesian System $x'y'$ counterclockwise, then the $x'y'-$MMSG of P changes only when y' reaches or moves away from a line perpendicular or parallel to a line passing through two points of P.*

Proof. If we rotate the Cartesian System $x'y'$ counterclockwise then the $RV(p, x', y')$ for a point $p \in P$ changes only when y' reaches or moves away from a line perpendicular or parallel to a line passing through two points of P; e.g. see Figure 1. From the previous and since the $RV(p, x', y'), p \in P$, equals to the set of adjacent vertices of p in the $x'y'-$MMSG of P (Corollary 1), we obtain the Proposition. □

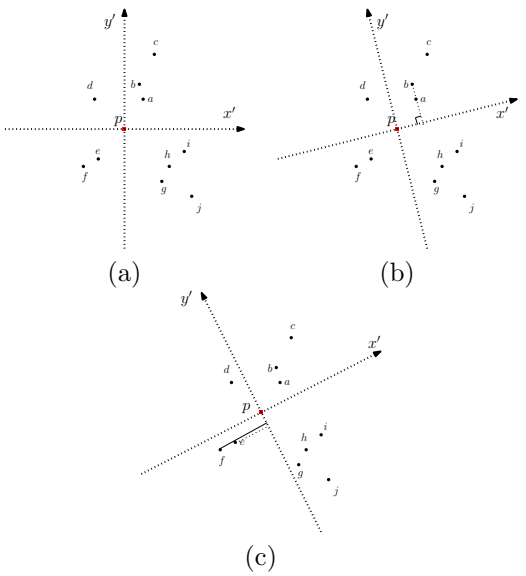


Figure 1: In (a) $RV(p, x', y') = \{a, b, d, e, g, h, i\}$. In (b) the y' becomes parallel to the $\overline{ab}$ and now b is not rectangularly visible from p. Finally, in (c) the y' has left the position where it previously was orthogonal to the $\overline{ef}$ and now f becomes rectangularly visible from p.

Let $S = \{s \in [0, \frac{\pi}{2}) :$ a line of slope s is perpendicular or parallel to a line passing through two points of $P\}$. Let $S = \{s_1, s_2, \ldots, s_l\}$ with $l = \binom{|P|}{2}$ such that $0 \leq s_1 < s_2 < \ldots < s_l < \frac{\pi}{2}$. We now define the set $S_{\text{sufficient}}$ to be equal to $\{s_1, \frac{s_1+s_2}{2}, s_2, \frac{s_2+s_3}{2}, \ldots, s_l, \frac{s_l+\frac{\pi}{2}}{2}\}$. Let $x_1y_1, x_2y_2, \ldots, x_{2l}y_{2l}$ be the Cartesian Systems in which the vertical axis has slope in $S_{\text{sufficient}}$, ordered w.r.t. the slope of their vertical axis.

Theorem 3 *The uniform $2D-$monotone minimum spanning graph of a point set P can be computed in $O(|P|^3)$ time.*

Proof. From Proposition 2 and the previous definitions we obtain the following Proposition.

Proposition 4 *The uniform $2D-$monotone minimum spanning graph of P is one of the $x'y'-$MMSG of P over all Cartesian Systems $x'y'$ with y' of slope in $S_{sufficient}$.*

We now give a $O(|P|^3)$ time rotational sweep algorithm. The algorithm initially computes the x_1y_1-MMSG of P and then it obtains each $x_{i+1}y_{i+1}-$MMSG of P from the x_iy_i-MMSG of P. Throughout the procedure the Cartesian System $x^{\text{opt}}y^{\text{opt}}$ in which the algorithm encountered the minimum cost solution so far is stored. In its last step, the algorithm recomputes the $x^{\text{opt}}y^{\text{opt}}-$MMSG of P, which since it is equal to the rectangle of influence graph $G = (P, E)$ w.r.t. the Cartesian System $x^{\text{opt}}y^{\text{opt}}$ (Corollary 1) it can be computed in $O(|P| \cdot \log |P| + |E|)$ time [25]. The crucial proposition (which we show later) that makes the time complexity of the algorithm equal to $O(|P|^3)$ is that each transition from the x_iy_i-MMSG of P to the $x_{i+1}y_{i+1}-$MMSG of P takes $O(|P|)$ time.

For each two points p, q of P let $I(p, q, x_i, y_i)$ be the number of points of $P \setminus \{p, q\}$ that are included in the rectangle w.r.t. the Cartesian System x_iy_i with opposite vertices p and q. Then, $RV(q, x_i, y_i)$ can be equivalently defined using the quantities $I(p, q, x_i, y_i), p \in P \setminus \{q\}$, as follows: $p \in RV(q, x_i, y_i)$ if $I(p, q, x_i, y_i) = 0$.

We store the $RV(q, x_i, y_i)$, $q \in P$, $i = 1,2, \ldots, 2l$ in the data structure $rv(q)$ which is implemented as an array of $|P|$ booleans. We also store the $I(p, q, x_i, y_i)$, $p, q \in P$, $i = 1,2, \ldots, 2l$ in the variable $i(p, q)$.

Computing the Cartesian Systems x_iy_i, $i = 1, 2, \ldots, 2l$ can be done in $O(|P|^2 \log |P|)$ time. Accompanied with each Cartesian System x_iy_i is the pair of points (p_i, q_i) such that $\overline{p_iq_i}$ is either parallel or perpendicular to the y_i axis or the y_{i-1} axis.

Ichino and Sklansky [14] noted that employing a range tree [7, 9] that contains the points of P one can calculate i) the rectangle of influence graph of P, and ii) the $I(p, q, x, y), p, q, \in P$, for a Cartesian System xy. Applying the previously mentioned approach, noted by Ichino and Sklansky [14], are obtained i) the rectangle of influence graph of P w.r.t. the Cartesian System x_1y_1 (which by Corollary 1 equals to the x_1y_1-MMSG of P), and ii) the $I(p, q, x_1, y_1)$, $p, q, \in P$.

We now show that we can update all the $rv(p), p \in P$, such that from equal to $RV(p, x_{i-1}, y_{i-1}), p \in P$, they become equal to $RV(p, x_i, y_i), p \in P$, in $O(|P|)$ total time. For each $p \in P \setminus \{p_i, q_i\}$ the update of $rv(p)$ takes $O(1)$ time. This is true, since only the points p_i and q_i have to be tested for inclusion to or removal from $rv(p)$. More specifically, we have to test if for one of them, say p_i, the rectangle with corners p and p_i contains (or it does not contain) q_i w.r.t. the Cartesian System x_iy_i while it did not contain (or it contained) it w.r.t. $x_{i-1}y_{i-1}$. If this is true, then the $i(p_i, p)$ changes and p_i has to be tested for membership in $rv(p)$ and included to or removed from $rv(p)$. Regarding $rv(p_i)$, the update takes $O(|P|)$ time, since for each other point $q \in P \setminus \{p_i, q_i\}$ we have to test if the rectangle with corners q and p_i contains (or it does not contain) q_i w.r.t. the Cartesian System x_iy_i while it did not contain it (or it contained it) w.r.t. the $x_{i-1}y_{i-1}$ and if so update both the $i(q, p_i)$ and the existence of q in $rv(p_i)$ if necessary. Similarly, $rv(q_i)$ can be updated in $O(|P|)$ time. □

We note that the procedure of obtaining the $2D-$UMMSG can be trivially modified such that the uniform $2D$-monotone spanning graph of a point set P with the least number of edges can be obtained in $O(|P|^3)$ time. Since for an arbitrary Cartesian System $x'y'$ the $x'y'-$MMSG of P is equal to the $x'y'-$monotone spanning graph of P with the least number of edges (Corollary 1), the only modification which is necessary is that in the transition from the Cartesian System x_iy_i to the Cartesian System $x_{i+1}y_{i+1}$ we check if the $x_{i+1}y_{i+1}-$monotone spanning graph of P with the least number of edges has the least number of edges among all the produced solutions so far.

In Figure 2 is given a point set P for which the $2D-$UMMSG of P is different from the uniform $2D-$monotone spanning graph of P with the least number of edges.

In Figure 3 we give a point set P for which the (non-uniform) $2D-$monotone spanning graph of P with the least number of edges does not coincide to the (non-uniform) $2D-$monotone spanning graph of P that has the minimum cost.

Regarding recognizing uniform $2D$-monotone graphs, we note that the $O(|E| \cdot \log |P|)$ time rotational sweep algorithm given in [22], which decides if a geometric graph $G = (P, E)$ with a specified vertex r as root is rooted uniform $2D-$monotone, can be easily extended into a $O(|P|^2 \cdot \log |P| + |P| \cdot |E|)$ time rotational sweep algorithm that decides if G is uniform $2D-$monotone. More specifically, in order to decide if G is uniform $2D-$monotone, the $|P|$ rooted geometric graphs (p_1, G), (p_2, G), ..., $(p_{|P|}, G)$ where (p_i, G) is the geometric graph G with root p_i and $\{p_1, p_2, \ldots, p_{|P|}\}$ is the vertex set of G, are considered. A Cartesian System $x'y'$ is rotated counterclockwise. From Fact 2, it follows that one of these $|P|$ rooted geometric graphs becomes rooted $x'y'-$monotone while previously it was not, or vice versa, only when the y' axis becomes (or leaves the position where it was previously) parallel or orthogonal to a line passing through two points of P. Hence, $O(|P|^2)$ Cartesian Systems need to be considered, which can be computed in $O(|P|^2 \log |P|)$ time. When the y' becomes (or leaves the position that it previously was) parallel or perpendicular to a line passing through the points $p, q \in P$ then by Fact 2 the status, i.e. being rooted $x'y'-$monotone, of the rooted geometric graphs (p, G) and (q, G) may change. Hence, the steps of the rooted uniform 2D-monotone recognition algorithm given in [22] for handling the event associated with the current Cartesian System $x'y'$ regarding the rooted geometric graphs (p, G) and (q, G), are applied. Furthermore, if $\overline{pq} \in E$ then by Fact 2 it follows that the status, i.e. being rooted $x'y'-$monotone, of each $(r, G), r \in P \setminus \{p, q\}$, may also change. Hence, for each $(r, G), r \in P \setminus \{p, q\}$, the steps of the rooted uniform 2D-monotone recognition algorithm given in [22] for handling the event associated with the current Cartesian System $x'y'$ are applied. Since, the remaining steps, i.e. after the calculation of the sufficient axes, of the rooted uniform 2D-monotone recognition algorithm, given in [22], regarding any of these $|P|$ rooted

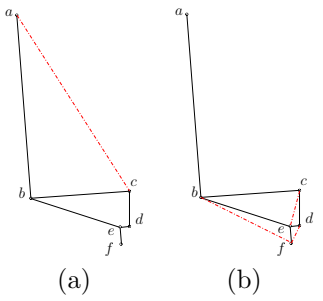


Figure 2: The points a,b and c form a right angle. Additionally, the points d, e and f form a right angle. The slope of $\overline{de}$ is smaller than the slope of $\overline{bc}$. The uniform $2D-$monotone spanning graph with the least number of edges is obtained when the y' axis becomes perpendicular to the $\overline{de}$ and is shown in (a). On the other hand the $2D-$UMMSG is obtained when the y' axis becomes perpendicular to the $\overline{bc}$ and is shown in (b).

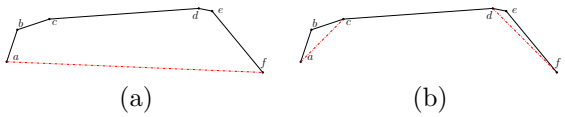


Figure 3: The slope of $\overline{ac}$ is $\frac{\pi}{4}$ while the slope of $\overline{fd}$ is $\frac{3\pi}{4}$. In (a) is depicted the $2D-$monotone spanning graph of P with the least number of edges. In (b) is illustrated the $2D-$monotone spanning graph of P that has the minimum cost.

geometric graphs take $O(|E|)$ time (Fact 3), applying the remaining steps regarding all these $|P|$ rooted geometric graphs, takes $O(|P| \cdot |E|)$ total time.

4 A 2−Approximation Algorithm for the k−Rooted y−Monotone Minimum Spanning Graph Problem

We now study the problem of producing the k−rooted y−monotone minimum spanning graph of a k−rooted point set P, where $1 < k < |P|$. We assume that no two points have the same y coordinate.

Let P be a point set and $a, b \in \mathbb{R}$ then $P_{y>a}$ is the subset of points of P whose y coordinate is greater than a. Similarly are defined $P_{y\geq a}$, $P_{y<a}$ and $P_{y\leq a}$. $P_{a<y<b}$ is the subset of points of P whose y coordinate is between a and b. Similarly are defined $P_{a<y\leq b}$, $P_{a\leq y<b}$ and $P_{a\leq y\leq b}$.

In [22, Lemma 1] it is noted that the rooted y−monotone minimum spanning graph of a rooted point set P with root r is the union of the rooted y−monotone minimum spanning graphs of (i) $P_{y\leq y(r)}$ and (ii) $P_{y\geq y(r)}$. The previous Fact is extended to the following Lemma.

Lemma 5 *Let P be a k−rooted point set, with $1 < k < |P|$, where r_1, r_2, ..., r_k are the roots of P such that $y(r_1) < y(r_2) < \ldots < y(r_k)$. The k−rooted y−monotone minimum spanning graph of P is the union of*

1. *the rooted y−monotone minimum spanning graph of $P_{y\leq y(r_1)}$.*
2. *the rooted y−monotone minimum spanning graph of $P_{y\geq y(r_k)}$.*
3. *the 2−rooted y−monotone minimum spanning graph of $P_{y(r_i)\leq y\leq y(r_{i+1})}$, $1 \leq i \leq k-1$.*

Theorem 6 *Given a k−rooted point set P, with $1 < k < |P|$, we can obtain in $O(|P| \cdot \log^2 |P|)$ time a k−rooted y−monotone spanning graph of P with cost at most twice the cost of the k−rooted y−monotone minimum spanning graph of P.*

Proof. For a 2−rooted point set P with roots r_1 and r_2 that are the lowest and highest points of the point set, respectively, we prove the following Lemma.

Lemma 7 *Given a 2−rooted point set P with roots r_1 and r_2 that are the lowest and highest points of the point set, respectively, we can obtain in $O(|P|\cdot\log^2 |P|)$ time a 2−rooted y−monotone spanning graph of P with cost at most twice the cost of the 2−rooted y−monotone minimum spanning graph of P.*

Proof. Initially, we employ Fact 4 to P considering it to have only the root r_1 and obtain the geometric graph G_1. Then, we employ Fact 4 to P considering it to have only the root r_2, obtaining G_2. In the final step we return the union of G_1 and G_2. $G_1 \cup G_2$ is 2−rooted y−monotone since G_1 (G_2) is rooted y−monotone with root r_1 (resp., r_2). We now show that $G_1 \cup G_2$ has cost at most twice the cost of the 2−rooted y−monotone minimum spanning graph G^{opt} of P. Since, in G^{opt} all the points p are connected with r_1 (r_2) by y−monotone paths it follows that its cost is greater than or equal to the cost of G_1 (resp., G_2). Hence, the cost of $G_1 \cup G_2$ which is less than or equal to the sum of the costs of G_1 and G_2 is at most twice the cost of G^{opt}. □

From Lemma 5, Fact 4 and Lemma 7 we obtain the Theorem. □

A 2−rooted planar geometric graph $G = (P, E)$ with roots r_1, r_2 s.t. $y(r_1) < y(p) < y(r_2), p \in P \setminus \{r_1, r_2\}$, is 2−rooted y−monotone if and only if for each $p \in P \setminus \{r_1, r_2\}$ there exist $q_1, q_2 \in \text{Adj}(p)$ with $y(q_1) < y(p) < y(q_2)$ [16]. Furthermore, a rooted geometric graph $G = (P, E)$ with a (single) root r that is not the highest or lowest point of P is rooted y−monotone if and only if for each $p \in P \setminus \{r\}$ there exists $q \in \text{Adj}(p)$ such that $y(q)$ is between $y(r)$ (inclusive) and $y(p)$ [22]. We extend the previous two Propositions to the following equivalent characterization of k−rooted y−monotone graphs where the latter implies an efficient recognition algorithm for k−rooted y−monotone graphs.

Proposition 8 *Let $G = (P, E)$ be a k−rooted geometric graph, where $1 < k < |P|$, with roots r_1, r_2, ..., r_k such that $y(r_1) < y(r_2) < \ldots < y(r_k)$. G is k−rooted y−monotone if and only if*

1. *for each $p \in P_{y<y(r_1)}$ there exists $q \in Adj(p)$ s.t. $y(q) \in (y(p), y(r_1)]$.*
2. *for each $p \in P_{y>y(r_k)}$ there exists $q \in Adj(p)$ s.t. $y(q) \in [y(r_k), y(p))$.*
3. *for each $p \in P_{y(r_i)<y<y(r_{i+1})}$ there exist $q_1, q_2 \in Adj(p)$ s.t. $y(q_1) \in [y(r_i), y(p))$ and $y(q_2) \in (y(p), y(r_{i+1})]$, $i = 1, 2, \ldots, k-1$.*
4. *there exists $q \in Adj(r_1)$ s.t. $y(q) \in (y(r_1), y(r_2)]$.*
5. *there exists $q \in Adj(r_k)$ s.t. $y(q) \in [y(r_{k-1}), y(r_k))$.*
6. *there exist $q_1, q_2 \in Adj(r_i)$ s.t. $y(q_1) \in [y(r_{i-1}), y(r_i))$ and $y(q_2) \in (y(r_i), y(r_{i+1})]$, $2 \leq i \leq k-1$.*

5 Further Research Directions

Given a point set P can the $2D$−monotone spanning graph of P that has the least number of edges be produced in polynomial time?

Does there exist a t−approximation algorithm, $t < 2$, for the k−rooted y−monotone minimum spanning graph problem?

Acknowledgement: I would like to thank Professor Antonios Symvonis for his valuable contribution in developing the results presented in this paper.

References

[1] S. Alamdari, T. M. Chan, E. Grant, A. Lubiw, and V. Pathak. Self-approaching graphs. In W. Didimo and M. Patrignani, editors, *Graph Drawing - GD 2012*, volume 7704 of *LNCS*, pages 260–271. Springer, 2013.

[2] N. Alon, Z. Füredi, and M. Katchalski. Separating pairs of points by standard boxes. *Eur. J. Comb.*, 6(3):205–210, 1985.

[3] P. Angelini. Monotone drawings of graphs with few directions. *Inf. Process. Lett.*, 120:16–22, 2017.

[4] P. Angelini, E. Colasante, G. Di. Battista, F. Frati, and M. Patrignani. Monotone drawings of graphs. *J. Graph Algorithms Appl.*, 16(1):5–35, 2012.

[5] P. Angelini, W. Didimo, S. Kobourov, T. Mchedlidze, V. Roselli, A. Symvonis, and S. Wismath. Monotone drawings of graphs with fixed embedding. *Algorithmica*, 71(2):233–257, 2015.

[6] Y. Bahoo, S. Durocher, S. Mehrpour, and D. Mondal. Exploring increasing-chord paths and trees. In J. Gudmundsson and M. Smid, editors, *Proceedings of the 29th Canadian Conference on Computational Geometry, CCCG 2017*, pages 19–24. Carleton University, Ottawa, Ontario, Canada, 2017.

[7] J. L. Bentley and H. A. Maurer. Efficient worst-case data structures for range searching. *Acta Inf.*, 13:155–168, 1980.

[8] N. Bonichon, P. Bose, P. Carmi, I. Kostitsyna, A. Lubiw, and S. Verdonschot. Gabriel triangulations and angle-monotone graphs: Local routing and recognition. In Y. Hu and M. Nöllenburg, editors, *Graph Drawing and Network Visualization - 24th International Symposium, GD 2016, Athens, Greece, September 19-21, 2016, Revised Selected Papers*, volume 9801 of *LNCS*, pages 519–531. Springer, 2016.

[9] M. de Berg, O. Cheong, M. J. van Kreveld, and M. H. Overmars. *Computational geometry: algorithms and applications*. Springer, 3rd edition, 2008.

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

[11] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan. Triangulating a simple polygon. *Inf. Process. Lett.*, 7(4):175–179, 1978.

[12] N. Guttmann-Beck and R. Hassin. On two restricted ancestors tree problems. *Inf. Process. Lett.*, 110(14-15):570–575, 2010.

[13] D. He and X. He. Optimal monotone drawings of trees. *SIAM Journal on Discrete Mathematics*, 31(3):1867–1877, 2017.

[14] M. Ichino and J. Sklansky. The relative neighborhood graph for mixed feature variables. *Pattern Recognition*, 18(2):161–167, 1985.

[15] D. G. Larman and P. McMullen. Arcs with increasing chords. *Mathematical Proceedings of the Cambridge Philosophical Society*, 72:205–207, September 1972.

[16] D. T. Lee and F. P. Preparata. Location of a point in a planar subdivision and its applications. *SIAM J. Comput.*, 6(3):594–606, 1977.

[17] G. Liotta, A. Lubiw, H. Meijer, and S. Whitesides. The rectangle of influence drawability problem. *Comput. Geom.*, 10(1):1–22, 1998.

[18] A. Lubiw and D. Mondal. Angle-monotone graphs: Construction and local routing. *CoRR*, abs/1801.06290, 2018.

[19] A. Lubiw and J. O'Rourke. Angle-monotone paths in non-obtuse triangulations. In J. Gudmundsson and M. Smid, editors, *Proceedings of the 29th Canadian Conference on Computational Geometry, CCCG 2017*, pages 25–30. Carleton University, Ottawa, Ontario, Canada, 2017.

[20] K. Mastakas. Drawing a Rooted Tree as a Rooted $y-$Monotone Minimum Spanning Tree. *CoRR*, abs/1806.04720, 2018.

[21] K. Mastakas and A. Symvonis. On the construction of increasing-chord graphs on convex point sets. In *6th Int. Conf. on Information, Intelligence, Systems and Applications, IISA 2015, Corfu, Greece, July 6-8, 2015*, pages 1–6. IEEE, 2015.

[22] K. Mastakas and A. Symvonis. Rooted uniform monotone minimum spanning trees. In D. Fotakis, A. Pagourtzis, and V. Th. Paschos, editors, *Algorithms and Complexity - 10th International Conference, CIAC 2017, Athens, Greece, May 24-26, 2017, Proceedings*, volume 10236 of *LNCS*, pages 405–417, 2017. Full Version:arXiv:1607.03338v2, 2017.

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

[24] A. Oikonomou and A. Symvonis. Simple compact monotone tree drawings. In F. Frati and K.-L. Ma, editors, *Graph Drawing and Network Visualization - 25th International Symposium, GD 2017, Boston, MA, USA, September 25-27, 2017, Revised Selected Papers*, volume 10692 of *LNCS*, pages 326–333. Springer, 2017.

[25] M. H. Overmars and D. Wood. On rectangular visibility. *J. Algorithms*, 9(3):372–390, 1988.

[26] G. Rote. Curves with increasing chords. *Mathematical Proceedings of the Cambridge Philosophical Society*, 115:1–12, January 1994.

[27] M. I. Shamos and D. Hoey. Closest-point problems. In *16th Annual Symposium on Foundations of Computer Science, Berkeley, California, USA, October 13-15, 1975*, pages 151–162. IEEE Computer Society, 1975.

On Nonogram and Graph Planarity Puzzle Generation

Marc van Kreveld*

There are many puzzles in the world, both physical and digital ones. From the computational perspective, a lot of attention has been given to combinatorial puzzles and how to (algorithmically) solve them. We will focus on puzzles with a geometric component where techniques from discrete and computational geometry can be employed to generate them. We also present new nonogram and graph planarity puzzles.

1 Nonogram puzzles

In a basic nonogram, each row and each column has a clue that gives an abstract description as to which cells are to be filled. Suppose a row is eight cells long. Then a clue 1 3 specifies that in the row, one cell should be filled, and later in that row another three cells in sequence. Between the one and the three filled cells, there is at least one non-filled cell. Also, before the one filled cell and after the three filled cells, there are zero or more non-filled cells. In binary, 00101110 and 10011100 are both valid filling choices for the row of cells corresponding to the clue. A nonogram puzzle originally consists of a grid with no cells filled, see Fig. 1. A solution (or filling) is correct if for every row and column, the filled cells are an option for the given clue.

The scientific study of nonograms usually focuses on the algorithmic complexity of solving them [1, 2, 11, 13].

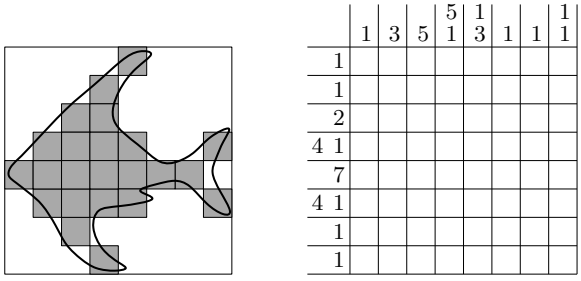


Figure 1: Basic nonogram (right) of the pixel image shown left.

In this talk we will discuss the problem of converting a simple polygon, representing some shape, into a nonogram. In essence, the polygon will become a low-res pixel image. We analyze how a pixel polygon can be formed that is simple and has small Hausdorff or Fréchet distance to the input. It is based on research by Bouts et al. [4].

We will also introduce new types of nonograms that generalize the basic type. Puzzles can be constructed based on any set of lines and even of curves. For these new types of generalized nonograms we review the design choices and let these inspire an automated method to generate a nonogram from a drawing. For nonograms based on curves, the curves replace the grid lines of basic nonograms and form an *arrangement* of cells with varying shape and size. Since rows and columns no longer exist, we need a new way to give clues indicating which cells should be filled by the puzzler. It allows for new reasoning steps that do not exist in a grid-based nonogram. This part is based on research by van de Kerkhof et al. [7].

2 Graph planarity puzzles

Planarity [10] is a popular abstract puzzle game that is widely available. The idea is that a tangled graph is given with intersecting edges, and the objective is to untangle the graph by dragging vertices to other locations. If the graph is planar, then the objective can always be realized, and we never need more vertex drags than there are vertices.

Algorithmically, planarity of a graph can be tested in linear time, and the algorithm returns an embedding of the graph in which it is drawn planar. So for an algorithm, an instance of Planarity is easily solvable in linear time. Minimizing the number of moves, however, is NP-hard [6, 12], see also [3].

In this talk we propose several variations on the game Planarity. These variations essentially limit the freedom of the operations that can be done on the drawn graph. Since the puzzle type is abstract, it is preferable that the interaction and operations themselves be simple. The puzzle might then become an elegant abstract puzzle of which there are many already (Move, Lines/Flow, Zengrams, Nintaii, Fling, ...).

Besides interacting with a vertex like in Planarity, it is natural to interact with an edge. Clicking or selecting is arguably the easiest interaction. We list a number of ways in which the drawing can change when an edge is selected:

- *Swap:* the two endpoints of the selected edge swap locations. Intuitively, the edge turns around while the endpoints drag all incident edges with them.

*Department of Information and Computing Sciences, Utrecht University, m.j.vankreveld@uu.nl

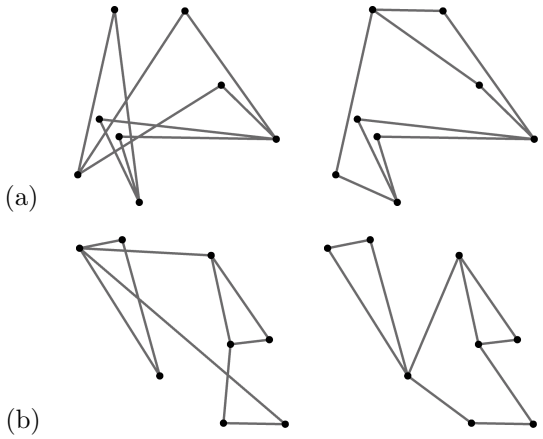


Figure 2: (a) Puzzle and solution after one swap (the left edge). (b) Puzzle and solution after two swaps.

- *Rotate:* like swap, but now the selected edge rotates over 90 degrees around its center. Since a single edge can be selected consecutively three times, it does not matter whether we rotate clockwise or counter-clockwise.
- *Stretch:* the selected edge is scaled by a factor 2 from its center, or by a factor 1/2.
- *Mid collapse:* the endpoints of the selected edge are united. The united vertex is placed in the middle of the edge and gets all edges incident to the original vertices. The selected edge is removed.
- *End collapse:* Same but the united vertex is placed at a selected endpoint.

We will investigate the first of the new variations closely: Swap Planarity. Examples are shown in Fig. 2. We show that quadratically many swaps are sometimes necessary (even if the input has just one edge crossing) and always sufficient; the latter follows from [14]. The decision (solvability) question is NP-complete for general graphs; this follows from [5]. Simple graphs like trees can always be made planar by swaps, but minimizing the number of swaps needed is NP-hard.

We discuss the automated generation of good puzzle instances for Swap Planarity by describing a five-step process which yields such a puzzle instance. Some of the considerations of a good instance are puzzle (complexity) based and some are geometry based. This part is based on research by Kraaijer et al. [8].

Acknowledgements. Research is supported by the Netherlands Organisation for Scientific Research (NWO) on grant no. 612.001.651

References

[1] Kees Joost Batenburg and Walter A. Kosters. Solving Nonograms by combining relaxations. *Pattern Recognition* 42(8):1672–1683, 2009.

[2] Daniel Berend, Dolev Pomeranz, Ronen Rabani, and Ben Raziel. Nonograms: Combinatorial questions and algorithms. *Discrete Applied Mathematics* 169:30–42, 2014.

[3] Prosenjit Bose, Vida Dujmovic, Ferran Hurtado, Stefan Langerman, Pat Morin, and David R. Wood. A polynomial bound for untangling geometric planar graphs. *Discrete & Computational Geometry* 42(4):570–585, 2009.

[4] Quirijn W. Bouts, Irina Kostitsyna, Marc van Kreveld, Wouter Meulemans, Willem Sonke, and Kevin Verbeek. Mapping polygons to the grid with small Hausdorff and Fréchet distance. In *24th Annual European Symposium on Algorithms, ESA*. LIPIcs, 22:1–22:16, 2016.

[5] Sergio Cabello. Planar embeddability of the vertices of a graph using a fixed point set is NP-hard. *Journal of Graph Algorithms and Applications* 10(2):353–363, 2006.

[6] Xavier Goaoc, Jan Kratochvíl, Yoshio Okamoto, Chan-Su Shin, and Alexander Wolff. Moving vertices to make drawings plane. In *Graph Drawing, 15th International Symposium, GD 2007*, volume 4875 of *Lecture Notes in Computer Science*, pages 101–112. Springer, 2008.

[7] Mees van de Kerkhof, Tim de Jong, Marc van Kreveld, Maarten Löffler, Raphael Parment, and Amir Vaxman. *Design and automated generation of Japanese picture puzzles.* Manuscript, 2018.

[8] Rutger Kraaijer, Marc van Kreveld, Wouter Meulemans, and André van Renssen. Geometry and generation of a new graph planarity game. In *Proceedings of the IEEE Conference on Computational Intelligence and Games*. To appear, 2018.

[9] Emilio G. Ortíz-García, Sancho Salcedo-Sanz, José M. Leiva-Murillo, Ángel M. Pérez-Bellido, and José Antonio Portilla-Figueras. Automated generation and visualization of picture-logic puzzles. *Computers & Graphics* 31(5):750–760, 2007.

[10] John Tantalo. Planarity. http://planarity.net/, 2007. Accessed: 2018-05-25.

[11] Jinn-Tsong Tsai. Solving Japanese nonograms by Taguchi-based genetic algorithm. *Applied Intelligence* 37(3):405–419, 2012.

[12] Oleg Verbitsky. On the obfuscation complexity of planar graphs. *Theoretical Compututer Science*, 396(1-3):294–300, 2008.

[13] I-Chen Wu, Der-Johng Sun, Lung-Ping Chen, Kan-Yueh Chen, Ching-Hua Kuo, Hao-Hua Kang, and Hung-Hsuan Lin. An Efficient Approach to Solving Nonograms. *IEEE Transactions on Computational Intelligence and AI in Games* 5(3):251–264, 2013.

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

Threadable Curves*

Joseph O'Rourke Emmely Rogers†

Abstract

We define a plane curve to be *threadable* if it can rigidly pass through a point-hole in a line L without otherwise touching L. Threadable curves are in a sense generalizations of monotone curves. We have two main results. The first is a linear-time algorithm for deciding whether a polygonal curve is threadable—$O(n)$ for a curve of n vertices—and if threadable, finding a sequence of rigid motions to thread it through a hole. We also sketch an argument that shows that the threadability of algebraic curves can be decided in time polynomial in the degree of the curve. The second main result is an $O(n \operatorname{polylog} n)$-time algorithm for deciding whether a 3D polygonal curve can thread through a hole in a plane in $\mathbb{R}^3$, and if so, providing a description of the rigid motions that achieve the threading.

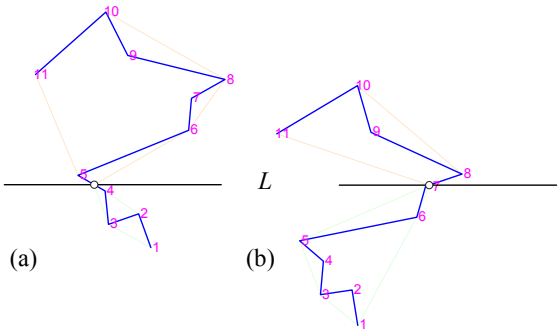


Figure 1: Two snapshots of a 10-segment polygonal chain passing through a point-hole in the x-axis.

1 Introduction

We define a simple (non-self-intersecting) open planar curve C to be *threadable* if there exists a continuous sequence of rigid motions that allows C to pass through a point-hole o in an infinite line L without any other point of C ever touching L. For fixed L, we will take L to be the x-axis and o to be the origin; equivalently we can view C as fixed and L moving (Lemma 1). C could be a polygonal chain or a smooth curve. C is open in the sense that it is not closed to a cycle. An example is shown in Fig. 1; animations are available at `http://cs.smith.edu/~jorourke/Threadable/`.

Note that our definition requires "strict threadability" in the sense that no other point of C touches L. So, for example, the curve illustrated in Fig. 2 is not threadable.

This notion has appeared in the literature in another guise. In particular, a threadable curve C corresponds to a "generalized self-approaching curve" with width π in both directions, as defined in [AAI+01]. However, those authors do not explore that concept, and in any case, our explorations focus on different properties of C. Nevertheless, our algorithms are quite similar to those for recognizing self-approaching curves and increasing-chord paths in [ACG+12].[1]

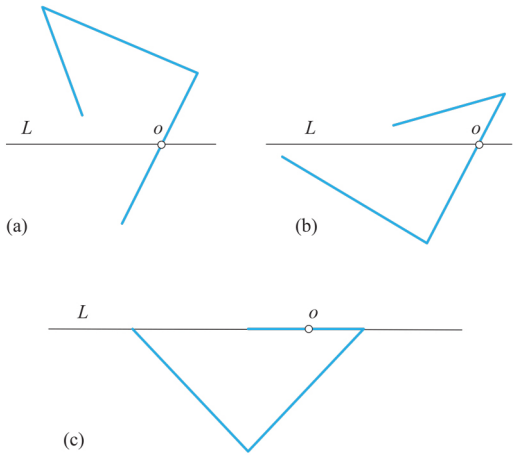


Figure 2: A curve that is not threadable. To pass completely through o, an edge would have to lie on L.

*Full version: `https://arxiv.org/abs/1801.08003`, v3.

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

[1]We thank Anna Lubiw for these references.

One could view our topic as a specialized motion-planning problem, but it seems not directly addressed in the literature. Work of Yap [Yap87], discussed in Section 7, can be viewed as a higher dimensional version. Research examining the fabrication of hydraulic tubes [AFM03], as well as work on "producible protein chains" [DLO06], lead to workspace-clearance concerns, to which we return in Section 8. We will see that classical computational geometry tools suffice to address our problems, but some interesting questions are raised.

1.1 Definition Consequences

We now explore a few consequences of the definition.

Lemma 1 *If a curve C is threadable, then through every point $p \in C$ there is a line L that meets C in exactly p: $L \cap C = \{p\}$, and L properly crosses C at p.*

Note that L tangent to C is insufficient for threadability, for then C would locally lie on one side of L. This is why the lemma insists on proper crossings.

What is perhaps not immediate is the implication in the other direction to Lemma 1:

Lemma 2 *If a curve C has the property that through every point $p \in C$ there is a line L that meets C in exactly p, and L properly crosses C at p, then C is threadable.*

The reason this is not immediate, is that it is conceivable that the orientation of the line changes discontinuously at some point $p \in C$, requiring an instantaneous rigid "jump" motion of C to pass through L, rather than a continuous rigid motion. A proof is deferred until we can rule out this discontinuity (Section 3).

1.2 Monotone Curves

A *monotone curve* C is defined as one that meets all lines parallel to some line L in a single point (if *strictly monotone*), or which intersects every such line in either a point or a segment (if *non-strictly monotone*). Every strictly monotone curve is threadable, and one can view threadability as a generalization of monotonicity, allowing the orientation of L to vary.

2 Butterflies

Define the *butterfly* $\mathrm{bf}(p)$ for $p \in C$ to be the set of all lines L satisfying the threadability condition at p: those lines that meet C in exactly p and properly cross C at p. Let L be one line in $\mathrm{bf}(p)$, and view C as passing through L at p. Then the convex hull H^+ of the chain from p upward is above L and meets L exactly at p, and the hull H^- of the chain from p downward is below L and again meets L exactly at p. (Here "upward" and "downward" are not meant literally, but just convenient shorthand for the two portions of the curve delimited by a roughly horizontal L.) If either hull met L in more than just p, then strict threadability would be violated at L. Now rotate L counterclockwise about p until it hits C at some point other than p, and similarly clockwise. The stopping points determine the butterfly *wing-lines*. See Fig. 3.

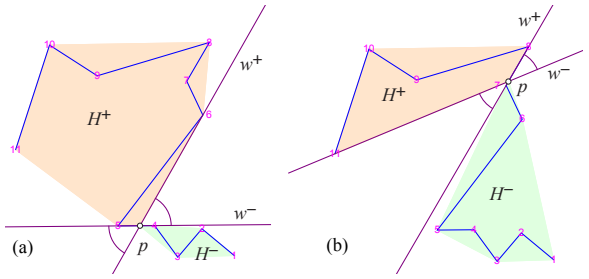


Figure 3: Here C is fixed, and two $\mathrm{bf}(p)$'s are shown. Note the hulls H^+ and H^- meet at exactly p. (a) The stopping point ccw is vertex 6 and cw it is vertices $4, 5$.

Thus $\mathrm{bf}(p)$ is an open double wedge. Its two boundary wing-lines w^+ and w^- (which are not part of $\mathrm{bf}(p)$) must both be externally supported by points of C distinct from p. Each wing must touch C on at least one of its two halves with respect to p. Note by our definition, $\mathrm{bf}(p)$ can never be a line; rather it becomes empty when the wing-lines merge to one line.

3 Upper and Lower Hulls

It is not difficult to see that the upper convex hull H^+ changes continuously (say, under the Hausdorff distance measure) as p moves along C, and similarly for H^-. This has long been known in the work on computing "kinetic" convex hulls of continuously moving points (although we have not found an explicit statement). Roughly, because each point in the convex hull of a finite set of points is a convex combination of those points, moving one point p a small amount ε changes the hull by at most a small amount δ. For more detail, see [Nie17].

Because the hulls change continuously, the butterflies change continuously as well. So we have finally established Lemma 2: If there is a line through every $p \in C$ meeting the threadability criteria, then indeed C is threadable: there are continuous rigid motions that move C through a point-hole in a line.

And now this is an immediate consequence of Lemma 2 and our definition of $\mathrm{bf}(p)$:

Lemma 3 *A curve C is threadable if and only if $\mathrm{bf}(p)$ is never empty for any $p \in C$.*

We can also now see the following characterization, which is the basis of the algorithm in the next section:

Lemma 4 *A curve C is threadable iff, for every $p \in C$, the upper and lower hulls intersect in exactly p, i.e., $H^+ \cap H^- = \{p\}$.*

Proof. ($\Rightarrow$:) Suppose C is threadable, but $H^+ \cap H^- \neq \{p\}$. We then show C could not be threadable.

- Case 1: $H^+ \cap H^-$ is a 2D region (Fig. 4(a)). Then p is strictly interior to one of H^+ or H^-. So, the butterfly $= \varnothing$. Therefore C is not threadable by Lemma 3.
- Case 2: $H^+ \cap H^-$ is a segment (Fig. 4(b)). Note the intersection could not consist of ≥ 2 segments, for that would violate the convexity of convex hulls. So, the butterfly wings reduce to a line; so the butterfly is empty. And again, C is not threadable by Lemma 3.

($\Leftarrow$:) Assume $H^+ \cap H^- = \{p\}$ for every p. Then, by the definition of $\mathrm{bf}(p)$, for every p the butterfly is non-empty, because one could rotate a line through p until it hits $H^{\pm}$. So Lemma 3 implies that C is threadable. □

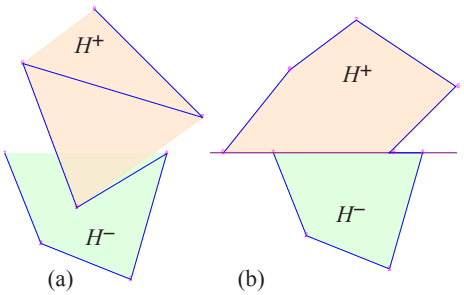


Figure 4: (a) An example of Case 1: $H^+ \cap H^-$ is a 2D region. (b) An example of Case 2: $H^+ \cap H^-$ is a segment.

4 Algorithm for Threadability

In light of Lemma 4, we can detect whether a polygonal chain is threadable by computing H^+ and H^- for all p along C, and verifying that p never falls inside either hull, i.e., ceases to be a nonflat vertex of either hull. Let p be a point on $C = (v_1, v_2, \ldots, v_n)$, which we view as moving "vertically downward" from v_1 (top) to v_n (bottom). Let the edges of C be $e_i = (v_{i-1}v_i)$. We concentrate on constructing $H = H^+$ as p moves downward along C. Clearly the same process can be repeated to construct H^-.

As p moves down along C, $H = \mathrm{hull}\{v_1, \ldots v_{i-1}, p\}$ grows in the sense that the hulls form a nested sequence. Thus once a vertex of C leaves ∂H, it never returns to ∂H (where ∂H is the boundary of H.) At any one time, p is a vertex of H. Let a_1, a_2 be the vertices of H right-adjacent to p, and b_1, b_2 the vertices left-adjacent, so that (b_2, b_1, p, a_1, a_2) are consecutive vertices of H. Finally, let A and B be the lines through a_1a_2 and b_1b_2 respectively. See Fig. 5.

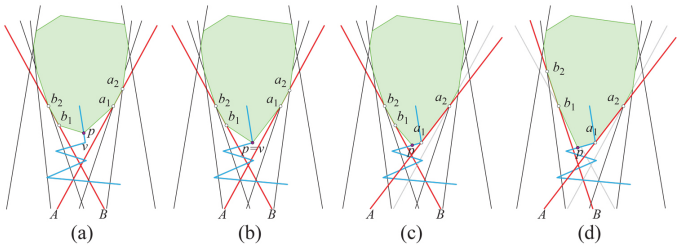


Figure 5: Algorithm snapshots. (a) H grows without combinatorial change until p reaches v. (b) $p = v$ event. (c) a_1, a_2 updated. e_i crosses B. (d) b_1, b_2 updated.

We now walk through the algorithm, whose pseudocode is displayed in the full version. Let p be on the interior of an edge $e_i = (v_{i-1}v_i)$. The portion of e_i already passed by p must lie inside H, and the remaining portion outside H. As long as p remains within the wedge region delimited by A, B, and ∂H, the combinatorial structure of H remains fixed (Fig. 5a). If p crosses A or B—say A—then a_1 leaves H and a_1, a_2 become the next two vertices counterclockwise around ∂H. If p reaches the endpoint v_i of e_i, then if e_{i+1} angles outside H, v_i becomes a new a_1 or b_1 depending on the direction of e_{i+1}. If instead, e_{i+1} turns inside H, advancing p would enter H and we have detected that C is not threadable by Lemma 4.

All the updates just discussed are constant-time updates: detecting if e_i crosses A or B, updating a_1, a_2 and b_1, b_2, and detecting if e_{i+1} turns inside H, entering $\triangle b_1 v_i a_1$.

At the end of the algorithm, H is the hull of C. It may seem surprising that we can compute the hull of C in linear time (rather than $O(n \log n)$), but Melkman showed long ago that the hull of any simple polygonal chain can be computed in linear time [Mel87]. The chain C acts almost as a pre-sorting of the points, leading to an $O(n)$ algorithm for threadability.

4.1 Rigid Motions

At any stage where the butterfly $\mathrm{bf}(p)$ is non-empty, we could choose the line L to bisect $\mathrm{bf}(p)$. This choice was used to produce the online animations cited in Section 1. To prepare for an analogous 3D-computation in Section 6, we explain the bisection choice in terms of vectors normal to L. Fig. 6(a) shows the possible L choices through p dictated by the two incident edges of H^+ and the two incident edges of H^-, illustrated by rightward rays from p along L. Rotating these 90° in (b) of the figure yields the possible vectors normal

to L. The intersection of the H^+ and H^- constraints yields an interval corresponding to bf(p), which is then bisected to select a particular N and therefore L. (The intersection always yields an interval [rather than two intervals] because each of the H^+ and H^- constraints is $\leq$ a semicircle.)

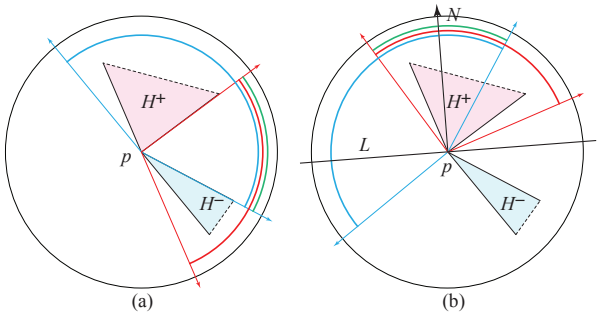


Figure 6: (a) Rightward rays along possible L's. (b) Normal vector N to L (black).

Let H_j^+ and H_j^-, $j = 1, \ldots, m$ be the sequence of hulls at the points at which there is a combinatorial change in either. Let $r_j \subseteq e$ be the range of p along edge e of C between $\{H_j^+, H_j^-\}$ and $\{H_{j+1}^+, H_{j+1}^-\}$. Then as p moves along r_j, the wings of the butterfly bf(p) have the same set of tangency points on the hulls. With L chosen as the bisector of bf(p), translation of p along r_j leads to translation and rotation of L. It is not difficult to see that the rotation implied by p moving along r_j reverses at most once, from clockwise to counterclockwise or vice versa. This is evident in Fig. 7, where the butterfly angle θ bisected to yield L has at most one local maximum. Thus each slide of p along r_j leads to at most two monotonic rotations. We call a slide and a simultaneous monotonic rotation an *elementary rigid motion*. But note that, although "elementary," these motions are not pure rotations and pure translations, but rather the particular mix determined by the slide and the butterfly bisection. We leave these elementary motions as the output rigid motions, not further analyzed into explicit analytical expressions.

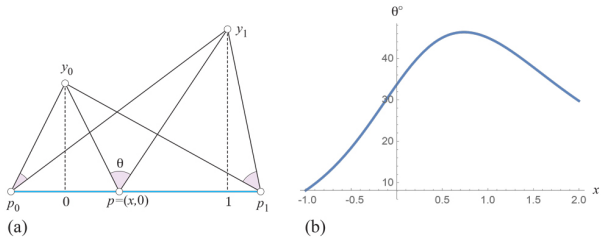


Figure 7: (a) p slides along edge e from p_0 to p_1, $r_j = (p_0, p_1) \subseteq e$. (b) The butterfly angle θ has at most one local maximum throughout the range.

Thus the sequence of $O(n)$ hulls provides a set of $O(n)$ elementary rigid motions to thread C, which we used to produce the online animations.

4.2 Difficult-to-Thread Curves

One easy consequence of our analysis is that a threadable curve need never "back-up" while threading through a hole, because p never enters $H^{\pm}$ as it progresses along the chain. However, one could define the "difficulty" of threading by, say, integrating the absolute value of the back-and-forth rotations necessary to thread. Then variations on the curve shown in Fig. 8 are difficult to thread in this sense. For each pair of adjacent spikes require a rotation by θ, and with many short spikes, there is no bound on $\sum |\theta|$ even for a fixed-length chain.[2]

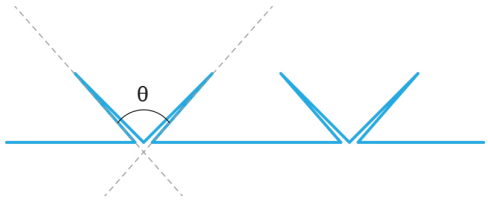


Figure 8: A threadable curve that requires repeated rotations. Animation: `http://cs.smith.edu/~jorourke/Threadable/`, Example 2.

5 Algebraic Curves

In the full version, we sketch an argument that shows detection of threadability for algebraic curves is achievable in time $O(d^4)$, where d is the degree of the curve.

6 Threadable Curves in 3D

The results in Section 4 can be extended to $\mathbb{R}^3$, asking whether a 3D polygonal chain C can pass through a point-hole in a plane. First we roughly sketch an algorithm. We claim without proof that the natural generalization of the 2D lemmas hold in 3D as well.

Again Lemma 4 is the key: we need that $H^+ \cap H^- = \{p\}$ holds for all p on C. Again computing H^+ and H^- will suffice to answer all questions; see Fig. 9. But now what was the simple wedge region between hull supporting lines A, B, and ∂H, becomes a more complex region R bounded by $O(n)$ hull-supporting planes, and the portion of ∂H formed by the faces incident to p, i.e., what is called star(p) in simplicial-complex theory (which has size $O(n)$). Setting aside complexity issues temporarily, the next edge e_{i+1} on which p will travel

[2] Thanks to Anna Lubiw for this observation.

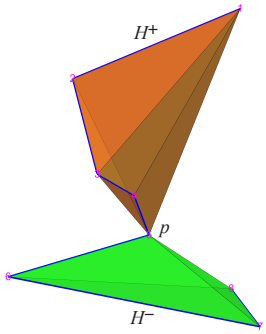


Figure 9: Upper and lower hulls for a 3D polygonal chain. Animation: `http://cs.smith.edu/~jorourke/Threadable/`, Example 6.

must be intersected with the planes bounding this region R, to determine whether R changes combinatorially, and if so, which supporting plane is first pierced by e_{i+1}.

The planes bounding R that are not determined by faces in star(p) are the planes incident to an edge of link(p), i.e., the edges of star(p) not incident to p, which form a topological circle. See Fig. 10. When e_{i+1} pierces a plane A supporting face $\triangle abc$ of H, with ab an edge of link(p), then ab is deleted from the link, and ac and cb added, and the planes incident to these new link edges are added to those defining R.

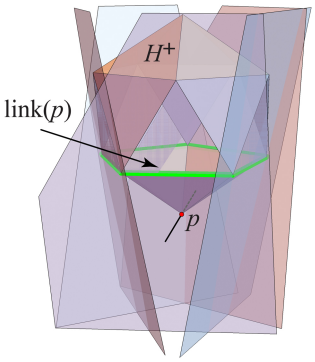


Figure 10: Faces sharing an edge with link(p) are extended to form the lower part of R. H^+ does not change combinatorially until p crosses one of those planes.

This allows H to be maintained throughout the movement of p along C. As in 2D, C is threadable if and only if p never enters either hull.

If C is threadable, selecting planes in the more complex bf(p) regions and determining rigid motions that achieve the threading are more complicated tasks than in 2D.

6.1 Updating the hull H quickly

Timothy Chan's powerful dynamic data structure for updating 3D convex hulls [Cha10] provides the tools needed to update the hull H quickly. Here "quickly" means in amortized expected $O(\text{polylog}\, n)$ time. His "nonvertical ray shooting" queries permit determining if the next edge e_{i+1} intersects a supporting plane of the region R described above, and if so, which one. Then that plane can be deleted, and new planes inserted according to the new link(p), as identified above. Thus the computation of the hulls H^+ and H^-—and therefore threadability detection—can be achieved in $O(n \,\text{polylog}\, n)$ time.

6.2 Butterfly "bisecting" planes

The equivalent of the butterfly bf(p) in 3D is a more complicated region than in 2D, and choosing a plane P through p separating H^+ and H^- (the analog of L) is correspondingly more complicated. As in 2D, we identify P by its normal vector N, say, pointing toward H^+. The outward normals to the faces of H^- incident to p form a convex geodesic polygon on the Gaussian sphere, with each node a face normal, and each geodesic arc corresponding to the dihedral angle along the edge shared by two adjacent faces. See, e.g., [BLS07]. Any point within this geodesic polygon corresponds to a normal vector whose plane supports H^- at p. Repeating this for H^+ yields another geodesic polygon corresponding to the faces of H^+ incident to p. Using outward face normals leads to normals pointing toward H^-; reflecting this geodesic polygon through the origin then orients the normals for H^+ and H^- consistently. See Fig. 11. Then the butterfly region bf(p) is determined by the intersection I of these two geodesic polygons.

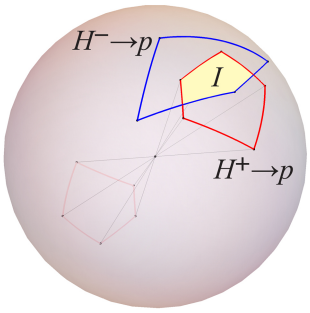


Figure 11: Gaussian sphere. The blue polygon represents the faces of H^- incident to p, and the red polygon the faces of H^+ incident to p. Any point in the (yellow) intersection I is the normal vector N of a plane P in bf(p).

The equivalent of bisecting bf(p) in 2D would be choosing the centroid of the intersection region I on

the Gaussian sphere; of course any point in the interior of I would suffice. In 2D we argued that, as p slides along an edge e between combinatorial changes in either H^+ or H^-, the rigid rotation reverses direction at most once, which led to a linear-size description of the rigid motions. In 3D, even with p on one edge e between combinatorial changes, it seems that the intersection region I on the Gaussian sphere might change $\Omega(n)$ times, requiring recalculation of $N \in I$. This complicates describing the rigid motions in a concise manner. We leave finding a clean notion of what should constitute a "elementary rigid motion" in 3D to future work, but we note that the rigid motions for threading are analytically determined and could be detailed to any precision desired.

7 Higher Dimensional Generalizations

There are two natural generalizations to higher dimensions, but neither seems a fruitful line of future inquiry. The first retains the curve as a 1-dimensional object which must pass through a hole in a hyperplane in $\mathbb{R}^d$.

The second generalization replaces the curve with a polygon P, which must pass through a slit in L. This topic has been explored previously, in two versions. We cite [Yap87] and [BVK05] and leave further discussion to the full version.

8 Open Problems

1. In $\mathbb{R}^3$, can finding a plane P separating H^+ and H^-, as sketched in Section 6, be achieved in $O(n \operatorname{polylog} n)$ time? In other words, can the intersection I of the two geodesic polygons be maintained in amortized expected $O(\operatorname{polylog} n)$ time?
2. Is there a natural definition of what constitutes an "elementary rigid motion" in $\mathbb{R}^3$, and how many such motions are needed to thread a polygonal curve of n segments?
3. If C were a hydraulic tube, it would be necessary to ensure clearance regions above and/or below L are empty of other objects to avoid collisions [AFM03]. If C represents a polygonal protein chain, clearance within a cone is important in some models [DLO06]. Finding minimum clearance regions requires more careful selection of L in $\mathrm{bf}(p)$, rather than just using the bisector as we suggest in Section 4.1. The question is most relevant in $\mathbb{R}^3$.
4. Suppose instead of C passing through a line, C were to pass through a point hole in a polygonal k-chain. What is the complexity of finding a threading motion as a function of n and k?
5. If C is not threadable, what is the shortest slit in L through which C could pass? Or, in $\mathbb{R}^3$, the smallest radius hole in a plane? Likely Yap's door width algorithm [Yap87] could apply to the 2D problem, but it would be attractive to find a hull-based approach in 2D and 3D.

Acknowledgements. We thank Mikkel Abrahamsen, Anna Lubiw, Joseph Mitchell, and the referees for helpful suggestions.

References

[AAI+01] O. Aichholzer, F. Aurenhammer, C. Icking, R. Klein, E. Langetepe, and G. Rote. Generalized self-approaching curves. *Discrete Appl. Math.*, 109:3–24, 2001.

[ACG+12] S. Alamdari, T.M. Chan, E. Grant, A. Lubiw, and V. Pathak. Self-approaching graphs. In *Internat. Symp. Graph Drawing*, pages 260–271. Springer, 2012.

[AFM03] E.M. Arkin, S.P. Fekete, and J.S.B. Mitchell. An algorithmic study of manufacturing paperclips and other folded structures. *Comput. Geom. Theory Appl.*, 25:117–138, 2003.

[BLS07] T. Biedl, A. Lubiw, and M. Spriggs. Cauchy's theorem and edge lengths of convex polyhedra. *Algorithms Data Structs.*, pages 398–409, 2007.

[BVK05] P. Bose and M. Van Kreveld. Generalizing monotonicity: On recognizing special classes of polygons and polyhedra. *Internat. J. Comput. Geom. & Appl.*, 15(06):591–608, 2005.

[Cha10] T.M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. *J. ACM*, 57(3):16, 2010.

[DLO06] E.D. Demaine, S. Langerman, and J. O'Rourke. Geometric restrictions on producible polygonal protein chains. *Algorithmica*, 44(2):167–181, 2006.

[Mel87] A.A. Melkman. On-line construction of the convex hull of a simple polyline. *Info. Proc. Letters*, 25(1):11–12, 1987.

[Nie17] M. Nientker. Convex hulls change continuously as one point moves continuously, October 2017. `https://math.stackexchange.com/q/2529897`.

[Yap87] C.-K. Yap. How to move a chair through a door. *IEEE J. Robotics Automation*, 3(3):172–181, 1987.

Looking for Bird Nests: Identifying Stay Points with Bounded Gaps

Ali Gholami Rudi*

Abstract

A stay point of a moving entity is a region in which it spends a significant amount of time. In this paper, we identify all stay points of an entity in a certain time interval, where the entity is allowed to leave the region but it should return within a given time limit. This definition of stay points seems more natural in many applications of trajectory analysis than those that do not limit the time of entity's absence from the region. We present an $O(n \log n)$ algorithm for trajectories in R^1 with n vertices and a $(1+\epsilon)$-approximation algorithm for trajectories in R^2 for identifying all such stay points. Our algorithm runs in $O(kn^2)$, where k depends on ϵ and the ratio of the duration of the trajectory to the allowed gap time.

1 Introduction

The question, asking where a moving entity, like an animal or a vehicle, spends a significant amount of its time is very common in trajectory analysis [1]. These regions are usually called popular places, hotspots, interesting places, stops, or stay points in the literature. There are several definitions of stay points and different techniques have been presented to find them [2, 3, 4, 5, 6]. However, from a geometric perspective, which is the focus of the present paper, few papers are dedicated to this problem.

Benkert et al. [2] defined a popular place to be an axis-aligned square of fixed side length in the plane which is visited by the most number of distinct trajectories. They modelled a visit either as the inclusion of a trajectory vertex or the inclusion of any portion of a trajectory edge, and presented optimal algorithms for both cases. Gudmundsson et al. [3] introduced several different definitions of trajectory hotspots. In some of these definitions, a hotspot is an axis-aligned square that contains a contiguous sub-trajectory with the maximum duration and in others it is an axis-aligned square in which the entity spends the maximum possible duration but its presence may not be contiguous. For hotspots of fixed side length, for the former they presented an $O(n \log n)$ algorithm and for the latter they presented an algorithm with the time complexity $O(n^2)$, where n is the number of trajectory vertices. Damiani et al. [7], like some of the cases considered by Gudmundsson et al. [3], allowed gaps between stay point and presented heuristic algorithms for finding them.

There are applications in which we need to identify regions that are regularly visited. Djordjevic et al. [8] concentrated on a limited form of this problem and presented an algorithm to decide if a region is visited almost regularly (in fixed periods of time) by an entity. However, in many applications that require spatio-temporal analysis, these definitions are inadequate. For instance, a bird needs to return to its nest regularly to feed its chicks. In other words, the bird may leave its nest but it cannot be away for a long time. We would like to find all possible locations for its nest.

Arboleda et al. [6] studied a problem very similar to the focus of the present paper, except that they assumed the algorithm takes as input, in addition to the trajectories, a set of polygons as potential stay points or interesting sites. They presented a simple algorithm to identify stay points among the given interesting sites; their algorithm computes the longest sub-trajectory visiting each interesting site for each trajectory, while allowing the entity to leave the site for some predefined amount of time. They also mentioned motivating real world examples to show that in some applications, it makes sense to allow the entity to leave the site for short periods of time, like leaving a cinema for the bathroom.

Our goal is identifying all trajectory stay points, i.e. axis-aligned squares in which the entity is always present, except for short periods of time, where both the side length of the squares and the allowed gap time are specified as inputs of the algorithm and assumed to be fixed. Note that we ignore the duration in which the entity stays in a region. If, for instance, a region with the maximum duration among our stay points is desired, our algorithm can be combined with those that find a stay point with the maximum duration, but allow unbounded entity absence, like the ones presented by Gudmundsson et al. [3].

This paper is organized as follows. In Section 2, we introduce the notation and define some of the main concepts of this paper. In Section 3, we handle trajectories in R^1 and present an algorithm to find all stay points of such trajectories with the time complexity $O(n \log n)$. We focus on trajectories in R^2 in Section 4 and present an approximation algorithm for finding their stay points. We conclude this paper by showing that the complexity of the stay map of two-

*Department of Electrical and Computer Engineering, Bobol Noshirvani University of Technology, gholamirudi@nit.ac.ir

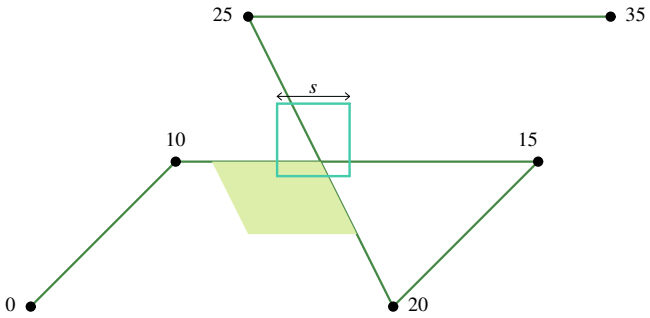


Figure 1: An example two-dimensional trajectory. The number near each vertex shows its timestamp. The green region is the stay map and the green square is a stay point ($g = 15$).

dimensional trajectories can be $\Theta(n^2)$.

2 Preliminaries

A trajectory T describes the movement of an entity in a certain time interval. Trajectories can be modelled as a set of vertices and edges in the plane. Each vertex of T represents a location at which the entity was observed. The time of this observation is indicated as the timestamp of the vertex. We assume that the entity moves in a straight line and with constant speed from a vertex to the next; the edges of the trajectory connect its contiguous vertices. A sub-trajectory of T for a time interval (a, b) is denoted as $T(a, b)$, and describes the movement of the entity from time a to time b. Except possibly the first and the last vertices of a sub-trajectory, which may fall on an edge of T, its set of vertices is a subset of those of T. The stay points considered in this paper are formally described in Definition 1. We use the symbols defined here, such as g and s, throughout the paper without repeating their description. Also, any square that appears in the rest of this paper is axis-aligned and has side length s.

Definition 1 *A stay point of a trajectory T in R^2 is a square of fixed side length s in the plane such that the entity never spends more than a given time limit g outside it continuously.*

The goal of this paper is identifying all stay points of a trajectory, or its stay map (Definition 2). Note that the parameters s and g are assumed to be fixed and specified as inputs of the algorithm.

Definition 2 *The stay map M of a trajectory T in R^2 is a subset of the plane such that every square of side length s whose lower left corner is in M is a stay point of T, and the lower left corners of all stay points of T are in M.*

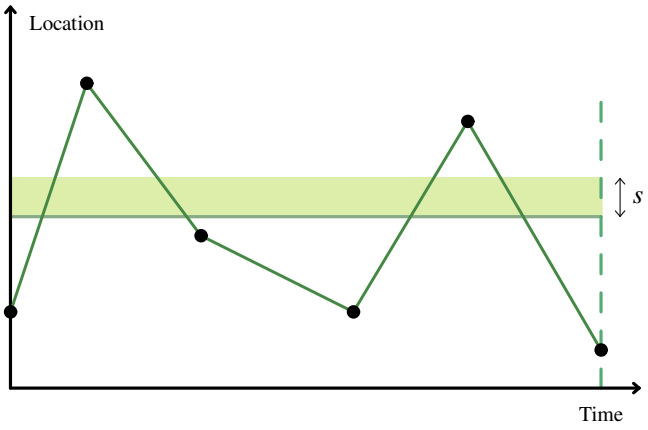


Figure 2: Mapping a one-dimensional trajectory to the time-location plane. The green rectangle of height s shows a possible stay point.

Figure 1 shows an example trajectory, its stay map, and one of its stay points. Note that every square, whose lowest left corner is in the stay map, is a stay point. Although these definitions are presented for trajectories in R^2, they can be trivially adapted for one-dimensional trajectories, as we do in Section 3.

3 Stay Maps of One-Dimensional Trajectories

Let T be a trajectory in R^1. A stay point of T is an interval of length s such that the entity never leaves it for a period of time longer than g. The stay map M of T is the region containing the left end points of all stay points of T. In this section, we present an algorithm for finding M.

Lemma 3 *The stay map M of a trajectory T in R^1 is continuous.*

Proof. To obtain a contradiction, let points p and q be inside M and v be outside it such that $p < v < q$ (our assumption that M is non-continuous implies the existence of this triple). Let r_p, r_q, and r_v be three segments of length s, whose left corners are at p, q, and v, respectively. Clearly, r_p and r_q are stay points while r_v is not. Whenever the entity moves to the left of v, it must return to q before the time limit g to visit r_q. Also, whenever the entity moves beyond the right end point of r_v (which is outside r_p), it must return to r_p before the time limit. Therefore, it can never be outside r_v for more than time g and this implies that v is also a stay point and inside M, which yields the desired contradiction. □

Lemma 4 *Given a trajectory T with n vertices in R^1, we can answer in $O(n)$ time whether a point p is in the stay map or not, and if not, whether the stay map is on its left side or on its right side.*

Proof. Define r as segment pq, in which q is $p+s$. Testing each trajectory edge in order, we can compute the duration of each maximal sub-trajectory outside r and check if it is at most g. Therefore, we can decide if p is the left end point of a stay point in $O(n)$ time. If it is not a stay point, there is at least one time interval, in which the entity spends more than time g on the left or on the right side of r. Without loss of generality, suppose it does so on the left side. Then, no point on the right of r can be a stay point and therefore the whole stay map of T must appear on the left of p. This again can be tested in $O(n)$ time by processing trajectory edges. □

An event point of a trajectory T in R^1 is a point on the line in which one of the following occurs: i) a trajectory vertex lies on that point, ii) the time gap between two contiguous visits to that point is exactly g.

Lemma 5 *The stay map M of a trajectory T starts and ends at an event point or at distance s from one.*

Proof. By Lemma 3, M is continuous. Let p be the left end point of the stay map M. Let $r = pq$ be a segment such that $q = p+s$. Whenever the entity leaves r through p, it returns by passing it again within the time limit g. Similarly, if the entity leaves r through q, it visits q again within time g. Suppose, for the sake of contradiction, that p is not an event point. Then, we can move r slightly to the left to obtain r'. r' must also be a stay point because every time the entity leaves it from either of its end points, it returns within time g, because neither p nor q is an event point (the time between the contiguous visits of the entity is not exactly g and they are not on a trajectory vertex). This contradicts the choice of p. A similar argument shows that the right endpoint of M must also be an event point or at distance s from one. □

Lemma 6 *The set of event points of a trajectory with n vertices can be computed in $O(n \log n)$ time.*

Proof. We map the trajectory to a plane such that a trajectory vertex at position p with timestamp t is mapped to point (t, p) (see Figure 2). Obviously, the polygonal path representing the trajectory in this plane is y-monotone. We perform a plane sweep by sweeping a line parallel to the x-axis in the positive direction of the y-axis in this plane.

The edges in this plane chop the sweep line into several segments. We maintain the length of every such segment during the sweep line algorithm. When the sweep line intersects a trajectory vertex v, an event point is recorded and, based on the other end point of the edges that meet at that vertex, one of the following cases occurs:

1. If v is the lowest end point of both edges, two new segments are introduced. Based on the slope of the edges bounding each segment, we record an event at which the distance between the edges is exactly g, if they are long enough.

2. If v is the highest end point of both edges that meet at v, three segments on the sweep line are merged (when the sweep line is before v, three segments are created by the edges incident to v, at v, there are two such segments, and after v, they merge into one). We also record an event for the location at which the length of the remaining segment becomes g in the plane.

3. If v is the highest end point of one edge and the lowest end point of another, the event scheduled for the location at which the length of each of the two incident segments on the sweep line are g may need to be updated.

Note that since the sweep line stops at n vertices and at each vertex only a constant number of event points are added, the total number of event points is $O(n)$. □

Theorem 7 *The stay map M of a trajectory T with n vertices in R^1 can be computed in $O(n \log n)$ time.*

Proof. Lemma 6 implies that the set of event points of T can be computed with the time complexity $O(n \log n)$. From this set, we can obtain an ordered sequence of event points and points at distance exactly s from them in $O(n \log n)$ time (note that the length of this sequence is still $O(n)$). Based on Lemma 5, M starts and ends at a point of this sequence. Also, Lemma 4 implies that we can decide if any of the end points of M appears before or after any point in $O(n)$ time. Therefore, we can perform a binary search on the sequence obtained from the event points of T to find the left and the right end points of M. Since the length of the sequence is $O(n)$, the time complexity of the binary search is $O(n \log n)$. □

Unfortunately, this algorithm cannot be adapted for two-dimensional trajectories, because their stay maps may no longer be continuous.

4 Stay Maps of Two-Dimensional Trajectories

We use the notation $P(a, b)$ to denote the region that contains the lower left corners of all squares of side length s that contain at least one point of the sub-trajectory $T(a, b)$. We also use $M(a, b)$ to indicate the stay map of the sub-trajectory $T(a, b)$. We assume that trajectory T starts at time 0 and has total duration D. It is clear that every point in the stay map of T must appear in $P(t, t+g)$ for any value of t, where $0 \leq t \leq D-g$ (because the entity cannot be outside a stay point of T for more than time g). Therefore, the stay map of T is the intersection of $P(t, t+g)$ for every possible value

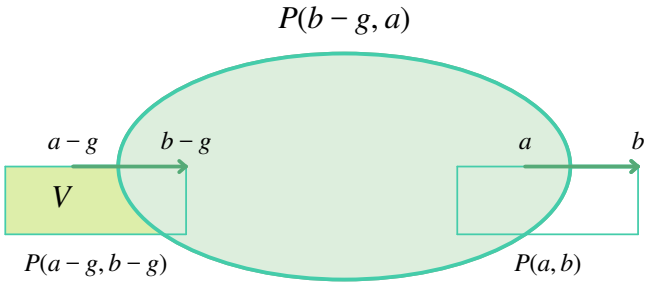


Figure 3: The difference V in Algorithm 1, when $P(a-g, b-g)$ and $P(a, b)$ do not overlap.

of t, $0 \leq t \leq D-g$. This suggests the general scheme demonstrated in Algorithm 1 for finding the stay map of a two-dimensional trajectory, assuming $D > g$.

Algorithm 1 *Let T be two-dimensional trajectory with n edges and total duration D. Compute the stay map of T ($M(0, D)$) as follows.*

1. *Compute $P(0, g)$, as the union of polygons $P(u, v)$, for all edges uv in $T(0, g)$.*

2. *Let $M(0, g)$ be $P(0, g)$. This is not strictly correct as $M(0, t)$ must include the complete plane when $t \leq g$ and its value changes to a subset of $T(0, g)$ for any value of $t > g$. This simplifying assumption, however, does not affect the correctness of the algorithm, since $D > g$.*

3. *Incrementally compute $M(0, D)$ as follows. Compute $M(0, b)$ from $M(0, a)$, in which $M(0, a)$ is the last computed stay map and b is the smallest value after a, such that $b - g$ or b is the timestamp of a trajectory vertex. Let V be the difference between $M(0, a)$ and $M(0, b)$ (note again that $M(0, b)$ is a subset of $M(0, a)$). After computing V, we obtain $M(0, b)$ by excluding V from $M(0, a)$.*

The core of Algorithm 1 is the computation of the difference V. By the choice of b, $T(a-g, b-g)$ and $T(a, b)$ are both line segments. The value of V depends on these segments and $T(b-g, a)$.

Let r be a square, whose lower left corner is in V and let $a - g + \delta$ be the time of entity's departure from r before time $b - g$. Since the lower left corner of r is in V, r is not visited by the entity in the sub-trajectory $T(a-g+\delta, a+\delta)$. In other words, any point not in $P(a-g+\delta, b-g)$, $P(b-g, a)$, and $P(a, a+\delta)$ for any value of δ in $0 \leq \delta \leq g$ cannot be a stay point.

To make the computation of V easier, we define V' as follows (V' is very similar to V, except that it ignores $P(b-g, a)$):

$$V' = \bigcup_{0 \leq \delta \leq g} P(a-g, a-g+\delta) \setminus (P(a-g+\delta, b-g) \cup P(a, a+\delta))$$

V' contains the lower left corners of all squares that have been visited during the interval $(a-g, a-g+\delta)$, but have not been visited in $(a-g+\delta, b-g)$ or $(a, a+\delta)$ for some δ in $0 \leq \delta \leq g$. Then, $V = V' \setminus P(b-g, a)$.

If no square intersects both $T(a-g, b-g)$ and $T(a, b)$, V' is $P(a-g, b-g)$. This case is shown in Figure 3, in which V' is the rectangle on the left. Otherwise, V' depends on the relative speed of the entity in these sub-trajectories. In both cases, V' is a polygon of constant complexity and can be computed in constant time. We do not discuss the details of the computation of V' in this paper, however. Since $T(b-g, a)$ consists of $O(n)$ edges, $P(b-g, a)$ is the union of $O(n)$ simple polygons. Therefore, $V' \setminus P(b-g, a)$ is also the union of a set of polygons with the total complexity $O(n)$. Let V_t be the union of the differences V for all iterations of the third step of Algorithm 1 (note that the complexity of V_t is $O(n^2)$). When the algorithm finishes, $M(0, D)$ is $P(0, g) \setminus V_t$. Since the computation of V_t requires finding the union of polygons with the total complexity $O(n^2)$, an $O(n^2)$ implementation of this exact algorithm seems unlikely.

4.1 Approximate Stay Maps of Two-Dimensional Trajectories

In Algorithm 2, we consider $P(t, t+g)$ for limited discrete values of t to compute *approximate stay maps* of a trajectory (Definitions 8 and 9), to improve the time complexity of Algorithm 1.

Definition 8 *A $(1+\epsilon)$-approximate stay point of a trajectory T in R^2 is a square of fixed side length s, such that the entity is never outside it for more than $g + \epsilon g$ time.*

Definition 9 *A $(1+\epsilon)$-approximate stay map of a trajectory T in R^2 is the region containing the lower left corners of all exact stay points of T and possibly the lower left corners of some of its $(1+\epsilon)$-approximate stay points.*

Algorithm 2 *Let T be a trajectory in R^2 with n edges and total duration D and let ϵ be any real positive constant no greater than D/g. Compute a $(1+\epsilon)$-approximate stay map of T as follows.*

1. *Compute $P(t, t+g)$ for $t = i\lambda$ for integral values of i from 0 to D/λ, where λ is ϵg. We call $P(t, t+g)$ for any value of t a snapshot of T.*

2. *Compute the intersection of these snapshots. For this, we can use the topological sweep of Chazelle and Edelsbrunner [9] on the subdivision of the plane induced by the edges of the snapshots and include in the output the faces present in all snapshots.*

Figure 4: The entity leaves a square at t_b and returns at t_e. If $t_e - t_b$ is larger than $g + g\epsilon$, there is a snapshot in which the entity is outside the square.

Theorem 10 *For trajectory T in R^2 with n edges and total duration D and any real positive constant ϵ no greater than D/g, Algorithm 2 computes a $(1+\epsilon)$-approximate stay map of T.*

Proof. Since the output of Algorithm 2 is the intersection of different snapshots of T, the lower left corner of every stay point must be inside it. Therefore, it suffices to show that every point in the output of the algorithm is the lower left corner of a $(1+\epsilon)$-approximate stay point.

Let r be a square whose lower left corner is in the region reported by this algorithm. Suppose that the entity leaves r at t_b and reenters r at t_e. We can set $t_b = 0$ for handling the initial part of the trajectory, and, if the entity never returns to r, we can set $t_e = D$. To prove the approximation factor, we show that $t_e \leq t_b + g + \epsilon g$. Let i be the largest index such that $\lambda i \leq t_b$ and let $t_1 = \lambda i$. We show that the entity must return before time $t_1+\lambda+\lambda/\epsilon$. Otherwise, $P(t_1+\lambda, t_1+\lambda+\lambda/\epsilon)$, which is a snapshot since λ/ϵ is equal to g, does not contain the lower left corner of r (this is demonstrated in Figure 4) and this contradicts the assumption that it is included in the region returned by the algorithm. Therefore, the entity cannot be outside r for longer than $\lambda/\epsilon + \lambda$, and $t_e \leq t_b + g + \epsilon g$. □

Theorem 11 *The time complexity of Algorithm 2 is $O(n^2/\epsilon^2 + \sigma^2/\epsilon^2)$, in which σ is D/g.*

Proof. A subdivision of the plane by m line segments has $O(m^2)$ faces and can be swept with the same time complexity [9]. Moreover, the number of the segments of each snapshot depends on the number of vertices of the sub-trajectory inside that snapshot (the region containing the lower left corners of the squares that intersect an edge of the sub-trajectory is a polygon with a constant number of sides). We, therefore, count the total number of vertices of the sub-trajectories in all snapshots. There are two types of trajectory vertices in each snapshot: those present in the original trajectory T and the end points of the snapshot, which may not coincide with a trajectory vertex. Since the duration of each snapshot is g and the difference between the start time of contiguous snapshots is ϵg, each trajectory vertex appears in at most $1/\epsilon$ snapshots. Therefore, the total number of vertices is at most $n/\epsilon + 2D/(\epsilon g)$ and the time complexity of Algorithm 2 is $O(n^2/\epsilon^2 + \sigma^2/\epsilon^2)$. □

It is not difficult to see that the stay map of a two-dimensional trajectory may contain $\Theta(n^2)$ faces and therefore we cannot expect an algorithm with the worst-case time complexity $o(n^2)$. In what follows, we demonstrate a trajectory with $O(n)$ edges and a stay map of $\Theta(n^2)$ faces. Trajectory edges are added incrementally, as demonstrated in Figure 5, in which filled regions represent the stay map (except for $t \leq g$, in which they represent $P(0,t)$) and arrows show trajectory edges. We assume that the entity starts at time 0 and position $(0,0)$.

Generate m vertical strips as follows. Add the second vertex at $(2s, 0)$ with timestamp $g/2$ (Figure 5.a). Move the entity to its initial position using three vertices as shown in Figure 5.b; the position of the last vertex is $(0,0)$ and its timestamp is $g - g/2n$. Create the vertical strips as follows: after every g/n time, quickly move the entity by s/n to the right (Figures 5.c–5.e). After n such steps and waiting for at least g, the current stay map consists of n vertical strips (Figure 5.f).

The same trajectory we used for creating vertical strip can be used for creating horizontal strips after rotating the trajectory 90 degrees. If this is performed after the previous step, however, this would result in a stay map (Figure 5.g), which consists of $\Theta(n^2)$ small squares.

5 Concluding Remarks

The definition of stay points with bounded gaps can be easily extended to multiple trajectories. A multi-trajectory stay point is a square that is visited by at least one of the entities in any interval of duration g. It seems possible to compute such stay maps, by modifying Algorithm 2 to compute the intersection of the union of the snapshots of different entities. However, the time complexity of this algorithm may no longer be $O(n^2)$, where n is the total number of trajectory vertices. Finding an efficient exact algorithm for the multi-trajectory version of the problem seems interesting.

As shown in Section 4, the complexity of a stay map can be $\Theta(n^2)$, rendering an algorithm with the time complexity $o(n^2)$ impossible. This bound however is not tight and a natural question is whether it is possible to find the exact stay map of two-dimensional trajectories in $O(n^2)$ time. Also, by limiting the size of the output, for instance by finding only one of the stay points, a more efficient algorithm is not unlikely. Furthermore, it seems interesting to study the problem in higher dimensions.

Acknowledgement

We thank Neda Ahmadzadeh Tori for the inspiring discussions that led to the study of this problem.

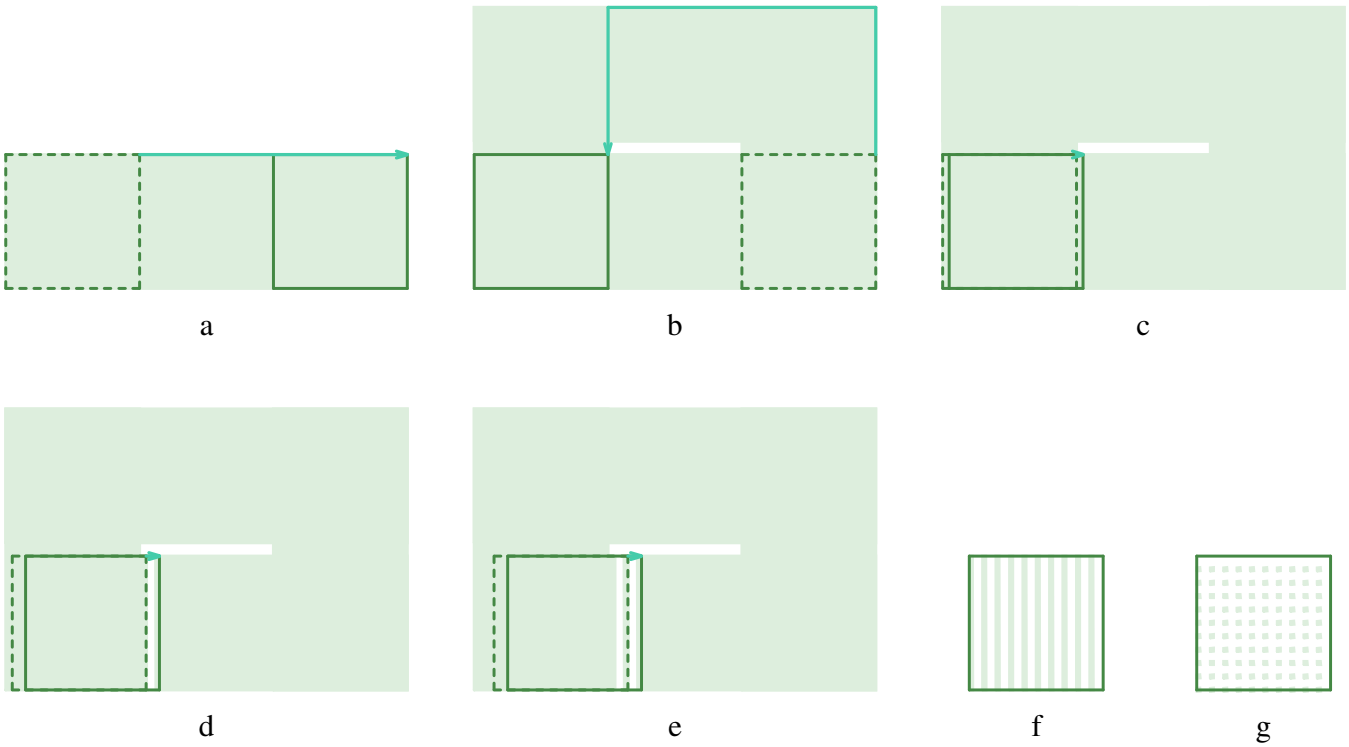


Figure 5: A trajectory with a stay map of $O(n^2)$ faces. The arrows indicate trajectory edges and filled regions indicate the stay map at each step.

References

[1] Y. Zheng. Trajectory data mining - an overview. *ACM Transactions on Intelligent Systems and Technology*, 6(3):29:1–29:41, 2015.

[2] M. Benkert, B. Djordjevic, J. Gudmundsson, and T. Wolle. Finding popular places. *International Journal of Computational Geometry and Applications*, 20(1):19–42, 2010.

[3] J. Gudmundsson, M. J. van Kreveld, and F. Staals. Algorithms for hotspot computation on trajectory data. In *SIGSPATIAL/GIS*, pages 134–143, 2013.

[4] M. Fort, J. A. Sellarès, and N. Valladares. Computing and visualizing popular places. *Knowledge and Information Systems*, 40(2):411–437, 2014.

[5] R. Prez-Torres, C. Torres-Huitzil, and H. Galeana-Zapin. Full on-device stay points detection in smartphones for location-based mobile applications. *Sensors*, 16(10):1693, 2016.

[6] F. J. M. Arboleda, V. Bogorny, and H. Patio. Smot+ncs - algorithm for detecting non-continuous stops. *Computing and Informatics*, 3(2):283–306, 2017.

[7] M. L. Damiani, H. Issa, and F. Cagnacci. Extracting stay regions with uncertain boundaries from gps trajectories - a case study in animal ecology. In *SIGSPATIAL/GIS*, pages 253–262, 2014.

[8] B. Djordjevic, J. Gudmundsson, A. Pham, and T. Wolle. Detecting regular visit patterns. *Algorithmica*, 60(4):829–852, 2011.

[9] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *Journal of the ACM*, 39(1):1–54, 1992.

When Can We Treat Trajectories as Points?

Parasara Sridhar Duggirala* & Donald R. Sheehy†

Abstract

In the formal verification of dynamical systems, one often looks at a trajectory through a state space as a sample behavior of the system. Thus, metrics on trajectories give important information about the different behavior of the system given different starting states. In the important special case of linear dynamical systems, the set of trajectories forms a finite-dimensional vector space. In this paper, we exploit this vector space structure to define (semi)norms on the trajectories, give an isometric embedding from the trajectory metric into low-dimensional Euclidean space, and bound the Lipschitz constant on the map from start states to trajectories as measured in one of several different metrics. These results show that for an interesting class of trajectories, one can treat the trajectories as points while losing little or no information.

1 Introduction

The starting point for many problems in computational geometry is a discrete set of points in a Euclidean space. Alternatively, many interesting questions arise from sets of paths or trajectories, and usually, such problems require very different ideas and methods. In this paper, we consider a class of trajectories that arise naturally in the field of formal verification of cyber-physical systems (CPSs)[1] in which one can transform a collection of trajectories into a set of points while losing little or no information.

Motivated partly by recent work on using algorithms from computational geometry to analyze trajectories through the state space of a CPS [12], we highlight an interesting class of systems studied in that field, for which natural metrics on trajectories can be nicely embedded into low-dimensional Euclidean space. Generally, it would be inefficient to treat trajectories as points. Even though a discrete trajectory in the plane broken into k pieces can be thought of as a single point in $\mathbb{R}^{2k}$, the blowup in dimension can be prohibitive for most geometric algorithms, especially those where the low-dimensional (i.e. $d = 2$ or 3) structure can be exploited.

Control software in safety critical CPSs such as autonomous vehicles and power plants should always satisfy the prescribed safety specification. One of the main challenges in verifying CPS safety properties is that they involve a mix of continuous and discrete behaviors. Even if we ignore the discrete switching, the continuous dynamics in most real world CPSs are highly nonlinear and difficult to analyze. For example, when we "simplify" the nonlinear dynamics to a linear approximation, verifying such linear systems of high dimensions is still challenging due to the curse of dimensionality. In dynamic analysis techniques, a few sample executions, also called trajectories of the systems are computed. Whether the system satisfies the desired property or not is inferred after carefully analyzing the generated executions. As these techniques purely depend on sample executions, they can be easily integrated into the testing and debugging phase of CPS design.

In many CPSs, the state space is modeled as a Euclidean space. As in other dynamical systems, the next state is a function of the current state and often some inputs or controls. The executions that form the primary data are trajectories in a Euclidean space. For this work, we consider the simplest case where the system is governed by a linear dynamical system. That is, the derivative of the state is a linear function of the current state. We will show how one can model the geometry of the space of trajectories as a set of points. Naturally, this will depend on a choice of a metric on the space of trajectories. We will consider several different metrics on trajectories including L_p-type metrics as well as the Fréchet distance and the Skorokhod distance.

The main goal is to show when and to what extent one can study the class of trajectories arising from a CPS using algorithms and data structures designed for points. The hope is that this will open the door to more applications of classical computational geometry of points in Euclidean space to problems in formal verification.

2 Metrics, Norms, and Samples

In this section, we review several notions that will be very familiar to most readers. We include the formal definitions for completeness, because our results are given in considerable generality and the fine distinctions

*Computer Science Department, University of Connecticut, psd@uconn.edu

†Computer Science Department, University of Connecticut, don.r.sheehy@uconn.edu

[1] For this paper, we can identify the buzzword "cyberphysical systems" used in the verification literature with the more general notion of a dynamical system.

in the definitions (e.g. pseudometrics and seminorms) will be important.

A *metric space* $\mathcal{X}$ is a pair (X, d) where X is a set and $\mathrm{d} : X \times X \to \mathbb{R}$ is a function satisfying the following conditions.

1. Nonnegative: $\mathrm{d}(x, y) \geq 0$
2. Symmetric: $\mathrm{d}(x, y) = \mathrm{d}(y, x)$
3. Triangle Inequality: $\mathrm{d}(x, z) \leq \mathrm{d}(x, y) + \mathrm{d}(y, z)$
4. Identity of Indiscernables: $\mathrm{d}(x, y) = 0$ if and only if $x = y$.

A function d satisfying the first three properties is called a *pseudometric*.

Let V be a vector space over some subfield of the complex numbers $\mathbb{C}$. A *norm* V is a function $\|\cdot\| : V \to \mathbb{R}$ satisfying the following conditions.

1. Nonnegative: $\|v\| \geq 0$
2. Absolutely scalable: $\|cv\| = |c|\|v\|$
3. Triangle Inequality: $\|u + v\| \leq \|u\| + \|v\|$
4. Definite: If $\|v\| = 0$, then $v = 0$.

A function satisfying the first three conditions of a norm is called a *seminorm*. Every norm on V induces a metric on V defined as $\mathrm{d}(u, v) = \|u - v\|$. Similarly, a seminorm induces a pseudometric.

A function between (pseudo)metric spaces, $f : (X, \mathrm{d}_X) \to (Y, \mathrm{d}_Y)$ is λ-Lipschitz if for all $x, x' \in X$, we have

$$\mathrm{d}_Y(f(x), f(x')) \leq \lambda \mathrm{d}_X(x, x').$$

This is a basic stability condition on mappings between metric spaces. It is most often used to describe real-valued functions using the standard metric on $\mathbb{R}$.

An *ε-sample* of a subset U of a metric space is a subset $S \subseteq U$ such that for all $u \in U$, there exists $s \in S$ such that $\mathrm{d}(u, s) \leq \varepsilon$. This notion is closely related to the *Hausdorff distance*, a (pseudo)metric on compact subsets of a (pseudo)metric space.[2] It is defined as follows.

$$\mathrm{d}_H(S, T) = \max\{\max_{s \in S} \min_{t \in T} \mathrm{d}(s, t), \max_{t \in T} \min_{s \in S} \mathrm{d}(s, t)\}$$

Using this definition, an ε-sample of U is a subset $S \subseteq U$ such that $\mathrm{d}_H(S, U) \leq \varepsilon$.

[2] Compactness here is primarily required for distance minimizers to exist.

3 States and Trajectories

Let $\mathcal{X} = (X, \mathrm{d})$ denote a metric space. For this paper, $\mathcal{X}$ will represent the space of *states* of some system. Usually, we will only consider states in $\mathbb{R}^d$, but it is useful to give the following definitions in full generality.

A trajectory in $\mathcal{X}$ is a continuous function $f : [0, 1] \to \mathcal{X}$. We are only considering maps from the unit interval, though many of the results in this paper generalize naturally to other finite length intervals. We use $\mathrm{Tr}(\mathcal{X})$ to denote the set of all trajectories in $\mathcal{X}$.

3.1 Sampling trajectories

A dynamical system may be viewed as a function from states to trajectories. If we endow both the state space and the trajectory space with metrics, we can ask when such a function (the dynamical system) is Lipschitz. Having a bound on the Lipschitz constant associated to such a system justifies sampling trajectories by sampling start states. For indeed, the Lipschitz condition implies that a good sample of the valid start states will give a correspondingly good sample of the trajectories in the following precise sense.

Proposition 1 *Let $\mathcal{X}$ be a set of states and let $\Theta \subset \mathcal{X}$ be a set of start states. Let $\mathrm{Tr}(\mathcal{X})$ be a metric space of trajectories in $\mathcal{X}$ with metric T. If $S \subseteq \Theta$ is an ε-sample of Θ and $\xi : \mathcal{X} \to \mathrm{Tr}(\mathcal{X})$ is λ-Lipschitz, then*

$$\xi(S) := \{\xi(s) \mid s \in S\} \text{ is a } \lambda\varepsilon\text{-sample of } \xi(\Theta).$$

Proof. Fix any $\gamma \in \xi(\Theta)$. Then $\gamma = \xi(x)$ for some $x \in \Theta$. So, there exists $s \in S$ such that $\mathrm{d}(s, x) \leq \varepsilon$. If $\gamma' = \xi(s)$, then,

$$\begin{aligned} T(\gamma, \gamma') &= T(\xi(x), \xi(s)) \\ &\leq \lambda \mathrm{d}(x, s) \\ &\leq \lambda\varepsilon. \end{aligned}$$

So, for all $\gamma \in \xi(\Theta)$, there exists $\gamma' \in \xi(S)$ such that $T(\gamma, \gamma') \leq \lambda\varepsilon$. We conclude that $\xi(S)$ is a $\lambda\varepsilon$-sample of $\xi(\Theta)$ as desired. □

3.2 Vector Spaces of Trajectories

The set of all trajectories mapping the interval $[0, 1]$ to the state space $\mathbb{R}^d$ naturally forms a vector space. Let $c \in \mathbb{R}$ be a scalar and let $\phi, \psi : [0, 1] \to \mathbb{R}^d$ be trajectories. Scalar multiplication and vector addition are defined as

$$\begin{aligned} (c\phi)(t) &:= c\phi(t), \text{ and} \\ (\phi + \psi)(t) &:= \phi(t) + \psi(t). \end{aligned}$$

In general, the dimension of this vector space is infinite. However, the following important case yields a finite-dimensional space of trajectories.

Consider systems that evolve in $\mathbb{R}^d$ in continuous time. At a given instant in time, the system state is denoted as a vector $x \in \mathbb{R}^d$ and its evolution is given as a linear differential equation, i.e.,

$$\dot{x} = Ax, \tag{1}$$

where $A \in \mathbb{R}^{d\times d}$. By using an extra variable and allowing A to be singular, this includes also the case of *affine* systems

$$\dot{x} = Ax + B, \tag{2}$$

where $B \in \mathbb{R}^{d\times 1}$. The system of trajectories, denoted as $\xi : \mathbb{R}^d \times \mathbb{R}_{\geq 0} \to \mathbb{R}^d$ are solutions of the initial value problem of the differential equation given in Equation (1). Given an initial state x_0, $\xi(x_0, t)$ denotes the state of the system at time instance t. The closed form expression for the trajectories is given in Equation (3) below.

$$\xi(x_0, t) = e^{At}x_0 + \int_0^t e^{A(t-\tau)}Bd\tau, \tag{3}$$

where $e^{At} = I + \frac{At}{1!} + \frac{(At)^2}{2!} + \frac{(At)^3}{3!} + \dots$ represents the matrix exponential operation. We denote the trajectory starting from initial state x_0 as ξ_{x_0}, i.e., $\xi_{x_0}(t) = \xi(x_0, t)$ so that it fits with our previous definition of trajectory.

Definition 1 *Trajectories of linear dynamical systems (such as given in Equation (1)) satisfy the* superposition principle. *Given any state $x_0 \in \mathbb{R}^d$, vectors $v_1, v_2, \dots, v_m \in \mathbb{R}^d$, and scalars $\alpha_1, \alpha_2, \dots, \alpha_m \in \mathbb{R}$, the following equality is satisfied.*

$$\xi(x_0+\sum_{i=1}^{m}\alpha_i v_i, t) = \xi(x_0, t)+\sum_{i=1}^{m}\alpha_i(\xi(x_0+v_i, t)-\xi(x_0, t))$$

The observation made in Definition 1 follows from the closed form solution given in Equation (3). Using the superposition principle, we can infer that for any pair of states $x_0, x_1 \in \mathbb{R}^n$ and for any vector $v \in \mathbb{R}^d$,

$$\xi(x_0 + v, t) - \xi(x_0, t) = \xi(x_1 + v, t) - \xi(x_1, t)$$

These equations reveal the low dimensional vector space structure of the space of trajectories. Indeed, given any basis $(b_1, \dots, b_n)$ for the state space, a trajectory starting from $x = \sum_{i=1}^n x_i b_i$ is the corresponding linear combination of *basis trajectories*:

$$\xi_x = \sum_{i=1}^{n} x_i \xi_{b_i}.$$

When the matrix A determining the system is non-singular, then the closed form for the trajectory (Equation (3)) simplifies to

$$\xi_x(t) = e^{At}x.$$

For the rest of the paper, we will assume this case for simplicity.

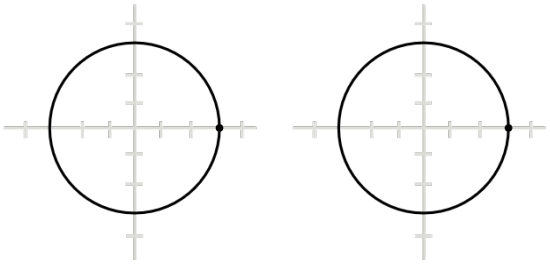


Figure 1: Two different trajectories with the same image. Left: $3e^{10\pi it}$. Right: $3e^{2\pi it}$.

4 Metrics Spaces of Trajectories

In this section, we will define several classes of metrics on trajectories. If one is content to view a trajectory $f : [0, 1] \to \mathbb{R}^d$ as merely a set of points in the state space, i.e.

$$\text{im } f := \{f(t) \mid t \in [0, 1]\},$$

then the Hausdorff distance provides an easy to define metric on trajectories. That is, one can define

$$T_H(f, g) := \mathrm{d}_H(\text{im } f, \text{im } g).$$

The Hausdorff distance has a natural geometric interpretation as the minimum radius r such that expanding im f by r would cover im g and vice versa. Unfortunately, the Hausdorff distance ignores the continuous structure of the input trajectories. For example, it sees no difference between the two trajectories of Figure 1, the first of which makes five revolutions and the second makes only one.

Indeed, the Hausdorff distance only give a pseudometric on trajectories as the distance between a trajectory f and its reverse trajectory $g(t) = f(1 - t)$ is precisely 0.

The L_p-type metrics on trajectories are defined for an integer $p \geq 1$ as follows:

$$L_p(f, g) := \left(\int_0^1 \|f(t) - g(t)\|_p^p dt\right)^{\frac{1}{p}}.$$

This includes, in particular, the L_∞ distance:

$$L_{\mathrm{d},\infty}(f, g) := \max_{t\in[0,1]} \mathrm{d}(f(t), g(t)),$$

where $\mathrm{d}(\cdot, \cdot)$ can be any metric on the state space. We abuse notation and write L_∞ to denote $L_{\mathrm{d},\infty}$ with $\mathrm{d}(x, y) := \|x - y\|_2$. We do this primarily because it is such a popular metric and can also be used to define other metrics as we will see.

A drawback of L_p-type metrics (usually L_∞) is their inability to recognize similarity of trajectories that differ only by some continuous reparameterization of time.

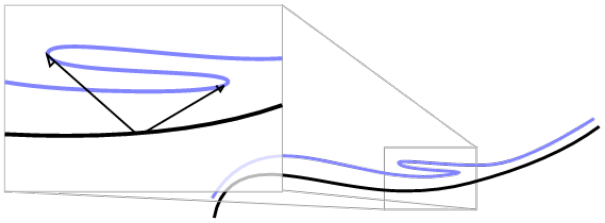


Figure 2: In this example the Fréchet distance between the curves is less than the $\mathcal{L}_\infty$ distance because reparameterization allows the lower trajectory to slow down in the neighborhood of the upper trajectory's zigzag.

For this reason, the Fréchet distance is often considered. It is defined as

$$\mathrm{d}_F(f,g) := \min_{h\in H} L_\infty(f, g\circ h),$$

where H is the set of orientation-preserving homeomorphisms $h : [0,1] \to [0,1]$, i.e. the continuous reparameterizations of time.

The Fréchet distance is also called the "dog walking distance" using the metaphor that a person walks along one trajectory and the dog walks along another [1, 13]. The person may adjust their speed (reparameterize time) so as to minimize the length of the leash (the L_∞ distance). For example in Figure 2, the lower trajectory can slow down to lessen the impact of the zigzag in the upper trajectory. It is an interesting exercise to show that using a different $\mathcal{L}_p$-type metric instead of L_∞ in the definition, does not result in a metric.

Technically, the Fréchet distance gives a pseudometric on trajectories. The trajectories f and $f \circ h$ have Fréchet distance zero despite being different functions. The triangle inequality follows by composing homeomorphisms.

The minimization in the definition of the Fréchet distance allows a substantial amount of freedom to align trajectories, sometimes too much freedom to be realistic. The *Skorokhod* distance addressed this issue by penalizing excessive time reparameterization [11]. This distance may be viewed as treating time as another spatial parameter. One starts with a metric on the space×time product $\mathbb{R}^n \times [0,1]$. For example, one could use the ℓ_p-product metric

$$\ell_p((x,s),(y,t)) := (\mathrm{d}(x,y)^p + |s-t|^p)^{1/p}.$$

This includes the ℓ_∞ metric

$$\ell_\infty((x,s),(y,t)) := \max\{\mathrm{d}(x,y), |s-t|\}.$$

Note that the definition does not specify a particular metric d on the state space. The graph of a trajectory ξ is the trajectory

$$\mathrm{Gr}(\xi)(t) = (\xi(t), t).$$

One defines a Skorokhod distance (assuming a product metric is fixed) as

$$\mathrm{d}_S(f,g) := \mathrm{d}_F(\mathrm{Gr}(f), \mathrm{Gr}(g)).$$

5 Hilbert Spaces of Trajectories

Just as with finite-dimensional vector spaces, the L_2-norm on trajectories results in a Hilbert space. The inner product is simply the integral of the standard Euclidean inner product, i.e.

$$\langle \xi_x, \xi_y \rangle := \int_0^1 \xi_x(t)^\top \xi_y(t) dt.$$

If we only consider the trajectories coming from a linear, dynamical system $\dot{x} = Ax$, the resulting Hilbert space only has the dimension of the state space as previously observed, but moreover, the induced metric is Euclidean and can be computed explicitly.

Theorem 2 *Given a dynamical system in $\mathbb{R}^d$ governed by $\dot{x} = Ax$, there exists a matrix $L \in \mathbb{R}^{d\times d}$ such that for any $x, y \in \mathbb{R}^d$,*

$$L_2(\xi_x, \xi_y) = \|Lx - Ly\|_2.$$

Proof.

$$\begin{aligned}\|\xi_x\|_2 &= \int_0^1 \xi_x(t)^\top \xi_x(t) dt \\ &= \int_0^1 (e^{At}x)^\top (e^{At}x) dt \\ &= x^T \left(\int_0^1 (e^{At})^\top (e^{At}) dt \right) x\end{aligned}$$

Let M be the matrix $\int_0^1 (e^{At})^\top (e^{At}) dt$ so that $\|\xi_x\|_2 = x^\top M x$. As the matrices are positive definite for all t, it follows that M is also positive definite. Thus, the Cholesky decomposition $M = LL^\top$ exists and the matrix L has the property that

$$\|\xi_x\|_2 = (Lx)^\top (Lx).$$

This fact about the L_2 norm immediately implies the corresponding claim about the L_2 metric. □

Computational Issues and Implications Following the proof of Theorem 2, a natural approach to computing L, at least approximately, is to discretize the integral $\int_0^1 (e^{At})^\top (e^{At}) dt$ and compute the pieces using the leading terms of the expansion of the matrix exponential, $e^{At} = I + \frac{At}{1!} + \frac{(At)^2}{2!} + \cdots$. The result is a positive definite matrix M whose Cholesky decomposition gives the desired linear operator L.

One immediate use for Theorem 2 is in the analysis of collections of trajectories. Any data analysis depending on the distances between trajectories, such as clustering or subsampling, would naturally require computing many pairwise distances. If the trajectories are discretized into k pieces, the straightforward computation of the L_2 distance would require $O(kd)$ time. If instead, one first computes L, then the time to compute these distances is reduced to $O(d^2)$. For high-fidelity measurements with large k, this can be a substantial speedup. This idea, though technically simple, has not been exploited in the literature on formal verification of these systems.

6 The Lipschitz Bound

It is only a small consolation if a car that always crashes, spends most of its time in an "uncrashed" state. Unfortunately, it is the nature of L_2 metrics to average distances over time, so such bad behaviors of a system could be missed. For this reason, a max-norm like L_∞ is often preferable. However, an isometric embedding of the trajectories into Euclidean space as in Theorem 2 is not possible (consider $A = 0$ for example). The alternative we propose in this section is to at least bound the Lipschitz constant of the system when viewed as a mapping from a metric on states to a metric on trajectories. We give such a bound in considerable generality; it applies to any seminorm on trajectories and we show its implications for other pseudometrics including the Fréchet and Skorokhod distance.

Theorem 3 *Let $\mathbb{R}^d$ be the states equipped with a norm $\|\cdot\|$. Let $\dot{x} = Ax$ be a linear dynamical system and let $\mathrm{Tr}_A(\mathbb{R}^d)$ be the trajectories in $\mathbb{R}^d$ arising from A endowed with some seminorm, $\|\cdot\|_{\mathrm{Tr}}$. Let $(b_1, \ldots, b_d)$ be a basis for $\mathbb{R}^d$ and let $(\xi_1, \ldots, \xi_d)$ be the corresponding basis for $\mathrm{Tr}_A(\mathbb{R}^d)$. Let $t = \|(\|\xi_1\|_{\mathrm{Tr}}, \ldots, \|\xi_d\|_{\mathrm{Tr}})\|$ Then, the mapping from $\mathbb{R}^d$ to $\mathrm{Tr}_A(\mathbb{R}^d)$ is t-Lipschitz. That is, for any pair of states $x, y \in \mathbb{R}^d$,*

$$\|\xi_x - \xi_y\|_{\mathrm{Tr}} \leq t\|x - y\|.$$

Proof. Write the states x and y in terms of the basis $(b_1, \ldots, b_d)$ as follows.

$$x = \sum_{i=1}^{d} x_i b_i \text{ and } y = \sum_{i=1}^{d} y_i b_i.$$

Then, the trajectories ξ_x and ξ_y can be written in the corresponding basis of trajectories as follows.

$$\xi_x = \sum_{i=1}^{d} x_i \xi_i \text{ and } \xi_y = \sum_{i=1}^{d} y_i \xi_i.$$

We can now bound $\|\xi_x - \xi_y\|_{\mathrm{Tr}}$ as follows.

$$\begin{aligned}
\|\xi_x - \xi_y\|_{\mathrm{Tr}} &= \|\sum_{i=1}^{d}(x_i - y_i)\xi_i\|_{\mathrm{Tr}} && \text{[by definition]} \\
&\leq \sum_{i=1}^{d}\|(x_i - y_i)\xi_i\|_{\mathrm{Tr}} && \text{[triangle ineq.]} \\
&= \sum_{i=1}^{d}(x_i - y_i)\|\xi_i\|_{\mathrm{Tr}} && \text{[norms are linear]} \\
&\leq t\|x - y\| && \text{[Cauchy-Schwarz]}
\end{aligned}$$

□

The hypothesis about working with (semi)norms in the theorem above are necessary to apply the Cauchy-Schwarz inequality. Thus, it's not clear how to duplicate this proof while replacing the norms on trajectories with a more elaborate distance on trajectories. However, the theorem naturally extends to give a Lipschitz bound when the distances on trajectories are measured using either the Fréchet distance or the Skorokod distance by using the relationship between these metrics and the L_∞ norm.

Theorem 4 *Let $\mathbb{R}^d$ be the states equipped with a norm $\|\cdot\|$. Let A be a linear dynamical system and let $\mathrm{Tr}_A(\mathbb{R}^d)$ be the trajectories in $\mathbb{R}^d$ arising from A endowed with either the Fréchet distance or the Skorokhod distance. Let $(b_1, \ldots, b_d)$ be a basis for $\mathbb{R}^d$ and let $(\xi_1, \ldots, \xi_d)$ be the corresponding basis for $\mathrm{Tr}_A(\mathbb{R}^d)$. Let $t = \|(\|\xi_1\|_\infty, \ldots, \|\xi_d\|_\infty)\|$ Then, the mapping from $\mathbb{R}^d$ to $\mathrm{Tr}_A(\mathbb{R}^d)$ is t-Lipschitz.*

Proof. It suffices to observe that for any pair of trajectories $f, g : [0, 1] \to \mathbb{R}^d$ and any ℓ_p-product metric on $\mathbb{R}^d \times [0, 1]$, the following inequalities hold.

$$\mathrm{d}_S(f, g) \leq \mathrm{d}_F(f, g) \leq L_\infty(f, g)$$

These inequalities hold by replacing the minimization over homeomorphisms with the specific choice of the identity homeomorphism. Thus, the theorem follows from Theorem 3 using the L_∞ metric on trajectories. □

7 Related Work, Conclusion, and Future Work

Proving properties of software systems while leveraging the data generated from the sample executions has been a well studied topic in the domain of formal verification [10, 14, 7, 15, 9]. However, these techniques do not deal with the CPS where the dynamics of the physical environment is of utmost importance. Recent techniques to integrate the information generated from sample trajectories for proving properties of CPS have

been investigated [4, 5]. In a recent work [6], the fact that the trajectories form a vector space has been leveraged to improve the scalability of verification by two orders of magnitude [2]. Techniques similar to [5] to provide probabilistic guarantees about trajectories of CPS have been investigated in [8].

This work attempts to address the gap between the data-driven verification technique and computational geometry. The focus on linear dynamical systems is because of two reasons. First, linear dynamical systems describe a large set of control systems that are in deployment. Second, these systems enjoy a rich set of properties (such as the superposition principle) that can be readily exploited to represent trajectories as points. Mapping from trajectories to points would also help us in performing topological data analysis [3] over trajectories.

While in this paper we considered a specific sub-class of dynamical systems, in our future work, we intend to apply similar techniques to nonlinear dynamics. Our goal is to eventually perform data-driven analysis of trajectories where partial or no model information is available.

References

[1] Helmut Alt and Michael Godau. Computing the fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl*, 5(1 & 2):75–91, 1995.

[2] Stanley Bak and Parasara Sridhar Duggirala. Simulation-equivalent reachability of large linear systems with inputs. In *International Conference on Computer Aided Verification*, pages 401–420. Springer, 2017.

[3] Gunnar Carlsson. Topology and data. *Bulletin of the American Mathematical Society*, 46(2):255–308, 2009.

[4] Alexandre Donzé and Oded Maler. Systematic simulation using sensitivity analysis. In *International Workshop on Hybrid Systems: Computation and Control*, pages 174–189. Springer, 2007.

[5] Parasara Sridhar Duggirala, Sayan Mitra, and Mahesh Viswanathan. Verification of annotated models from executions. In *Proceedings of the Eleventh ACM International Conference on Embedded Software*, page 26. IEEE Press, 2013.

[6] Parasara Sridhar Duggirala and Mahesh Viswanathan. Parsimonious, simulation based verification of linear systems. In *International Conference on Computer Aided Verification*, pages 477–494. Springer, 2016.

[7] Michael D Ernst, Jeff H Perkins, Philip J Guo, Stephen McCamant, Carlos Pacheco, Matthew S Tschantz, and Chen Xiao. The daikon system for dynamic detection of likely invariants. *Science of Computer Programming*, 69(1-3):35–45, 2007.

[8] Chuchu Fan, Bolun Qi, Sayan Mitra, and Mahesh Viswanathan. Dryvr: Data-driven verification and compositional reasoning for automotive systems. In *International Conference on Computer Aided Verification*, pages 441–461. Springer, 2017.

[9] Pranav Garg, Christof Löding, P Madhusudan, and Daniel Neider. Ice: A robust framework for learning invariants. In *International Conference on Computer Aided Verification*, pages 69–87. Springer, 2014.

[10] Patrice Godefroid, Nils Klarlund, and Koushik Sen. Dart: directed automated random testing. In *ACM Sigplan Notices*, volume 40, pages 213–223. ACM, 2005.

[11] Rupak Majumdar and Vinayak S. Prabhu. Computing the skorokhod distance between polygonal traces. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 199–208, 2015.

[12] Rupak Majumdar and Vinayak S Prabhu. Computing distances between reach flowpipes. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pages 267–276. ACM, 2016.

[13] Günter Rote. Computing the fréchet distance between piecewise smooth curves. *Computational Geometry: Theory and Applications*, 37(3):162–174, 2007.

[14] Koushik Sen, Darko Marinov, and Gul Agha. Cute: a concolic unit testing engine for c. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 263–272. ACM, 2005.

[15] Rahul Sharma. *Data-driven Verification*. PhD thesis, Stanford University, 2016.

Compatible 4-Holes in Point Sets

Ahmad Biniaz* Anil Maheshwari† Michiel Smid†

Abstract

Counting interior-disjoint empty convex polygons in a point set is a typical Erdős-Szekeres-type problem. We study this problem for convex 4-gons. Let P be a set of n points in the plane and in general position. A subset Q of P, with four points, is called a 4-*hole* in P if Q is in convex position and its convex hull does not contain any point of P in its interior. Two 4-holes in P are *compatible* if their interiors are disjoint. We show that P contains at least $\lfloor 5n/11 \rfloor - 1$ pairwise compatible 4-holes. This improves the lower bound of $2\lfloor (n-2)/5 \rfloor$ which is implied by a result of Sakai and Urrutia (2007).

1 Introduction

Throughout this paper, an *n-set* is a set of n points in the plane and in general position, i.e., no three points are collinear. Let P be an n-set. A *hole* in P is a subset Q of P, with at least three elements, such that Q is in convex position and no element of P lies in the interior of the convex hull of Q. A *k-hole* in P is a hole with k elements. By this definition, a 3-hole in P is an empty triangle with vertices in P, and a 4-hole in P is an empty convex quadrilateral with vertices in P.

The problem of finding and counting holes in point sets has a long history in discrete combinatorial geometry, and has been an active research area since Erdős and Szekeres [14, 15] asked about the existence of k-holes in a point set. In 1931, Esther Klein showed that any 5-set contains a convex quadrilateral [15]; it is easy to see that it also contains a 4-hole. In 1978, Harborth [17] proved that any 10-set contains a 5-hole. In 1983, Horton [18] exhibited arbitrarily large point sets with no 7-hole. The existence of a 6-hole in sufficiently large point sets has been proved by Nicolás [22] and Gerken [16]; a shorter proof of this result is given by Valtr [26].

Two holes Q_1 and Q_2 are *disjoint* if their convex hulls are disjoint, i.e., they do not share any vertex and do not overlap. We say sat Q_1 and Q_2 are *compatible* if the interiors of their convex hulls are disjoint, that is, they can share vertices but do not overlap. A set of holes is called disjoint (resp. compatible) if its elements are pairwise disjoint (resp. compatible). See Figure 1.

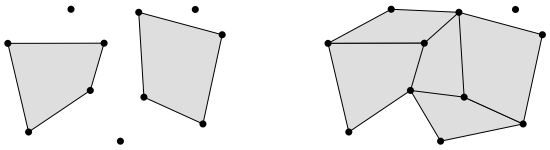


Figure 1: Two disjoint 4-holes (left), and five compatible 4-holes (right).

Since every three points form the vertices of a triangle, by repeatedly creating a triangle with the three leftmost points of an n-set we obtain exactly $\lfloor n/3 \rfloor$ disjoint 3-holes. However, this does not generalize to 4-holes, because the four leftmost points may not be in convex position. Obviously, the number of disjoint 4-holes in an n-set is at most $\lfloor n/4 \rfloor$. Hosono and Urabe [19] proved that the number of disjoint 4-holes is at least $\lfloor 5n/22 \rfloor$; they improved this bound to $(3n-1)/13$ when $n = 13 \cdot 2^k - 4$ for some $k \geqslant 0$. A variant of this problem where the 4-holes are vertex-disjoint, but can overlap, is considered in [29]. As for compatible holes, it is easy to verify that the number of compatible 3-holes in any n-set is at least $n-2$ and at most $2n-5$; these bounds are obtained by triangulating the point set: we get $n-2$ triangles, when the point set is in convex position, and $2n-5$ triangles, when the convex hull of the point set is a triangle. Sakai and Urrutia [24] proved among other results that any 7-set contains at least two compatible 4-holes. In this paper we study the problem of finding the maximum number of compatible 4-holes in an n-set.

Devillers *et al.* [13] considered some colored variants of this problem. They proved among other results that any bichromatic n-set has at least $\lceil n/4 \rceil - 2$ compatible monochromatic 3-holes; they also provided a matching upper bound. As for 4-holes, they conjectured that a sufficiently large bichromatic point set has a monochromatic 4-hole. Observe that any point set that disproves this conjecture does not have a 7-hole (regardless of colors). For a bichromatic point set $R \cup B$ in the plane, Sakai and Urrutia [24] proved that if $|R| \geqslant 2|B|+5$, then there exists a monochromatic 4-hole. They also studied the problem of blocking 4-holes in a given point set R; the goal in this problem is to find a smallest point set B such that any 4-hole in R has a point of B in its interior. The problem of blocking 5-holes has been studied by Cano *et al.* [12].

*University of Waterloo, Canada. Supported by NSERC Postdoctoral Fellowship. ahmad.biniaz@gmail.com

†Carleton University, Canada. Supported by NSERC. {anil, michiel}@scs.carleton.ca

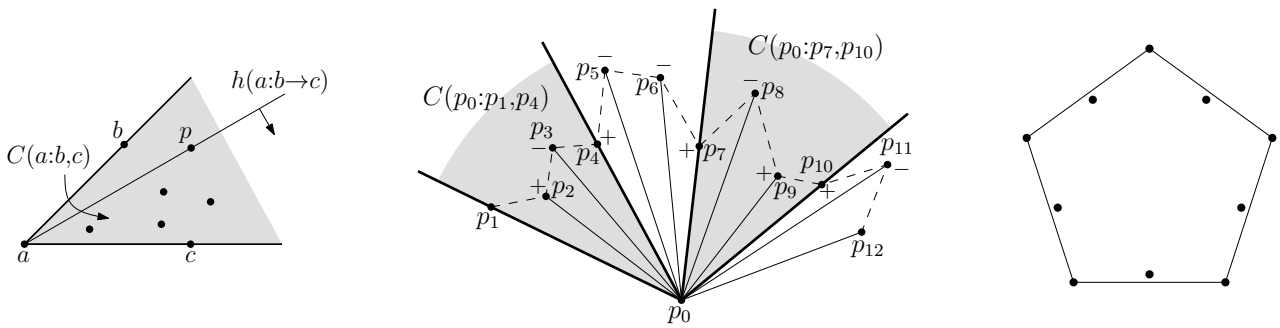


Figure 2: The point p is the attack point of $h(a{:}b{\to}c)$ (left). The radial ordering of points around p_0 (middle). A 10-set with at most three compatible 4-holes (right).

Aichholzer *et al.* [3] proved that every 11-set contains either a 6-hole, or a 5-hole and a disjoint 4-hole. Bhattacharya and Das [6] proved that every 12-set contains a 5-hole and a disjoint 4-hole. They also proved the existence of two disjoint 5-holes in every 19-set [7]. For more results on the number of k-holes in small point sets and other variations, see the paper by Aichholzer and Krasser [4], a summary of recent results by Aichholzer *et al.* [5], and B. Vogtenhuber's doctoral thesis [27]. Researchers also have studied the problem of counting the number of (not necessarily empty nor compatible) convex quadrilaterals in a point set; see, e.g., [2, 11, 21, 28].

A quadrangulation of a point set P in the plane is a planar subdivision whose vertices are the points of P, whose outer face is the convex hull of P, and every internal face is a quadrilateral; in fact the quadrilaterals are empty and pairwise compatible. Similar to triangulations, quadrangulations have applications in finite element mesh generation, Geographic Information Systems (GIS), scattered data interpolation, etc.; see [9, 10, 23, 25]. Most of these applications look for a quadrangulation that has the maximum number of convex quadrilaterals. To maximize the number of convex quadrilaterals, various heuristics and experimental results are presented in [9, 10]. This raises another motivation to study theoretical aspects of compatible empty convex quadrilaterals in a planar point set.

In this paper we study lower and upper bounds for the number of compatible 4-holes in point sets in the plane. A trivial upper bound is $\lfloor n/2 \rfloor - 1$ which comes from n points in convex position. The $\lfloor 5n/22 \rfloor$ lower bound on the number of disjoint 4-holes that is proved by Hosono and Urabe [19], simply carries over to the number of compatible 4-holes. Also, as we will see in Section 2, the lower bound of $2\lfloor (n-2)/5 \rfloor$ on the number of compatible 4-holes is implied by a result of Sakai and Urrutia [24]. After some new results for small point sets, we prove non-trivial lower bounds on the number of compatible 4-holes in an n-set. We prove that every 9-set (resp. 11-set) contains three (resp. four) compatible 4-holes. Using these results, we prove that every n-set contains at least $\lfloor 5n/11 \rfloor - 1$ compatible 4-holes. Our proof of this lower bound is constructive, and immediately yields an $O(n \log^2 n)$-time algorithm for finding this many compatible 4-holes.

Since the initial presentation of this work [8], the problem has attracted further attention. Most prominently, the lower bound on the number of compatible 4-holes has been improved to $\lceil \frac{n-3}{2} \rceil$ by Cravioto-Lagos, González-Martínez, Sakai, and Urrutia [1]. The same bound is claimed in an abstract by Lomeli-Haro, Sakai, and Urrutia in Kyoto International Conference on Computational Geometry and Graph Theory (CGGT2007) [20]. However, this result has not been published yet.

2 Preliminaries

First we introduce some notation from [19]. We define the *convex cone* $C(a{:}b, c)$ to be the region of the angular domain in the plane that is determined by three non-collinear points a, b, and c, where a is the apex, b and c are on the boundary of the domain, and $\angle bac$ is acute (less than $\pi/2$). We denote by $h(a{:}b{\to}c)$ the rotated half-line that is anchored at a and rotates, in $C(a{:}b, c)$, from the half-line ab to the half-line ac. If the interior of $C(a{:}b, c)$ contains some points of a given point set, then we call the first point that $h(a{:}b{\to}c)$ meets the *attack point* of $h(a{:}b{\to}c)$; see Figure 2-left.

Let P be an n-set. We denote by $CH(P)$ the convex hull of P. Let p_0 be the bottommost vertex on $CH(P)$. Without loss of generality assume that p_0 is the origin. Label the other points of P by $p_1, \ldots, p_{n-1}$ in clockwise order around p_0, starting from the negative x-axis; see Figure 2-middle. We refer to the sequence $p_1, \ldots, p_{n-1}$ as the *radial ordering* of the points of $P \setminus \{p_0\}$ around p_0. We denote by $l_{i,j}$ the straight line through two points with indexed labels p_i and p_j.

It is easy to verify that the number of 4-holes in an n-set in convex position is exactly $\lfloor n/2 \rfloor - 1$. Figure 2-right, that is borrowed from [19], shows an example of a 10-set that contains at most three compatible 4-holes; by removing a vertex from the convex hull, we obtain a 9-set with the same number of 4-holes. This example can be extended to larger point sets, and thus, to the following proposition.

Proposition 1 *For every $n \geqslant 3$, there exists an n-set that has at most $\lceil n/2 \rceil - 2$ compatible 4-holes.*

Proposition 2 *The number of compatible 4-holes in an n-set is at most $n-3$.*

Proof. Let P be an n-set. Consider the maximum number of compatible 4-holes in P. The point set P together with an edge set, that is the union of the boundary edges of these 4-holes, introduces a planar graph G. Every 4-hole in P corresponds to a 4-face (a face with four edges) in G, and vice versa. Using Euler formula for planar graphs one can verify that the number of internal 4-faces of G is at most $n-3$. This implies that the number of 4-holes in P is also at most $n-3$. □

Theorem 1 (Klein [15]) *Every 5-set has a 4-hole.*

Theorem 2 (Sakai and Urrutia [24]) *Every 7-set has at least two compatible 4-holes.*

As a warm-up, we show that the number of 4-holes in an n-set P is at least $\lfloor (n-2)/3 \rfloor$. Let p_0 be the bottommost point of P and let $p_1, \ldots, p_{n-1}$ be the radial ordering of the other points of P around p_0. Consider $\lfloor (n-2)/3 \rfloor$ cones $C(p_0{:}p_1, p_4), C(p_0{:}p_4, p_7), C(p_0{:}p_7, p_{10}), \ldots$ where each cone has three points of P (including p_0) on its boundary and two other points in its interior. See Figure 2-middle. Each cone contains five points (including the three points on its boundary), and by Theorem 1 these five points introduce a 4-hole. Since the interiors of these cones are pairwise disjoint, we get $\lfloor (n-2)/3 \rfloor$ compatible 4-holes in P. We can improve this bound as follows. By defining the cones as $C(p_0{:}p_1, p_6), C(p_0{:}p_6, p_{11}), C(p_0{:}p_{11}, p_{16}), \ldots$, we get $\lfloor (n-2)/5 \rfloor$ cones, each of which contains seven points. By Theorem 2, the seven points in each cone introduce two compatible 4-holes, and thus, we get $2 \cdot \lfloor (n-2)/5 \rfloor$ compatible 4-holes in total. Intuitively, any improvement on the lower bound for small point sets carries over to large point sets.

3 Compatible 4-holes in small point sets

In this section we provide lower bounds on the number of compatible 4-holes in 9-sets and 11-sets. In Subsection 3.2 we prove that every 9-set contains at least three compatible 4-holes and every 11-set contains at least four compatible 4-holes. Both of these lower bounds match the upper bounds given in Proposition 1. Due to the nature of this type of problems, our proofs involve case analysis. The case analysis gets more complicated as the number of points increases. To simplify the case analysis, we use two observations and a lemma that are given in Subsection 3.1. To simplify the case analysis further, we prove our claim for 9-sets first, then we use this result to obtain the proof for 11-sets. In this section we may use the term "quadrilateral" instead of 4-hole.

Let P be an n-set. Let p_0 be the bottommost point of P and let $p_1, \ldots, p_{n-1}$ be the radial ordering of the other points of P around p_0. For each point p_i, with $i \in \{2, \ldots, n-2\}$, we define the *signature* $s(p_i)$ of p_i to be "+" if, in the quadrilateral $p_0 p_{i-1} p_i p_{i+1}$, the inner angle at p_i is greater than π, and "−" otherwise; see Figure 2-middle. We refer to $s(p_2)s(p_3)\ldots s(p_{n-2})$ as the *signature sequence* of P with respect to p_0. We refer to $s(p_{n-2})\ldots s(p_3)s(p_2)$ as the *reverse* of $s(p_2)s(p_3)\ldots s(p_{n-2})$. A *minus subsequence* is a subsequence of − signs in a signature sequence. A *plus subsequence* is defined analogously. For a given signature sequence δ, we denote by $m(\delta)$, the number of minus signs in δ.

3.1 Two observations and a lemma

In this section we introduce two observations and a lemma to simplify some case analysis in our proofs, which come later. Notice that if $s(p_i)\ldots s(p_j)$ is a plus subsequence, then the points $p_{i-1}, p_i, \ldots, p_j, p_{j+1}$ are in convex position and the interior of their convex hull does not contain any point of P. Also, if $s(p_i)\ldots s(p_j)$ is a minus subsequence, then the points $p_0, p_{i-1}, p_i, \ldots, p_j, p_{j+1}$ are in convex position and the interior of their convex hull does not contain any point of P. Therefore, the following two observations are valid.

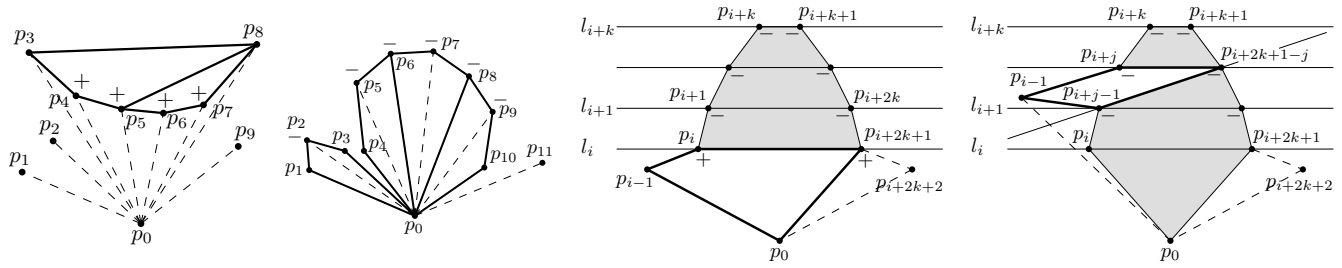
Observation 1 *Let $s(p_i)\ldots s(p_j)$ be a plus subsequence of length $2k$, with $k \geqslant 1$. Then, the convex hull of $p_{i-1}, \ldots, p_{j+1}$ can be partitioned into k compatible 4-holes. See Figure 3(a).*

Observation 2 *Let $s(p_i)\ldots s(p_j)$ be a minus subsequence of length $2k+1$, with $k \geqslant 0$. Then, the convex hull of $p_0, p_{i-1}, \ldots, p_{j+1}$ can be partitioned into $k+1$ compatible 4-holes. See Figure 3(b).*

Lemma 3 *Let $s(p_{i+1})s(p_{i+2})\ldots s(p_{i+2k})$ be a minus subsequence of length $2k$, with $k \geqslant 1$, and let p_i and p_{i+2k+1} have + signatures. Then, one can find $k+1$ compatible 4-holes in the convex hull of $p_0, p_{i-1}, \ldots, p_{i+2k+2}$.*

Proof. Refer to Figures 3(c) and 3(d). For every $j \in \{0, \ldots, k\}$ let l_{i+j} be the line through p_{i+j} and $p_{i+2k+1-j}$. These lines might intersect each other, but, for a better understanding of this proof, we visualized them as parallel lines in Figures 3(c) and 3(d).

Notice that the points $p_0, p_i, \ldots, p_{i+2k+1}$ are in convex position. If p_{i-1} is below l_i, then we get a 4-hole $p_0 p_{i-1} p_i p_{i+2k+1}$ and k other compatible 4-holes in the convex hull of the points $p_i, \ldots, p_{i+2k+1}$; see Figure 3(c). Assume p_{i-1} is above l_i. If p_{i-1} is below some lines in the sequence $l_{i+1}, \ldots, l_{i+k}$, then let l_{i+j} be the first one



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

Figure 3: (a) Plus subsequence $s(p_4)s(p_5)s(p_6)s(p_7)$ of length four. (b) Minus subsequences $s(p_2)$ and $s(p_5)\dots s(p_9)$ of lengths one and five. The point p_{i-1} is (c) below l_i, and (d) below l_{i+j} and above all lines $l_i, \dots, l_{i+j-1}$.

in this sequence, that is, p_{i-1} is below l_{i+j} but above all lines $l_i, \dots, l_{i+j-1}$. Notice that in this case p_{i-1} is also above the line through p_{i+j-1} and $p_{i+2k+1-j}$. In this case we get a 4-hole $p_{i-1}p_{i+j}p_{i+2k+1-j}p_{i+j-1}$, and $k-j$ compatible 4-holes in the convex hull of $p_{i+j} \dots, p_{i+2k+1-j}$, and j compatible 4-holes in the convex hull of $p_0, p_i, \dots, p_{i+j-1}, p_{i+2k+1-j}, \dots, p_{i+2k+1}$; see Figure 3(d). Thus, we get $k+1$ compatible 4-holes in total. Similarly, if p_{i+2k+2} is below one of the lines l_{i+j} for $j \in \{0, \dots, k\}$ we get $k+1$ compatible 4-holes. Thus, assume that both p_{i-1} and p_{i+2k+2} are above all lines $l_i, \dots, l_{i+k}$. In this case we get a 4-hole $p_{i-1}p_{i+2k+2}p_{i+k+1}p_{i+k}$ and k other compatible 4-holes in the convex hull of $p_i, \dots, p_{i+2k+1}$. Thus, we get $k+1$ compatible 4-holes in total. □

Quadrilaterals obtained by Observations 1 and 2 do not overlap because quadrilaterals obtained by Observation 1 lie above the chain $p_1, \dots, p_{n-1}$ while quadrilaterals obtained by Observation 2 lie below this chain. However, the quadrilaterals obtained in the proof of Lemma 3 might lie above and/or below this chain. The quadrilaterals obtained by this lemma can overlap the quadrilaterals obtained by Observations 1 or 2 in the following two cases:

- Consider the first case in the proof of Lemma 3 when p_{i-1} lies below l_i and we create the quadrilateral $p_0p_{i-1}p_ip_{i+2k+1}$. If $s(p_{i-1})$ belongs to a minus subsequence, and we apply Observation 2 on it, then the quadrilateral $p_0p_{i-2}p_{i-1}p_i$ obtained by this observation overlaps the quadrilateral $p_0p_{i-1}p_ip_{i+2k+1}$. Similar issue may arise when $s(p_{i+2k+2})$ belongs to a minus subsequence.

- Consider the last two cases in the proof of Lemma 3 when p_{i-1} lies above l_i. If $s(p_{i-1})$ belongs to a plus subsequence, and we apply Observation 1 on it, then the quadrilaterals obtained by this observation might overlap either the quadrilateral $p_{i-1}p_{i+j}p_{i+2k+1-j}p_{i+j-1}$ or the quadrilateral $p_{i-1}p_{i+2k+2}p_{i+k+1}p_{i+k}$ that is obtained by Lemma 3. Similar issue may arise when $s(p_{i+2k+2})$ belongs to a plus subsequence.

As such, in our proofs, we keep track of the following two assertions when applying Lemma 3 on a subsequence $s(p_{i+1})s(p_{i+2})\dots s(p_{i+2k})$:

Assertion 1. Do not apply Observation 1 on a plus subsequence that contains $s(p_{i-1})$ or $s(p_{i+2k+2})$.

Assertion 2. Do not apply Observation 2 on a minus subsequence that contains $s(p_{i-1})$ or $s(p_{i+2k+2})$.

3.2 Compatible 4-holes in 9-sets and 11-sets

Here we count compatible 4-holes in 9-sets and 11-sets.

Theorem 4 *Every 9-set contains at least three compatible 4-holes.*

Theorem 5 *Every 11-set contains at least four compatible 4-holes.*

In the rest of this section we prove Theorem 4. The proof of Theorem 5, which is given in the full version of our paper [8], has the same structure as of Theorem 4, and make more use of Observations 1-2 and Lemma 3.

Let P be a 9-set. Let p_0 be the bottommost point of P and let $p_1, \dots, p_8$ be the radial ordering of the other points of P around p_0. Let δ be the signature sequence of P with respect to p_0, i.e., $\delta = s(p_2)\dots s(p_6)s(p_7)$. Depending on the value of $m(\delta)$, i.e., the number of minus signs in δ, we consider the following seven cases. Notice that any proof of this theorem for δ carries over to the reverse of δ as well. So, in the proof of this theorem, if we describe a solution for a signature sequence, we skip the description for its reverse.

- $m(\delta) = 0$: In this case δ is a plus subsequence of length six. Our result follows by Observation 1.

- $m(\delta) = 1$: In this case δ has five plus signs. By Observation 2, we get a quadrilateral by the point with $-$ signature. If four of the plus signs are consecutive, then by Observation 1 we get two more quadrilaterals. Otherwise, δ has two disjoint subsequences of plus signs, each of length at least two. Again, by Observation 1 we get a quadrilateral for each of these subsequences. Therefore, in total we get three 4-holes; these 4-holes are pairwise non-overlapping.

- $m(\delta) = 2$: Notice that δ has a plus subsequence of length at least two. If the two minus signs are non-consecutive, then we get two quadrilaterals by Observation 2 and one by Observation 1. Assume the two minus signs are consecutive. If the four plus signs are consecutive or partitioned into two subsequences of lengths two, then we get two quadrilaterals by Observation 1 and one by Observation 2. The remaining sequences are $+--+++$ and $+++--+$, where the second one is the reverse of the first one. By splitting the first sequence as $+--+|++$ we get two quadrilaterals for the subsequence $+--+$, by Lemma 3. If in this lemma we land up in the last case where both p_{i-1} and p_{i+2k+2} are above l_{i+k}, then we get a third compatible quadrilateral $p_1p_6p_7p_8$, otherwise we get $p_4p_6p_7p_8$. Notice that Assertion 1 holds here.

- $m(\delta) = 3$: If the three minus signs are pairwise non-consecutive, then we get three quadrilaterals by Observation 2. If the three minus signs are consecutive, then δ has a plus subsequence of length at least two. Thus, we get two quadrilaterals by Observation 2 and one by Observation 1. Assume the minus signs are partitioned into two disjoint subsequences of lengths one and two. Then, we get two quadrilaterals for the minus signs. If δ has a plus subsequence of length at least two, then we get a third quadrilateral by this subsequence. The remaining sequences are $+--+-+$ and its reverse.

  We show how to get three compatible 4-holes with the sequence $+--+-+$. See Figure 4. First we look at p_1. If p_1 is below $l_{2,5}$ then the three quadrilaterals $p_0p_1p_2p_5$, $p_2p_3p_4p_5$, and $p_0p_5p_6p_7$ are compatible. Assume p_1 is above $l_{2,5}$. If p_1 is below $l_{3,4}$ then the quadrilaterals $p_1p_3p_4p_2$, $p_0p_2p_4p_5$, and $p_0p_5p_6p_7$ are compatible. Assume p_1 is above $l_{3,4}$. Now, we look at p_6. If p_6 is above $l_{3,4}$ then $p_1p_6p_4p_3$, $p_2p_3p_4p_5$, and $p_0p_5p_6p_7$ are compatible. If p_6 is below $l_{3,4}$ and above $l_{2,5}$ as in Figure 4-left, then $p_0p_2p_3p_5$, $p_3p_4p_6p_5$, and $p_0p_5p_6p_7$ are compatible. Assume p_6 is below $l_{2,5}$ as in Figure 4-right; consequently p_7 is also below $l_{2,5}$ because p_6 has $-$ signature. Since p_5 has $+$ signature, p_4 is above $l_{5,6}$. Now, we look at p_8. If p_8 is above $l_{5,6}$, then $p_4p_8p_6p_5$, $p_2p_3p_4p_5$, and $p_0p_5p_6p_7$ are compatible. If p_8 is below $l_{5,6}$ and above $l_{5,7}$, then $p_5p_6p_8p_7$, $p_2p_3p_4p_5$, and $p_0p_2p_5p_7$ are compatible. Assume p_8 is below $l_{5,7}$ as in Figure 4-right. In this case $p_2p_3p_4p_5$, $p_2p_5p_6p_7$, and $p_0p_2p_7p_8$ are compatible.

- $m(\delta) = 4$: If the two plus signs in δ are consecutive, then we get one quadrilateral by Observation 1 and two by Observation 2. Assume the two plus signs are non-consecutive. If the minus signs are partitioned into three subsequences or two subsequences of lengths one and three, then we get three compatible 4-holes by Observation 2. The remaining sequences are $+----+$, $+--+--$ and its reverse. For the sequence $+----+$ we get three quadrilaterals by Lemma 3. The sequence $+--+--$ can be handled by splitting as $+--+|-|-$, where we get two quadrilaterals for the subsequence $+--+$, by Lemma 3, and one quadrilateral for the last minus sign, by Observation 1. Notice that Assertion 2 holds here as we apply Observation 1 on the last minus sign.

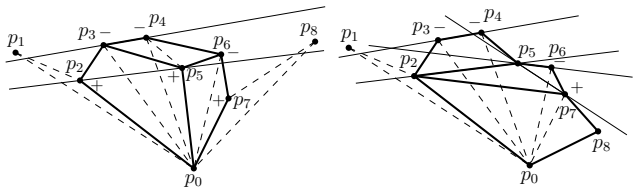


Figure 4: Signature sequence $+--+-+$. Left: p_1 is above $l_{3,4}$, and p_6 is below $l_{3,4}$ and above $l_{2,5}$. Right: p_1 is above $l_{3,4}$, p_6 is below $l_{2,5}$, and p_8 is below $l_{5,7}$.

- $m(\delta) = 5$: If the five minus signs are consecutive, then we get three compatible quadrilaterals by Observation 2. Otherwise, δ has two minus subsequences, one of which has size at least three. By Observation 2 we get three quadrilaterals with these two subsequences.

- $m(\delta) = 6$: The six minus signs are consecutive and our result follows by Observation 2.

4 Compatible 4-holes in n-sets

In this section we prove our main claim for large point sets, that is, every n-set contains at least $\lfloor 5n/11 \rfloor - 1$ compatible 4-holes. As in Section 2, by combining Theorems 4 and 5 with the idea of partitioning the points into some cones with respect to their radial ordering about a point p_0, we can improve the lower bound on the number of compatible 4-holes in an n-set to $3 \cdot \lfloor (n-2)/7 \rfloor$ and $4 \cdot \lfloor (n-2)/9 \rfloor$, respectively. In the rest of this section, we first prove a lemma, that can be used to improve these bounds further. We denote by ab the straight-line through two points a and b. We say that a 4-hole Q is *compatible with* a point set A if the interior of Q is disjoint from the interior of the convex hull of A.

Lemma 6 *For every $(r+s)$-set, with $r, s \geqslant 4$, we can divide the plane into two internally disjoint convex regions such that one region contains a set A of at least s points, the other region contains a set B of at least r points, and there exists a 4-hole that is compatible with A and B.*

Before proving this lemma, we note that a similar lemma has been proved by Hosono and Urabe (Lemma 3 in [19]) for disjoint 4-holes, where they obtain a set A' of $s-2$ points, a set B' of $r-2$ points, and a 4-hole Q that is disjoint from A' and B'. However, their lemma does not imply our Lemma 6, because it might not be

possible to add two points of Q to A' to obtain a set A of s points such that Q is compatible with A.

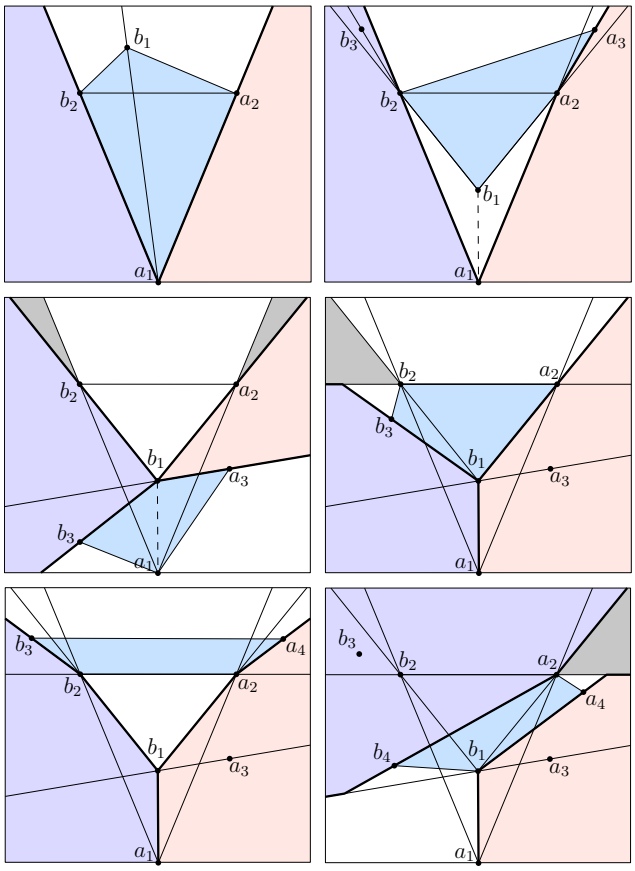


Figure 5: Illustration of Lemma 6. The convex regions with r and s points are shown in light purple and light orange, respectively. The compatible 4-holes with these regions are in blue color. The gray regions are empty.

In the following proof, if there exist two internally disjoint convex regions such that one of them contains a set A of s points, the other contains a set B of r points, and there exists a 4-hole that is compatible with A and B, then we say that A and B are *good*.

Proof of Lemma 6. Consider an $(r+s)$-set. In this proof a "point" refers to a point from this set. Also when we say a convex shape is "empty" we mean that its interior does not contain any point from this set.

Let a_1 be a point on the convex hull of this set, and without loss of generality assume that a_1 is the lowest point. Let a_2 be the point such that $s-2$ points are to the right side of the line a_1a_2. Let A be the set of points that are on or to the right side of a_1a_2, and let B be the set of other points. Notice that A contains s points and B contains r points. Let b_1 be the point of B such that the interior of $C(a_1{:}a_2, b_1)$ does not contain any point. Let b_2 be the point of B such that the interior of $C(a_1{:}a_2, b_2)$ contains only b_1. See Figure 5(top-left).

If b_1 is not in the interior of the triangle $\triangle a_1a_2b_2$, then $a_1a_2b_1b_2$ is a 4-hole that is compatible with A and $(B \setminus \{b_1\}) \cup \{a_1\}$. As shown in Figure 5(top-left), the interiors of the convex hulls of these two sets are disjoint, and thus, these two sets are good. Assume that b_1 is in the interior of $\triangle a_1a_2b_2$. We consider two cases depending on whether or not $C(b_1{:}b_2, a_2)$ is empty.

- $C(b_1{:}b_2, a_2)$ is not empty. If $C(b_1{:}b_2, a_2)$ contains a point of A, then let a_3 be such a point that is the neighbor of a_2 on $CH(A)$; see Figure 5(top-right). Then $b_1b_2a_3a_2$ is a 4-hole, and A and $(B \setminus \{b_1\}) \cup \{a_1\}$ are good. If $C(b_1{:}b_2, a_2)$ contains a point of B, then let b_3 be such a point that is the neighbor of b_2 on $CH(B)$. Then $b_1b_2b_3a_2$ is a 4-hole, and A and $(B \setminus \{b_1\}) \cup \{a_1\}$ are good.
- $C(b_1{:}b_2, a_2)$ is empty. Let a_3 be the attack point of $h(b_1{:}a_1{\to}a_2)$, i.e., the first point that $h(b_1{:}a_1{\to}a_2)$ meets. If the attack point of $h(b_1{:}a_1{\to}b_2)$ is below b_1a_3, then let b_3 be that point; Figure 5(middle-left). In this case $b_1a_3a_1b_3$ is a 4-hole, and $(A \setminus \{a_1\}) \cup \{b_1\}$ and B are good. Assume that the attack point of $h(b_1{:}a_1{\to}b_2)$ is above b_1a_3. We consider the following two cases depending on whether or not there is a point of B above the line a_2b_2.
  - No point of B is above a_2b_2. Let b_3 be the attack point of $h(b_1{:}b_2{\to}a_1)$ as in Figure 5(middle-right). Then $b_1b_3b_2a_2$ is a 4-hole, and $A \cup \{b_1\}$ and $(B \setminus \{b_2\}) \cup \{a_1\}$ are good.
  - Some point of B is above a_2b_2. Let b_3 be such a point that is the neighbor of b_2 on $CH(B)$. If some point of A is above a_2b_2, then let a_4 be such a point that is the neighbor of a_2 on $CH(A)$; see Figure 5(bottom-left). Then $a_2b_2b_3a_4$ is a 4-hole, and $A \cup \{b_1\}$ and $B \cup \{a_1\}$ are good. Assume that no point of A is above a_2b_2. Let a_4 be the attack point of $h(b_1{:}a_2{\to}a_3)$ and b_4 be the attack point of $h(a_2{:}b_1{\to}b_2)$ as in Figure 5(bottom-right). Notice that it might be the case that $b_4 = b_2$. In either case, $b_1b_4a_2a_4$ is a 4-hole, and $(A \setminus \{a_2\}) \cup \{b_1\}$ and $(B \setminus \{b_1\}) \cup \{a_2\}$ are good. □

Theorem 7 *Every n-set contains at least $\lfloor 5n/11 \rfloor - 1$ compatible 4-holes.*

Proof. Let P be an n-set. Our proof is by induction on the number of points in P. The base cases happen when $|P| \leqslant 14$. If $|P| \leqslant 13$, then our claim follows from one of Theorems 1, 2, 4, or 5. If $|P| = 14$, then by applying Lemma 6 on P with $r = s = 7$ we get a 4-hole together with two sets A and B each containing at least 7 points. By Theorem 2 we get two 4-holes in each of A and B. Thus, we get five compatible 4-holes in total. This finishes our proof for the base cases.

Assume that $|P| \geqslant 15$. By applying Lemma 6 on P with $r = n-11$ and $s = 11$ (notice that r is at least four

as required by this lemma) we get a 4-hole together with two sets A and B such that the interiors of their convex hulls are disjoint, A contains at least 11 points, and B contains at least $n-11$ points. By Theorem 5 we get four compatible 4-holes in $CH(A)$. By induction, we get $\lfloor 5(n-11)/11 \rfloor - 1$ compatible 4-holes in $CH(B)$. Therefore, in total, we get

$$1 + 4 + \left(\left\lfloor \frac{5(n-11)}{11} \right\rfloor - 1 \right) = \left\lfloor \frac{5n}{11} \right\rfloor - 1$$

compatible 4-holes in P. □

An $O(n \log^2 n)$-time algorithm for computing this many 4-holes follows from the proofs, by using a dynamic convex hull data structure for computing the sets A and B in Lemma 6.

References

[1] Personal communication with J. Urrutia.

[2] O. Aichholzer, F. Aurenhammer, and H. Krasser. On the crossing number of complete graphs. *Computing*, 76(1):165–176, 2006.

[3] O. Aichholzer, C. Huemer, S. Kappes, B. Speckmann, and C. D. Tóth. Decompositions, partitions, and coverings with convex polygons and pseudo-triangles. *Graphs and Combinatorics*, 23(5):481–507, 2007.

[4] O. Aichholzer and H. Krasser. The point set order type data base: A collection of applications and results. In *Proceedings of the 13th Canadian Conference on Computational Geometry*, pages 17–20, 2001.

[5] O. Aichholzer, R. F. Monroy, H. González-Aguilar, T. Hackl, M. A. Heredia, C. Huemer, J. Urrutia, P. Valtr, and B. Vogtenhuber. On k-gons and k-holes in point sets. *Comput. Geom.*, 48(7):528–537, 2015.

[6] B. B. Bhattacharya and S. Das. On the minimum size of a point set containing a 5-hole and a disjoint 4-hole. *Studia Scientiarum Mathematicarum Hungarica*, 48(4):445–457, 2011.

[7] B. B. Bhattacharya and S. Das. Disjoint empty convex pentagons in planar point sets. *Periodica Mathematica Hungarica*, 66(1):73–86, 2013.

[8] A. Biniaz, A. Maheshwari, and M. Smid. Compatible 4-holes in point sets. *CoRR*, abs/1706.08105, 2017.

[9] P. Bose, S. Ramaswami, G. T. Toussaint, and A. Turki. Experimental results on quadrangulations of sets of fixed points. *Computer Aided Geometric Design*, 19(7):533–552, 2002.

[10] P. Bose and G. T. Toussaint. Characterizing and efficiently computing quadrangulations of planar point sets. *Computer Aided Geometric Design*, 14(8):763–785, 1997.

[11] A. Brodsky, S. Durocher, and E. Gethner. Toward the rectilinear crossing number of K_n: new drawings, upper bounds, and asymptotics. *Discrete Mathematics*, 262(1-3):59–77, 2003.

[12] J. Cano, A. G. Olaverri, F. Hurtado, T. Sakai, J. Tejel, and J. Urrutia. Blocking the k-holes of point sets in the plane. *Graphs and Combinatorics*, 31(5):1271–1287, 2015.

[13] O. Devillers, F. Hurtado, G. Károlyi, and C. Seara. Chromatic variants of the Erdős-Szekeres theorem on points in convex position. *Comput. Geom.*, 26(3):193–208, 2003.

[14] P. Erdős. Some more problems on elementary geometry. *Austral. Math. Soc. Gaz.*, 5:52–54, 1978.

[15] P. Erdős and G. Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.

[16] T. Gerken. Empty convex hexagons in planar point sets. *Discrete & Computational Geometry*, 39(1):239–272, 2008.

[17] H. Harborth. Konvexe Fünfecke in ebenen Punktmengen. *Elemente der Mathematik*, 33:116–118, 1978.

[18] J. D. Horton. Sets with no empty convex 7-gons. *Canad. Math. Bull.*, 26(4):482–484, 1983.

[19] K. Hosono and M. Urabe. On the number of disjoint convex quadrilaterals for a planar point set. *Comput. Geom.*, 20(3):97–104, 2001.

[20] M. Lomeli-Haro, T. Sakai, and J. Urrutia. Convex quadrilaterals of point sets with disjoint interiors. In *Abstracts of Kyoto International Conference on Computational Geometry and Graph Theory (KyotoCGGT2007)*.

[21] L. Lovász, K. Vesztergombi, U. Wagner, and E. Welzl. Convex quadrilaterals and k-sets. In J. Pach, editor, *Towards a Theory of Geometric Graphs*, volume 342 of *Contemporary Mathematics*, pages 139–148. 2004.

[22] C. M. Nicolás. The empty hexagon theorem. *Discrete & Computational Geometry*, 38(2):389–397, 2007.

[23] S. Ramaswami, P. A. Ramos, and G. T. Toussaint. Converting triangulations to quadrangulations. *Comput. Geom.*, 9(4):257–276, 1998.

[24] T. Sakai and J. Urrutia. Covering the convex quadrilaterals of point sets. *Graphs and Combinatorics*, 23(Supplement-1):343–357, 2007.

[25] G. T. Toussaint. Quadrangulations of planar sets. In *Proceedings of the 4th International Workshop on Algorithms and Data Structures* (*WADS*), pages 218–227, 1995.

[26] P. Valtr. On empty hexagons. In J. E. Goodman, J. Pach, and R. Pollack, editors, *Surveys on Discrete and Computational Geometry: Twenty Years Later*, pages 433–441. 2008.

[27] B. Vogtenhuber. *Combinatorial aspects of colored point sets in the plane*. PhD thesis, Graz University of Technology, November 2011.

[28] U. Wagner. On the rectilinear crossing number of complete graphs. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms* (*SODA*), pages 583–588, 2003.

[29] L. Wu and R. Ding. On the number of empty convex quadrilaterals of a finite set in the plane. *Appl. Math. Lett.*, 21(9):966–973, 2008.

Some Heuristics for the Homological Simplification Problem

Erin W. Chambers * Tao Ju † David Letscher ‡ Mao Li § Christopher Topp ¶ Yajie Yan ‖

Abstract

In this paper, we consider heuristic approaches for solving the homological simplification problem. While NP-Hard in general, we propose an algorithm that in practice significantly reduces topological noise from large datasets, such as those from medical or biological imaging.

1 Introduction

In this paper, we will consider the homological simplification problem. Introduced in [5], this asks: given a pair of simplicial complexes (C, N) where $C \subset N$, can the persistent homology group of this pair be realized as the homology of some intermediate complex? This problem is one way to approach the problem of topologically accurate simplification, where the goal is to take a "noisy" shape and simplify it to reach some desired topological structure. Such algorithms are useful in a wide range of applications, as any surface or region reconstruction algorithm on scanned input data is apt to contain errors, and hence a post processing phase to simplify it is necessary.

For filtrations of closed and orientable 2-manifolds, the homological simplification problem is solvable [11]; this work actually solves the more general problem of finding an ϵ-simplification in a filtration. However, such existence results do not hold in 3-manifolds since there are filtrations of manifolds that do not have ϵ-simplifications [11]. For the homological simplification problem, it is NP-Hard to determine if a simplification exists for 3-manifolds, even if the complex is embedded in $\mathbb{R}^3$ [5].

In this paper, we consider a 2-phase heuristic algorithm to simplify voxelized shapes. We first use a persistent homology-based algorithm to identify candidate simplifications. This approach is inspired by prior work to find "nice" generators for homology groups on surfaces [8], but we expand to find not just the generators of homology group, but also representatives in the larger space that kills those generators. Phase 2 is then a validation: given such a candidate simplification, we must check that it does result in a global simplification, since adding such things can introduce new topological features. We apply this algorithm to a number of types of input data, to assess how successful the heuristic is in practice. We find that our simplifications are able to remove over 99% of topological errors in several real-world data sets.

*Department of Computer Science, Saint Louis University, echambe5@slu.edu.

†Department of Computer Science and Engineering, Washington University, St. Louis taoju@cse.wustl.edu.

‡Department of Computer Science and Engineering, Saint Louis University, letscher@slu.edu.

§Donald Danforth Plant Science Center, mli@danforthcenter.org .

¶Donald Danforth Plant Science Center, ctopp@danforthcenter.org .

‖Department of Computer Science and Engineering, Washington University, St. Louis yajieyan@wustl.edu.

2 Related Work

2.1 Cubical complexes

A *cubical complex* is built from a collection of cells that are points, intervals, squares, cubes and higher dimensional analogs, where the intersection of any two cells is also a cell of the cubical complex. Formally, the cells are built from products of intervals, either the unit interval $[k, k+1]$ or a degenerate interval $[k, k]$, where k is an integer (so that our complex is aligned with an integer lattice); in d-dimensional space, a cube is a product of d elementary intervals. Given 2 cubes x and y, x is a face of y if $x \subseteq y$. A cubical complex of dimension d is a collection of cubes that is closed under taking faces such that the intersection of any two faces is a common subface. In this paper, we will use cubical complexes to represent the topology of our shapes.

2.2 Digital topology

In imaging and computer graphics, voxelizations are among the most common shape representations. With 3-dimensional Euclidean space divided using a cubical lattice, each individual cube is a single *voxel* and a voxelization of a shape is a set of these voxels. A common connectivity model of voxels is called 6-connectivity, which considers two voxels adjacent if they share a common 2-dimensional face [15]. This connectivity model results in a different topology than just taking the actual union of the voxels, which would typically connect anything that shares an edge or vertex as well. To be topologically consistent with the union of voxels, we will work in a dual complex where there is a vertex for each

voxel, and edge whenever two voxels share a common face, a square or 2-cell for any 4 voxels around an edge, and a 3-cell or voxel whenever there are 8 voxels surrounding a common vertex. Note that this is still a cubical complex, but it is not a pure cubical complex, since not every cell of dimension 1 or 2 belongs to 3-cell.

2.3 Homology

Homology groups and persistent homology on filtrations are commonly used tools to find topological features in spaces; due to space constraints, we refer the reader to recent books covering the topic for definitions of homology and persistent homology groups [9, 17]. For two spaces $C \subseteq N$, these persistent homology groups simply capture some of the topological features that are present in C and still remain in N. Most relevant to our setting, cubical persistent homology has been considered in some prior work [21], including optimized data structures to compute persistent homology groups of such complexes.

The p^{th} Betti number of a space X, $\beta_p(X)$ is defined to be the rank of the p^{th} homology groups. Given an inclusion map $f : C \to N$, we can define a persistent notion of Betti number, where the p^{th} Betti number of f is the rank of the induced map on homology: $\beta_p(C \to N) = rk(f^*(H_p(C)))$. Extending this to filtrations of more than 2 spaces precisely gives the notion of persistent homology groups (or their ranks).

2.4 Homological simplification

Consider two spaces $C \subset N \subset \mathbb{R}^3$. The homological simplification problem is generally phrased in terms of Betti numbers: we wish to find a space X such that the p-dimensional Betti number, $\beta_p(X)$ is equal to the Betti number of the inclusions $C \to N$, $\beta_p(C \to N)$. In three dimensions, this problem has been shown to be NP-Hard [5].

2.5 Topological repair

Various methods have been developed in different research communities for removing topological errors of surfaces in $\mathbb{R}^3$. In computer graphics, algorithms exist for modifying a given surface to either remove features smaller than a given size or achieve a specific topology (e.g., a single connected component with a prescribed genus) [22, 24, 6]. These methods make decisions of where and how to modify the surface solely based on the shape of the given surface, whereas homological simplification bases its decision on the persistence of topological features between two spaces. In medical image analysis, there has been active research on rectifying the topology of reconstructions of biological structures, such as the cortical surface in the human brain [19, 13, 18, 23]. Most of these methods are specialized for removing redundant handles and cannot deal with other types of topological noises such as disconnected components or cavities. Finally, in scientific visualization, a line of research aims at simplifying the topological structure (e.g., the Morse-Smale complex) of a scalar function [7, 11, 12, 20]. While these methods effectively remove topological noise on *all* level sets of the function, they are unnecessarily expensive if the goal is to fix the topology of one level set. In addition, methods designed to work on functions in $\mathbb{R}^3$ are limited to reducing the set of maxima or minima, and hence are less useful for removing topological handles on the level sets.

3 Heuristic Shape Simplification

Since we will use voxelized representatives of our input shapes, we will focus on the restriction of the homology simplification problem to voxelized shapes. We restate the problem as follows where we refer to C as the core, and N as the neighborhood, and we wish to find a space X which is somehow "in between" them with homology equal to the persistent homology of $C \to N$.

3.1 Voxelized homological simplification

Suppose $C \subset N \subset \mathbb{R}^3$ are voxel regions. Determine if there exists X such that $C \subset X \subset N$ and $\beta_p(X) = \beta_p(C \to N)$ for all p.

Proposition 1 *The voxelized homological simplification problem is NP-hard.*

Proof. [Proof sketch] We omit details due to space constraints, but just briefly note that the proof for simplicial complexes embedded in $\mathbb{R}^3$ is a reduction from 3SAT [5]. They construct gadgets that are easily modified to be built from voxels on an $O(m)$ by $O(n)$ grid, where n is the number of booleans variables and m is the number of clauses in the 3SAT instance. □

It is typically impractical to solve the homological simplification problem; however, we are concerned with simplifying a shape as much as possible. In particular, we will try to find a shape X where $C \subset X \subset N$ with each of the Betti numbers β_p as close as possible to the persistent Betti numbers $\beta_p(C \to N)$. If they are equal, we have a solution to the homological simplification problem. Even if we do not achieve maximal simplification, we will typically simplify our shape significantly.

Our simplification procedure will be a two phase process. It starts with the core C, which our algorithm will always include in the result, and then tries to expand to remove topological noise. First, a modified version of the standard persistence algorithm [10] is used to find

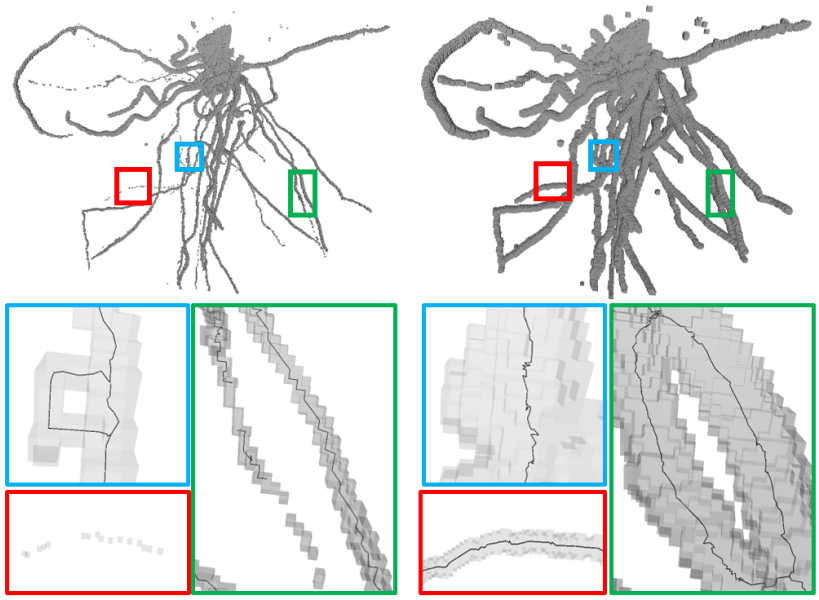


Figure 1: The core and neighborhood for the root system of corn. Note that in the inclusion of core into the neighborhood some loops are filled (blue inset), components connect (red inset) and new loops form (green inset). A solution to the homological simplification problem will accept the first two types of modifications and reject the creation of new loops and voids.

sets of voxels whose addition would remove topological features that are not in the neighborhood. However, these candidates might add new features that are not desired. The second phase examines the modified shape and checks if the persistence Betti numbers have been reduced. If so, the changes are kept. This process of candidate generation and validation is repeated until no more valid additions can be found. There are no guarantees that this will remove all of the undesired topological features; however, our experiments in Section 5 show that in real world examples the accuracy of the simplification is very high.

3.2 Types of topological errors

We now examine the types of topological errors in more detail, and our candidate simplifications. Consider $C \subset N \subset \mathbb{R}^3$ and the maps induced by inclusion $f_p^* : H_p(C) \to H_p(N)$ for $p = 0, \ldots, 3$. If every f_p^* is an injection then C is already a solution to the homological simplification problem. Assuming we do not have a solution, we know that not all the f_p^* are injective; in addition, we know that any extraneous topological noise must appear in the kernel of some f_p^*. In order to get a solution to the homological simplification problem, we therefore need to remove all such cycles.

To this end, consider a cycle $\alpha \neq 0$, where $f_p(\alpha) = 0$. In this 3-dimensional setting, we have several possibilities: α might connect two components, or it might fill a void or loop in the large space. To demonstrate what these look like, assume α is *irreducible*, that is it cannot be written as a non-trivial sum of other elements of the kernel. Depending on the dimension p, we can interpret the different types of noise we observe, and our algorithm will suggest different ways of removing that α in order to construct a better candidate solution (or near-solution, if we cannot generate an optimal one) to the homological simplification problem. We note that each of these cases can be observed in the root data in Figure 1, and hence we expect them all to exist in practice, depending on the data set. The cases to consider:

- $\alpha \in \ker f_0^*$: Here α is represented by two points lying in different components of C. These components can be connected by a path γ in N that connects the two points. In this case, γ is a one dimensional chain in N. By adding γ to C we obtain a new space $C \cup \gamma$ where the kernel of $H_0(C \cup \gamma) \to H_0(N)$ has rank one less than $H_0(C) \to H_0(N)$.
- $\alpha \in \ker f_1^*$: In this case α is a curve that follows a non-trivial topological feature in C, such as going around a handle, but which‘‘ bounds some surface in N that kills that homological feature. If this surface is a disk D then $H_1(C \cup D) \to H_1(N)$ has rank one less than $H_1(C) \to H_1(N)$. In other words, we can add the disk D to remove the handle that is present in C. However, if the curve bounds a surface with genus, then if we were to add the surface in, a generator of the kernel is removed but two or more generators are added (since this surface has

its own handles). Hence the resulting object is not simpler in terms of homology, and we cannot use this chain to simplify the space.

- $\alpha \in \ker f_2^*$: Here α is a surface bounding a hollow region (a "void") in C, but which is not hollow in N. This void could be a simple ball, but could also contain topology. In either case, we can fill this in to simplify the topology. If the surface is a sphere, Alexander's theorem implies that it will bound a ball B in N [14]. In this case, it is easy to simplify since $H_2(C \cup B) \to H_2(N)$ has rank one less than $H_2(C) \to H_2(N)$. If the surface does not bound a ball, then the simplification may also simplify $\ker f_1$, as some loops will disappear when the void disappears.

The commonality of all of these cases is that our algorithm must find a $(k+1)$-dimensional structure in N whose boundary is in the kernel of f_k^* and whose addition simplifies the topological structure. Our techniques are based on this observation, repeatedly adding simplifications until no more can be found; we describe the generation of these candidates in the next subsection. Ideally, after adding these $(k+1)$-dimensional structures, we obtain a space X with $C \subset X \subset N$ and $H_k(C) \to H_k(X)$ is a surjection and $H_k(X) \to H_k(N)$ is an injection for all k; in this case, we would actually have a solution to the homological simplification problem. Our algorithm will only simplify the shape, although we have no guarantees of optimality or approximation ratio. In Section 5, we will discuss our implementation and show that it works well on several data sets.

3.3 Candidate generation

We now describe our algorithm to find generators in the kernel as well as to find candidate cycles that they bound in order to simplify the shape. Consider $C \subset X \subset N$ where X is initially equal to C but will be extended to remove topological features that are not shared by C and N. We will build a boundary matrix, ∂_p, for calculating the p-dimensional persistent homology of the filtration $C \subset X \subset N$. There is a column of ∂_p for each $(p+1)$-cell of X and a non-zero entry in that column for each p-cell that is a face of the $(p+1)$-cell. The rows and columns are ordered so that cells of C occur first, then cells of X and finally cells of N. The standard persistence algorithm [10] adds columns to ones to their right to obtain a canonical reduced form.

After these matrix reductions, the columns of ∂_p represent cycles involving the cells with non-zero entries. If that column corresponds to cells of N and the non-zero entries involved include d-cells of C, then this cycle, z, is a feature of C that does not persist in N and should

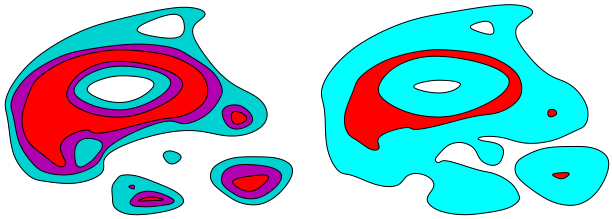


Figure 2: (top) Three intensity threshold input: $t-\epsilon, t, t+\epsilon$. (bottom) Corresponding core and neighborhood.

be removed. To remove this feature we need to find a $(d+1)$ chain, b, in X such that $\partial_p b = z$. We note that of course there are potentially many candidates for z; we find one by creating a copy of the identity matrix M with an entry for each column of ∂_p and performing the same column operations as we did in the reduction. An invariant of these column operations is that if v is the i-th column of M, then $\partial_p v$ is the i-th column of ∂_p. So we choose b to be the union of cells with non-zero entries in the column of M corresponding to the cycle z; recall that b is actually a chain in the dual cubical complex. We build a candidate simplification S which is the set of voxels that contain b. When S is added to X, the cycle z will become trivial; however, additional features could be added, so we cannot add S and guarantee reduction of Betti numbers.

We note that many improvements to the standard persistence algorithm have been made to make this calculation more efficient [2, 1]. These algorithms focus only on finding birth and death times, however, and discard any non-essential information in order to make calculations faster. In particular, they either discard or never compute information about generators in order to speed up the calculations; we need not only the generators, but also the cycles they bound, which we know of no way to track directly in the faster algorithms.

In addition, it is worth re-iterating that there are many possible cycles that could be added in order to kill generators in the kernel; we have merely computed the ones which can be calculated with a simple modification of the standard persistence algorithm. Surprisingly, in our experiments the algorithm nonetheless produces geometrically pleasing results, particularly for smaller topological errors which are common in real-world data.

3.4 Candidate validation

To determine if the addition of S will actually simplify the shape, recall that our algorithm tries to minimize the sum of Betti numbers $B(X) = \sum_{p=0}^{3} \beta_p(X)$, with the goal of reducing this complexity to be equal to $\sum_{p=0}^{3} \beta_p(C \to N)$. If S is set of voxels that form a candidate simplification, we will calculate $B(X \cup S)$. If $B(X \cup S) < B(X)$ and $\beta_p(X \cup S) \leq \beta_p(X)$ for all

p, then we will consider this a *valid simplification*. We cannot be sure that the addition of S has not created new topological feature, but our check does guarantee that more features are removed than were introduced.

Our algorithm We will repeat the process of finding candidate simplifications using the modified standard persistence algorithm and adding them one at a time if they are valid simplifications, until no more are possible.

4 Removing Topological Errors in 3D Imaging

In a 3D imaging technology such as CT or MRI, the output is a grid of voxels each with an intensity value. A typical representation would be a map $f : \{0, \ldots, k-1\}^3 \to \mathbb{R}$. Shapes of interest, in theory, can be extracted by selecting voxels in a given intensity range. A typical segmentation might try to extract all voxels with intensity over a specified threshold, e.g. $X = f^{-1}([t, \infty))$. However, there can be a variety of errors in this segmentation.

Define $N_\delta(X)$ as a neighborhood of the set X for positive δ, or as points with a $|\delta|$ neighborhood contained in the set X for negative δ. Formally:

$$N_\delta(X) = \begin{cases} \cup_{x \in X} B_\delta(x) & \delta \geq 0 \\ \{x \mid B_{-\delta(x)} \subset X\} & \delta < 0 \end{cases}$$

Then there is a natural two dimensional filtration of the region where $X_{t,\delta} = N_\delta\left(f^{-1}([t, \infty))\right)$. Here, t specifies the intensity threshold, and N_δ is in fact a morphological dilation when $\delta > 0$ or erosion when $\delta < 0$, using the ball of radius $|\delta|$ as the structuring element[16].

If t is a threshold for an initial segmentation $X = f^{-1}([t, \infty))$, we can give two parameters, ϵ and δ, representing noise levels on the threshold value and geometric scale, respectively. We will define the core $C = X_{t+\epsilon,-\delta}$ and the neighborhood $N = X_{t-\epsilon,\delta}$. This collapses the two dimensional filtration into a pair of spaces $C \subset N$. In this case the core C represents voxels that we want to include in the final shape, they all have neighborhoods meeting a higher intensity threshold. And the neighborhood N contains all the voxels near some voxel with a lower intensity threshold. See Figure 2 for an example.

A good solution to the homological simplification with this C and N would have features that do not appear or disappear with a small change in threshold, expanding or contraction of the shape, or a combination of these changes. In practice, a partial solution to the homological simplification problem might only remove some of the noise; in our experiments (described in the next section), this still results in an improvement over simple thresholding techniques. This improvement is quite evident in Figure 1, for example, where thresholding any any level (in the top pictures) leaves errors such as disconnected fragments and cycles; since this is a root and hence a simply connected space, these must be sampling errors and not actual features of the data. In the following section, we will discuss our observations of a reduction in more than 99% of the noise in several real world datasets.

5 Experimental Results

We experimented on three different collections of data: CT scans of corn root systems, synthetic root systems and brain volumes reconstructed from histological sections. The corn data was from a single variety of corn, with three different scans each viewed at two different resolutions. The synthetic root was designed to roughly resemble a root system and was studied at nine different resolutions. The brain scans were of the BigBrain dataset [4] downsampled at ten different resolutions. Figure 3 gives some information about the datasets, and our code is available [3]. The largest regions had close to 400 million voxels; this forced some additional techniques described below to handle the scale of the data sets. The largest core in these experiments has 2.5 million voxels and the largest neighborhood had over 4 million voxels. The most complicated example had approximately 8 thousand components, 17 thousand loops and one thousand voids. All but approximately 100 of those where noise features. In general, the complexity of the initial shapes was very high and there were very few features that were shared between the core and the neighborhood.

5.1 Practical concerns

The shapes that we have considered have regions as large as several hundred million voxels. As a result, we utilize a sparse representation for our shapes, so that the neighborhoods have about 6 million voxels at most; instead of storing actual values of the intensity, we have 3 hash sets that simply mark the core C, the current shape, and the neighborhood N. Even with this compression, there are speed issues. While there are faster persistent homology implementations [1, 2] that can handle inputs of this size, the standard persistence algorithm [10] has trouble on this size of inputs. However, we have no way to generate candidate simplifications using these faster algorithms, as discussed in Section 3.3. In order to deal with the size of the data, we did this calculation on windows of data at most 250^3 voxels in size, where the windows were chosen to overlap and cover the shape. This of course makes it likely we will miss some larger errors, but made the algorithm computationally feasible. We performed the calculations on the largest shapes in under five hours on a Linux desktop with a 2.20 GHz processor I5 processor and 64 GB RAM; see Figure 4.

	Region size	Core size	Neighborhood size	Complexity
Corn	$2.3 \times 10^6 - 1.5 \times 10^8$	$1.6 \times 10^3 - 2.4 \times 10^5$	$2.4 \times 10^3 - 4.6 \times 10^6$	$109 - 6,713$
Synthetic	$3.9 \times 10^6 - 3.8 \times 10^8$	$1.8 \times 10^3 - 1.7 \times 10^5$	$2.3 \times 10^3 - 2.7 \times 10^6$	$208 - 26,605$
Brain	$3.9 \times 10^5 - 1.7 \times 10^7$	$5.1 \times 10^4 - 2.5 \times 10^6$	$6.1 \times 10^5 - 3.6 \times 10^6$	$388 - 15,393$

Figure 3: Characteristics of the data analyzed, including the number of voxels in the region examined, core and neighborhood and the initial complexity of the shapes.

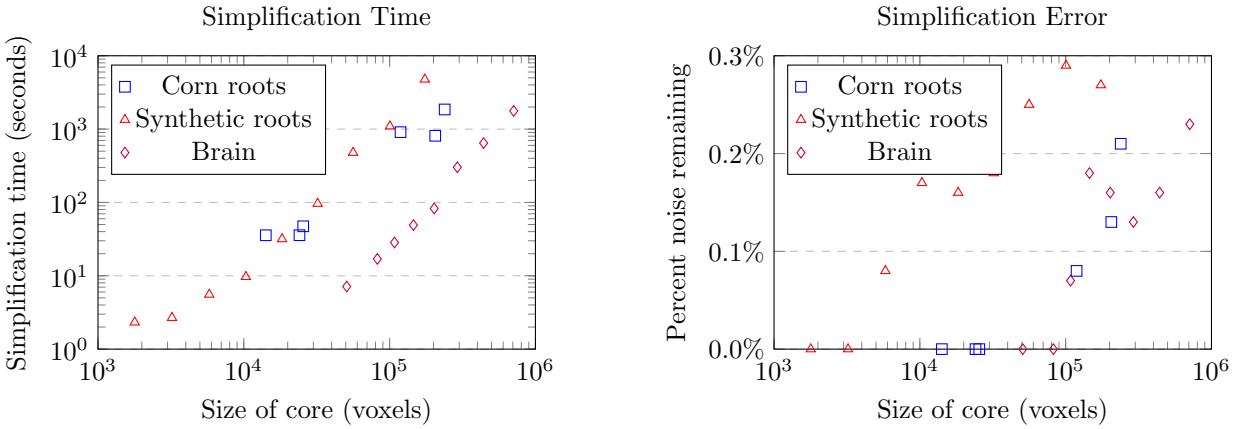


Figure 4: (a) Experimental runtime and (b) error rates on three datasets.

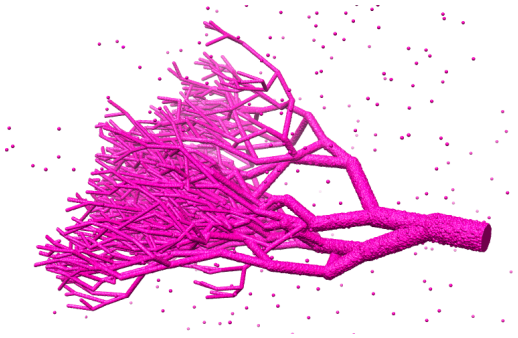


Figure 5: Synthetic root input shape with noise.

5.2 Accuracy

For all of the test shapes, over 99.7% of the topological errors were removed by our method; it is worth noting that much of this success is perhaps because many of the errors were quite small, consisting of just a few missing or extra voxels. In several cases, our algorithm was able to find solutions to the homological simplification problem. See Figure 4 for the error rates in the experiments. We note that it is computationally infeasible to determine in the other cases if solutions to the homological simplification problem exist.

6 Future Work

There are several natural directions to pursue next. First, we note that the algorithm described here picks some candidate to remove a particular error, but does not necessarily choose a geometrically nice generator. In fact, we have found examples where our algorithm adds a cycle that is obviously not ideal, or misses larger topological features due to the windows used in our algorithm. We plan to consider a more robust set of candidate simplifications that might be able to reduce the errors that are not repaired; at the same time, we would like to restrict to simplifications that have nice geometric properties as well as topological properties. In general, calculating "optimal" generators is impossible. However, for data sets such as the roots, we can take advantage of prior knowledge about the physical structure to prefer certain reconstructions, such as those that would reconnect two nearly crossing roots so as to maintain roughly the same local feature size along each. Second, on the more practical end, we would like to continue scaling the algorithms to larger datasets through optimizations and potential parallelization. Finally, more on the theoretical side, it is interesting to consider the notion of hardness of approximation or any approximation guarantees of heuristics such as our greedy approach.

Acknowledgements: This material is based upon work supported by the National Science Foundation under Award numbers: (PGRP) IOS-1638507, (EPSCoR) IIA-1355406, (AF) 1614562, and the linked collaborative awards (ABI) 1759807, 1759836 and 1759796.

References

[1] DIPHA. https://github.com/DIPHA/dipha.

[2] GUHDI c++ library. http://gudhi.gforge.inria.fr/.

[3] Voxelized homological simplification implementation. http://git.cs.slu.edu/public-repositories/shape-simplification-software.

[4] Katrin Amunts, Alan Evans, and Karl Zilles. Big-brain dataset.

[5] Dominique Attali, Ulrich Bauer, Olivier Devillers, Marc Glisse, and André Lieutier. Homological reconstruction and simplification in r3. In *Proceedings of the Twenty-ninth Annual Symposium on Computational Geometry*, SoCG '13, pages 117–126, New York, NY, USA, 2013. ACM.

[6] Marco Attene, Marcel Campen, and Leif Kobbelt. Polygon mesh repairing: An application perspective. *ACM Computing Surveys (CSUR)*, 45(2):15, 2013.

[7] P-T Bremer, Bernd Hamann, Herbert Edelsbrunner, and Valerio Pascucci. A topological hierarchy for functions on triangulated surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 10(4):385–396, 2004.

[8] T. K. Dey, K. Li, and J. Sun. On computing handle and tunnel loops. In *Cyberworlds, 2007. CW '07. International Conference on*, pages 357–366, Oct 2007.

[9] Herbert Edelsbrunner and John Harer. *Computational Topology: An Introduction*. AMS Press, 2009.

[10] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. Topological persistence and simplification. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pages 454–463. IEEE, 2000.

[11] Herbert Edelsbrunner, Dmitriy Morozov, and Valerio Pascucci. Persistence-sensitive simplification functions on 2-manifolds. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 127–134. ACM, 2006.

[12] David Günther, Alec Jacobson, Jan Reininghaus, Hans-Peter Seidel, Olga Sorkine-Hornung, and Tino Weinkauf. Fast and memory-efficienty topological denoising of 2d and 3d scalar fields. *IEEE transactions on visualization and computer graphics*, 20(12):2585–2594, 2014.

[13] Xiao Han, Chenyang Xu, Ulisses Braga-Neto, and Jerry L Prince. Topology correction in brain cortex segmentation using a multiscale, graph-based algorithm. *IEEE Transactions on Medical Imaging*, 21(2):109–121, 2002.

[14] John Hempel. *3-manifolds*, volume 349. American Mathematical Soc., 2004.

[15] Reinhard Klette and Azriel Rosenfeld. *Digital geometry: Geometric methods for digital picture analysis*. Elsevier, 2004.

[16] Laurent Najman and Hugues Talbot. *Mathematical morphology: from theory to applications*. John Wiley & Sons, 2013.

[17] Steve Y. Oudot. *Persistence Theory: From Quiver Representations to Data Analysis*, volume 209 of *Mathematical Surveys and Monographs*. American Mathematical Society, 2015.

[18] Florent Ségonne, Jenni Pacheco, and Bruce Fischl. Geometrically accurate topology-correction of cortical surfaces using nonseparating loops. *IEEE transactions on medical imaging*, 26(4):518–529, 2007.

[19] David W Shattuck and Richard M Leahy. Automated graph-based analysis and correction of cortical volume topology. *IEEE transactions on medical imaging*, 20(11):1167–1177, 2001.

[20] Maxime Soler, Melanie Plainchault, Bruno Conche, and Julien Tierny. Topologically controlled lossy compression. *arXiv preprint arXiv:1802.02731*, 2018.

[21] Hubert Wagner, Chao Chen, and Erald Vuçini. *Efficient Computation of Persistent Homology for Cubical Data*, pages 91–106. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[22] Zoë Wood, Hugues Hoppe, Mathieu Desbrun, and Peter Schröder. Removing excess topology from isosurfaces. *ACM Transactions on Graphics (TOG)*, 23(2):190–208, 2004.

[23] Rachel Aine Yotter, Robert Dahnke, Paul M Thompson, and Christian Gaser. Topological correction of brain surface meshes using spherical harmonics. *Human brain mapping*, 32(7):1109–1124, 2011.

[24] Qian-Yi Zhou, Tao Ju, and Shi-Min Hu. Topology repair of solid models using skeletons. *IEEE Transactions on Visualization and Computer Graphics*, 13(4), 2007.

Isomorphism Elimination by Zero-Suppressed Binary Decision Diagrams*

Takashi Horiyama† Masahiro Miyasaka† Riku Sasaki‡

Abstract

In this paper, we focus on the isomorphism elimination. More precisely, our problem is as follows: Given a graph G with labeled edges and a family $\mathcal{F}$ of its subgraphs, we extract all automorphisms $\mathrm{Aut}G = \{\pi_1, \pi_2, \ldots, \}$ on the given graph, define the lexicographically largest subgraph for each set of the mutually isomorphic subgraphs on each automorphism π_i, and select the lexicographically largest subgraphs on any of the automorphisms. In this paper, the families of subgraphs are manipulated by ZDDs. We also apply our algorithms to the enumeration of nonisomorphic developments of Platonic and Archimedean solids and d-dimensional hypercubes. Experimental results show that the proposed method is more than 300 times faster and 3,000 times less memory than the conventional method in the best case. Our algorithms are applicable to many other enumeration problems with eliminating isomorphic solutions.

1 Introduction

Suppose that we are given a cube. By cutting along the set of edges $\{e_2, e_3, e_4, e_6, e_{10}, e_{11}, e_{12}\}$ of the cube in Figure 1(a), we can obtain the development in Figure 1(c). When we rotate the positions of cut edges by 90 degrees, i.e., by cutting along the set of edges $\{e_1, e_3, e_4, e_7, e_9, e_{11}, e_{12}\}$ as depicted in Figure 1(b), we can also obtain the development in Figure 1(c). Are these the same? If we focus on the fact that the edges are *labeled*, the positions of cut edges are different, and thus we can say they are different. If we do not care about the labels, i.e., the edges are *unlabeled*, the shape of the developments are the same, and thus we can say they are *isomorphic*.

A cube has 384 labeled developments, and they are classified into 11 nonisomorphic developments (i.e., essentially different unlabeled developments). Here, a development and its mirror shape are regarded as isomorphic. As for the labeled developments, we can count their numbers by combining the following two theorems:

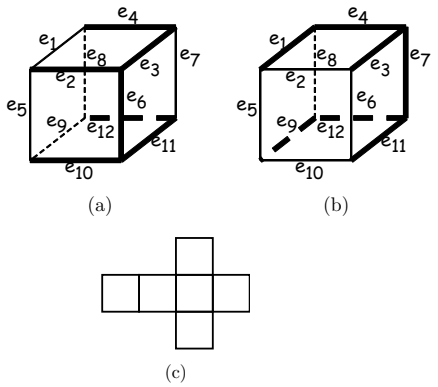


Figure 1: The developments of a cube by different cut edges (a) and (b) are isomorphic.

Theorem 1 *(See, e.g., [[5], Lemma 22.1.1]) The cut edges of a development of a polyhedron form a spanning tree of the 1-skeleton (i.e., the graph formed by the vertices and the edges) of the polyhedron, and vice versa.*

Theorem 2 *Matrix Tree Theorem [12]: The number of spanning trees of a graph is equal to any cofactor of the Laplacian matrix of the graph.*

By applying the theorems, Brown et al. showed that a Buckminsterfullerene (also known as an icosahedral C_{60}, or a truncated icosahedron) has 375,291,866,372,898,816,000 (approximately 3.75×10^{20}) labeled developments [1]. The numbers of labeled developments of Handballene (truncated dodecahedral C_{60}) and Archimedene (truncated icosidodecahedral C_{120}) are given in [2].

As for counting the nonisomorphic developments, the numbers for Platonic solids are obtained in the 1970s [7][10]. Recently, Horiyama and Shoji proposed a technique for counting the number of nonisomorphic developments of any polyhedron (including nonconvex polyhedron) [9]. By applying this method, they also listed the number of nonisomorphic (and also labeled) developments of all regular-faced convex polyhedra (i.e., Platonic solids, Archimedean solids, Johnson-Zalgaller solids, Archimedean prisms, and antiprisms), Catalan solids, bipyramids and trapezohedra. For example, a Buckminsterfullerene (i.e., a truncated icosahedron) has 3,127,432,220,939,473,920 (approximately 3.13×10^{18})

*A preliminary version was presented at AAAC2018.

†Graduate School of Science and Engineering, Saitama University, {horiyama,miyasaka}@al.ics.saitama-u.ac.jp

‡Faculty of Engineering, Saitama University, sasaki@al.ics.saitama-u.ac.jp

nonisomorphic developments. We here note that the technique in [9] counts the number of nonisomorphic developments without enumerating developments.

If we turn to the developments of polytopes in 4 (or more) dimensions. We can apply the matrix tree theorem to any polytope, and thus we can count the number of the labeled developments. As for the number of the nonisomorphic developments, Gardner asked to enumerate all of the nonisomorphic developments of a 4-dimensional hypercube [6], and Turney enumerated 261 nonisomorphic developments by hands [19]. He also says "As far as I know, the only way is to exhaustively examine the possibilities" in [19]. Later, a technique for counting the number of the nonisomorphic developments of 4-dimensional regular convex polytopes [4] is proposed. The technique is an extension of those for the Platonic solids [7][10], and is further extended to that for any 3-dimensional polyhedron [9]. These techniques avoid explicitly enumerating the developments, but count their numbers by exploiting Polya's counting theorem [17].

As for the enumeration of nonisomorphic developments, an efficient exhaustive search technique using BDDs (Binary Decision Diagrams) is proposed in [8], where a BDD [3] is a succinct data structure that represents a family of sets by a graph. In [8], a method to construct a BDD corresponding to a family of labeled developments is proposed, where each development are represented as a set of labeled edges that form a spanning tree. Then, by omitting mutually isomorphic developments, the nonisomorphic developments are obtained.

Later, a sophisticated technique called a "frontier-based search" [11] is proposed for constructing BDDs/ZDDs representing all constrained subgraphs, and we can adopt this technique to the first step of the method in [8]. A ZDD (Zero-suppressed Binary Decision Diagram) [16] is a variant of BDDs, and also represents a family of sets. The frontier-based search is an extension of Simpath algorithm [13] by Knuth for enumerating all st-paths (i.e., simple paths from vertex s to t) in a given graph. The method can be considered as one of DP-like algorithms, and it constructs the resulting BDDs/ZDDs in a top-down manner. By applying this method to the first step in [8], we can speed-up the construction of the BDD/ZDD representing a family of spanning trees.

Our contribution. In this paper, we focus on the second step of the method in [8], i.e., the isomorphism elimination. More precisely, our problem is as follows: Given a graph G with labeled edges and a family $\mathcal{F}$ of its subgraphs, we extract all automorphisms $\mathrm{Aut}G = \{\pi_1, \pi_2, \dots, \}$ on the given graph, define the lexicographically largest subgraph for each set of the mutually isomorphic subgraphs on each automorphism π_i, and select the lexicographically largest subgraphs on any of the automorphisms. In this paper, both of the given and resulting families of subgraphs are in the form of ZDDs, and the computation are performed on ZDDs. This is because (1) ZDDs can represent a family of sets compactly, (2) the manipulation of ZDDs are faster than the other representations in many cases.

In general, the first step for extracting all automorphisms on a given graph is not tractable: It is still open whether the graph automorphism problem (i.e., the problem deciding whether a given graph has a nontrivial automorphism or not) is in P or in NP-complete [15]. Fortunately, however, we can solve the problem in polynomial time if the degrees of vertices in a graph graph are bounded by a constant [14].

Our main issue is to select the lexicographically largest subgraphs on any of the automorphisms. In [8], BDDs $G_1, G_2, \dots$ are constructed so that G_i represents a family of the lexicographically largest subgraphs on automorphism π_i, and their intersection is taken for selecting a family of subgraphs that appear in all of the families of $G_1, G_2, \dots$ Unfortunately, the method was proposed before the era of the frontier-based search algorithms. Thus, similarly to the BDD/ZDD algorithms in those days, it obtains the resulting BDD by an old-fashioned manner, i.e., by the repetition of so-called "apply operations." In this paper, we renovate this step by introducing the framework of the frontier-based search: We propose algorithms for the top-down construction of the ZDD representing a family of the lexicographically largest subgraphs on π_i.

2 Enumeration by Zero-Suppressed Binary Decision Diagrams

A *zero-suppressed binary decision diagram (ZDD)* [16] is directed acyclic graph that represents a family of sets. As illustrated in Figure 2, it has the unique source node[1], called *the root node*, and has two sink nodes 0 and 1, called *the 0-node* and *the 1-node*, respectively (which are together called the constant nodes). Each of the other nodes is labeled by one of the variables $x_1, x_2, \dots, x_n$, and has exactly two outgoing edges, called *0-edge* and *1-edge*, respectively. On every path from the root node to a constant node in a ZDD, each variable appears at most once in the same order. The size of a ZDD is the number of nodes in it.

Every node v of a ZDD represents a family of sets $\mathcal{F}_v$, defined by the subgraph consisting of those edges and nodes reachable from v. If node v is the 1-node (respectively, 0-node), $\mathcal{F}_v$ equals to $\{\{\}\}$ (respectively, $\{\}$). Otherwise, $\mathcal{F}_v$ is defined as $\mathcal{F}_{0\text{-}succ(v)} \cup \{S \mid S = \{var(v)\} \cup S', S' \in \mathcal{F}_{1\text{-}succ(v)}\}$, where 0-$succ(v)$ and

[1] We distinguish *nodes* of a ZDD from *vertices* of a graph (or a 1-skeleton).

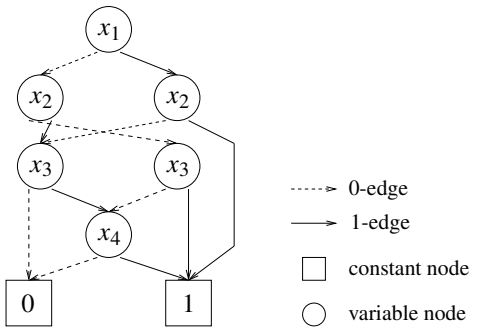


Figure 2: A ZDD representing $\{\{1,2\},\{1,3,4\},\{2,3,4\},\{3\},\{4\}\}$.

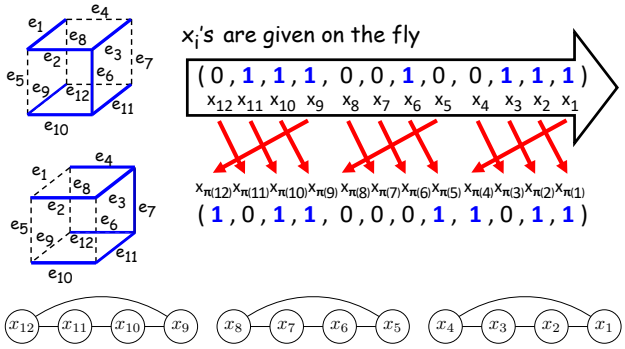


Figure 3: Comparison of x and $\pi(x)$, and propagation graph G_π.

1-$succ(v)$, respectively, denote the nodes pointed by the 0-edge and the 1-edge from node v, and $var(v)$ denotes the label of node v. The family $\mathcal{F}$ of sets represented by a ZDD is the one represented by the root node. Figure 2 is a ZDD representing $\mathcal{F} = \{\{1,2\},\{1,3,4\},\{2,3,4\},\{3\},\{4\}\}$. Each path from the root node to the 1-node, called *1-path*, corresponds to one of the sets in $\mathcal{F}$.

The frontier-based search [11] constructs ZDDs in a top-down manner, and it can be considered as one of DP-like algorithms. We can modify DP algorithms for recognition (i.e., testing whether a given instance satisfies some property) to the frontier-based search algorithm that construct a ZDD representing the family of the yes-instances of the property. Thus, in Section 3, we mainly focus on the method in the form of DP algorithms. The key of the frontier-based search is to share ZDD-nodes by simple "knowledge" of partially given input, and not to traverse the same subproblems more than once. In the context of DP, this means that "internal state" for partially given input should be small. For more details, see [11].

3 Isomorphism Elimination

Let π be a permutation on $\{1,2,\ldots,n\}$, and $\preceq$ be a lexicographical order on $x = (x_n, x_{n-1}, \ldots, x_1) \in \{0,1\}^n$. For any x, we can obtain $\pi(x) = (x_{\pi(n)}, x_{\pi(n-1)}, \ldots, x_{\pi(1)})$, and thus we can define a family $\mathcal{F}_\pi$ of lexicographically larger x's as $\mathcal{F}_\pi = \{x \mid x \succeq \pi(x)\}$. Here, we regard a vector x as a set $\{x_i \mid x_i = 1\}$, which implies that $\mathcal{F}_\pi$ can be regarded as a family of sets $\{x_{i_1}, x_{i_2}, \ldots\}$ $(\subseteq \{x_n, x_{n-1}, \ldots, x_1\})$ that are lexicographically larger than their π-mapped set $\{x_{\pi(i_1)}, x_{\pi(i_2)}, \ldots\}$. Given a set of permutations $\mathrm{Aut}G = \{\pi_1, \pi_2, \ldots\}$, by taking the intersection of $\mathcal{F}_{\pi_1}, \mathcal{F}_{\pi_2}, \ldots$, we can obtain a family of sets each of which is the lexicographically largest on $\mathrm{Aut}G$. By our algorithms described below, we can construct ZDDs of $\mathcal{F}_{\pi_1}, \mathcal{F}_{\pi_2}, \ldots$ in the top-down manner. By combining the top-down construction of the ZDD for spanning trees $\mathcal{F}_s$, we can directly construct the ZDD of their intersection $\mathcal{F}_s \cap \mathcal{F}_{\pi_1} \cap \mathcal{F}_{\pi_2} \cap \cdots$ in the top-down manner [18].

Now, we discuss a DP algorithm for recognizing $\mathcal{F}_\pi$. As illustrated in Figure 3, $x_n, x_{n-1}, \ldots, x_1$ are given on-the-fly. In other words, x_i is given in time slot i $(i = n, n-1, \ldots, 1)$. We compare x and $\pi(x)$, and output 1 if and only if $x \succeq \pi(x)$ holds.

The outline of our algorithm is as follows. The algorithm consists of two phases. In Phase I, $x_n, x_{n-1}, \ldots, x_1$ are given on-the-fly. In the comparison of x and $\pi(x)$, x_i is compared with $x_{\pi(i)}$. In case $i > \pi(i)$, since $x_{\pi(i)}$ will be given in the future, we store x_i in the memory until $x_{\pi(i)}$ is given. On the other hand, in case $i < \pi(i)$, $x_{\pi(i)}$ is already stored in the memory, and thus we can compare x_i and $x_{\pi(i)}$. We transfer the result (denoted as c_i) of the comparison to Phase II. In case $i = \pi(i)$, we compare x_i and $x_{\pi(i)}$, and transfer $c_i :=$ '=' (i.e., equivalent) to Phase II.

In Phase II, the results of the comparisons $C = \{c_n, c_{n-1}, \ldots, c_1\}$ are given from Phase I. Note that the given order of c_i is not $c_n, c_{n-1}, \ldots, c_1$. The order is defined by π. Let π' denote the order of c_i's given to Phase II: c_i's are given in the order of $c_{\pi'(n)}, c_{\pi'(n-1)}, \ldots, c_{\pi'(1)}$. We also note that no c_i may be given in some time slot, and that two c_i and $c_{i'}$ may be given in the same time slot. In Phase II, by checking such c_i's, we conclude whether $x \succeq \pi(x)$ holds or not.

Now, we move to the details of the algorithm. In Phase I, x_i is stored until $x_{\pi(i)}$ appears. At the same time, x_i is required to compare with $x_{\pi^{-1}(i)}$. Thus, precisely speaking, x_i is stored into the memory if $i > \min\{\pi(i), \pi^{-1}(i)\}$ holds, and it is stored until $x_{\min\{\pi(i),\pi^{-1}(i)\}}$ is given. We define *the propagation graph* G_π as the graph $G_\pi = (V, E)$ with $V = \{x_n, x_{n-1}, \ldots, x_1\}$ and $(x_i, x_{\pi(i)}) \in E$. From G_π, we can estimate the memory consumption. (Note that the ordering of the variables is fixed.)

Proposition 3 *To store x_i's in Phase I, w bit is enough, where w is the cut width of the propaga-*

Algorithm 1: Preparation of Phases I and II

Input : n, π
Output: UpdateMemory[ ], cutwidth, Compare[ ]

1 Prepare an empty array until[ ]
2 **for** $i := n, n-1, \ldots, 1$ **do**
3 **if** $i > \min\{\pi(i),\ \pi^{-1}(i)\}$ **then** // It is necessary to store x_i in the memory
4 $k := \begin{cases} \min\{j \mid i \le \text{until}[j]\} & \text{if } \exists j \text{ s.t. } i \le \text{until}[j] \\ (\text{cardinality of until}[\,]) + 1 & \text{otherwise} \end{cases}$
5 until$[k] := \min\{\pi(i),\ \pi^{-1}(i)\}$ // $M[k]$ should be kept until the level of $x_{\pi(i)}$ or $x_{\pi^{-1}(i)}$
6 position$[i] := k$ // x_i is stored in $M[k]$
7 UpdateMemory$[i]$:= UpdateMemory$[i] \cup \{(k, \text{'store'})\}$
8 UpdateMemory[until$[k]$] := UpdateMemory[until$[k]] \cup \{(k, \text{'erase'})\}$
9 cutwidth := cardinality of until[ ]
10 **for** $i := n, n-1, \ldots, 1$ **do**
11 **if** $i > \pi(i)$ **then** // x_i is stored until $x_{\pi(i)}$ is given
12 Compare$[\pi(i)]$:= Compare$[\pi(i)] \cup \{(i, \text{position}[i], \text{'input'})\}$
13 **else if** $i < \pi(i)$ **then** // $x_{\pi(i)}$ is stored until x_i is given
14 Compare$[i]$:= Compare$[i] \cup \{(i, \text{'input'}, \text{position}[\pi(i)])\}$
15 **else** // x_i and $x_{\pi(i)}$ are the same variable
16 Compare$[i]$:= Compare$[i] \cup \{(i, \text{'input'}, \text{'input'})\}$

tion graph G_π with respect to the variable ordering $x_n, x_{n-1}, \ldots, x_1$.

Algorithm 1 summarizes the preparation necessary for Phases I and II. If $i > \min\{\pi(i), \pi^{-1}(i)\}$ holds in Line 3, we plan to store the value of x_i in $M[k]$ and keep $M[k]$ until $x_{\min\{\pi(i),\pi^{-1}(i)\}}$ is given (Lines 4–6). In Line 4, we assign the position k in the first-fit manner. That is, we set the smallest j as k, where $M[j]$ is not used in time slot i. We use a variable-length array until[ ] to indicate that $M[k]$ should be kept until time slot $\min\{\pi(i), \pi^{-1}(i)\}$ (Lines 1 and 5). In Line 4, if no j satisfies $i \le$ until$[j]$, we prepare a new position. In Line 6, position$[i]$ is used to indicate that x_i is in $M[k]$. In Line 7 (respectively, Line 8), we record the plan for storing x_i in $M[k]$ (respectively, erasing $M[k]$). These plans UpdateMemory[ ] are actually executed in Lines 7–11 of Algorithm 2 (Phase I).

The plan for comparing x_i and $x_{\pi(i)}$ is recorded in Lines 10–16, and it is actually executed in Lines 3–6 of Algorithm 2. Compare$[i]$ is a set of the plans for the comparison in time slot i. In case $i > \pi(i)$ (Lines 11 and 12), we keep x_i in $M[\text{position}[i]]$ until $x_{\pi(i)}$ will be given in time slot $\pi(i)$. Thus, we store our plan in Compare$[\pi(i)]$. Plan $(i, \text{position}[i], \text{'input'})$ indicates that c_i (i.e., the comparison of x_i and $x_{\pi(i)}$) can be obtained by comparing $M[\text{position}[i]]$ and 'input' (i.e., $x_{\pi(i)}$) in time slot $\pi(i)$. In case $i < \pi(i)$ (Lines 13 and 14), since we already have $x_{\pi(i)}$ in $M[\text{position}[\pi(i)]]$, we can compare x_i and $x_{\pi(i)}$ in time slot i. Thus, we store our plan in Compare$[i]$. Plan $(i, \text{'input'}, \text{position}[\pi(i)])$ indicates that c_i is obtained by comparing 'input' in time slot i (i.e., x_i) and $M[\text{position}[\pi(i)]]$. Otherwise, since x_i and $x_{\pi(i)}$ are the same variable, we can compare x_i and $x_{\pi(i)}$ in time slot i. We store our plan $(i, \text{'input'}, \text{'input'})$ in Compare$[i]$, where the plan in-

Algorithm 2: Phase I

Input : UpdateMemory[ ], cutwidth, Compare[ ], $x = (x_n, x_{n-1}, \ldots, x_1)$
Output: $(c_n, c_{n-1}, \ldots, c_1)$

1 Prepare an array $M[\,]$ of size cutwidth
2 **for** $i := n, n-1, \ldots, 1$ **do**
3 **foreach** $(i', p_0, p_1) \in$ Compare$[i]$ **do**
4 $m_0 := \begin{cases} x_i & \text{if } p_0 = \text{'input'} \\ M[p_0] & \text{otherwise} \end{cases}$
5 $m_1 := \begin{cases} x_i & \text{if } p_1 = \text{'input'} \\ M[p_1] & \text{otherwise} \end{cases}$
6 $c_{i'} := \begin{cases} \text{'}>\text{'} & \text{if } m_0 > m_1 \\ \text{'}<\text{'} & \text{if } m_0 < m_1 \\ \text{'}=\text{'} & \text{if } m_0 = m_1 \end{cases}$
7 **foreach** $(k, \text{behavior}) \in$ UpdateMemory$[i]$ **do**
8 **if** behavior = *'store'* **then**
9 $M[k] := x_i$ // Store x_i in $M[k]$
10 **else** // In case behavior = 'erase',
11 $M[k] := 0$ // erase $M[k]$

Algorithm 3: Phase II

Input : $(c_n, c_{n-1}, \ldots, c_1)$ and a permutation π'

Output: $\begin{cases} 1: & \text{if } x \succeq \pi(x) \\ 0: & \text{otherwise} \end{cases}$

```
1  (i_s, c_{i_s}) := (∞, '=')        // Set the initial state
2  for j := n, n − 1, ..., 1 do
3  |  i' := π'(j)
4  |  if i' > i_s then
   |  |  // The position of c_{i'} is higher than that of c_{i_s}
5  |  |  if c_{i'} ≠ '=' then
6  |  |  |  (i_s, c_{i_s}) := (i', c_{i'})
7  |  else
   |  |  // The position of c_{i_s} is higher than that of c_{i'}
8  |  |  if c_{i_s} = '=' and c_{i'} ≠ '=' then
9  |  |  |  (i_s, c_{i_s}) := (i', c_{i'})
10 if c_{i_s} = '>' or '=' then
11 |  Output 1                       // x ⪰ π(x) holds
12 else
13 |  Output 0                       // x ⋡ π(x) holds
```

dicates that c_i is obtained by comparing 'input' and 'input' (i.e., both are x_i's) in time slot i.

Algorithm 2 executes the plans in Compare[i] and UpdateMemory[i] in each time slot i. In Line 6, $c_{i'}$ = '>' means $x_{i'} > x_{\pi(i')}$. The notions $c_{i'}$ = '<' and '=' are also defined similarly.

Algorithm 3 describes Phase II. Recall that c_n, c_{n-1}, $\ldots, c_1$ may not be given in this order. For convenience, we introduce permutation π' denoting that c_i's are given in the order of $c_{\pi'(n)}, c_{\pi'(n-1)}, \ldots, c_{\pi'(1)}$. (This ordering is implicitly given by Lines 2 and 3 of Algorithm 2, and thus, it is just for convenience, and we will avoid it later by combining Phases I and II.)

Suppose $i' = \pi'(j)$ as in Line 3 of Algorithm 3. At this moment, we are checking $c_{i'}$. Note that some of the already checked $c_{\pi'(n)}, c_{\pi'(n-1)}, \ldots, c_{\pi'(j)}$ may be in the higher position than $c_{i'}$, and others may be in the lower position than $c_{i'}$. To avoid storing all of them, we use (i_s, c_{i_s}) as an internal state. In case c_{i_s} = '>' (respectively, '<'), all of the already checked c_k's satisfying $k > i_s$ are '=' and already checked c_{i_s} is '>' (respectively, '<'). In this case, if c_k is in the lower position than c_{i_s} (i.e., $k < i_s$), it does not affect the result in the comparison of x and $\pi(x)$. In case c_{i_s} = '=', all of the already checked c_k's are '=', and thus, they do not affect the result in the comparison of x and $\pi(x)$. In Line 1 of Algorithm 3, we set $(i_s, c_{i_s}) := (\infty, '=')$ as an initial state. ($i_s = \infty$ means no c_k's are checked.)

By checking $c_{i'}$, we update (i_s, c_{i_s}): If $c_{i'}$ is in the higher position than c_{i_s} (i.e., $i' > i_s$ holds), $c_{i'}$ is prior to c_{i_s}. Thus, in case $c_{i'}$ is not '=', we store $(i', c_{i'})$ as a new state (Lines 4–6). If c_{i_s} is in the higher position than $c_{i'}$, c_{i_s} is prior to $c_{i'}$. Thus, only in case c_{i_s} is '=' and $c_{i'}$ is not '=', we have a chance to store $(i', c_{i'})$ as a new state (Lines 7–9). After all $c_{i'}$ are checked, we can conclude whether $x \succeq \pi(x)$ holds or not according to the final c_{i_s} (Lines 10–13).

Now, we combine Phases I and II. Line 1 of Algorithm 3 is an initialization of state (i_s, c_{i_s}), and it should be inserted in the beginning of Algorithm 2. Lines 4–9 of Algorithm 3 receive $c_{i'}$, and thus they should be inserted just after Line 6 in the foreach-loop of Algorithm 2. As we mentioned above, we do not need π' since i' in Algorithm 3 is given as the i' in Algorithm 2. Lines 10–13 of Algorithm 3 decide the output according to the final c_{i_s}, and thus they should be inserted just after the last part of Algorithm 2. Given $x_n, x_{n-1}, \ldots, x_1$ on-the-fly, by Algorithm 1 and Algorithm 2+3 (i.e., combined version of Algorithms 2 and 3), we can conclude whether $x \succeq \pi(x)$ holds or not.

In the frontier-based search, we construct ZDDs in a top-down manner. Each node of the resulting ZDD has its internal state (i_s, c_{i_s}) and $M[\,]$. The root node of the resulting ZDD is prepared with $(i_s, c_{i_s}) := (\infty, '=')$. We do not care about $M[\,]$ since we are not given any input x_i. The label of the root node is x_n, which indicates that we are checking x_n. For each i in $\{n, n-1, \ldots, 1\}$ and for each node v labeled x_i, we try both of the cases $x_i = 0$ and 1. In case $x_i = 0$, from node v, we prepare node 0-$succ(v)$. The internal state of 0-$succ(v)$ can be obtained by applying Lines 3–11 of Algorithm 2 (combined with Lines 4–9 of Algorithm 3) to the state of v. We can perform similarly in case $x_i = 1$. If two nodes have the same label x_i and the same internal state, by following the definition of ZDD, we merge the two nodes. From this observation, we can estimate the upper bound on the size of the resulting ZDD. Furthermore, since the execution of Lines 3–11 of Algorithm 2 and Lines 4–9 of Algorithm 3 for each node can be done in constant time, we can also evaluate the time complexity.

Theorem 4 *The size of the resulting ZDD is $O(n^2 2^w)$, where w is the cut width of the propagation graph G_π with respect to the variable ordering $x_n, x_{n-1}, \ldots, x_1$. The computation time for the construction is proportional to the size of the resulting ZDD.*

4 Experimental Results

Experimental results are given in Tables 1 and 2. The computation time is measured on Intel(R) Xeon(R) E7-2830 2.13GHz, 2TB Memory, Red Hat Enterprise Linux Server release 6.6.

In table 1, the developments of 5 Platonic solids and 5 out of 13 Archimedean solids (a cuboctahedron, a truncatedtetrahedron, a truncatedoctahedron, a truncatedcube, and a rhombicuboctahedron) are enumer-

Table 1: Summary of the results for Platonic and Archimedian solids.

Polyhedron	$\|E\|$	|Aut|	#(Labeled Developments)	#Developments	Computation Time (s)		Required Memory (MB)	
					Conventional	Proposed	Conventional	Proposed
Tetrahedron	6	24	16	1	0.01	0.00	30	2
Cube	12	48	384	11	0.02	0.01	30	2
Octahedron	12	48	384	11	0.02	0.01	30	2
Dodecahedron	30	120	5,184,000	43,380	9.10	0.54	529	5
Icosahedron	30	120	5,184,000	43,380	5.73	0.51	282	10
Cuboctahedron	24	48	331,776	6,912	0.35	0.06	36	3
Truncatedtetrahedron	18	24	6,000	261	0.03	0.01	30	2
Truncatedoctahedron	36	48	101,154,816	2,108,512	75.59	2.67	11,192	23
Truncatedcube	36	48	32,400,000	675,585	133.63	2.10	2,078	35
Rhombicuboctahedron	48	48	301,056,000,000	6,272,012,000	$>$ 3 H	1,913.97		11,182

Table 2: Summary of the results for d-dimensional hypercubes.

d	$\|E\|$	|Aut|	#(Labeled Developments)	#Developments	Computation Time (s)		Required Memory (MB)	
					Conventional	Proposed	Conventional	Proposed
2	4	8	4	1	0.02	0.00	36	2
3	12	48	384	11	0.10	0.01	36	2
4	24	384	82,944	261	3.00	0.09	150	2
5	40	3,840	32,768,000	9,694	1166.52	3.96	36,036	10
6	60	46,080	20,736,000,000	502,110	$>$ 3 H	478.39	$>$ 140,000	208

ated. The second column $|E|$ in Table 1 gives the number of edges in the 1-skeleton of a polyhedron. The third column |Aut| gives the number of automorphisms of a polyhedron. The fourth and fifth columns give the number of labeled and nonisomorphic (i.e., unlabeled) developments, respectively. For example, as for a rhombicuboctahedron, we have 301,056,000,000 labeled developments. By checking the graph isomorphism for all of these labeled developments among 48 automorphisms, we obtained 6,272,012,000 nonisomorphic developments. The size of the required memory is summarized in the eighth and ninth column.

As the conventional method, we used the algorithm in [8] combined with the frontier-based search [11] for enumerating labeled developments. The difference between the conventional and our proposed methods is as follows: The algorithm in [8] was proposed before the era of the frontier-based search algorithms. Thus it is necessary to construct the ZDDs of $\mathcal{F}_{\pi_1}, \mathcal{F}_{\pi_2}, \ldots$ completely and then make an intersection of the ZDDs. On the other hand, in our proposed method, we can directly construct the ZDD of the intersection without constructing the intermediate ZDDs of $\mathcal{F}_{\pi_1}, \mathcal{F}_{\pi_2}, \ldots$. The proposed method requires less memory than the conventional method in many cases.

In table 2, the developments of d-dimensional hypercubes are enumerated. Similarly to the case of taking dual of a 3-dimensional polyhedron, we prepare the facet-adjacency graph whose vertices and edges corresponds to the $(d-1)$-dimensional hypercubes and their adjacency of the original hypercube. The facet-adjacency graph of d-dimensional hypercube is a complete d-partite graph with $2d$ vertices and $4\binom{d}{2}$ edges. The automorphism Aut has $2^d\binom{d}{2}$ permutations. Table 2 tells that the proposed method is more than 300 times faster and 3,000 times less memory than the conventional method in case $d = 5$. As for the case $d \geq 6$, we believe the speed-up ratio is more than 300.

5 Conclusion

We have address the issue of the isomorphism elimination by proposing the top-down construction method for the ZDDs of lexicographically largest instances. Experimental results show that the proposed method is more than 300 times faster and 3,000 times less memory than the conventional method in the best case. Our algorithms are applicable to many other enumeration problems with eliminating isomorphic instances.

References

[1] T. J. N. Brown, R. B. Mallion, P. Pollak, B. R. M. de Castro, J. A. N. F. Gomes, The number of spanning trees in buckminsterfullerene, *Journal of Computational Chemistry*, vol. 12, pp. 1118–1124, 1991.

[2] T. J. N. Brown, R. B. Mallion, P. Pollak, A. Roth, Some Methods for Counting the Spanning Trees in Labelled Molecular Graphs, examined in Relation to Certain Fullerenes, *Discrete Applied Mathematics*, vol. 67, pp. 51–66, 1996.

[3] R. E. Bryant, Graph-based algorithms for Boolean function manipulation, *IEEE Transactions on Computers*, vol. C-35, pp. 677–691 (1986).

[4] F. Buekenhout, M. Parker, The Number of Nets of the Regular Convex Polytopes in Dimension ≤ 4, *Discrete Mathematics*, vol. 186, pp. 69–94, 1998.

[5] E. D. Demaine, J. ORourk, Geometric Folding Algorithms: Linkages, Origami, Polyhedra, Cambridge University Press (2007).

[6] M. Gardner, Mathematical Games: Is It Possible to Visualize a Four-Dimensional Figure?, *Scientific American*, 214, pp. 138–143, 1966.

[7] C. Hippenmeyer, Die Anzahl der inkongruenten ebenen Netze eines regulären Ikosaeders, *Elemente der Mathematik*, vol. 34, pp. 61–63, 1979.

[8] T. Horiyama and W. Shoji, Edge Unfoldings of Platonic Solids Never Overlap, In *Proc. of the 23rd Canadian Conference on Computational Geometry (CCCG 2011)*, pp. 65–70, 2011.

[9] T. Horiyama and W. Shoji, The Number of Different Unfoldings of Polyhedra, In *Proc. of the 24th International Symposium on Algorithms and Computation (ISAAC 2013)*, *Lecture Notes in Computer Science*, 8283, pp. 623–633, Springer-Verlag, 2013.

[10] M. Jeger, Über die Anzahl der inkongruenten ebenen Netze des Würfels und des regulären Oktaeders, *Elemente der Mathematik*, vol. 30, pp. 73–83, 1975.

[11] J. Kawahara, T. Inoue, H. Iwashita, and S. Minato. Frontier-based Search for Enumerating All Constrained Subgraphs with Compressed Representation. IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol. E100-A, no. 9, pp. 1773–1784, 2017.

[12] G. Kirchhoff, Über die Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Verteilung galvanischer Ströme gefuhrt wird, Annalen der Physik und Chemie, 72, pp. 497–508, 1847.

[13] D. E. Knuth, The Art of Computer Programming, vol. 4, fascicle 1, Bitwise Tricks & Techniques, Binary Decision Diagrams, Addison-Wesley (2009).

[14] E. M. Luks, Isomorphism of graphs of bounded valence can be tested in polynomial time, Journal of Computer and System Sciences, 25 (1), pp. 42–65, 1982.

[15] A. Lubiw, Some NP-complete problems similar to graph isomorphism, SIAM Journal on Computing, 10 (1), pp. 11–21, 1981.

[16] S. Minato. Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems. In *Proc. of the 30th ACM/IEEE Design Automation Conference (DAC'93)*, pp. 272–277, 1993.

[17] G. Pólya, Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen, Acta Mathematica, 68 (1), pp. 145–254, 1937.

[18] TdZdd: A top-down/breadth-first decision diagram manipulation framework, `https://github.com/kunisura/TdZdd`

[19] P. D. Turney, Unfolding the Tesseract, *Journal of Recreational Mathematics*, 17 (1), pp. 1–16, 1984–85.

On error representation in exact-decisions number types

Martin Wilhelm*

Abstract

Accuracy-driven computation is a strategy widely used in exact-decisions number types for robust geometric algorithms. This work provides an overview on the usage of error bounds in accuracy-driven computation, compares different approaches on the representation and computation of these error bounds and points out some caveats. The stated claims are supported by experiments.

1 Introduction

In Computational Geometry, many algorithms rely on the correctness of geometric predicates, such as orientation tests or incircle tests. In contrast to various other areas of computing, a small error in computation often does not imply a small error in the result. Instead, algorithms may fail, produce drastically wrong output, or no output at all if a predicate returns the wrong result [8].

To mitigate the consequences of this problem, exact-decisions number types have been developed, based on the concept of accuracy-driven-computation and the Exact Geometric Computation Paradigm [12]. Examples for such number types are `leda::real` from the LEDA library [2], `Core::Expr` [3, 14] and `Real_algebraic` [4]. They store the expressions involved in directed acyclic graphs, called expression dags, and maintain approximations and error bounds for each subexpression. When a decision has to be made, the accuracy of the subexpressions is increased until the value can be separated from zero or value can be guaranteed to be zero through a separation bound [1]. This concept will be explained in slightly more detail in Section 2.1.

Approximations are usually stored in arbitrary-precision floating-point number types, or short, *bigfloats*. In both `leda::real` and `Real_algebraic`, error bounds are stored in a bigfloat as well, in form of an absolute error bound. In `Core::Expr`, error bounds are stored as a combination of upper and lower bound for the most significant bit of the value.

In this paper we will show advantages and disadvantages of different error bound representation forms for accuracy-driven computation. In Section 2, various possibilities are evaluated and sensible choices for an implementation are proposed. We present an experimental comparison for these choices in Section 3 based on the number type `Real_algebraic`.

*Institut für Simulation und Graphik, Otto-von-Guericke-Universität Magdeburg, martin.wilhelm@ovgu.de

2 Error Representation

There are more aspects to the representation of an error bound than might be apparent at first glance. One natural way to represent an error is by storing a value e_{dir}, such that $\hat{x} - e_{dir} \leq x \leq \hat{x} + e_{dir}$, if x represents the real value of the expression and $\hat{x}$ its approximation. We call this the *direct* error representation.

Since an error bound can get quite small, a direct representation must be stored in a non-primitive data type, most commonly in a bigfloat. Computations involving bigfloats are expensive. Another natural way of storing an error bound is to store an exponent e_{log}, such that $\hat{x} - 2^{e_{log}} \leq x \leq \hat{x} + 2^{e_{log}}$. It then suffices to store e_{log} as a primitive integer data type, such as `long`. We call this the *logarithmic* representation.

The obvious advantage of a direct over a logarithmic value is that the direct representation can be much more precise than the logarithmic one. If, for example, a large sum is computed with error e at the m operands, error propagation with a direct representation leads to an error bound of me (assuming exact operations). A logarithmic integer bound, on the other hand, increases by at least 1 for each addition. So error propagation leads to an error bound of $2^{\log(e)+m}$ if the additions are processed sequentially.

To keep the bound small while avoiding bigfloats, a third approach on error representation would be to store a logarithmic error bound in a floating-point primitive, such as `double`. While not as precise as the direct representation, the error bound can be increased in smaller steps, avoiding an exponential increase as in the previous example. In contrast to the error representation as an integer value, it might also enable more elaborate evaluation strategies, such as proposed by van der Hoeven [9].

There are various other ways to represent an error. While all three methods above store the radius of an error interval, the interval can be stored directly through an upper and lower bound, such as in `Core::Expr`. Furthermore one can imagine various combined approaches, where the representation can change based on the size of the error. In this paper, however, we will focus on the three variants proposed above. We also only consider absolute error bounds, although there are good

reasons to compute the bound relative to the size of the approximation or even combine an absolute and a relative error [7].

2.1 Errors in accuracy-driven computation

We introduced three different ways of representing an error bound in a number type. The complexity rises if we consider different combinations of the approaches during the computation process. The usefulness of error bound representations may change depending on the current task. So it could be advantageous to switch between different representations during the computation.

In this section we will shortly describe the concept of accuracy-driven computation as implemented in `Real_algebraic` and the usage of error bounds within. Accuracy-driven computation was first introduced by Yap and Dubé under the name "precision-driven computation" [13]. It describes a lazy approach on exact-decisions computation. Computations are not done directly on invocation, but stored in an *arithmetic expression dag*, i.e., a rooted ordered directed acyclic graph whose nodes are either a floating-point number or an operator with its operands as children.

Once a decision has to be made, the computation is started. Since every decision can be translated to the decision, whether a value is positive, negative or zero, it is sufficient to determine the sign of the root node. This can be done by using a separation bound, i.e., a number $\mathrm{sep}(E)$ for an expression E, such that $|\operatorname{value}(E)| > 0 \Rightarrow |\operatorname{value}(E)| > \mathrm{sep}(E)$. If $\mathrm{sep}(E)$ is not part of the error interval of an approximation, then the sign of the approximation is correct.

The strategy in accuracy-driven computation is to start with a desired accuracy q at a (root) node, compute the accuracy needed at its children to guarantee a respective error bound, and recurse on the children with the new desired accuracies. To determine whether an expression E is zero, an accuracy of $q_{\max} = \lfloor \log(\mathrm{sep}(E)) \rfloor$ is needed. If $\operatorname{value}(E) > 0$, however, a separation is usually possible with a much larger error as soon as zero is not contained anymore in the error interval. So usually q is chosen to be a small negative number[1] in the beginning, with an exponential increase until $|q| > |q_{\max}|$.

Before the top-down computation is started, an initial value for each node must be computed bottom-up, i.e., with a small fixed precision. This is necessary, since sometimes an estimate for the value of a child node is needed to compute the required accuracy. During this process, an accuracy-driven-computation for a subexpression may be invoked, if the value of a divisor or the operand of a root needs to be separated from zero.

Error bounds mainly occur in three different places:

1. In each node a current error bound is stored to prevent recomputations of approximations that are already sufficiently accurate.
2. A desired error bound is computed and propagated top-down through the expression dag.
3. An initial error bound for each node is computed bottom-up.

In `Real_algebraic`, a direct error bound is used in the first and the third case, while the top-down propagation is done with a logarithmic integer error bound. The number type `leda::real` uses direct error bounds in all three cases, whereas `Core::Expr` uses a logarithmic integer bound in the second and a logarithmic integer interval, i.e., a combination of upper and lower bound in the third case. Both representations are saved inside each node.

The initial bottom-up computation is done with small precision. Bigfloat operations are less expensive then, whereas the influence of a weak error bound is increased. A direct error bound is therefore a sensible choice for this part of the algorithm. In contrast, the main accuracy-driven parts of the computation require high precisions, causing the maintenance of a direct error bound to be too expensive compared to its benefits [6].

2.2 Switching between direct and logarithmic bound

When parts of the algorithm are computed with differing error representations, the algorithm must switch between those representations. Since floating-point data types are stored as a mantissa and an exponent, computing a direct bound e_{dir} from a logarithmic bound e_{log} can be done fast and without loss of precision by setting the exponent of e_{dir} to e_{log}, i.e., $e_{dir} = \Phi(e_{log}) := 2^{e_{log}}$. For the reverse process, e_{log} must be computed as $e_{log} = \hat{\Phi}(e_{dir}) := \lceil \log(e_{dir}) \rceil$, losing some precision in the progress. In particular, it cannot be expected that $e_{dir} = \Phi(\hat{\Phi}(e_{dir}))$. However, we should expect $e_{log} = \hat{\Phi}(\Phi(e_{log}))$ to be true.

As previously mentioned, computing Φ is cheap. What about $\hat{\Phi}$? The mantissa m and exponent b of a floating-point value x are usually chosen, such that $m \in [0.5, 1)$, $b \in \mathbb{Z}$ and $x = m2^b$. So it seems natural to choose $\hat{\Phi}(x) = b$ as a cheap conversion function, as done in `Real_algebraic`.

However, there is a significant drawback to this approach. If x is a power of two, $\hat{\Phi}$ overestimates $\lceil \log(x) \rceil$ by one, since then $m = 0.5$. While this does not affect overall correctness of the algorithm and the case seems very special, implementing $\hat{\Phi}$ like that can lead to massive drops in performance, since then $e_{log} \neq \hat{\Phi}(\Phi(e_{log}))$ for every value of e_{log}.

After an error bound is guaranteed by the accuracy-driven computation, the stored error is set to this error bound. If later on the same error bound is needed for

[1] We will refer to accuracies as "small" if their absolute value is small, although they are usually negative numbers. We fear that the opposite notion would be even more misleading.

the respective node, the algorithm checks whether a recomputation is necessary. If the representations of the stored bound and the requested bound differ, as present in `Real_algebraic`, the check fails and the computation needs to be executed again.

A drastic example for this effect arises if a power is computed through repeated squaring. Each node gets recomputed along each of the 2^n paths from the root to the leaf, leading to an exponential increase in running time (cf. Figure 1). For $x = \sqrt{13} + \sqrt{17}$ and $n = 15$ operations, `Real_algebraic` takes 87.84 seconds with the "inexact" implementation to evaluate the expression up to an accuracy of $q = 50000$, compared to about 0.01 seconds with an "exact" one[2,3]. While in this example the problem can be avoided by switching to a topological evaluation order (see Mörig et al. [5]), it persists if checks need to be repeated in the main algorithm.

Figure 1: The arithmetic expression dag resulting from computing x^{2^n} through repeated squaring. There are 2^n different paths from the topmost multiplication to the common leaf x. If recomputation checks fail due to conversion errors, the evaluation time increases drastically.

If the conversion should be exact, considerable effort needs to be done. We need to check, whether the value of the mantissa is exactly 0.5 and, if so, decrease the result by one. When using `mpfr` bigfloats for example, this changes the one-liner

```
mpfr_exp_t ceil_log2(const mpfr_t& a){
  return mpfr_get_exp(a);
}
```

to the more elaborate method

```
mpfr_exp_t ceil_log2(const mpfr_t& a){
  mpfr_exp_t e = mpfr_get_exp(a);
  mpfr_t rop; mpfr_init(rop);
  mpfr_div_2si(rop,a,e,MPFR_RNDA);
  if (mpfr_cmp_d(rop,0.5) == 0) --e;
  mpfr_clear(rop);
  return e;
}
```

In the second method, the mantissa of the bigfloat is accessed, which can potentially be large. For large computations it can be expected to be significantly slower than the first method. If the undesirable effects described above should be avoided, it may therefore prove more efficient to avoid transformations between different error representations entirely.

[2]We use the term "exact" in this context to express that the method correctly implements the function $\hat{\Phi}(e_{dir}) := \lceil \log(e_{dir}) \rceil$.

[3]Specifications on the test configuration can be found at the beginning of Section 3.

Note that even then the conversion function is widely used during the computation. Approximations of the value of a (sub)expression must be stored in bigfloats. In accuracy-driven computation, often a bound for the magnitude of the result is needed, which is then computed by the above function. Since, in contrast to error transformation, a worse bound for the magnitude only causes a small difference in performance, it is reasonable to use the inexact, but fast transformation method in those cases.

2.3 Logarithmic floating-point error bounds

In the previous section, difficulties are pointed out that may arise when switching between a direct and a logarithmic error representation. In contrast, switching between a floating-point representation and an integer representation for logarithmic errors is cheap and natural. This raises the question whether the fixed precision computations can be done more efficiently through a logarithmic floating-point bound.

Most of the computations involved in error propagation with a fixed precision can be broken down to the sum of two or three errors. So with a logarithmic bound we have to find a value c for two error representations a and b, such that $2^c \geq 2^a + 2^b$. With an integer value the error bound doubles with each such summation, since this is the smallest step in which the bound can be increased. So we have $c = \max(a, b) + 1$.

If a, b, c are floating-point values, we may find a better bound by setting $c = \max(a, b) + \log(1 + 2^{-|a-b|})$, where we have to make sure that each operation is rounded up (towards infinity). With repeated additions, the floating-point error bound increases much more slowly than the integer bound, although computing the logarithm in each step makes it also a lot more expensive.

Can the error propagation during accuracy-driven computation benefit from more precision in the exponent? Surprisingly, the answer is no, at least not directly. When deciding which accuracy is needed at the child nodes to guarantee a certain accuracy at the parent node, up to three error terms need to be balanced. Besides the desired accuracies of the one or two children, the precision at which the operation at the parent node is computed must be chosen. A higher accuracy at the child nodes can then be used to reduce the precision needed for the bigfloat operation.

This decision is done locally, i.e., the parent node does neither know of what size, nor of which form its subtrees are. Without this information it cannot be decided to what extent the precision of the operation needs to be increased in order to require a smaller overall accuracy. So if the decision should be made locally, the gain from switching to a floating-point exponent is marginal and will probably not cover the additional costs associated with it.

Nevertheless, benefitting may be possible if a global error propagation strategy is implemented. The overall accuracy needed could then be kept small through balancing of error terms, which in turn has the potential to drastically improve the performance of the number type, especially for unbalanced expression dags [9, 10].

2.4 Errors and separation bounds

During the accuracy-driven part of the computation, it is beneficial to convert subgraphs to a single bigfloat node if their approximation is already exact, i.e., if their error is zero. This is especially useful if the value of a subgraph is found to be zero. If after a computation the approximation of a node is close to zero, `Real_algebraic` computes a separation bound for this node and checks whether the true value can be declared zero. An approximation is considered close to zero if zero is part of its error interval.

If the error bound is bad, this check happens (and fails) more often. This can have significant consequences for the performance, since for computing the separation bound of a subexpression, the whole subtree must be traversed. Existing bounds for the child nodes cannot be used, since they may misrepresent the algebraic degree of the expression if common subexpressions exist (cf. Figure 2) and a higher algebraic degree drastically worsens the separation bound.

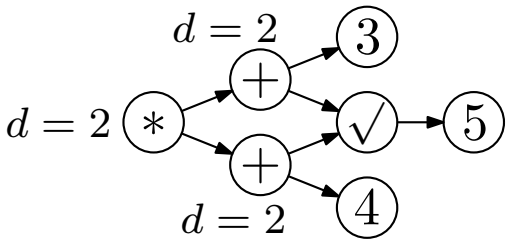


Figure 2: The two children of the root node share a common subexpression with a square root operation. Although both subexpression at the child nodes have algebraic degree two, the algebraic degree of the full expression is still only two.

In addition, the separation bound cannot be assumed to stay the same during the whole computation, since subgraphs may be converted to bigfloat nodes. While a previous separation bound stays valid after such a conversion, in some cases a much better bound can be computed, e.g. if roots can be eliminated. So it is advisable to keep the bound flexible.

To solve this problem, once computed separation bounds can be cached together with a global timestamp. Whenever a bigfloat conversion happens, all previous timestamps get invalidated. The advantage of this method is an easy implementation without much overhead, leading to good results, if the usage of the number type is limited to few large expressions. However, if many different usages of the number type exist at the same time, a separation bound might get invalidated by a bigfloat conversion in a completely disjoint expression dag. An alternative approach based on topological evaluation which leads to similar results can be found in [11].

3 Experimental Results

The experiments are performed on an Intel Core i5 660, 8GB RAM, under Ubuntu 17.10. For `Real_algebraic` we use Boost interval arithmetic as floating-point-filter and MPFR bigfloats for the bigfloat arithmetic. The code is compiled using g++ 7.2.0 with C++11 on optimization level `O3` and linked against Boost 1.62.0 and MPFR 3.1.0. Test results are averaged over 25 runs each. The variance for each data point is negligible.

We compare three main strategies with different representations for (a) storing, (b) error propagation in accuracy-driven computation and (c) fixed precision evaluation (cf. Section 2.1):

1. The default strategy of `Real_algebraic`, i.e., direct error representation for (a) and (c) and logarithmic integer representation for (b), named `def`.

2. Logarithmic integer representation for all three parts (a), (b) and (c), named `lgi`.

3. Logarithmic floating-point representation for (a) and (c), logarithmic integer representation for (b), named `lgd`.

Note, that in each case we use a logarithmic integer representation for (b). It has already been shown that direct error bounds are very expensive compared to the additional benefit [6]. As described in Section 2.3, advanced strategies would be needed for error propagation to benefit from a logarithmic floating-point representation. While without such strategies, no interesting results are to be expected, implementing them heavily reduces comparability to the other approaches presented in this paper. Therefore we leave it aside for future work.

3.1 Comparison of separation bound strategies

First, we show the effects of the interactions between error bound representation and the computation of separation bounds, as described in Section 2.4. We test for the equality

$$F_n = \frac{\phi^n - \bar{\phi}^n}{\sqrt{5}},$$

where F_n represents the n-th Fibonacci number and $\phi = 1 - \bar{\phi} = \frac{1+\sqrt{5}}{2}$. We compute both sides of the equation in a simple loop as in the code below.

```
template <class NT>
bool fibonacci_test(const int n){
  NT sqrt5  = sqrt(NT(5));
  NT phi    = (NT(1) + sqrt5) / NT(2);
  NT phibar = (NT(1) - sqrt5) / NT(2);

  NT phiN = phi; NT phibarN = phibar;
  NT fib0 = 0; NT fib1 = 1; NT tmp;

  for(int i = 1; i < n; i++){
    tmp = fib1; fib1 += fib0; fib0 = tmp;
    phiN *= phi; phibarN *= phibar;
  }

  NT res = NT(1)/sqrt5 * (phiN-phibarN);
  return fib1 == res;
}
```

We show running times for each of the three aforementioned strategies with and without a cached separation bound computation (indicated by an additional `s`).

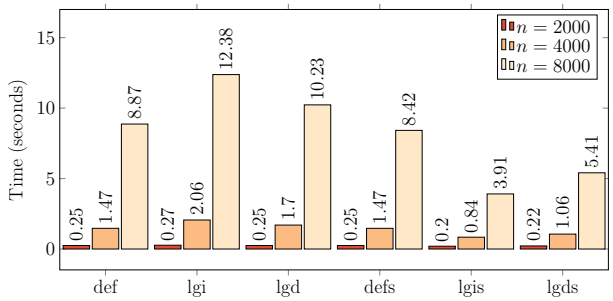


Figure 3: Results for `fibonacci_test` with different strategies. A logarithmic error bound increases the running time due to repeated computation of separation bounds. If the separation bounds are cached, the running time can be significantly decreased.

The results are shown in Figure 3. Representing the error logarithmically has a negative impact on the performance due to a heavy increase in separation bound computations. For $n = 8000$, a separation bound computation is started 1149 times with `def` compared to 29092 and 13939 times with `lgi` and `lgd`. When the cost for those computations can be reduced due to caching, the logarithmic error bounds outperform the direct representation. Note that the logarithmic floating-point representation ranges in between the other two representations in each scenario, which underlines its character of slightly more precision for slightly more overhead compared to a logarithmic integer representation.

3.2 Geometric problems

In the previous section a first impression of the behavior of the three different representations is obtained. In this section we test this impression against more realistic geometric problems. For this, we recreate several experiments from Mörig et al. [4].

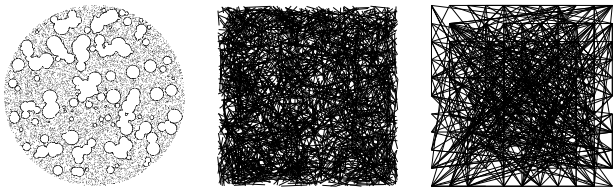


Figure 4: Examples from the test data for the delaunay triangulation and intersection points computation. Three types of data are shown: A point cloud with 50% of its points on a union of disks (left), a random collection of short segments (middle) and a random collection of segments with endpoints on a grid (right).

We compare the three strategies introduced at the beginning of Section 3. In none of the experiments, a timestamped separation bound computation significantly worsened the performance. So we show only results with a timestamped computation enabled. Instead we show the impact of an exact implementation of `ceil_log2` (cf. Section 2.2), indicated by an `x`.

We first compute the delaunay triangulation of a set of 20000 points, of which between 50% and 100% lie on a union of disks with no points inside (cf. Figure 4).

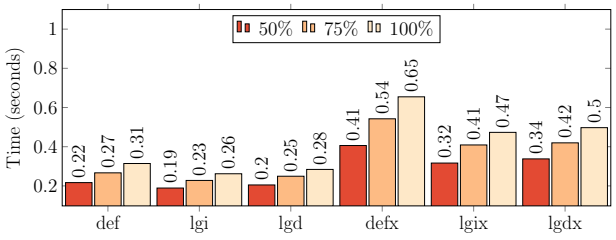


Figure 5: Experimental results for the computation of a delaunay triangulation on 20000 random points with different percentages of them distributed along the boundary of a union of disks and none inside. More degeneracies lead to more performance gain through a logarithmic error bound. An exact computation of `ceil_log2` causes an increase in running time in all cases.

The results are shown in Figure 5. The data shows a small performance gain by switching from a direct error representation to a logarithmic one. Using exact transformations between bigfloats and integer types shows to be very expensive in all cases, but even more so in the case of direct error representation. In `lgi` and `lgd` errors are not transformed through this method. Still, both experience an increase in running time, since the computation of the magnitude of an *approximation* is affected by this change as well (cf. Section 2.2). It can be reasoned that for `lgi` and `lgd` the inexact transformation can be used, since the problems from Section 2.2

are solved by design. While not fully comparable, the difference between `defx` and `lgi` is worth noting.

In Figure 6 the results for the computation of intersection points on different test data are shown. The test data consists of long or short random segments, segments with endpoints on a grid and segments which are parallel to the axes (cf. Figure 4). The intersection tests are performed with homogenous coordinates. Results for Cartesian coordinates are not shown, but look similar. The same number of segments as in Mörig et al. are used in the four scenarios to make the data somewhat comparable.

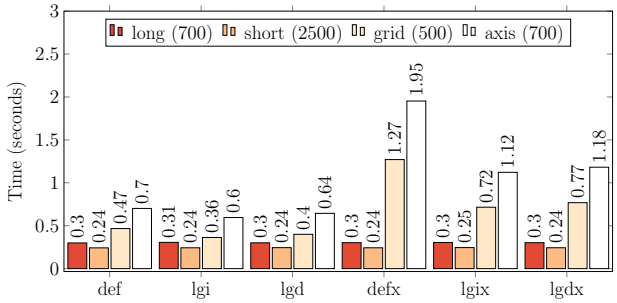


Figure 6: Results for computing the intersection points of segments in various constellations. Neither the form of error representation, nor the exactness of the `ceil_log2` operation have an influence on the running time for random long or short segments. If segments are placed on a grid or parallel to the axes, a logarithmic error bound leads to better results. Whether the `ceil_log2` operation is performed exactly has significant influence on the performance in these cases.

For long or short random segments, no difference between the three forms of error representation is apparent. Because of the random distribution of the segments, the data sets have almost no degeneracies. Therefore almost all relevant signs can be decided through a floating-point filter, without invoking the accuracy-driven computation in the first place.

For segments on a grid or axis-parallel segments, significant differences are present between different error representations, especially if the transformation between representations is exact. In these cases lots of degeneracies occur, causing the computation to switch to accuracy-driven computation more often.

Summarizing the results, in each geometric experiment the logarithmic representation performed at least as well as the direct representation, while avoiding possible drawbacks due to a faulty error conversion. The more degeneracies occur, the larger is the gain compared to other strategies. Switching to an exact implementation of `ceil_log2` is too expensive to be a reasonable alternative for solving the problems resulting from an inexact transformation.

4 Conclusion

Choosing a representation model for error bounds in accuracy-driven computation means balancing a tradeoff between the quality of the bound and the efficiency of its computation. Our results suggest that in most cases it is not worth the effort to compute a better error bound to increase the overall efficiency of the number type. A simple logarithmic integer bound outperforms the direct error bound representation as well as the logarithmic floating-point bound in all cases if accompanied by some cautionary mechanisms regarding the separation bound computation.

Mixed forms of error representation suffer from problems arising during the transformation between the representation or, respectively, from the transformation overhead. In conclusion, we suggest using logarithmic integer error bounds by default in exact-decisions number types based on accuracy-driven computation.

5 Future work

The additional precision gained through the usage of a logarithmic floating-point representation proved to be not sufficient to compensate for the additional cost associated with its operations in the fixed-precision initialization phase of the algorithm. For accuracy-driven computation it seems promising to combine this approach with a global error propagation strategy as described in Section 2.3. To a lesser extent, global strategies may also be used to improve the error bound computed with integer exponents.

References

[1] C. Burnikel, S. Funke, K. Mehlhorn, S. Schirra, and S. Schmitt. A separation bound for real algebraic expressions. *Algorithmica*, 55(1):14–28, 2009.

[2] C. Burnikel, K. Mehlhorn, and S. Schirra. The leda class real number. Report MPI-I-1996-1-001, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1996.

[3] V. Karamcheti, C. Li, I. Pechtchanski, and C. Yap. A core library for robust numeric and geometric computation. In *Proceedings of the Fifteenth Annual Symposium on Computational Geometry, Miami Beach, Florida, USA, June 13-16, 1999*, pages 351–359, 1999.

[4] M. Mörig, I. Rössling, and S. Schirra. On design and implementation of a generic number type for real algebraic number computations based on expression dags. *Mathematics in Computer Science*, 4(4):539–556, 2010.

[5] M. Mörig and S. Schirra. Precision-driven computation in the evaluation of expression-dags with common subexpressions: Problems and solutions. In *6th International Conference on Mathematical Aspects of Computer and Information Sciences, MACIS*, pages 451–465, 2015.

[6] M. Mörig. *Algorithm Engineering for Expression Dag Based Number Types*. Dissertation, Otto-von-Guericke Universität Magdeburg, 2015.

[7] K. Ouchi. Real/expr: Implementation of an exact computation package. *Master's thesis, New York University, Department of Computer Science, Courant Institute*, 1997.

[8] S. Schirra. Robustness and precision issues in geometric computation. In *Handbook of Computational Geometry*, pages 597–632. Elsevier, 2000.

[9] J. van der Hoeven. Computations with effective real numbers. *Theor. Comput. Sci.*, 351(1):52–60, 2006.

[10] M. Wilhelm. Balancing expression dags for more efficient lazy adaptive evaluation. In *Mathematical Aspects of Computer and Information Sciences - 7th International Conference, MACIS 2017, Vienna, Austria, November 15-17, 2017, Proceedings*, pages 19–33, 2017.

[11] M. Wilhelm. Multithreading for the expression-dag-based number type Real_algebraic. Technical Report FIN-001-2018, Otto-von-Guericke-Universität Magdeburg, 2018.

[12] C. Yap. Towards exact geometric computation. *Comput. Geom.*, 7:3–23, 1997.

[13] C. Yap and T. Dubé. The exact computation paradigm. In *Computing in Euclidean Geometry*, pages 452–492. World Scientific, 1995.

[14] J. Yu, C. Yap, Z. Du, S. Pion, and H. Brönnimann. The design of core 2: A library for exact numeric computation in geometry and algebra. In *Proceedings of the Third International Congress on Mathematical Software, ICMS*, pages 121–141, 2010.

Author Index