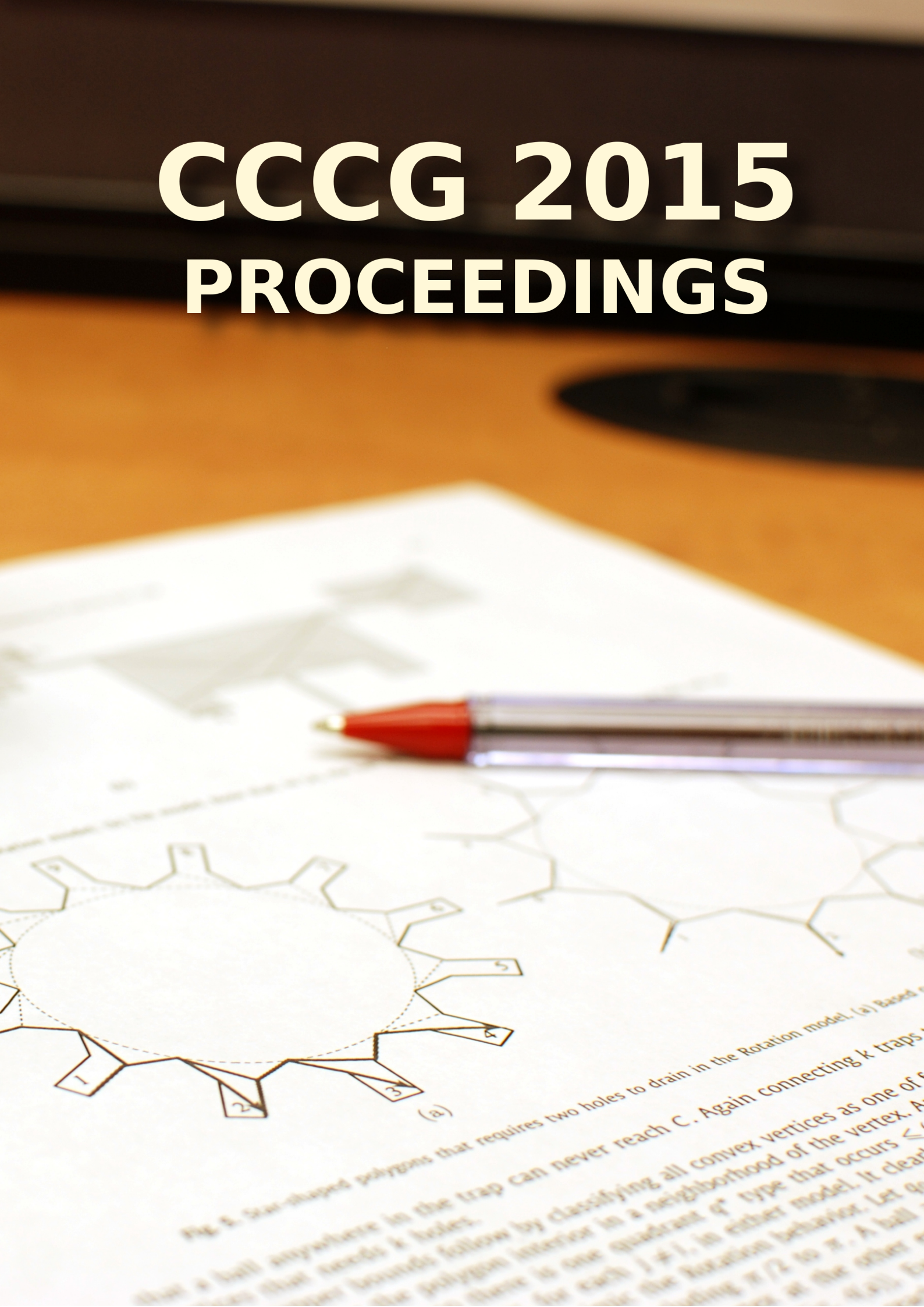
CCCG 2015
PROCEEDINGS



Proceedings of the 27th Canadian Conference on Computational Geometry

August 10-12, 2015
Queen's University
Kingston, Ontario
Canada

Sponsored by

Archived as Queen's University School of Computing Technical Report number 2015-626.

Foreword

The Twenty-Seventh Canadian Conference on Computational Geometry took place from August 10-12, 2015 at Queen's University in Kingston, Ontario. This annual conference attracts researchers in computational geometry from around the world to Canada for an open exchange of ideas and results. This volume contains 43 contributed papers as well as 3 invited talks. These proceedings are available on line at `http://research.cs.queensu.ca/cccg2015/` and at the central CCCG web site `http://www.cccg.ca`.

The organizing committee would like to thank the invited speakers Jit Bose (Ferran Hurtado Memorial Lecture), Bruce Reed (Paul Erdős Memorial Lecture), and Jonathan Shewchuk. We also thank all of those who contributed their papers to the conference.

Thanks go out to the program committee and the local arrangements committee for their organizational assistance. Special thanks to Ben Cecchetto and Bahram Kouhestani. Ben designed the CCCG 2015 web page and the cover of these proceedings. Furthermore, Ben is the webmaster and Bahram prepared these proceedings from the submitted LaTeX source files.

We gratefully acknowledge financial support from the Fields Institute for Research in Mathematical Sciences, the Queen's University Office of Research Services, the Queen's University Faculty of Arts and Science, and the School of Computing at Queen's University.

David Rappaport

Avant-Propos

La vignt-septième conférence sur la géométrie algorithmique a eu lieu du 10 au 12 août 2015, à l'Université Queen's à Kingston, en Ontario. Cette conférence annuelle réunit au Canada des chercheurs en géométrie algorithmique du monde entier, pour un échange d'idées et de résultats. Ce volume contient les textes de 43 articles communiqués à la conférence ainsi que les résumés de 3 articles présentés sur invitation. Ces actes sont disponibles en lignes à `http://research.cs.queensu.ca/cccg2015/` et à la page Web centrale de CCCG, `http://www.cccg.ca`.

Le comité organisateur remercie les conférenciers invités Jit Bose (Ferran Hurtado Memorial Lecture), Bruce Reed (Paul Erdős Memorial Lecture), et Jonathan Shewchuk. Nous remercions aussi les auteurs des communications.

Merci au comité de programme et le comité des préparatifs locaux pour l'ensemble de leur aide organisationnelle. Un merci spécial à Ben Cecchetto et Bahram Kouhestani. Ben a conçu la page Web CCCG 2015 et la couverture de ces actes. De plus, Ben est le webmaster et Bahram a préparé ces actes à partir des fichiers sources LaTeX qui nous furent soumis.

Nous désirons exprimer notre reconnaissance aux organismes suivants de leur support financier: Fields Institute for Research in Mathematical Sciences, Queen's University Office of Research Services, Queen's University Faculty of Arts and Science, et School of Computing at Queen's University.

David Rappaport

Invited Speakers

Jit Bose	Carleton University
Bruce Reed	McGill University
Jonathan Shewchuk	University of California at Berkeley

Program Committee

Mohammad Ali Abam	Sharif University
Oswin Aichholzer	University of Technology Graz
Greg Aloupis	Tufts University
Binay Bhattacharya	Simon Fraser University
Therese Biedl	University of Waterloo
Paz Carmi	Ben-Gurion University
Mirela Damian	Villanova University
Stephane Durocher	University of Manitoba
William Evans	University of British Columbia
Robin Flatland	Siena College
Joachim Gudmundsson	University of Sydney
Meng He	Dalhousie University
John Howat	Carleton University
John Iacono	New York University
Hiro Ito	The University of Electro-Communications (UEC)
Matias Korman	National Institute of Informatics
Alejandro López-Ortiz	University of Waterloo
Anil Maheshwari	Carleton University
Pat Morin	Carleton University
Asish Mukhopadhyay	University of Windsor
David Rappaport (Chair)	Queen's University
Carlos Seara	Universitat Politécnica de Catalunya (UPC)
Ryuhei Uehara	Japan Advanced Institute of Science and Technology (JAIST)
Stephen Wismath	University of Lethbridge
Norbert Zeh	Dalhousie University

Organizing Comittee

Benjamin Cecchetto	Queen's University
Daniel Goc	Queen's University
Bahram Kouhestani	Queen's University
Timothy Ng	Queen's University
David Rappaport (Chair)	Queen's University
Kai Salomaa	Queen's University
Thomas Vaughan	Queen's University

Additional Reviewers

Ahmad Biniaz
Aritra Banik
Shervin Daneshpajouh
Martin Derka
Stefan Huber
Shahin Kamali
Mohammad Reza Kazemi
Bahram Kouhestani
Daniela Maftuleac
Bengt J. Nilsson
Yoshio Okamoto
Yota Otachi
Alexander Pilz
Andr van Renssen
Marcel Roeloffzen
Natan Rubin
Manfred Scheucher
Saeed Seddighin
Dimitrios Skrepetos
Akihiro Uejima
Haitao Wang
Gelin Zhou

Contents

Session 2A

Session 2B

Session 3A

Session 3B

Tuesday 11 August

Session 4A

Session 4B

Session 5A

Session 5B

Wednesday 12 August

Session 6A

Session 6B

Connectivity Preserving Iterative Compression

Bruce Reed (McGill University)

Abstract

When applying iterative compression to solve an optimization problem, we construct a smaller auxiliary graph from the input graph, solve the problem on this smaller graph, and then use the solution to solve the original problem. We present a variant of this approach in which the smaller graph inherits the connectivity properties of the original graph. This is especially useful when trying to solve embedding or routing problems. We present three applications which illustrate this. To warm up we present a simple linear time algorithm which tests planarity and embeds a planar input in the plane. We then present, for every fixed k, a linear time algorithm which determines if an input graph has crossing number at most k, and obtains a corresponding embedding if it does. Finally we present a linear time algorithm which given four vertices s1,s2,t1,t2 of a graph G determines if there are 2 vertex disjoint paths P1 and P2 such that Pi contains si and ti. It either finds the desired two paths or an embedding up to 3-cuts of G in a disc which proves that the paths cannot exist. As well as introducing the new technique, our talk will highlight some of the connections between graph minor theory and embedding algorithms.

(This is a joint work with Kenichi Kawarabayashi and Zhentao Li).

An Algorithm for the Maximum Weight Independent Set Problem on Outerstring Graphs*

J. Mark Keil† Joseph S. B. Mitchell‡ D. Pradhan§ Martin Vatshelle¶

Abstract

Outerstring graphs are the intersection graphs of curves that lie inside a disk such that each curve intersects the boundary of the disk. Outerstring graphs are among the most general classes of intersection graphs studied. To date, no polynomial time algorithm is known for any of the classical graph optimization problems on outerstring graphs; in fact, most are NP-hard. It is known that there is an intersection model for any outerstring graph that consists of polygonal arcs attached to a circle. However, this representation may require an exponential number of segments relative to the size of the graph.

Given an outerstring graph and an intersection model consisting of polygonal arcs with a total of N segments, we develop an algorithm that solves the Maximum Weight Independent Set problem in $O(N^3)$ time. If the polygonal arcs are restricted to single segments, then outersegment graphs result. For outersegment graphs, we solve the Maximum Weight Independent Set problem in $O(n^3)$ time where n is the number of vertices in the graph.

1 Introduction

A graph G is a *geometric intersection graph* if the vertex set of G is a set of geometric objects and two such objects are adjacent in G if and only if they intersect. An *independent set* in a geometric intersection graph corresponds to a set of disjoint geometric objects in the intersection model. The Maximum (Weight) Independent Set problem in intersection graphs of geometric objects in the plane has many applications, including train dispatching [8], map labelling [1], and data mining [13]. In the railroad dispatching problem studied by Filer, Mihalák, Schöbel, Widmayer and Zych [8], we are given a set of paths (strings) in the plane and asked for a maximum set of non-conflicting train routes, i.e. a maximum independent set in a string graph. Due to the fact that most trains either leave or enter the station area, it is natural to consider outerstring graphs in this application. This is the problem we solve in this paper.

String graphs are the intersection graphs of curves in the plane and they are among the most general geometric intersection graphs that have been studied. String graphs are a superclass of planar graphs [25], chordal graphs, co-comparability graphs [15], subtree filament graphs [14] and circle graphs. Indeed the intersection graph of any collection of connected sets in the plane is a string graph. As early as 1959, Benzer [4] encountered string graphs in his study of genetic structures. Since then they have been extensively studied and have many applications. Kratochvíl et al. [19] showed that every string graph can be realized by a family of polygonal arcs with a finite number of intersections. However in 1991, Kratochvíl and Matoušek [21] constructed string graphs on n vertices that require at least 2^{cn} intersection points in any realization. This also implies that a representation of a string graph with a family of polygonal arcs may require an exponential number of bends in the polygonal arcs. In 1991, Kratochvíl [18] proved that the problem of recognizing string graphs is NP-hard, but more than a decade passed before Schaefer et al. [24] showed that recognizing string graphs is in NP.

In 1966 Sinden [25] showed that all planar graphs are string graphs, thus the Maximum Independent Set problem became known to be NP-hard on string graphs when it was proven to be NP-hard in planar graphs. Recently, Fox and Pach [10] provided approximation algorithms and exact sub-exponential algorithms for the Maximum Independent Set problem in string graphs. In 1976, the 3-Colorability problem for string graphs was proven NP-complete by Ehrlich et al. [7], even when a geometric representation is given as the input. The Maximum Clique problem has long been known to be NP-hard [22, 23] on string graphs. Indeed most of the classical NP-hard graph optimization problems remain NP-hard when restricted to string graphs, even when given a geometric representation.

It seems that one must somehow restrict string graphs to achieve polynomial time algorithms. The two most natural ways to restrict string graph are to either limit the shapes of the strings, or to limit the positions of the strings. The most commonly studied such restrictions are to limit the strings to be straight line segments or

*J. M. Keil supported by NSERC; J. Mitchell is partially supported by NSF (CCF-1018388) and the US-Israel Binational Science Foundation (Grant 2010074).

†Dept. of Computer Science, University of Saskatchewan
‡Dept. of Appl. Math. and Statistics, Stony Brook University
§Dept. of Applied Math, Indian School of Mines, Dhanbad
¶Dept. of Informatics, University of Bergen

to require that each string touches the infinite face of the plane. We first consider each restriction separately and then the case combining them.

Segment graphs are the intersection graphs of line segments in the plane. This restriction to line segments still allows the graphs to be useful in many applications, but unfortunately most of the classical NP-complete graph problems remain intractable on segment graphs. Since all planar graphs are segment graphs [6], the 3-Colorability problem and the Maximum Independent Set problem remain NP-complete on segment graphs. Kratochvíl and Nešetřil [22] proved that the Maximum Independent Set problem in segment graphs is NP-hard even if all the segments are restricted to lie in at most two directions in the plane. It has recently been shown that the Maximum Clique problem is NP-hard on segment graphs [5]. Thus even a severe limiting of the shapes of the strings in a string graph does not lead to polynomial time algorithms.

The restriction that each string touches the infinite face of the plane was explored in 1991 [17] by Kratochvíl who defined *outerstring graphs* to be the intersection graphs of curves that lie inside a disk such that each curve intersects the boundary of the disk in one of its endpoints. Although outerstring graphs have been studied for more than 20 years [11, 12, 17, 20], when we consider the classical NP-hard graph optimization problems on outerstring graphs, we again do not find any known polynomial time algorithms. For outerstring graphs the NP-completeness of Minimum Clique Cover, Colorability, Minimum Dominating Set, and Hamiltonian Cycle follow from the fact that they contain circle graphs. The Maximum Clique problem was recently shown to be NP-hard on ray graphs [5], a subclass of outerstring graphs. The Maximum Independent Set problem remains open on outerstring graphs.

In their study of train dispatching, Flier et al. [9] consider subclasses of outerstring graphs, in particular the intersection graphs of segments lying inside a disk having one endpoint attached to the boundary of the disk, called *outersegment graphs*. Applying the additional restriction that each segment is either horizontally or vertically aligned, they are able to obtain a polynomial time algorithm for the Maximum Independent Set problem given a geometric representation of the graph.

In the next section, we describe a dynamic programming algorithm for the Maximum Weight Independent Set problem in an outerstring graph which runs in time polynomial in the size of the geometric input representation of the graph. Finally, we show how our algorithm can be used to find a maximum weight set of disjoint boundary rectangles in $O(n^3)$ time. This problem has applications in PCB routing [16].

2 Outerstring graphs

Outerstring graphs are the intersection graphs of curves in the plane that lie inside a circle such that each curve intersects the boundary of the circle in one of its endpoints. Let $G = (V, E)$ be an outerstring graph with n weighted vertices. In order to find the Maximum Weight Independent Set of G, our algorithm assumes the input is a polygonal geometric representation of G. The circle is represented as a simple polygon P with $O(n)$ vertices. Lying completely inside P, each string s corresponding to a vertex of G is represented by a non-self-intersecting polygonal line with one endpoint, $start(s)$, coinciding with a unique vertex of P. We call the vertices of s that are different from $start(s)$ the interior vertices of s, as they lie in the interior of P. Let S be the set of polygonal lines corresponding to the vertices of V. Then $R(G) = (P, S)$ is a representation of G. See the top left of Figure 1. Let N be the total number of segments used to represent the strings in S, and the polygon P. In many applications of outerstring graphs this polygonal geometric representation is the natural input.

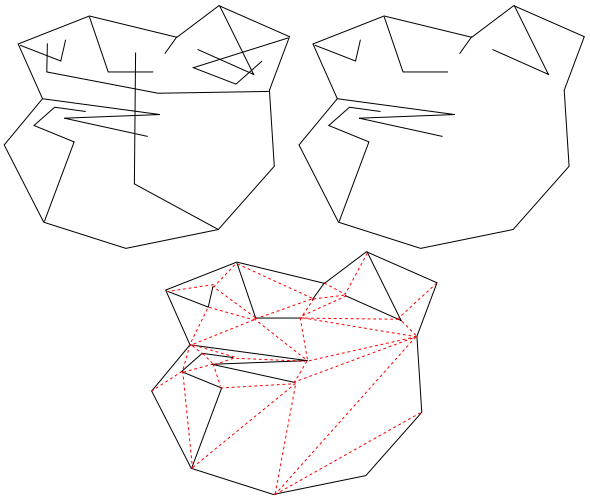


Figure 1: The representation of an outerstring graph, a maximum set S^* of disjoint strings, and a triangulation of P respecting S^*.

An optimal solution to the Maximum Weight Independent Set problem for an outerstring graph G with geometric representation $R(G) = (P, S)$ appears as a set of disjoint polygonal lines $S^* \subseteq S$ inside P. See the top right of Figure 1. Together, $P \cup S^*$ form a connected planar straight line graph that can be triangulated. The bottom drawing of Figure 1 show the triangles that are inside P in a triangulation of $P \cup S^*$. Let u and v be vertices of either P or of the polygonal lines of S. In order to define the subproblems used in

our dynamic program, we will use u and v to define a subpolygon $P(u, v)$ of P. The subproblem associated with $P(u, v)$ is that of finding a maximum weight independent set of strings of S that lie wholly inside $P(u, v)$, where $P(u, v)$ is treated as a closed set. It is sufficient to consider all u and v such that uv forms an edge in a triangulation of the planar straight line graph $P \cup S^*$, where $S^* \in S$ is the set of strings in an optimal solution. We do not know S^*, but if u is an interior vertex of string S_u and v is an interior vertex of string S_v, then if uv is to be a diagonal in a triangulation of $P \cup S^*$ strings S_u and S_v must be in the optimal solution, and thus be disjoint. Thus if u or v are interior vertices of strings of S, we also insist on including those strings in the solution to the subproblem for $P(u, v)$ whether or not they lie completely in $P(u, v)$. Also segment uv cannot intersect any segment in S_u or S_v.

If u and v are both vertices of P, we define $P(u, v)$ to be the part of P to the left of the directed edge $\vec{uv}$. See the top drawing in Figure 2. If v is an interior vertex of a string A in S, and u lies on polygon P such that string A does not intersect segment uv then we define $P(u, v)$ to be the polygonal region bounded by the portion of P clockwise from u to $start(A)$, the portion of string A from $start(A)$ to v, and the edge uv. See the top drawing in Figure 3 for an example. If u is an interior vertex of a string and v lies on P then $P(u, v)$ is defined analogously. If both u and v are interior vertices of distinct strings of P, $P(u, v)$ is defined to be the polygonal region bounded by the portion of P clockwise from $start(S_u)$ to $start(S_v)$, the portion of S_v from $start(S_v)$ to v, segment uv, and the portion of S_u from u to $start(S_u)$. See for example the top drawing in Figure 4. Let $f(u, v)$ be the optimal value of a solution to the subproblem associated with $P(u, v)$. The dynamic programming algorithm considers the subpolygons $P(u, v)$ in increasing order of area, and for each computes the optimal weight $f(u, v)$ for a solution to the subproblem.

The type of a subproblem depends on whether u or v or both are interior vertices of strings of S.

Type 0: In a type 0 subproblem neither u nor v is an interior vertex of a string of S; thus, both are vertices of P. See Figure 2.

In a triangulation of $P(u, v)$ where $P(u, v)$ contains the strings of an optimal solution, the third vertex w of the triangle containing u and v may be a vertex of P. In this case the optimal solution is the disjoint union of the solution to the subproblem associated with $P(u, w)$ and the solution to the subproblem associated with $P(w, v)$.

If w is an interior vertex of a string A then the solution to $P(u, v)$ will depend upon whether or not string A contains u or v as $start(A)$. If neither u nor v is $start(A)$, then the subproblem for $P(u, w)$ is of type 1 and the solution contains string A. Likewise the solution to the subproblem for $P(w, v)$ is of type 1 and contains string A. The solution to the subproblem for $P(u, v)$ is the union of the solution to the subproblem for $P(u, w)$ and the solution to subproblem for $P(w, v)$. In the union, the string A is only included once; thus, $f(u, v) = f(u, w) + f(w, v) - weight(A)$. If $start(A)$ is u, then the region cut off by $\vec{uw}$ is a "pocket" of string A that cannot contain any other string. See the fourth situation of Figure 2. The only subproblem that exists is that for subpolygon $P(w, v)$; thus, $f(u, v) = f(w, v)$.

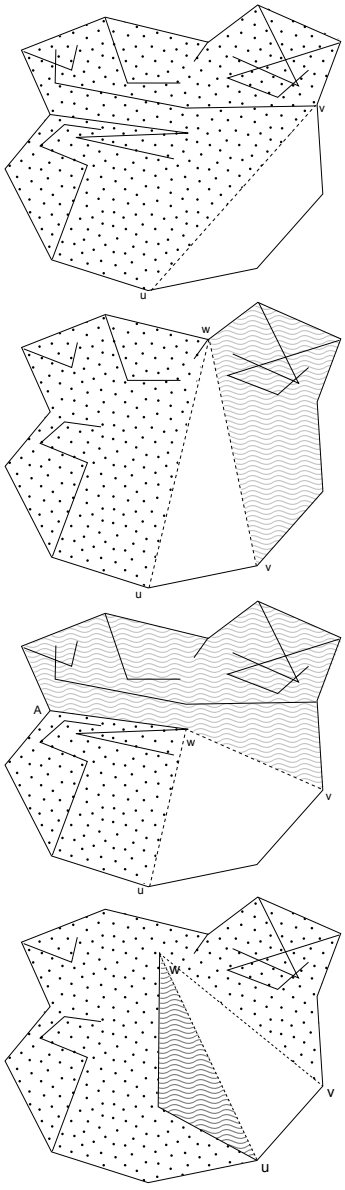


Figure 2: In a type 0 subproblem, the third vertex w of triangle uwv may lie on P or on a string A of S.

Type 1: In this case, exactly one of u and v lie on P. Without loss of generality, assume that u lies on P and v is an interior vertex of a string A. Hence, the subproblem for $P(u, v)$ includes string A plus the solution to the Maximum Weight Independent Set problem for strings that lie inside P, left of $\vec{uv}$ that avoid string A. See Figure 3. The third vertex w of the triangle that contains u and v in a triangulation of $P(u, v)$ may lie on P, A or another string B. If w lies on P then we have reduced the problem to a smaller type 0 subproblem associated with $P(u, w)$ and a smaller type 1 subproblem associated with $P(w, v)$. If w lies on A then we have reduced the problem to a smaller type 1 subproblem associated with $P(u, w)$. Note that the region cut off by $\vec{wv}$ is a "pocket" of string A that cannot contain any other string. See the third situation in Figure 3. If w lies on another string B, then the problem reduces to a subproblem of type 1 associated with $P(u, w)$, plus a subproblem of type 2 associated with $P(w, v)$, as in the final drawing of Figure 3.

Type 2: In a type 2 subproblem u is an interior vertex of a string A and v is an interior vertex of a string B. The third vertex w of a triangle containing u and v may lie on P, one of A or B, or on another string C. See Figure 4. If w lies on P then the solution to the subproblem for $P(u, v)$ is the union of the solutions to two type 1 subproblems, for $P(u, w)$ and for $P(w, v)$. If w lies on B, the same string as v, then the region cut off by segment wv cannot contain any other strings and the solution to for $P(u, v)$ will consist of the solution to the smaller subproblem for $P(u, w)$. See the third drawing of Figure 4. The situation where w lies on A is analogous. If w lies on a new string C, then the solution to the subproblem for $P(u, v)$ is the union of the two type 2 subproblems associated with $P(u, w)$ and $P(w, v)$, as in the final drawing of Figure 4.

Theorem 1 *Given the geometric representation $R(G) = (P, S)$ of a weighted outerstring graph G, the dynamic programming algorithm described above computes the maximum weight of an independent set for G in $O(N^3)$ time, where N is the number of segments used to represent the strings of S and the polygon P.*

Proof. The correctness of the computation of each $f(u, v)$ can be verified by induction. If $P(u, v)$ is a triangle then $f(u, v)$ is either zero or equal to the weight of the input string(s) containing u and/or v. Otherwise $f(u, v)$ can be computed in constant time from $f(u, w)$ and $f(w, v)$, for one of the $O(N)$ possible w in $P(u, v)$, where $P(u, w)$ and $P(w, v)$ are smaller area subpolygons such that the triangle uwv only intersects the strings in the optimal solution of the subproblem associated with $P(u, v)$ in vertices of the representation.

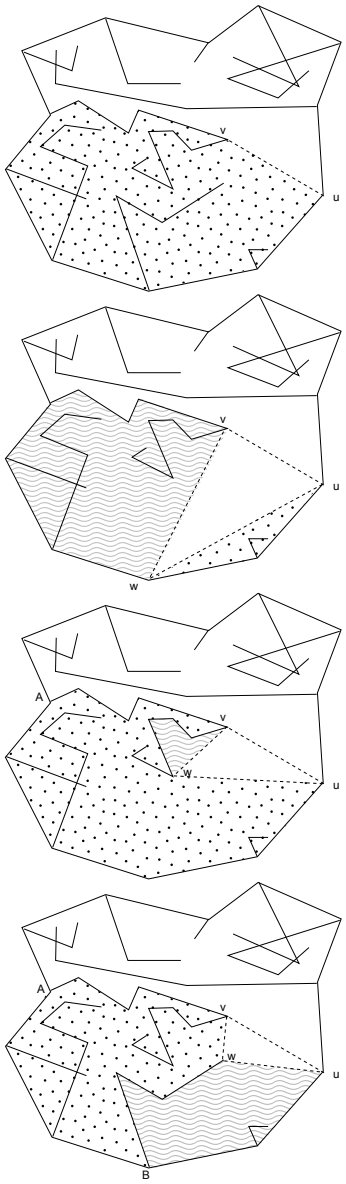


Figure 3: In a type 1 subproblem, the third vertex w of triangle uwv may lie on P, on the same string as v, or on a different string.

The $O(N^3)$ running time can be achieved by precomputing intersection information. There are $O(N^2)$ segments uv that may potentially define subpolygons $P(u, v)$. Each of these segments can be tested against the N segments in the representation to determine the strings of S that the segment intersects. In order for $P(u, v)$ to be a subpolygon, segment uv must not intersect with any segment in the string S_u containing u nor in the string S_v containing v. Further, there can be no intersection between strings S_u and S_v. For all S_i and S_j, it can be precomputed in $O(N^2)$ time whether

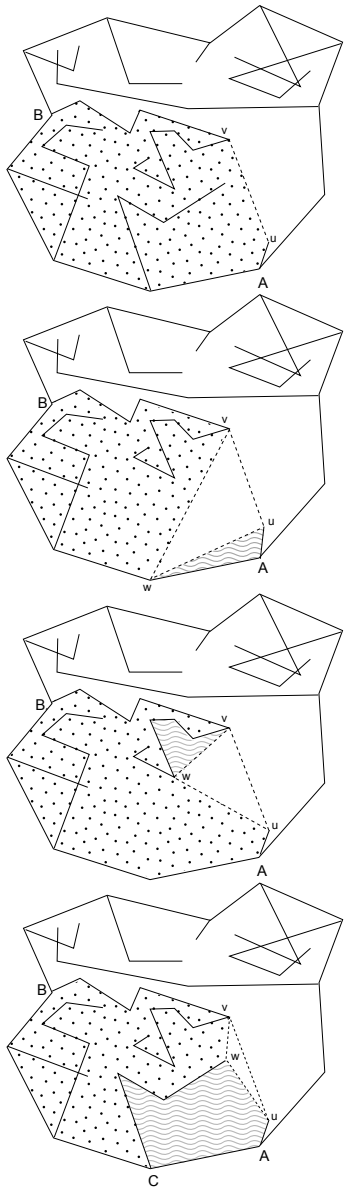


Figure 4: A type 2 subproblem

or not the strings S_i and S_j intersect, by first precomputing all the pairwise intersections of segments in the representation $R(G)$.

Given a subpolygon $P(u, v)$, where u and v are interior vertices of strings A and B respectively, a vertex w is potentially the third vertex of a triangle uwv in a triangulation of $P(u, v)$ containing the strings corresponding to an optimal solution for $P(u, v)$ if uw and wv do not intersect the strings A and B. These intersections have been precomputed and stored. It is also necessary to ensure that w is on the left side of $\vec{uv}$. □

If the strings in $R(G) = (P, S)$ are each single segments then G is an outersegment graph. The geometric representation of an outersegment graph with n vertices requires only n segments to represent S and P; thus:

Corollary 1 *Given the geometric representation of a weighted outersegment graph with n vertices, the dynamic programming algorithm computes the maximum weight of an independent set in $O(n^3)$ time.*

3 Application

We use our Maximum Weight Independent Set algorithm for outerstring graphs to find a maximum weight disjoint set of boundary rectangles. Given a rectangular region R, a rectangle r contained in R is a *boundary rectangle* with respect to R if at least one of the sides of r is a subset of a side of R. The problem of finding a maximum weight disjoint set of boundary rectangles has application in printed circuit board routing [16]. Kong et al. [16] provide the first polynomial time algorithm for the problem running in $O(n^6)$ time. This was improved to $O(n^4)$ in [3] and [2]. Using our Maximum Weight Independent Set algorithm for outerstring graphs, we can achieve $O(n^3)$ time.

Given a rectangle R and a set Q of n weighted boundary rectangles inside R, we create an instance of the Maximum Weight Independent Set for an outerstring graph G with geometric representation $R(G) = (P, S)$, where P is the boundary of R and each boundary rectangle r in Q maps to a four segment polygonal chain in S. Let s_{min} be the minimum side length of a rectangle in Q, and let δ be $\frac{s_{min}}{2}$.

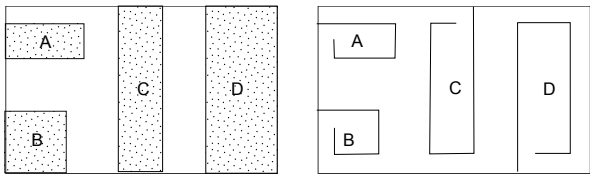


Figure 5: The boundary rectangles in Q are represented by strings in S

If a rectangle $r \in Q$ has one side contained in a side of R, then the corresponding string s has $start(s)$ at the clockwisemost intersection point of the boundary of r and R. See rectangle and string A in Figure 5. The first two segments in the polygonal chain of s coincide with the first two sides of r in a clockwise traversal of the boundary of r. The third segment of s is a subset of the third side of r stopping δ before the side of R. The fourth segment of s is parallel to the side of R containing $start(s)$ at distance δ from the boundary of R. There is a gap of δ between the end of the fourth segment of s and the first segment of s. When a rectangle $r \in Q$

shares two or three sides with R, the construction of the strings is illustrated in Figure 5.

The construction of the strings in S corresponding to the rectangles in R allows us to conclude that two rectangles r_1 and r_2 in Q intersect if and only if their corresponding strings s_1 and s_2 in S intersect.

Our $O(N^3)$ Maximum Weight Independent Set algorithm for outerstring graphs thus gives us the following theorem.

Theorem 2 *Given a rectangle R and a set Q of n weighted boundary rectangles inside R, a maximum weight disjoint set of rectangles in Q can be found in $O(n^3)$ time.*

References

[1] P. Agarwal, M. van Kreveld, and S. Suri. Label placement by maximum independent set in rectangles. *Computational Geometry: Theory and Applications*, 11(3):209–218, 1998.

[2] A. AhmadiNejad and H. Zarrabi-Zadeh. The maximum disjoint set of boundary rectangles. In *Proceedings of CCCG*, pages 302–307, 2014.

[3] S. Assadi, E. Emamjomeh-Zadeh, S. Yazdanbod, and H. Zarrabi-Zadeh. On the rectangle escape problem. In *Proceedings of CCCG*, pages 235–240, 2013.

[4] S. Benzer. On the topology of genetic fine structure. *Proceedings of the National Academy of Sciences*, 47:1607, 1959.

[5] S. Cabello, J. Cardinal, and S. Langerman. The clique problem in ray intersection graphs. *Discrete & Computational Geomtry*, 50(3):771–783, 2013.

[6] J. Chalopin and D. Gonçalves. Every planar graph is the intersection graph of segments in the plane. In *Proceedings of STOC*, pages 631–638, 2009.

[7] S. Ehrlich, S. Even, and R. Tarjan. Intersection graphs of curves in the plane. *J. Combin. Theory Ser. B.*, 21:8–20, 1976.

[8] H. Flier, M. Mihalák, A. Schöbel, P. Widmayer, and A. Zych. Vertex disjoint paths for dispatching in railways. In *Proceedings of ATMOS*, pages 61–73, 2010.

[9] H. Flier, M. Mihalák, P. Widmayer, and A. Zych. Maximum independent set in 2-direction outersegment graphs. In *Proceedings of WG*, pages 155–166, 2011.

[10] J. Fox and J. Pach. Computing the independence number of intersection graphs. In *Proceedings of SODA*, pages 1161–1165, 2011.

[11] J. Fox and J. Pach. Coloring K_k-free intersection graphs of geometric objects in the plane. *Eur. J. Comb.*, 33(5):853–866, 2012.

[12] J. Fox and J. Pach. String graphs and incomparability graphs. *Advances in Math.*, 230:1381–1401, 2012.

[13] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Data mining with optimized two-dimensional association rules. *ACM Trans. on Database Systems*, 26(2):179–213, 2001.

[14] F. Gavril. Maximum weight independent sets and cliques in intersection graphs of filaments. *Information Processing Letters*, 73:181–188, 2000.

[15] M. Golumbic, D. Rotem, and J. Urrutia. Comparability graphs and intersection graphs. *Discrete Math.*, 43:37–46, 1983.

[16] H. Kong, Q. Ma, T. Yan, and M. D. Wong. An optimal algorithm for finding disjoint rectangles and its applications to PCB routing. In *Proceedings of DAC*, pages 212–217, 2010.

[17] J. Kratochvíl. String graphs I. The number of critical nonstring graphs is infinite. *J. Comb. Theory, Ser. B*, 52(1):53–66, 1991.

[18] J. Kratochvíl. String graphs II. Recognizing string graphs is NP-hard. *J. Comb. Theory, Ser. B*, 52(1):67–78, 1991.

[19] J. Kratochvíl, M. Goljan, and P. Kučera. *String Graphs*. Academia, Prague, 1986.

[20] J. Kratochvíl, A. Lubiw, and J. Nešetřil. Noncrossing subgraphs in topological layouts. *SIAM J. Discrete Math.*, 4(2):223–244, 1991.

[21] J. Kratochvíl and J. Matoušek. String graphs requiring exponential representations. *J. Comb. Theory, Ser. B*, 53(1):1–4, 1991.

[22] J. Kratochvíl and J. Nešetřil. Independent set and clique problems in intersection-defined classes of graphs. *Comment. Mat. Univ. Carolinae*, 31:85–93, 1990.

[23] M. Middendorf and F. Pfeiffer. The max clique problem in classes of string-graphs. *Discrete Math.*, 108:365–372, 1992.

[24] M. Schaefer, E. Sedgwick, and D. Štefankovič. Recognizing string graphs is in NP. *J. Comput. Syst. Sci.*, 67(2):365–380, 2003.

[25] F. W. Sinden. Topology of thin film RC-circuits. *Bell System Tech. J.*, 45:1639–1662, 1966.

Duality for Geometric Set Cover and Geometric Hitting Set Problems on Pseudodisks*

Stephane Durocher† Robert Fraser†

Abstract

Given an instance of a geometric set cover problem on a set of points X and a set of objects $\mathcal{R}$, the dual is a geometric hitting set problem on a set of points P and a set of objects $\mathcal{Q}$, where there exists a one-to-one mapping from each $x_j \in X$ to a dual object $Q_j \in \mathcal{Q}$ and for each $R_i \in \mathcal{R}$ to a dual point in $p_i \in P$, so that a dual point p_i is contained in a dual object Q_j if and only if the corresponding primal point x_j is covered by the object R_i. In this work, we explore the setting of geometric duality for geometric set cover problems on pseudodisks. We first show that there does not always exist a geometric dual on pseudodisks. We initiate the search for a characterization of the class of objects that may be dualized by identifying a sufficient (but not necessary) property for a dual to exist on distinct pseudodisks, called the *pair-cover and crossing-quad free* property. We show that such problems may be dualized into hitting set instances on pseudodisks by building a planar support for the dual instance, and then constructing an orthogonal drawing of the support which we transform into a dual set of pseudodisks. A corollary of these results is a PTAS for dualizable set cover problems using the PTAS for hitting set on pseudodisks.

1 Introduction

Geometric duality is a beautiful and useful tool for computational geometers, as some problems are conceptually simpler to solve in the dual setting. The classic example is point-line duality in the plane, see e.g. [8, §8.2]. Duality has been the catalyst for breakthroughs such as the first optimal algorithm for the half-plane range query problem [6]. Our interest lies in geometric set cover and hitting set problems, motivated by the distinction that there exists a PTAS for the hitting set problem on pseudodisks [14],[1] while none is known for the set cover problem on pseudodisks. Therefore, we endeavoured to prove that any set cover problem on pseudodisks could be dualized to a hitting set problem on pseudodisks in polynomial time (and vice versa), which would permit the use of the PTAS to obtain an approximate solution.

Unfortunately, it turns out that the dualization we desired is not always possible, and we open our study by proving this fact. We show that dualization is straightforward for problems where all regions in the instance are translations of a geometric object. We proceed by studying dualization on pseudodisks having the *pair-cover and crossing-quad free* property, and we prove that such objects may always be dualized.

1.1 Definitions and Nomenclature

We begin by reviewing concepts required for our discussion.

Definition 1 Pseudodisk: *A pseudodisk is a region of the plane bounded by a closed Jordan curve, with the restriction that the boundaries of any two pseudodisks in a given instance may intersect only transversely and at most twice.*

Definition 1 could be generalized to allow pseudodisks to intersect exactly once (tangentially at a single point), but we ignore this detail for clarity of exposition.

Definition 2 Geometric Set Cover Problem: *Given a set of points X and a set of geometric objects $\mathcal{R}$, the* geometric set cover problem *is to find a subset $\mathcal{R}^\star \subseteq \mathcal{R}$ of minimum cardinality so that all elements of X are covered by $\mathcal{R}^\star$, i.e. $X \subseteq \cup_{R \in \mathcal{R}^\star} R \cap X$.*

Definition 3 Geometric Hitting Set Problem: *Given a set of points P and a set of geometric objects $\mathcal{Q}$, the* geometric hitting set problem *is to find a subset $P^\star \subseteq P$ of minimum cardinality so that all objects in $\mathcal{Q}$ contain at least one element of $P^\star$, i.e. $\forall Q \in \mathcal{Q}, Q \cap P^\star \neq \emptyset$.*

Definition 4 Geometric Dual: *Given an instance of the set cover problem $\mathcal{S} = (X, \mathcal{R})$ (the* primal *setting), an instance of the hitting set problem $\mathcal{H} = (P, \mathcal{Q})$ is a* geometric dual *of $\mathcal{S}$ (the* dual *setting) if there are bijections between X and $\mathcal{Q}$ as well as $\mathcal{R}$ and P and any point $p_i \in P$ is contained in an object $Q_j \in \mathcal{Q}$ if and only if the corresponding point $x_j \in X$ is covered by the object $R_i \in \mathcal{R}$ in the primal setting. An optimal solution $P^\star$ for the dual setting corresponds exactly to*

*This work was supported in part by the Natural Sciences and Engineering Council of Canada (NSERC).

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

[1] A set of pseudodisks is equivalent to a set of the 2-admissible regions used in [14].

an optimal solution $\mathcal{R}^\star$ for the primal setting. A set cover instance dualizing a hitting set instance is defined analogously.

In this discussion, we usually omit the prefix "geometric" from the concepts defined here. We consistently use X and $\mathcal{R}$ for set cover and P and $\mathcal{Q}$ for hitting set to differentiate the two settings. We write ∂R_i to denote the boundary of a pseudodisk R_i. We assume that all points and pseudodisks are distinct.

1.2 Related Work

The conventional application of duality is to points and lines (or pseudolines, which are curves that intersect pairwise at most once, and such intersections must not be tangential) [2, 10]. A set of pseudolines can define the boundaries of a set of pseudo-halfplanes so that the setting becomes a set cover or hitting set problem, and the preservation of the "above-below" property in the dual means that duality exists and is well defined for our purposes. Our work seeks to extend these results to pseudodisks.

A corollary of our result is the derivation of a PTAS for some set cover problems on pseudodisks by showing that they may be dualized to hitting set problems on pseudodisks, allowing the hitting set PTAS of Mustafa and Ray [14] to be applied. This is an active area of research; a QPTAS for broad classes of set cover problems was recently found [13]. Our algorithm makes use of the arrangement of the boundaries of a set of pseudodisks (this set of boundaries is also known as a set of pseudo-circles); the combinatorial properties of such arrangements are well studied [1].

A *hypergraph* is a generalization of a standard graph, where a *hyperedge* may contain any number of points. Therefore, a hypergraph $H = (V, F)$ may be used as an abstract representation of a set cover problem $\mathcal{S} = (X, \mathcal{R})$ (or hitting set problem) by creating a vertex v_j in V for each point x_j in X, and mapping each edge $f_i \in F$ to an object $R_i \in \mathcal{R}$ so that the vertices in $f_i \cap V$ correspond exactly to the points in $R_i \cap X$. A *planar support* of a hypergraph $H = (V, F)$ is a planar graph $G = (V, E)$ where the subgraph of G induced by any hyperedge $f_i \in F$ is connected. Much of the research with respect to planar supports has been to determine whether a planar support exists for a given hypergraph [5, 12]. Our use of planar supports is reversed, since finding the hypergraph corresponding to the hitting set instance dualizing a given set cover instance is straightforward. We build a planar support for the hypergraph, and the support is used to create the dual hitting set instance on pseudodisks. To our knowledge, supports have not been used in a similar manner before.

2 A Counterexample for Duality on Pseudodisks

Theorem 1 *There exists a family of set cover problems on pseudodisks for which there is no dual hitting set formulation on pseudodisks (and equivalently, such hitting set problems on pseudodisks cannot be dualized into set cover instances on pseudodisks).*

See Appendix A for the proof of Theorem 1.

3 Geometric Dual of Set Cover

Our aim is to identify classes of set cover problems on pseudodisks that may always be dualized. We begin with a more general result that is fairly trivial to establish.

Theorem 2 *Problems defined on translates of any single object can always be dualized.*

See Appendix A for the proof of Theorem 2.

For the remainder of the paper, we outline a method for reducing an instance of another class of geometric set cover problems on pseudodisks to an instance of a geometric hitting set problem on pseudodisks (or vice versa, of course).

Definition 5 Pair-Cover Free Property: *The* Pair-Cover Free *(PF) property holds for a set of geometric objects $\mathcal{R}$ if $R_i \not\subseteq R_j \cup R_k$ for all $R_i, R_j, R_k \in \mathcal{R}$.*

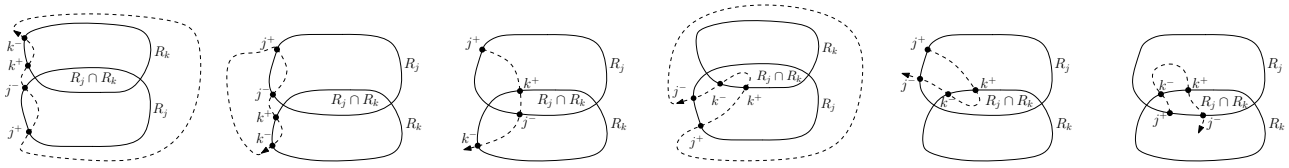
Definition 6 Crossing-Quad Free Property: *If $R_k \cap R_\ell \not\subseteq R_i \cup R_j$ for any four pseudodisks $\{R_i, R_j, R_k, R_\ell\} \subseteq \mathcal{R}$ where $R_i \cap R_j \subseteq R_k \cup R_\ell$, then the* Crossing-Quad Free *(CF) property holds.*[2]

We define the *pair-cover and crossing-quad free set cover* (PCF-SC) problem on pseudodisks as the set cover problem $\mathcal{S} = (X, \mathcal{R})$ where $\mathcal{R}$ is a set of pseudodisks with the PF and CF properties.

Theorem 3 *Any instance $\mathcal{S} = \{X, \mathcal{R}\}$ of PCF-SC may be reduced to an instance of a hitting set problem $\mathcal{H} = \{P, \mathcal{Q}\}$ in polynomial time, where P is a set of points, $\mathcal{Q}$ is a set of pseudodisks (both in $\mathbb{R}^2$), and $\mathcal{H}$ is a geometric dual of $\mathcal{S}$.*

The reduction progresses in two stages. First, the set cover instance is converted to a special graph known as a *planar support* $G = (V, E)$, where each vertex $v_i \in V$ corresponds to a pseudodisk $R_i \in \mathcal{R}$ and each point $x_j \in X$ maps to a connected induced subgraph S_j of G. Finally, we show how to fatten each of the subgraphs in the plane to form a pseudodisk, creating an instance of the hitting set problem on pseudodisks that is a geometric dual of the original set cover instance. We give an overview of the proof below, which is then proved formally in the remainder of Section 3:

[2] This property is only used in the proof of Lemma 9.



(a) Case 1a. R_i covers all of $R_j \cap R_k$.

(b) Case 1b. R_i covers none of $R_j \cap R_k$.

(c) Case 2 (and Case 5). R_i covers one point of $\partial R_j \cap \partial R_k$.

(d) Case 3 violates the PF property (R_i encloses $\partial R_j \cap \partial R_k$).

(e) Case 3 (and Case 4). R_i covers neither point in $\partial R_j \cap \partial R_k$.

(f) Case 6. This case is forbidden by the PF property.

Figure 1: Cases illustrating the different ways in which one pseudodisk may intersect two others. The figures are oriented so the walk on R_i is clockwise.

1. Convert the set cover instance to a planar support:

(a) Build the support by iterating over the regions of the plane in the arrangement of the pseudodisks in order of increasing depths (Algorithm 1).
(b) Show that the subgraph of the support imposed by any point is connected (Lemma 7).
(c) Show that the support is planar (Lemma 8).

2. Convert the planar support to a hitting set instance:

(a) Show that the support may be embedded orthogonally (Theorem 10).
(b) Show that each edge may be fattened to define a bounded region of the plane, and that these objects may be arranged and manipulated to become pseudodisks (Lemma 11).

3.1 Properties of Pair-Cover Free Set Cover

Various arguments in this discussion consider the possible ways in which three pseudodisks may interact, and so we enumerate all possible cases. Consider two pseudodisks R_j and R_k that each intersect a third pseudodisk R_i. Let j^+, j^-, k^+, k^- denote the set of events that occur during a clockwise walk around ∂R_i, where j^+ indicates the point of entry into R_j and j^- indicates the point of exit. We may arbitrarily begin our walk at j^+, and so there are 3! possible walks (see Figure 1), which we divide into cases. Note that for Cases 1–3, we begin outside R_k, while for Cases 4–6 we begin inside.

Case 1. j^+, j^-, k^+, k^-: A pseudodisk R_i may either cover $R_j \cap R_k$ completely or not at all depending on how the path is closed to create a pseudodisk. Call these Cases 1a (Figure 1a) and 1b (Figure 1b), respectively.

Case 2. j^+, k^+, j^-, k^-: R_i covers exactly one point in $\partial R_j \cap \partial R_k$, as shown in Figure 1c.

Case 3. j^+, k^+, k^-, j^-: R_i covers either both points or neither point in $\partial R_j \cap \partial R_k$. However, covering both points entails violating the PF property, since $R_k \subset R_i \cup R_j$ in this scenario (see Figure 1d). Therefore, the only valid scenario for Case 3 is that where R_i covers neither point in $\partial R_j \cap \partial R_k$, shown in Figure 1e.

Case 4. j^+, j^-, k^-, k^+: This is symmetric with Case 3 (swap the labels j and k).

Case 5. j^+, k^-, j^-, k^+: This is symmetric with Case 2.

Case 6. j^+, k^-, k^+, j^-: The beginning of the tour is in $R_j \cap R_k$. The next event is k^-, so the path is now in $R_j \setminus R_k$, and the following event is k^+, so both intersection points of $\partial R_i \cap \partial R_k$ are in R_j. The last event j^- is in R_k, and so both points of $\partial R_i \cap \partial R_j$ are in R_k, implying that $R_i \subseteq R_j \cup R_k$ which violates the PF property.

Therefore, the distinct cases are 1a, 1b, 2, and 3. Lemma 4 is an immediate consequence of Case 2:

Lemma 4 *Given any instance of PCF-SC, if a pseudodisk R_i intersects the boundaries of two other pseudodisks (say ∂R_j and ∂R_k) in the closed region $R_j \cap R_k$, then exactly one point of $\partial R_j \cap \partial R_k$ is covered by R_i.*

3.2 Building the Support

To build the support we describe how to construct an *adjacency list* that is a supergraph of the support and a subgraph of the intersection graph of $\mathcal{R}$; as we explain later, the support may be derived from the adjacency list. The support G has the property that if $(v_i, v_j) \in G$ then $R_i \cap R_j \neq \emptyset$, but the reverse is not necessarily true. Rule 1 holds for the adjacency list and the support G:

Rule 1 *For any three pseudodisks $\{R_i, R_j, R_k\}$, if $R_i \cap R_j \subseteq R_k$ then there is no edge (v_i, v_j) in the support G.*

The adjacency list is stored as a map from each vertex v_i to the set of neighbouring vertices, where a neighbour vertex corresponds to a pseudodisk that intersects R_i in the primal without violating Rule 1. A non-empty intersection between pseudodisks R_1 and R_2 does not necessarily imply the existence of the edge (v_1, v_2) in the adjacency list; it does, however, imply the following lemma:

Lemma 5 *For every $i \geq 1$ and $R' = \{R_{a_1}, \ldots, R_{a_i}\} \subseteq \mathcal{R}$, where R' is the set of all pseudodisks covering some cell of the arrangement, then G' is connected, where G' is the subgraph of the adjacency list induced by the vertices $v_{a_1}, v_{a_2}, \ldots, v_{a_i}$ associated with R'.*

Proof. Suppose otherwise. That is, there exists $R' = \{R_{a_1}, \ldots, R_{a_i}\} \subseteq \mathcal{R}$ such that $\bigcap_{1 \leq j \leq i} R_{a_j} \neq \emptyset$ and the corresponding induced subgraph of the adjacency list is disconnected. Suppose R_{a_a} and R_{a_b} in R' lie in separate components in the induced subgraph. Since there is no edge (v_{a_a}, v_{a_b}) in the subgraph, Rule 1 must have been applied such that $R_{a_a} \cap R_{a_b} \subseteq R_{a_c}$ for some R_{a_c}. Consequently, $R_{a_c} \in R'$. Without loss of generality, suppose R_{a_a} and R_{a_c} lie in separate components. Again, Rule 1 must have been applied such that $R_{a_a} \cap R_{a_c} \subseteq R_{a_d}$ for some $R_{a_d} \in R'$. Observe that $R_{a_a} \cap R_{a_b} \subseteq R_{a_a} \cap R_{a_c} \subseteq R_{a_d}$. Since R' is finite, this argument cannot be applied indefinitely, leading to a contradiction. □

The support is built iteratively using the *depths* of regions of the plane in the primal, where the depth is defined as the number of pseudodisks entirely covering that region of the plane (a *region* in this case is a cell in the arrangement of the set of pseudodisks), and so a region with depth k is covered by k pseudodisks. Let $\mathcal{A}(\mathcal{R})$ denote the arrangement defined by the pseudodisks in $\mathcal{R}$. Note that although some cells of the arrangement do not necessarily contain a point of X in the primal, we create a subgraph in the support for each cell in the arrangement. We show that a pseudodisk may be created in the dual for each cell, and those not needed in the dual may be discarded later. The algorithm for building the support is sketched in Algorithm 1.

Algorithm 1 BUILD-SUPPORT($\mathcal{S} = \{X, \mathcal{R}\}$)

1: **Input:** An instance of the PCF-SC problem $\mathcal{S}$.
2: **Output:** $G = (V, E)$, a planar support for the dual of $\mathcal{S}$.
3: Insert a vertex v_i in V for each R_i in $\mathcal{R}$.
4: Consider the arrangement of the plane imposed by the pseudodisks $\mathcal{R}$, call it $\mathcal{A}(\mathcal{R})$. Sort the cells of $\mathcal{A}(\mathcal{R})$ in order of increasing depth. An element $Z \in \mathcal{A}(\mathcal{R})$ is defined by a subset of $\mathcal{R}$.
5: For each vertex $v_i \in V$, compute the adjacency list.
6: For each region of depth 1, add a self-loop to the corresponding vertex in G (Figure 2).
7: **for** each region $Z \in \mathcal{A}(\mathcal{R})$ (in order of depth $= 2 \rightarrow |\mathcal{R}|$) **do**
8: If the subgraph of G induced by the vertices corresponding to the set of pseudodisks $\mathcal{R}'(\subseteq \mathcal{R})$ that cover Z has two or more connected components, then iteratively add edges to join pairs of components using edges selected from the adjacency lists until the induced subgraph is connected.
9: **end for**
10: **return** G

Each region of depth 1 corresponds to a pseudodisk that uniquely covers some region of the plane in the primal (one pseudodisk may cover many such regions). Each vertex in G corresponding to such a pseudodisk is given a self-loop in E so that each cell in the arrangement corresponds to a non-empty set of edges in E.

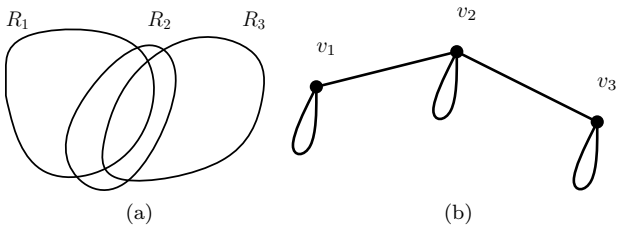


Figure 2: Building the neighbourhood graph. (a) Three pseudodisks R_1, R_2, R_3, where $R_1 \cap R_3 \subseteq R_2$. (b) Each pseudodisk has a self-edge in G, and there are edges (v_1, v_2) and (v_2, v_3). The region of depth 3 in (a) is covered by R_1, R_2, R_3, but in the graph v_1, v_2, v_3 form a connected subgraph of G, so no further edges are required.

The iterative procedure continues by considering regions of increasing depth, although edges are only added to G if the subgraph induced by the set of vertices corresponding to the region is not already connected. If the subgraph is not connected, then a pair of vertices (v_i, v_j) is selected so that v_i and v_j are in separate components of the subgraph and v_j is in the adjacency list for v_i. As shown in Lemma 5, some such pair must always exist, and so edges are added until the induced subgraph is connected. The algorithm can be made consistent by imposing a total ordering on the edges using their labels as keys, and always choosing the first edge in the ordering that connects components of the graph. The algorithm for connecting induced subgraphs operates somewhat analogously to Kruskal's minimum spanning tree algorithm, where edges already in G have zero weight, edges permitted by the adjacency lists are given unit weight, and those not permitted have infinite weight. Our approach has additional complexities, however, because we are operating on subgraphs of the support, and we must take care when adding edges to maintain the planarity of the support.

3.3 Planarity of the Support

The relative order of the edges around a vertex may be defined unambiguously by the objects in the primal. Call the pseudodisk R_2 a *neighbour* of pseudodisk R_1 if an edge between v_1 and v_2 exists in G. Let R_i be a region of the plane which is intersected by both R_j and R_k, as we examined in Figure 1. If there is an unambiguous sense that $R_j \cap \partial R_i$ is clockwise or counterclockwise of $R_k \cap \partial R_i$ w.r.t. any $R_\ell \cap \partial R_i$ on ∂R_i, then the edges (v_i, v_j) and (v_i, v_k) must have the same relative ordering w.r.t. (v_i, v_ℓ) around v_i in G, for each such R_ℓ where the edge (v_i, v_ℓ) exists. For brevity going

forward, we simply refer to the relative order of R_j and R_k on ∂R_i.

The boundaries ∂R_j and ∂R_k may intersect 0, 1, or 2 times in R_i. If 0 times, then we are in either Case 1b or Case 3 (refer to §3.1). If 1b, then $R_i \cap R_j \cap R_k$ is empty and the relative order of R_j and R_k on ∂R_i is unambiguous. Case 3 does not arise, as it implies that either $R_i \cap R_j \subseteq R_k$ or $R_i \cap R_k \subseteq R_j$, which means $(v_i, v_j) \notin G$ or $(v_i, v_k) \notin G$ respectively by Rule 1. If $|\partial R_j \cap \partial R_k \cap R_i| = 1$, i.e. Case 2, then necessarily $|\partial R_i \cap \partial R_k \cap R_j| = 1$ and $|\partial R_i \cap \partial R_j \cap R_k| = 1$ as well, by Lemma 4. In this case, there is an unambiguous sense where one of R_j or R_k is clockwise of the other on R_i. If $|\partial R_j \cap \partial R_k \cap R_i| = 2$, then Case 1a applies and so $R_j \cap R_k \subseteq R_i$, which means the relative ordering of the pseudodisks R_j and R_k on R_i is again unambiguous. Therefore, relative orderings may always be consistently applied to the edges of G in the embedding.

A cycle in G corresponds to a set of pseudodisks that partitions the plane into an unbounded region and a (possibly empty) set of bounded regions.

Lemma 6 *There is a deterministic method for creating a cycle C in the embedding of G so that the clockwise ordering of the vertices in C is defined by corresponding pseudodisks in the primal.*

See Appendix A for the proof of Lemma 6.

The support now contains subgraphs corresponding to cells of the arrangement in the primal, where each vertex of the support corresponds to a pseudodisk in the primal that covers the point corresponding to the subgraph. This is immediate from the construction, since the point must exist in one of the regions of the plane used to build the graph, and a subgraph is built for each region. G adheres to the definition of a support for the dual, since each necessary subgraph is a connected induced subgraph of G, which gives the following lemma:

Lemma 7 *For any point x_j in the primal, and all vertices $v_i \in V$ in the support corresponding to pseudodisks $R_i \in \mathcal{R}$ in the primal, there exists a connected subgraph $S_j \subseteq G$ where $v_i \in S_j$ if and only if $x_j \in R_i$.*

Lemma 8 *The support G is planar.*[3]

See Appendix A for the proof of Lemma 8.

3.4 Dual Properties of the Support

The support G encapsulates some of the combinatorial structure of the dual; to complete the dual we must construct a set of pseudodisks defined by the connected subgraphs on G that correspond to the points in the primal. The subgraphs have several characteristics that allow the creation of the dual hitting set instance.

Lemma 9 *Let C denote a cycle in an induced subgraph S of G in the embedding, where S corresponds to a point x in the primal and C corresponds to pseudodisks $\mathcal{R}_C$. Any vertices on the interior of the bounded region defined by C must correspond to pseudodisks that cover x in the primal.*

See Appendix A for the proof of Lemma 9.

3.5 Building the Hitting Set Instance

We now describe how to embed the support G in the plane and transform it into the dual hitting set instance. To begin, remove subgraphs from the support that do not correspond to points in X in the primal. We construct a planar orthogonal box drawing[4] of the support G using the following result:

Theorem 10 (Biedl and Kaufmann (1997) [4]) *Given a planar triconnected graph $G = (V, E)$, a planar orthogonal box drawing of G can be drawn in $O(m + n)$ time on a $(m - n + 1) \times \min\{m - n + 1, m/2\}$ grid with $m - n$ edge bends, where $n = |V|$ and $m = |E|$.*

The drawing of the support remains planar, and while G is not necessarily triconnected, one may add dummy edges to make it triconnected and then remove the dummy edges once the drawing is computed [9]. The placement of the point set P for the dual hitting set instance is simple: place a point inside each of the vertices (boxes) of the orthogonal drawing of the support, and these will dualize the corresponding pseudodisks of $\mathcal{R}$ in the primal (and so $|P| = |V|$). Now we describe how to create the set of pseudodisks $\mathcal{Q}$ for the dual hitting set instance.

A finer grid is imposed upon the orthogonal drawing with a resolution of $1/(2m+1)$, where m is the number of pseudodisks needed in the dual, i.e., the cardinality of X in the primal. The pseudodisks that are created for the dual are orthogonal polygons with edges incident upon the lines in this finer grid. An edge e is made *k-fat* by taking the Minkowski sum of e with $[-k, k] \times [-k, k]$. We fatten parts of the edges as necessary using points on the refined grid, so that parts of the edges may be $(k/(2m + 1))$-fat, for $k \in [0, \ldots, m]$, (i.e., all edges of the drawing are less than $(1/2)$-fat).

If all edges of a subgraph are grown to be k-fat (for possibly varying values of k), the subgraph defines a polygon. Any introduced holes are removed (Lemma 9

[3] Note that if our goal was to simply derive a PTAS for this class of set cover problems, we could stop here by showing that the PTAS of [14] applies given that the support has the requisite locality property. However our primary goal is to demonstrate the existence of duality, so we proceed nonetheless.

[4] A box drawing is a graph with orthogonal non-overlapping edges, where vertices may be drawn as rectangles in order to accommodate all incident edges.

established the validity of this action, since removing a hole never causes the simplified region to cover any new vertices of G).

The planar regions are constructed iteratively. For subgraph $S_1 = (V_1, E_1)$, create a polygon so that the subgraph is $(1/(2m+1))$-fat. Now S_2 is added, and any overlap must be resolved. Overlapping edges may be resolved by making one of the edges $(2/(2m+1))$-fat in the area of overlap. Repeating this operation of inserting and fattening existing edges for each subgraph creates a hitting set instance that dualizes the set cover instance, although the resulting objects are not necessarily pseudodisks. However, the removal of extraneous intersections is always possible so that all objects are pseudodisks. See Appendix B for details.

This establishes the following lemma:

Lemma 11 *All subgraphs of the support induced by a set of vertices corresponding to a point in the primal may be enclosed with a region of the plane so that all such regions are pseudodisks.*

Finally, any cell of the arrangement that contained k points in the primal requires $k-1$ additional pseudodisks in the dual. Since we are not concerned with the PF property in the dual, we nest the missing dual pseudodisks just inside the existing dual pseudodisk so that they all have the same combinatorial structure with respect to all other points and pseudodisks in the hitting set instance. These pseudodisks form the objects $\mathcal{Q}$ for the dual hitting set instance. If the dualization may be completed in polynomial time, then Theorem 3 follows.

Theorem 12 *Dualization of an instance of PCF-SC on pseudodisks to an instance of the hitting set problem on pseudodisks can be completed in $O(Im^5 \log m + mn)$ time, where $m = |\mathcal{R}| = |V| = |P|$, $n = |X| = |\mathcal{Q}|$, and I denotes the time required to compute the intersection points of a pair of pseudodisks.*

See Appendix A for the proof of Theorem 12.

4 Conclusions

Our examination of the geometric duality of set cover and hitting set problems on pseudodisks has revealed positive and negative results. Perhaps surprising is the fact that not all instances are dualizable. The construction of a geometric dual is possible on translates of an object, or when we restrict instances on pseudodisks to those that have what we call the pair-cover and quad-crossing free property. A corollary of the dualization is that there exists a PTAS for set cover problem on pseudodisks with this property. Our algorithm for the construction of the dual applies interesting techniques, as we make use of graph drawing techniques to build the dual from a planar support.

There remain several open questions. Our dualization technique requires that the primal setting have the PF property, while the dual instance that is created does not necessarily have this property. It would be preferable if the dual instance also had the PF property, but we conjecture that there exists a counter-example to show that such duality does not always exist. Finally, the dualization also requires that the quad-crossing free property applies, but this property does not seem tremendously important to the dualization. We conjecture that duality is possible on instances without the CF restriction.

References

[1] P.K. Agarwal, E. Nevo, J. Pach, R. Pinchasi, M. Sharir, and S. Smorodinsky. Lenses in arrangements of pseudo-circles and their applications. *J. ACM*, 51(2):139–186, 2004.

[2] P.K. Agarwal and M. Sharir. Pseudo-line arrangements: Duality, algorithms, and applications. *SIAM J. Comp.*, 34(3):526–552, 2005.

[3] I.J. Balaban. An optimal algorithm for finding segments intersections. In *Proc. SoCG*, 211–219. ACM, 1995.

[4] T. Biedl and M. Kaufmann. Area-efficient static and incremental graph drawings. In *Proc. ESA*, vol. 1284 *LNCS*, 37–52. Springer, 1997.

[5] K. Buchin, M. van Kreveld, H. Meijer, B. Speckmann, and K. Verbeek. On planar supports for hypergraphs. *J. Graph Alg. & Appl.*, 15(4):533–549, 2011.

[6] B. Chazelle, L.J. Guibas, and D.-T. Lee. The power of geometric duality. *BIT Num. Math.*, 25(1):76–90, 1985.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, 2nd ed.*. McGraw-Hill, 2001.

[8] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry*. Springer, 2010.

[9] H. Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990.

[10] J.E. Goodman. Proof of a conjecture of Burr, Grünbaum, and Sloane. *Disc. Math.*, 32(1):27–35, 1980.

[11] D. Bevan. Number of point subsets that can be covered by a disk. Math. Stack Exchange. http://math.stackexchange.com/q/75487.

[12] D.S. Johnson and H.O. Pollak. Hypergraph planarity and the complexity of drawing venn diagrams. *J. Graph Theory*, 11(3):309–325, 1987.

[13] N.H. Mustafa, R. Raman, and S. Ray. QPTAS for geometric set-cover problems via optimal separators. *CoRR*, abs/1403.0835, 2014.

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

[15] J. Steiner. *Gesammelte Werke*, vol. 1. Berlin, 1881.

A Appendix: Proofs

Proofs omitted from the main text due to space constraints appear in full in this section.

Theorem 1 *There exists a family of set cover problems on pseudodisks for which there is no dual hitting set formulation on pseudodisks (and equivalently, such hitting set problems on pseudodisks cannot be dualized into set cover instances on pseudodisks).*

Proof. Consider a set X of n points in the plane in general position. Let $\mathcal{R}$ be a maximal set of circular disks so that $R_i \cap X \neq R_j \cap X$ for any $R_i, R_j \in \mathcal{R}$; there are $\omega(n^2)$ such disks [11]. Since the arrangement of a set of n pseudodisks has at most $n^2 - n + 2$ cells [15], this is the maximum number of cells in the arrangement dualizing the n points of X. However, the $\omega(n^2)$ disks in the primal require $\omega(n^2)$ distinct cells in the arrangement, and so such an instance cannot be dualized. □

Theorem 2 *Problems defined on translates of any single object (including pseudodisks) may always be dualized.*

Proof. Given a canonical object C, choose a reference point r so that $r \in C$, and let $-C$ be the reflection of C through r. To create a dualization, replace every translated object with a similarly translated instance of r, and replace every point with a translation of $-C$ so that r is incident upon the point. Due to the reflection through r, every point (of the plane) in C maps to a unique point in $-C$, and this point maps back to the original point again. Therefore, a set R in the primal contains a point x if and only if the dual of R is in the dual of x. □

Lemma 6 *There is a deterministic method for creating a cycle C in the embedding of G so that the clockwise ordering of the vertices in C is defined by corresponding pseudodisks in the primal.*

Proof. Suppose we wish to add the edge (v_1, v_k) to the graph G, which will result in a new cycle C. Let (v_1, v_2) be the other edge incident upon v_1 in the cycle. We know that part of ∂R_1 lies outside of $R_2 \cup R_k$ by the PF property. Consider a very small pseudodisk R' that covers some point on $\partial R_1 \setminus R_2 \cup R_k$ as a reference, and assume that the edge (v_1, v') is required in G.

Any edge (v_i, v_{i+1}) in the cycle represents two vertices whose corresponding objects R_i and R_{i+1} in the primal have a non-empty area of intersection. By Rule 1, $R_i \cap R_{i+1}$ is not covered by any other object, and so there exists a point in $R_i \cap R_{i+1}$ outside of R_1. Therefore, we may place a point in $R_i \cap R_{i+1}$ in the primal for every edge (v_i, v_{i+1}) in the cycle. Given two consecutive edges of the cycle (v_i, v_{i+1}) and (v_{i+1}, v_{i+2}), there exists a path in the primal inside $R_{i+1} \setminus R_1$ from the point in $R_i \cap R_{i+1}$ to that in $R_{i+1} \cap R_{i+2}$, since the objects are pseudodisks (to prevent such a path, $R_{i+1} \setminus R_1$ would have to be disjoint). Let H be the path defined by joining all of the points defined by the cycle in this manner.

The points in $R_1 \cap R_2$ and $R_1 \cap R_k$ are in R_1, so the union of the path H with R_1 defines one unbounded region of the plane and at least one bounded region outside of R_1 (H need not be simple). The reference pseudodisk R' will either be on the boundary of the unbounded region or a bounded region. The relative order of the edges (v_1, v_2), (v_1, v_k), and (v_1, v') around v_1 is uniquely defined, as discussed earlier. Now the rule for closing the cycle is as follows: the cycle encloses v' in G if and only if R' is on the boundary of a bounded region of the plane defined by $H \cup R_i$ in the primal. □

Lemma 8 *The support G is planar.*

Proof. We establish that G is planar by demonstrating that edge crossings are never necessary under this scheme. Suppose the graph G has been built so that the support is planar so far, but the next edge to be added would violate planarity. Let one such edge be (v_1, v_2). Therefore, v_2 is in the adjacency list for v_1, and $R_1 \cap R_2$ contains the cell of the arrangement for which we are building an induced subgraph in the support.

If v_1 cannot be connected to v_2 with an edge while preserving planarity, then v_1 and v_2 belong to a connected component of G (otherwise we could place them in the same face), but they are separated by at least one cycle of edges C whose vertices are not part of the subgraph we are constructing (otherwise the vertex in the cycle would be part of some connected component that we could add an edge to). At minimum, the cycle C corresponds to a sequence of pseudodisks that are pairwise intersecting, and whose union is a bounded region of the plane and which may separate the plane into several regions. The cell of the arrangement must lie outside of all pseudodisks in C; any pseudodisk covering the cell has a vertex in the subgraph. One of the vertices in $\{v_1, v_2\}$ is on the interior of C and the other is not, and because the combinatorial structure of the primal is preserved in the support G, one of $\{R_1, R_2\}$ lies partially in the bounded region of the plane defined by the boundaries of the regions defining the cycle, and the other lies partially on the unbounded region. Therefore, either R_1 or R_2 must cross the pseudodisks in C to cover the cell in the arrangement. However, no pseudodisk may cross C. To do so would require either intersecting the boundary of a pseudodisk in C in at least four places (which violates the definition of a pseudodisk), or covering the area of intersection of at least two pseudodisks in C (which means that C is not a cycle, by Rule 1). □

Lemma 9 *Let C denote a cycle in an induced subgraph S of G in the embedding, where S corresponds to a point x in the primal and C corresponds to pseudodisks $\mathcal{R}_C$. Any vertices on the interior of the bounded region defined by C must correspond to pseudodisks that cover x in the primal.*

Proof. Consider a cycle C in S with a vertex v (corresponding to a pseudodisk R in the primal) where v has a neighbour

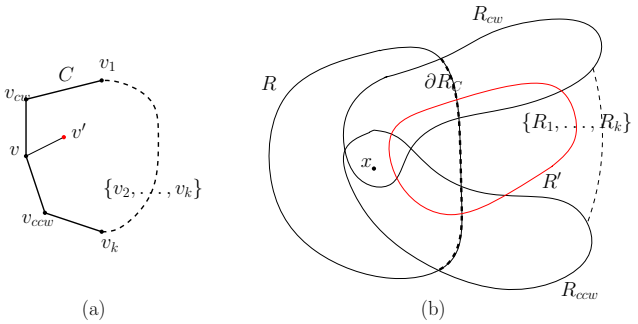


Figure 3: (a) Let C be a cycle in a subgraph S where a vertex $v' \notin S$ is in the interior of C, and the edge (v, v') exists in the support. v_{cw} (resp. v_{ccw}) is the clockwise (resp. counterclockwise) neighbour of v on the cycle. (b) All vertices in S correspond to pseudodisks covering a point x in the primal, and R' (the dual of v') does not cover x. The union of the pseudodisks dualizing the vertices in C (minus v) form a contiguous subset of the boundary of R which we call ∂R_C (drawn as the thick dashed line). R' covers a contiguous subset of ∂R_C.

v' on the interior of the cycle and $v' \notin S$. The pseudodisks in $\mathcal{R}_C \setminus R$ (in the primal) have a common area of intersection in R (since they all cover x and $x \in R$), and each edge of the cycle corresponds to a pair of pseudodisks that must intersect outside of R by Rule 1. It follows that the union of the pseudodisks in $\mathcal{R}_C$ covers a contiguous portion of the boundary of R, otherwise the boundaries of some pair of pseudodisks would have to intersect each other at four points. Let ∂R_C denote this portion of the boundary of R, as illustrated in Figure 3.

Now consider the two neighbours of v in C; call v_{cw} (resp. v_{ccw}) the clockwise (resp. counterclockwise) neighbour of v on C, and let R_{cw} (resp. R_{ccw}) denote the corresponding pseudodisk in the primal. Since v' lies on the interior of cycle C, by our construction algorithm, the entry point of R' lies between that of R_{cw} and R_{ccw} on ∂R_C. Consider a pseudodisk $R_i \in \mathcal{R}_C$ that contains the entry point of R'. The exit point of R' must be outside of R_i, otherwise either $R' \subset R_i \cup R$, which would violate the PF property, or $R \cap R' \subseteq R_i$, which would mean there is no edge (v, v') by Rule 1. Furthermore, the entry and exit points of R' cannot span those of any pseudodisk $R_j \in \mathcal{R}_C$. To do so would require that $|\partial R_j \cap \partial R' \cap R| = 2$, since R' cannot cover the point x in $R \cap R_j$, which would imply that $R_j \subseteq R \cup R'$, violating the PF property. Therefore, if the entry point of R' is in R_i, we can choose a pseudodisk R_j containing the exit point so that R_i and R_j are dualized by the vertices v_i and v_j and the edge (v_i, v_j) is in C.

Since the entry point of R' is in $R_i \setminus R_j$ and the exit point is in $R_j \setminus R_i$, we may conclude that $R_i \cap R_j \cap \partial R_C \subset R'$. Therefore, by Lemma 4, R' must cover either $R \cap R_i \cap R_j$ or $R_i \cap R_j \setminus R$. Of course, it cannot cover $R \cap R_i \cap R_j$ since this includes the point x, and we assumed that $x \notin R'$. However, R' cannot cover $R_i \cap R_j \setminus R$, since this implies that $\{R_i, R_j, R, R'\}$ is a crossing quad. Since either R' must cover x or v' must be outside the cycle C, we have a contradiction. □

Lemma 12 *Dualization of an instance of PCF-SC on pseudodisks to an instance of the hitting set problem on pseudodisks can be completed in $O(Im^5 \log m + mn)$ time, where $m = |\mathcal{R}| = |V| = |P|$, $n = |X| = |\mathcal{Q}|$, and I denotes the time required to compute the intersection points of a pair of pseudodisks.*

Proof. We show that the dualization can be completed in $O(Im^5 \log m + mn)$ time, where $m = |\mathcal{R}| = |V| = |P|$, $n = |X| = |\mathcal{Q}|$, and I denotes the time required to compute the intersection points of a pair of pseudodisks in the input set cover instance. Better analysis might establish a lower worst-case running time, but our goal is simply to establish polynomial running time.

The construction of the support takes $O(Im^5 \log m)$ time. The arrangement of a set of m pseudodisks has at most $m^2 - m + 2$ cells [15], and so the support has $O(m^2)$ subgraphs. In Algorithm 1, line 3 takes $O(m)$ time, line 4 may be completed in $O(Im^2 + m^2 \log m)$ time (sort the cells), line 5 may be done naïvely in $O(m^4)$ time, and line 6 may be completed in $O(m^2)$ time by traversing the arrangement. The loop iterates $O(m^2)$ times, and inside the loop we find the edges of the induced subgraph in G and then run Kruskal's minimum spanning tree algorithm to create a connected subgraph. Since we may have $O(m^2)$ edges in the graph, the running time is in $O(m^2 \log m)$ [7, p.570]. Checking whether a new edge violates planarity may be done in $O(m)$ time, but the bottleneck on the running time is when an edge closes a cycle. We can find the path in $O(Im)$ time, and the determination of whether the pseudodisk R' is on a closed face may be done in $O(Im)$ time by using one point in R' as a test for containment. This may be performed any time that an edge is added, so the work inside the loop takes $O(Im^3 \log m)$ time, giving the $O(Im^5 \log m)$ bound on the running time for building the support.

The algorithm for the construction of the hitting set instance from the support runs in $O(m^5 + mn)$ time. The orthogonal box drawing of the support runs in $O(m)$ time, and the determination of the positions of the points in P may be done at the same time. The insertion of a pseudodisk by fattening the edges of the corresponding subgraph may be done in $O(m^2)$ time, as each vertex of the subgraph may require that $O(m)$ other edges be fattened locally. There are $O(m^2)$ pseudodisks to be inserted, and so this may be done in $O(m^4)$ time. Any two objects have at most $O(m)$ intersections at this point, and each object is composed of $O(m)$ line segments, so the set of intersection points for the pair may be found in $O(m \log m)$ time [3], and these may placed in order around the boundary of one of the objects. Reducing the number of intersection points for the pair requires at most $O(m)$ iterations of the algorithm to remove pairs of intersection points. The algorithm may require determining which of two differences of the objects contains no points of P, and this may be done in $O(m^2)$ time by checking each point for containment, since each of the difference regions has $O(m)$ edges. Therefore, each pair may be

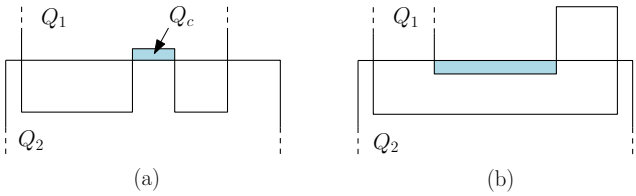


Figure 4: If ∂Q_1 and ∂Q_2 intersect in at least four places, each set of four consecutive intersections falls into one of two cases: (a) $Q_1 \setminus Q_2$ and $Q_2 \setminus Q_1$ are connected, or (b) $Q_1 \setminus Q_2$ and $Q_2 \setminus Q_1$ are not connected. The regions that may be resolved to reduce the number of intersections are shaded.

repaired into pseudodisks in $O(m^3)$ time, for a total upper bound of $O(m^5)$ time to convert all objects into pseudodisks. Finally, pseudodisks are added for each point in X that has no corresponding pseudodisk in $\mathcal{Q}$, i.e., from those cells in the arrangement containing more than one point from X. There are $O(n)$ such disks, and each may be placed in $O(m)$ time. □

B Appendix: Reducing the Number of Intersections

Consider a pair of objects, call them Q_1 and Q_2, enclosing S_1 and S_2 respectively, that are not pseudodisks. It has already been established that the objects are simple and enclosed by closed Jordan curves, so the only remaining possible violation is that the boundaries of Q_1 and Q_2 intersect more than twice. The sequence of events of a walk on ∂Q_1 (w.l.o.g.) must contain $2^+, 2^-, 2^+, 2^-$, and this gives rise to two cases to consider: local to these events, $Q_1 \setminus Q_2$ and $Q_2 \setminus Q_1$ are both either connected regions or not, as illustrated in Figure 4. The action taken to reduce the number of intersections while preserving the dual property depends on the case.

Case 1. If $Q_1 \setminus Q_2$ and $Q_2 \setminus Q_1$ are connected, then there must exist a bounded region of the plane Q_c outside of $Q_1 \cup Q_2$. We claim that we may remove the two points of intersection between Q_1 and Q_2 on the boundary of this region by moving the boundary of Q_1 (w.l.o.g.) to be just inside that of Q_2. This will not cause Q_1 to cover any additional points in $Q_2 \cap P$, so the only way that this move affects the hitting set combinatorially is if there exists any vertex v_c of G in Q_c, i.e. the bounded region of the plane that was formerly not covered by Q_1 or Q_2, but is now covered by Q_1.

Consider the cycle of $S_1 \cup S_2$ that encloses Q_c in the planar embedding of the support. As with the proof of Lemma 9, we note that if any vertex exists in Q_c, then at least one vertex v' exists that is a neighbour of a vertex v in $S_1 \cup S_2$, and say w.l.o.g. that $v \in S_1$. We define R_{cw} and R_{ccw} as before, and let R^i_{cw} be the first vertex in $S_1 \cap S_2$ on the cycle in a clockwise direction from v and R^i_{ccw} is defined analogously for the counterclockwise direction (R_{cw} and R^i_{cw} and also R_{ccw} and R^i_{ccw} are not necessarily distinct). There is a region of ∂R covered by the pseudodisks in the primal corresponding to the vertices of the cycle moving clockwise from R_{cw} to R^i_{cw}, and also analogously for R_{ccw} and R^i_{ccw}, call them ∂R_{cw} and ∂R_{ccw} respectively. In fact, since $R^i_{cw} \cap R^i_{ccw}$ contains points dualizing both S_1 and S_2, their pairwise area of intersection must extend across the boundary of R to cover points $x_1 \in R$ and $x_2 \notin R$ (S_i dualizes x_i), and so $\partial R_{cw} \cup \partial R_{ccw}$ defines a contiguous interval of ∂R. Now the same argument may be applied as in Lemma 9 to conclude that either R' covers x_1 (the dual of S_1), or else R' must be outside of the cycle. Therefore, removing intersections of the dual in this case may be done without covering additional vertices.

Case 2. If $Q_1 \setminus Q_2$ and $Q_2 \setminus Q_1$ are not connected, then one of the two regions of $Q_2 \setminus Q_1$ may be moved inside Q_1 unless both regions contain vertices of S_2. Suppose this is the case, and so an edge flip causes Q_1 to cover additional vertices of S_2. This implies that there is some path that is a subset of Q_1 for which Q_2 has vertices on both sides (i.e., the endpoints of the path could not be joined to create a cycle without enclosing vertices of Q_2). However, it was demonstrated that one subgraph cannot cross another in the proof of Lemma 9, so one of the regions of $Q_2 \setminus Q_1$ cannot contain vertices of S_2. Therefore, removing points of intersection may again be done without covering additional vertices. In both cases, there are no vertices in the regions where edges are moved to resolve conflicts. Therefore, these resolutions may be done without creating additional points of intersection with other objects by similarly translating other edges if necessary.

Conflict-free Covering*

Esther M. Arkin † Aritra Banik ‡ Paz Carmi ‡ Gui Citovsky † Matthew J. Katz ‡
Joseph S. B. Mitchell † Marina Simakov ‡

Abstract

Let $P = \{C_1, C_2, \ldots, C_n\}$ be a set of color classes, where each color class C_i consists of a set of points. In this paper, we address a family of covering problems, in which one is allowed to cover at most one point from each color class. We prove that the problems in this family are NP-complete (or NP-hard) and offer several constant-factor approximation algorithms.

1 Introduction

Let $P = \{C_1, C_2, \ldots, C_n\}$ be a set of color classes, where each color class C_i consists of a set of points. In this paper we address several closely related covering problems, in which one is allowed to cover at most one point from each color class. Before defining the problems, let us introduce some terminology. Let the set P of point color classes be on a line. We call an interval on the x-axis that contains at most one point from each color class a *conflict-free* interval (or CF-interval for short).

In this paper we consider the following problems.

Covering color classes with CF-intervals: Given a set P of point color classes on a line where each color class consists of a pair, find a minimum-cardinality set $\mathcal{I}$ of CF-intervals, such that at least one point from each color class is covered by an interval in $\mathcal{I}$.

Covering color classes with arbitrary unit squares: Given a set P of point color classes in the Euclidean plane where each color class consists of a vertically or horizontally unit separated pair of points, find a minimum-cardinality set $\mathcal{S}$ of unit squares (assuming a feasible solution exists), such that exactly one point from each color class is covered by a square in $\mathcal{S}$.

Covering color classes with a convex polygon: Given a set P of point color classes in the Euclidean plane where each color class consists of either a pair or a triple of points, decide whether or not there exists a convex polygon Q such that Q contains exactly one point from each color class. We also consider the related problem in which each color class consists of a pair of points and the goal is to maximize the number of color classes covered by a convex polygon Q, with Q containing exactly one point from each color class.

1.1 Related work

As far as we know, the first to consider a "multiple-choice" problem of this kind were Gabow et al. [7], who studied the following problem. Given a directed acyclic graph with two distinguished vertices s and t and a set of k pairs of vertices, determine whether there exists a path from s to t that uses at most one vertex from each of the given pairs. They showed that the problem is NP-complete. A sample of additional graph problems of this kind can be found in [2, 8, 13]. The first to consider a problem of this kind in a geometric setting were Arkin and Hassin [3], who studied the following problem. Given a set V and a collection of subsets of V, find a cover of minimum diameter, where a cover is a subset of V containing at least one representative from each subset. They also considered the multiple-choice dispersion problem, which asks to maximize the minimum distance between any pair of elements in the cover. They proved that both problems are NP-hard and gave $O(1)$-approximation algorithms. Recently, Arkin et al. [1] considered the following problem. Given a set S of n pairs of points in the plane, color the points in each pair by red and blue, so as to optimize the radii of the minimum enclosing disk of the red points and the minimum enclosing disk of the blue points. In particular, they consider the problems of minimizing the maximum and minimizing the sum of the two radii. In another recent paper, Consuegra and Narasimhan [4] consider several problems of this kind.

1.2 Our results

In Section 2 we consider the problem dealing with covering color classes, each consisting of a pair of points, with a minimum-cardinality set of CF-intervals. We prove that it is NP-hard by first proving that the following problem (covering color classes with a *given* set of CF-intervals) is NP-hard. Given a set P of point color classes and a set $\mathcal{I}$ of CF-intervals, find a minimum-

*Research supported by US-Israel Binational Science Foundation (project 2010074). E. Arkin and J. Mitchell partially supported by the National Science Foundation (CCF-1540890). A. Banik and M. Simakov partially supported by the Lynn and William Frankel Center for Computer Sciences.

†Applied Math & Statistics, Stony Brook University
‡Computer Science, Ben-Gurion University, Israel

cardinality set $\mathcal{I}' \subseteq \mathcal{I}$ (if it exists), such that, at least one point from each color class is covered by an interval in $\mathcal{I}'$. The latter proof is by a reduction from minimum vertex cover. The former proof also requires the following auxiliary result, which we state as an independent theorem. More precisely, we prove that minimum vertex cover remains NP-hard even when we restrict the underlying set of graphs to graphs in which each vertex is of degree at least $|V|/2$, where V is the set of vertices of the graph. We present a 4-approximation algorithm for this problem. We also present a 2-approximation algorithm for covering with arbitrary CF-intervals.

In Section 3 we consider the case where P is a set of point color classes in the Euclidean plane.

Suppose each color class consists of a pair and each pair of points from the same color class is unit distance apart, either vertically or horizontally separated. We show that finding a minimum-cardinality set $\mathcal{S}$ of axis parallel unit squares (assuming a feasible solution exists), such that exactly one point from each color class is covered by a square in $\mathcal{S}$ is NP-hard. We then present a 6-approximation algorithm.

We then consider the case that each color class consists of either a pair or triple of points. We show that deciding if there exists a convex polygon Q such that Q contains exactly one point from each color class is NP-complete. If each color class consists of a pair of points, we show that maximizing the number of color classes covered by Q is NP-hard. Finally, we consider the case that each color class consists of an arbitrary amount of points and all points from the same color class are vertically collinear. We (optimally) maximize the number of color classes covered (exactly one point from each color class) by Q in polynomial time.

2 Covering Color Classes

Let $P = \{C_1, C_2, \ldots, C_n\}$ be a set of n color classes, where each color class C_i is a pair of points $\{p_i, \overline{p_i}\}$ on the x-axis. We call an interval on the x-axis that contains at most one point from each color class a conflict free interval (CF-interval). A main goal in this section is to prove that the following problem is NP-hard; additionally, we give a 2-approximation.

Problem 1 Covering color classes with CF-intervals. *Find a minimum-cardinality set $\mathcal{I}$ of* arbitrary *CF-intervals, such that at least one point from each color class is covered by an interval in $\mathcal{I}$.*

Before presenting the proof, we prove that the problem in which one has to pick the covering CF-intervals from a given set of CF-intervals is NP-hard. We then use this result in our proof for Problem 1, together with an auxiliary result stated as Theorem 2 below.

2.1 Covering with a given set of CF-intervals

We prove that the following problem is NP-hard.

Problem 2 Covering color classes with a given set of CF-intervals. *Given a set $\mathcal{I}$ of CF-intervals, find a minimum-cardinality set $\mathcal{I}' \subseteq \mathcal{I}$ (if it exists), such that at least one point from each color class is covered by an interval in $\mathcal{I}'$.*

We describe a reduction from vertex cover. A *vertex cover* of a graph G is a subset of the vertices of G, such that each edge of G is incident to at least one vertex of the subset. Given a positive integer k, determining whether there exists a vertex cover of size k is an NP-complete problem [9]. Let $G = (V, E)$ be a graph, where $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$. We construct a set P of point color classes and a set $\mathcal{I}$ of CF-intervals, such that G has a vertex cover of size k if and only if there exists a subset $\mathcal{I}' \subseteq \mathcal{I}$ of size k that covers at least one point from each color class.

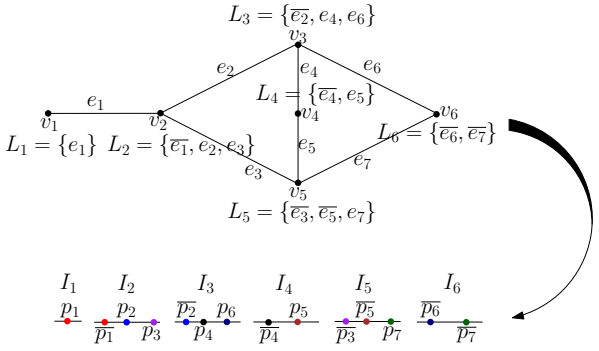


Figure 1: Reduction from vertex cover.

For each vertex v_i create an initially empty set L_i. For each edge $e_k = \{v_i, v_j\}$, where $i < j$, add e_k to L_i and $\overline{e_k}$ to L_j. Now, draw n disjoint intervals on the x-axis, one per set, such that interval I_{i+1} is to the right of interval I_i, $i = 1, \ldots, n-1$. Moreover, for each set L_i, draw $|L_i|$ arbitrary points on the interval I_i as follows. For each element in L_i, if it is of the form e_j, then add the point p_j to I_i, and if it is of the form $\overline{e_j}$, then add the point $\overline{p_j}$ to I_i. Finally, set $P = \{\{p_1, \overline{p_1}\}, \ldots, \{p_m, \overline{p_m}\}\}$ and $\mathcal{I} = \{I_1, \ldots, I_n\}$. See Figure 1 for an illustration.

It is easy to see that G has a vertex cover of size k if and only if there exist k intervals in $\mathcal{I}$ which together cover at least one point from each color class in P. Hence we have the following theorem.

Theorem 1 *Problem 2 is NP-hard.*

2.2 Covering with arbitrary CF-intervals

In order to show that Problem 1 is NP-hard, we first need to prove the following theorem, which says that minimum vertex cover remains NP-hard even when we restrict our attention to highly dense graphs.

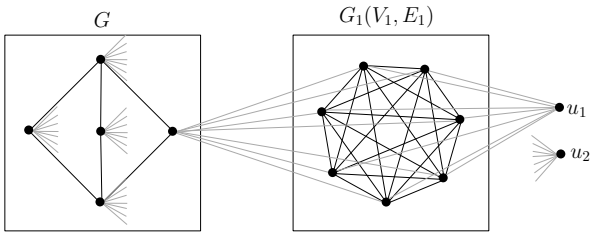


Figure 2: The graph G'.

Theorem 2 (Min vertex cover in dense graphs) *Finding a minimum vertex cover of a graph in which the degree of each vertex is at least $\frac{n}{2}$ is NP-hard, where n is the number of vertices in the graph.*

Proof. Let $G = (V, E)$ be any graph. We construct a new graph $G' = (V', E')$ in which the degree of each vertex is at least $\frac{|V'|}{2}$, and show that one can immediately obtain a minimum vertex cover of G from a minimum vertex cover of G' (and vice versa).

Let $G_1 = (V_1, E_1)$ be the complete graph of $|V| + 2$ vertices. We construct G' as follows. Set $V' = V \cup V_1 \cup \{u_1, u_2\}$, where u_1, u_2 are two new vertices. Set $E' = E \cup E_1 \cup E_2 \cup E_3$, where $E_2 = V \times V_1$ and $E_3 = V_1 \times \{u_1, u_2\}$ (see Figure 2). Notice that G' has the desired property, i.e., for each $v \in V'$, the degree of v (in G') is at least $\frac{|V'|}{2} = \frac{2|V|+4}{2} = |V| + 2$. (If v comes from V, then $deg_{G'}(v) = deg_G(v) + |V| + 2 \geq |V| + 2$, if v comes from V_1, then $deg_{G'}(v) = deg_{G_1}(v) + |V| + 2 \geq |V| + 2$, and if $v \in \{u_1, u_2\}$, then $deg_{G'}(v) = |V_1| = |V| + 2$.)

We now claim that given a minimum vertex cover of G', one can immediately obtain a minimum vertex cover of G, and vice versa. Let V^* be a minimum vertex cover of G'. We first show that $V_1 \subseteq V^*$. Since G' contains the complete graph G_1 of size $|V| + 2$, any minimum vertex cover of G' must include at least $|V| + 1$ vertices of V_1. If one of V_1's vertices, v, is not in V^*, then both u_1 and u_2 are necessarily in V^* (to cover the edges $\{v, u_1\}, \{v, u_2\}$). But, if so, V^* is not a minimum vertex cover, since $V^* \setminus \{u_1, u_2\} \cup \{v\}$ is also a vertex cover of G'. We conclude that $V_1 \subseteq V^*$. Notice that V_1 covers all the edges in E' except for the edges in E. Thus, the rest of the vertices in V^* consist of a minimum vertex cover of G. In other words, $V^* \cap V$ is a minimum vertex cover of G.

On the other hand, let $\tilde{V}$ be a minimum vertex cover of G, then $V_1 \cup \tilde{V}$ is a minimum vertex cover of G'. (Since, as shown above, V_1 is contained in any minimum vertex cover of G', and in order to cover the remaining uncovered edges, we need a minimum vertex cover of G.) □

Corollary 3 *Finding a minimum vertex cover of a graph $G = (V, E)$ in which the degree of each vertex is at least $\varepsilon|V|$, where $0 < \epsilon < 1$, is NP-hard.*

Proof. Similar to the proof of Theorem 2. □

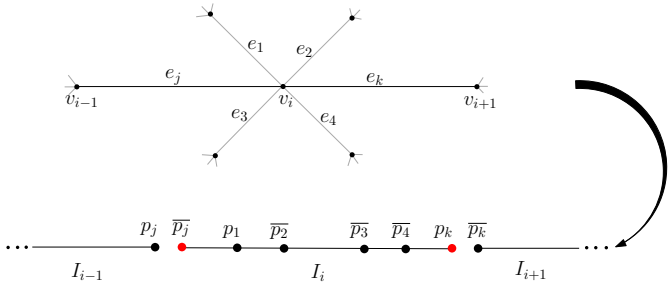


Figure 3: Illustration of Theorem 4.

We are now ready to prove that Problem 1 is NP-hard. We describe a reduction from minimum vertex cover in dense graphs (see Theorem 2 above). Let $G = (V, E)$ be any graph in which the degree of each vertex is at least $\frac{n}{2}$, where $n = |V|$. By Dirac's theorem [5] (or Ore's theorem [11]), G contains a Hamiltonian cycle; moreover, Palmer [12] presented a simple and efficient algorithm for computing such a cycle, under the conditions of Ore's theorem.

Let $v_1, v_2, \ldots, v_n, v_1$ be a Hamiltonian cycle in G. As for Problem 2, we construct a set P of point color classes. For each vertex $v_i \in V$, we construct a set L_i as follows. For each edge $e_k = \{v_i, v_j\}$ adjacent to v_i, we add e_k (resp., $\overline{e_k}$) to L_i, if $i < j$ (resp., $j < i$). We now draw n disjoint intervals on the x-axis, such that interval I_i corresponds to set L_i and precedes interval I_{i+1} (for $i < n$). We draw $|L_i|$ points in I_i as follows. Let $e_j = \{v_{i-1}, v_i\}$ and $e_k = \{v_i, v_{i+1}\}$. Then $\overline{e_j}, e_k \in L_i$. Place a point $\overline{p_j}$ corresponding to $\overline{e_j}$ at the left endpoint of I_i and place a point p_k corresponding to e_k at the right endpoint of I_i. In addition, place a point anywhere in the interior of I_i, for each of the other elements in L_i. For example, in Figure 3 e_j and e_k are the edges connecting v_i to v_{i-1} and to v_{i+1}, respectively, and e_1, e_2, e_3, e_4 are the other edges incident to v_i. The corresponding interval representation is shown in Figure 3.

Now, set $P = \{\{p_1, \overline{p_1}\}, \{p_2, \overline{p_2}\}, \ldots\}$ and $\mathcal{I} = \{I_1, \ldots, I_n\}$. Observe that I_i is conflict free (by construction), for $i = 1, \ldots, n$. Moreover, any other CF-interval is necessarily contained in one of the intervals already in $\mathcal{I}$ (since any interval that covers the right endpoint of I_i and the left endpoint of I_{i+1} is not conflict free). Thus, one might as well pick intervals from $\mathcal{I}$ when covering the color classes of P with a minimum number of arbitrary CF-intervals. But, by Theorem 1 this is NP-hard. Hence we have the following theorem.

Theorem 4 *Problem 1 is NP-hard.*

A 4-approximation algorithm for Problem 2. Let $P = \{C_1, C_2, \ldots, C_n\}$ be a set of point color classes (pairs, $C_i = \{p_i, \overline{p_i}\}$) on the set of points $\mathcal{P} = \bigcup_i C_i$. We assume there exists $\mathcal{I}' \subseteq \mathcal{I}$ such that $\mathcal{I}'$ covers at least one point from each color class and we provide a 4-

approximation algorithm for covering P with the fewest number of CF-intervals. For a given $p \in \mathcal{P}$, let $I_p \in \mathcal{I}$ be a CF-interval (if it exists) that covers p and extends farthest to the right among all intervals that cover p. Let $I_p^{(r)} \subseteq I_p$ be the subinterval of I_p that contains p and all points to the right of p.

Input: $P = \{C_1, C_2, \ldots, C_n\}$, a set of point color classes and $\mathcal{I}$, a set of CF-intervals.
Output: A subset $\mathcal{I}' \subseteq \mathcal{I}$ covering at least one point from each color class.
$\mathcal{T} = \emptyset$
while *there exists* $p \in \mathcal{P}$ *such that* p *is uncovered in* $\mathcal{T}$ *and there exists* $I \in \mathcal{I}$ *such that* $p \in I$ **do**
  Let p be the leftmost uncovered point in $\mathcal{P}$ that is contained in some interval in $\mathcal{I}$.
  $\mathcal{T} \leftarrow \mathcal{T} \cup I_p^{(r)}$
end
Compute a subset of intervals $\mathcal{T}$ to cover at least one point of each of the C_i's, using a low-frequency set cover approximation algorithm.
Let $\mathcal{I}'$ be the set of intervals $I_p \in \mathcal{I}$ corresponding to each $I_p^{(r)}$ of $\mathcal{T}$ in the resulting cover.

Algorithm 1: An algorithm for Problem 2.

Lemma 5 $|\mathcal{I}'| \leq 4|OPT|$.

Proof. Consider the set $\mathcal{T}$ of intervals at the end of the while loop. Let $OPT_{\mathcal{T}} \subseteq \mathcal{T}$ be an optimal set cover of the C_i's. First we claim that $|OPT_{\mathcal{T}}| \leq 2|OPT|$. Consider the leftmost point p in an arbitrary interval A of OPT. By the construction of Algorithm 1, we know that there must exist an interval $T \in \mathcal{T}$ that contains p. If there exists a point that is covered by A and not covered by T, then let q be the leftmost such point. We know there exists an interval $I_q^{(r)} \in \mathcal{T}$ that starts at q and extends at least as far to the right as does A. Thus, for any $A \in OPT$, there exist at most two intervals in T, the union of which entirely contains A.

Observe that since each newly added interval to $\mathcal{T}$ cannot contain a previously covered point, then, at the end of the while loop, each $p \in \mathcal{P}$ is contained in at most one interval of $\mathcal{T}$; thus, each pair C_i is covered by at most two intervals of $\mathcal{T}$ (one covering p_i, one covering $\overline{p_i}$). Therefore, we are approximating a low-frequency (at most 2) set cover instance, for which LP relaxation gives a 2-approximation [14] (pp. 119-120). Hence, we have $|\mathcal{I}'| \leq 2|OPT_{\mathcal{T}}| \leq 4|OPT|$. (For color classes of size at most c, we obtain a $2c$-approximation.) □

A 2-approximation algorithm for Problem 1. Let $P = \{C_1, C_2, \ldots, C_n\}$ be a set of point color classes (pairs, $C_i = \{p_i, \overline{p_i}\}$) on the set of points $\mathcal{P} = \bigcup_i C_i$. We provide a simple 2-approximation algorithm for covering P with arbitrary CF-intervals. For any point $p \in \mathcal{P}$, denote the maximal CF-interval starting at p and ending at a point of $\mathcal{P}$ to the right of p (or at p) by $I_{\max}(p)$.

Input: $P = \{C_1, C_2, \ldots, C_n\}$, a set of point color classes.
Output: A set $\mathcal{I}$ of CF-intervals.
$\mathcal{I} = \emptyset$
while $\mathcal{P} \neq \emptyset$ **do**
  Let p be the leftmost point in $\mathcal{P}$
  $\mathcal{I} \leftarrow \mathcal{I} \cup I_{\max}(p)$
  For each point of $\mathcal{P}$ that lies in $I_{\max}(p)$, remove it and its twin point from $\mathcal{P}$
end

Algorithm 2: A greedy algorithm for Problem 1.

Consider the set $\mathcal{I}$ computed by Algorithm 2. Clearly, $\mathcal{I}$ is a set of (disjoint) CF-intervals, such that at least one point from each color class is covered by the intervals of $\mathcal{I}$. It remains to prove that $\mathcal{I}$ is a 2-approximation of OPT, where OPT denotes any optimal solution.

Lemma 6 $|\mathcal{I}| \leq 2|OPT|$.

Proof. For any two points x and y, let $[x, y]$ (resp., (x, y)) denote the closed (resp., open) interval with endpoints x and y. Let $[p_a, p_b]$ and $[p_c, p_d]$ be two consecutive intervals in $\mathcal{I}$. Observe that since $[p_a, p_b]$ is a maximal CF-interval, there exists a point p_i (resp., $\overline{p_i}$) in $[p_a, p_b]$, such that $\overline{p_i}$ (resp., p_i) is in (p_b, p_c). Therefore any interval in OPT can intersect at most two intervals in $\mathcal{I}$. Moreover, since OPT must cover the color class $C_i = \{p_i, \overline{p_i}\}$, there exists an interval $I \in OPT$, such that $I \cap \{p_i, \overline{p_i}\} \neq \emptyset$. We thus conclude that $|OPT| \geq |\mathcal{I}|/2$. □

3 Two Dimensions

Let $P = \{C_1, C_2, \ldots, C_n\}$ be a set of n color classes in the Euclidean plane. We explore covering problems where exactly one point from each color class must be covered.

3.1 Unit Squares

Problem 3 Covering color classes with arbitrary unit squares. *Let $P = \{C_1, C_2, \ldots, C_n\}$ be in the Euclidean plane and let each color class C_i consist of a vertically or horizontally unit separated pair of points. Find a minimum-cardinality set $\mathcal{S}$ of axis-aligned unit squares (assuming a feasible solution exists), such that exactly one point from each color class is covered by a square in $\mathcal{S}$.*

Theorem 7 *Problem 3 is NP-hard.*

Proof. The reduction is from PLANAR 3-SAT [10], where one is given a formula in conjunctive normal form with at most three literals per clause, with the

objective of deciding whether or not the formula is satisfiable. Given variables $\{x_1, x_2, \ldots, x_n\}$ and clauses $\{c_1, c_2, \ldots, c_m\}$, we consider the graph whose nodes are the clauses and variables and whose edges join variable x_i with clause c_j if and only if $x_i \in c_j$ or $\neg x_i \in c_j$. The resulting bipartite graph, G, is planar.

In a manner similar to Fowler et al. [6], in a planar embedding of G we replace all of the edges incident to a variable node with a variable chain that visits the corresponding clauses and returns to the variable node to form a loop. The variable chains consist of a sequence of unit separated pairs (see Figure 4) and are designed in such a way that any minimum cardinality solution will either cover $\{\overline{a_{i+k}}, a_{i+k+1} : \text{k is even}\}$ or $\{\overline{a_{i+k}}, a_{i+k+1} : \text{k is odd}\}$. That is, for a given variable chain, either all blue unit squares or all red unit squares will be used. Using red (resp. blue) squares for variable x_i is equivalent to setting this variable to TRUE (resp. FALSE). Using planarity of the graph embedding, no two variable chains intersect, and any two points from different chains are spaced at least unit distance apart.

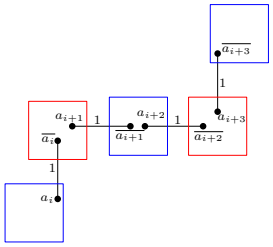


Figure 4: Variable chain.

Clause c_i consists of a single (green) pair (see Figure 5). If c_i evaluates to FALSE, then a square that is not associated with any variable loop will be needed to cover c_i. If c_i evaluates to TRUE, then a point from c_i can be covered by a square from an incoming loop whose literal evaluates to TRUE.

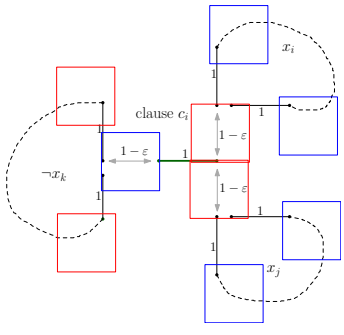


Figure 5: Clause gadget.

Let r_i be the number of pairs used in variable chain $i, 1 \leq i \leq 3m$. We design the variable chains so that r_i is even for all i. Let $r = \sum_i r_i$. It is now apparent that there exists a satisfying truth assignment in PLANAR 3-SAT if and only if a minimum cardinality covering with unit squares uses $\frac{r}{2}$ squares. □

Remark: If P is on a line and pairs are unit separated, we can minimize the number of unit intervals used in a complete cover (assuming a solution exists) in polynomial time using dynamic programming.

A 6-approximation algorithm. We lay out a grid with unit dimensions on top of our point set P and two-color the cells of the grid red and black in the style of a checkerboard. We say that a cell is occupied if it contains a point in P. Let R be the set of occupied red cells and B the set of occupied black cells. As a solution, we use the set of smaller cardinality.

Lemma 8 $\min\{|R|, |B|\} \leq 6|OPT|$.

Proof. Suppose w.l.o.g that $\min\{|R|, |B|\} = |R|$. Note that R is a feasible solution because any two points of a color class are unit separated either vertically or horizontally, thus one of the two points must occupy a red cell and the other must occupy a black cell. Therefore, R covers all color classes of points and no two points from the same color class are covered by R.

Now we claim that in the optimal solution, OPT, at least $\frac{1}{12}(|R| + |B|)$ unit squares are used. An arbitrary unit square, s, used in OPT stabs at most four cells of the checkerboard. These four cells are adjacent to at most eight other cells in total, each of which can be occupied by the pair of one of the points covered by s. Thus, at most 12 occupied cells of the checkerboard can be accounted for by any unit square used in OPT.

Combining the fact that $\min\{|R|, |B|\} \leq \frac{1}{2}(|R| + |B|)$ and $OPT \geq \frac{1}{12}(|R| + |B|)$, we have that $\min\{|R|, |B|\} \leq 6|OPT|$. □

3.2 Covering with a Convex Polygon

Problem 4 *Let $P = \{C_1, C_2, \ldots, C_n\}$ be in the Euclidean plane and let each color class C_i consist of either a pair or a triple of points. Decide whether or not there exists a convex polygon Q such that Q contains exactly one point from each color class.*

Problem 5 *Let $P = \{C_1, C_2, \ldots, C_n\}$ be in the Euclidean plane and let each color class C_i consist of a pair of points. Maximize the number of color classes covered by a convex polygon Q such that Q contains exactly one point from each covered color class.*

Theorem 9 *Problem 4 is NP-complete.*

Proof. Problem 4 is clearly in NP because we can check whether or not polygon Q is convex and whether or not Q contains exactly one point from each color class

in polynomial time. We present a reduction from EXACTLY 1-IN-3-SAT, where one is given a formula in conjunctive normal form with at most three literals per clause, with the objective of deciding whether or not the formula is satisfiable. In a satisfying assignment, every clause must contain exactly one TRUE literal.

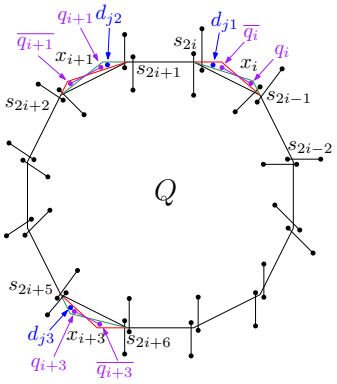


Figure 6: Construction of hardness for Problem 4.

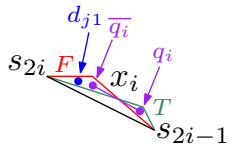


Figure 7: Close-up of variable gadget for Problem 4.

Given variables $\{x_1, x_2, \ldots, x_n\}$ and clauses $\{c_1, c_2, \ldots, c_m\}$, we start by considering $2n$ points, $S = \{s_1, s_2, \ldots, s_{2n}\}$, in the position of a regular $2n$-gon. These $2n$ points are not part of any color class; we use them to help explain the construction. We place two pairs of points around each point of S in such a way that convex polygon Q must have vertices at each point of S (see Figure 6). We create a variable gadget x_i in between points s_{2i-1} and s_{2i} for $1 \leq i \leq n$. Each variable gadget consists of color class that is a pair of points $\{q_i, \overline{q_i}\}$, $1 \leq i \leq n$ (see Figure 7). We place $\{q_i, \overline{q_i}\}$ so that Q can be expanded to cover either q_i (green lines in Figure 7) or $\overline{q_i}$ (red lines in Figure 7), while remaining convex. Setting x_i to TRUE (resp. FALSE) corresponds to expanding Q to cover q_i (resp. $\overline{q_i}$). If x_i (resp. $\neg x_i$) appears in clause c_j, a point from a color class that contains triple $\{d_{j1}, d_{j2}, d_{j3}\}$ will appear in the expansion of Q that covers q_i (resp. $\overline{q_i}$), and not in the expansion of Q that covers $\overline{q_i}$ (resp. q_i). It is now apparent that there exists a satisfying truth assignment in EXACTLY 1-IN-3-SAT if and only if convex polygon Q covers exactly one point from each color class. □

Theorem 10 *Problem 5 is NP-hard.*

Proof. The reduction is from MAX EXACTLY 1-IN-2-SAT where each clause has at most two literals and the objective is to maximize the number of clauses that evaluate to TRUE. A clause evaluates to TRUE if and only if it contains exactly one TRUE literal. Using the same construction as in Problem 4, it is easy to see that maximizing the number of TRUE clauses is equivalent to maximizing the number of color classes covered. □

References

[1] E. M. Arkin, J. M. Díaz-Báñez, F. Hurtado, P. Kumar, J. S. B. Mitchell, B. Palop, P. Pérez-Lantero, M. Saumell, and R. I. Silveira. Bichromatic 2-center of pairs of points. *Comput. Geom.*, 48(2):94–107, 2015.

[2] E. M. Arkin, M. M. Halldórsson, and R. Hassin. Approximating the tree and tour covers of a graph. *Information Processing Letters*, 47(6):275–282, 1993.

[3] E. M. Arkin and R. Hassin. Minimum-diameter covering problems. *Networks*, 36(3):147–155, 2000.

[4] M. E. Consuegra and G. Narasimhan. Geometric avatar problems. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2013*, volume 24 of *LIPIcs*, pages 389–400, 2013.

[5] G. A. Dirac. Some theorems on abstract graphs. *Proc. London Mathematical Society, series 3*, 2(1):69–81, 1952.

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

[7] 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*, 2(3):227–231, 1976.

[8] O. Hudec. On alternative p-center problems. *Zeitschrift fur Operations Research*, 36(5):439–445, 1992.

[9] R. M. Karp. *Reducibility among combinatorial problems*. Springer, 1972.

[10] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.

[11] O. Ore. Note on hamilton circuits. *The American Mathematical Monthly*, 67(1):p. 55, 1960.

[12] E. M. Palmer. The hidden algorithm of Ore's theorem on hamiltonian cycles. *Computers & Mathematics with Applications*, 34(11):113–119, 1997.

[13] S. L. Tanimoto, A. Itai, and M. Rodeh. Some matching problems for bipartite graphs. *Journal of the ACM (JACM)*, 25(4):517–525, 1978.

[14] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag New York, Inc., New York, NY, USA, 2001.

Space Filling Curves for 3D Sensor Networks with Complex Topology

Mayank Goswami* Siming Li† Junwei Zhang† Emil Saucan‡ David Xianfeng Gu† Jie Gao†

Abstract

Several aspects of managing a sensor network (e.g., motion planning for data mules, serial data fusion and inference) benefit once the network is linearized to a path. The linearization is often achieved by constructing a space filling curve in the domain. However, existing methods cannot handle networks distributed on surfaces of complex topology.

This paper presents a novel method for generating space filling curves for 3D sensor networks that are distributed densely on some two-dimensional geometric surface. Our algorithm is completely distributed and constructs a path which gets uniformly, progressively denser as it becomes longer. We analyze the algorithm mathematically and prove that the curve we obtain is dense. Our method is based on the Hodge decomposition theorem and uses holomorphic differentials on Riemann surfaces. The underlying high genus surface is conformally mapped to a union of flat tori and then a proportionally-dense space filling curve on this union is constructed. The pullback of this curve to the original network gives us the desired curve.

1 Introduction

In this paper we consider sensors deployed in 3D space such that the sensors are located densely on some underlying 2-dimensional geometric surface of possibly complex topology. This assumption models many practical scenarios in sensor deployment — sensors are often attached to the surfaces of terrains, exterior/interior of buildings [13], or other architectural structures, for easy installation and energy supplies, etc. In some other cases, the applications require sensors to be installed to monitor complex 3D structures, such as underground tunnels [14, 29] or pipes [22]. Therefore the sensors are located sparsely in 3D space but densely on a 2-dimensional surface (the "boundary" of some 3D objects) of possibly complex topology.

We are interested in innovative ways of managing such sensor networks, in the regime of using mobile entities to aid such management. Such mobile agents are often termed 'data mules', since one of the major applications is to use a mobile node to collect data from static sensors [1, 10, 17, 23, 27]. Static data sinks suffer from the well known problem of 'energy hole', as sensors near the sink are used more often and may run out of battery sooner than others. Mobile data sinks could get around this problem.

Our focus is to plan data mule along a path that uniformly traverses the entire field for a sensor network. This represents a periodic solution by which all sensors have a fair chance of being served by the data mule. It is better for scenarios when sensors have the same data generation rate, or when the sensor network requires a patrolling team to continuously monitor its general functioning and health. It can also be used for linear, logical operations in sensor networks such as data fusion [20] or sequential inference. Thus, the main questions is: given a unit-disk-graph G on n nodes (sensors) densely placed on a smooth two-dimensional manifold, construct a path γ that 1) passes through each node, and 2) for any integer $\ell = \Omega(\text{diameter}(G))$ and any subset A of nodes, the proportion of nodes among the first ℓ nodes in γ that lie in A is equal to the relative size of A.

The above problem is hard in general; however, if the sensors are densely distributed, one can attack a continuous version of it. The continuous version basically asks to construct a space filling curve on a two dimensional manifold M, which gets progressively dense (for any $\ell = \Omega(\text{diameter}(M))$ and any submanifold $A \subset M$, the proportion of γ_ℓ (the first ℓ length of γ) that lies inside A is equal to the relative area of A). The hope is that by "thickening" a solution to the continuous version, we might get a reasonable solution to the discrete version.

The representative work in this direction is by Ban *et al.* [4], which generalizes the idea of a space filling curve, often defined for a square, for a general 2D domain with holes. A space filling curve is a single curve that recursively 'fills up' the square, when the number of iterations goes to infinity [21]. For a sensor network with fixed density, a space filling curve nicely tours around all sensors with total travel length comparable to the traveling salesman solution. When the sensor network has holes, however, the space filling curve is broken and loses its nice properties. For a domain with a single hole, Ban *et al.* [4] proposed to map it to a torus such that a space filling curve can be easily found — by essentially following a line bouncing back and forth between the inner and outer boundaries. When there are two or more holes, all but one holes are mapped to 'slits', and the path bounces on these slits too. It is proved that this curve is dense, i.e., any point of the domain will be covered by the path of sufficiently long; and the curve has bounded density, i.e., it does not path through any point too many times.

However, one major limitation of the mechanism in Ban *et al.* [4] is its applicability to 2D domains with holes

*Max Planck Institute for Informatics, Saarbrücken, Germany gmayank@mpi-inf.mpg.de

†Computer Science Department, Stony Brook University, NY, USA {silli,junwezhang,gu,jgao}@cs.stonybrook.edu

‡Technion, Israel Institute of Technology; Max Planck Institute for Mathematics, Leipzig, Germany semil11@gmail.com

only. Terrains with holes can be handled via an additional mapping to 2D but general surfaces with high genus (multiple handles) cannot be handled. For underground deployment of sensors for monitoring tunnels, handles appear very often and we need a different scheme for generating the space filling curves.

Our contribution. The main result in this paper is a new linearization scheme for general sensor networks on 2D surfaces. We generate space filling curves with the same nice properties as those in [4]. In particular, the curves have 1) dense, progressive coverage – that as the curve gets longer, the distance from any point to the curve decreases quickly; 2) efficient coverage – a point is not visited more than a constant number of times.

Moving from a 2D domain to a general 2D surface in 3D really makes the problem harder. Note that before this work, there was no algorithm to construct the desired space-filling curve on a surface, even in the continuous setting. We remark that the problem is much easier if one drops the progressive density requirement. For that one can cut the 2D surface into small patches each mapped to a 2D domain such that previous methods can be applied.

To summarize, our contribution is not only on the algorithmic and application aspects, but also presents a theoretically proven, continuous-setting algorithm, that may be of independent interest. The algorithms presented here can be easily made to work in a distributed setting for a sensor network.

2 Related Work

In this section we survey other ideas that generate a path to visit all sensor nodes.

Space filling curves. Space filling curves have been used for linear/serial fusion [20] in sensor networks when the sensors are deployed uniformly in a square. There has been a heuristic algorithm that generalizes a Hilbert curve for an ellipse [11]. Technically the curves generated by Ban *et al.* [4] and the one in this paper are not going to completely fill up the surface (since topologically a curve is different from a surface) – but both the curves get infinitesimally closer to any given point.

Finding a tour. On a sensor network deployed in space, generating a tour of the graph, depending on the requirement, maps to either the Hamiltonian cycle/path problem or the traveling salesman tour. The former requires each vertex be visited exactly once and only the edges of the graph can be used. The second tries to minimizes the total travel distance instead. Both problems are NP-hard [9]. Euclidean TSP has good approximation schemes [3, 19] but these solutions suffer from two potential problems: 1) lack of progressive density; 2) cannot support multiple data mules easily.

Random walk. A practically appealing solution for visiting nodes in a network is by random walk. The downside is that we encounter the coupon collector problem. Initially a random walk visits a new node with high probability. After a random walk has visited a large fraction of nodes, it is highly likely that the next random node encountered has been visited before. Thus it takes a long time to aimlessly walk in the network and hope to find the last few unvisited nodes. Theoretically for a random walk to cover a grid-like network, the number of steps is quadratic in the size of the network [15]. For a random walk of linear number of steps, there are a lot of duplicate visits as well as a large number of nodes that are not visited at all. In the case of multiple random walks, since there is little coordination between the random walks, they may visit the same nodes and duplicate their efforts.

3 Theory of constructing space filling curves

For ease of exposition, we start by summarizing our method in this section. We then describe the theory behind our constructed curve, and end this section with the proof that the curve is dense. For the sake of completeness, we have provided all the theoretical material necessary for understanding our construction.

3.1 Informal discussion of techniques

Let us consider the mathematical problem of constructing a dense curve with the desired property of *proportional density* on a two dimensional manifold S. We first treat the surface as a one dimensional complex manifold, also called a Riemann surface. This basically means that locally our surface looks like an open set in the complex plane, and the transition maps from one such local "chart" to another are holomorphic.

With this point of view, we consider a *holomorphic differential* on our Riemann surface S. A holomorphic differential is basically an assignment of a complex-valued holomorphic function on each chart of the surface, that transforms line elements in the correct way; in complex coordinates z and $\bar{z}$, it is a tensor of type $(1, 0)$.

Using properties of certain special kinds of holomorphic differentials called Strebel differentials, we partition our surface into pieces, each of which is a flat torus with some holes removed. Each such piece is mapped to a parallelogram with slits (the boundaries of the holes map to the slits). In other words, we view the surface S as a union of parallelograms with slits, with slits being glued together in a certain way. This change of coordinates is mathematically termed a "branched covering".

In these coordinates, our curve is just a straight line on the cover. The slope of this line is either irrational, or chosen randomly, depending on the position of the slits and the sides of the parallelograms. Using several important and recent results in *Teichmüller theory*, we can prove density.

Note that although we partition the surface into pieces, we do not cover one piece first and then move on to the next. Instead our curve comes back into each piece infinitely often, increasing the density proportionally to the length.

3.2 Theoretic Background

Conformal Atlas Suppose $(S, \mathbf{g})$ is a surface with a Riemannian metric $\mathbf{g}$. Given any point $p \in S$, there is a neighborhood $U(p)$ where one can find the *isothermal coordinates* (i.e., local coordinates where the metric is conformal to the Euclidean metric) (x, y) on $U(p)$, such that

$$\mathbf{g} = e^{2\lambda(x,y)}(dx^2 + dy^2),$$

where the scalar function $\lambda : U(p) \to \mathbb{R}$ is the conformal factor function. The atlas consisting of isothermal coordinates is called a *conformal atlas*. In the following discussion, we always assume the local parameters are isothermal.

De Rham Cohomology De Rham cohomology theory is based on the existence of differential forms with certain prescribed properties. Suppose $f : S \to \mathbb{R}$; then its differential is given by

$$df(x, y) = \frac{\partial f(x, y)}{\partial x}dx + \frac{\partial f(x, y)}{\partial y}dy,$$

Suppose ω is a differential 1-form on the surface, which has local representation as $\omega(x, y) = f(x, y)dx+g(x, y)dy$. The *exterior differential operator* d acts on ω as,

$$d\omega(x, y) = (\frac{\partial g}{\partial x} - \frac{\partial f}{\partial y})dx \wedge dy.$$

If $d\omega = 0$, then ω is called a *closed 1-form*. If there exists a function $h : S \to \mathbb{R}$ such that $\omega = dh$, then ω is called an *exact 1-form*. Exact 1-forms are closed. The first De Rham cohomology group of the surface is the group of all non-exact closed 1-forms.

Hodge Decomposition The Hodge star operator on differential forms is defined as

$$^*\omega = {}^*(f(x, y)dx+g(x, y)dy) = (-g(x, y)dx+f(x, y)dy).$$

A differential 1-form is called a *harmonic*, if $d\omega = 0, d^*\omega = 0$. *The Hodge decomposition theorem* states that each cohomology class has a unique harmonic form. The group consisting of all the harmonic 1-forms is denoted as $H^1_\Delta(S, \mathbb{R})$; it is isomorphic to $H^1(S, \mathbb{R})$.

Holomorphic Differentials Let $\{(U_\alpha, z_\alpha\}$ be the conformal atla ygs, where the complex parameter $z_\alpha = x_\alpha + \sqrt{-1}y_\alpha$. Suppose (U_β, z_β) is another chart, the parameter transition function is $z_\beta(z_\alpha)$ is *holomorphic*, namely, it satisfies the following Cauchy-Riemann equations:

$$\begin{cases} \frac{\partial x_\beta}{\partial x_\alpha} &= \frac{\partial y_\beta}{\partial y_\alpha} \\ \frac{\partial x_\beta}{\partial y_\alpha} &= -\frac{\partial y_\beta}{\partial x_\alpha} \end{cases}$$

Let Ω be a complex differential form with local representation $\Omega(z_\alpha) = f(z_\alpha)dz_\alpha$, where $f(z_\alpha)$ is holomorphic. A holomorphic 1-form can be decomposed to a pair of conjugate harmonic real differential 1-forms, $\Omega = \omega + \sqrt{-1}^*\omega$, where ω is harmonic. All the holomorphic differentials form a group $\Omega(S)$, which is isomorphic to $H^1_\Delta(S, \mathbb{R})$.

Branched covering Let X, Y be compact connected topological spaces. A continuous mapping $f : X \to Y$ is called a branched covering if it is a local homeomorphism everywhere except a finite number of "branch" points. In the complex setting, this would mean that a branched covering is, locally at a point p, upto composition by biholomorphic maps, of the form $z \to z^{e_p}$, where $e_p > 1$ for finitely many branch points, and $e_p = 1$ everywhere else.

Trajectory Structure and Strebel differentials Given a holomorphic 1-form Ω on a genus g surface, there are $2g-2$ zero points. At each point $p \in S$, the tangent direction $d\gamma \in TM_p$ is called a *horizontal direction*, if $\Omega(d\gamma)$ is real. A curve $\gamma \subset S$ is called a *horizontal trajectory* of Ω, if at each point $p \in \gamma$, $d\gamma$ is along the horizontal direction. The horizontal trajectories through zeros of Ω are called *critical trajectories*. Similar to holomorphic 1-forms, one can consider quadratic differentials, which are tensors of type $(2, 0)$ in holomorphic coordinates. For quadratic differentials we define a direction to be horizontal if the differential is positive along it, and vertical if it is negative.

If the graph of vertical critical trajectories is compact, the quadratic differential is called Strebel. In the group of quadratic differentials, Strebel differentials are dense [5]. We will use a holomorphic-1 form whose square is Strebel. This will imply that the horizontal trajectories are closed curves.

3.3 Dense curve construction in continuous setting

We describe first the branched covering we use to construct our curve, and then prove the density.

Branched covering from a Strebel differential Given a Strebel differential Ω, the critical horizontal trajectories segment the surface to g connected components, denoted as $\{\Gamma_1, \Gamma_2, \cdots, \Gamma_g\}$. Each connected component Γ_k is of genus one with boundaries,

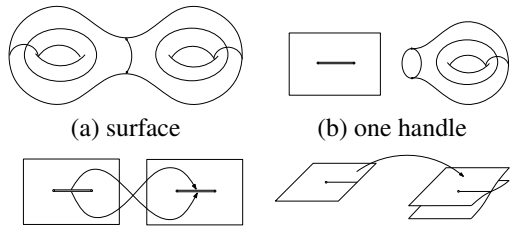
$$\partial\Gamma_k = b^1_k + b^2_k + \cdots + b^{n_k}_k.$$

The Strebel differential Ω induces a flat metric on each Γ_k, the integration of Ω on each boundary loop b^i_k maps the boundary loop to a straight line slit. Namely, then integration of Ω on each Γ_k maps Γ_k to a flat torus with straight line slits.

The mapping from the surface to the flat tori are diffeomorphic except at the zero points. Locally, the mapping at the zero points is similar to the complex power map $z \mapsto z^2$. Therefore, the zero points are the branch points.

The curve we use Suppose each flat torus is $\mathbb{R}^2/\Gamma_k$, $k = 1, 2, \cdots, g$. Here Γ_k represent lattice groups. Then we can find a line ℓ on the plane, such that ℓ does not go through any points in the union of lattices $\cup_k\Gamma_k$, since this union is countable. In particular if the lattice points are all rational then a line with irrational slope would do; otherwise we chose a random line. Denote the slope of ℓ as k.

On the"welded flat tori", start from one point draw a line γ with slope k. Then γ goes across the handles via the slits; when it hits a slit it moves from one handle to another and



(a) surface (b) one handle

(c) glue pattern (d) branch point

Figure 1: Branch covering map.

continues with the same slope k. We take care to chose k in such a way that this line does not pass through the endpoint of any slit. Again, this is easy to maintain since we only have finitely many of these endpoints.

Theorem 1. *Let γ be the curve constructed as above. Then γ is dense and does not go through any point more than once.*

Proof. Density would imply aperiodicity of γ, which in turn would imply that γ does not pass through any point twice. This is because on each torus, γ is a line with slope k, and if it visited a point twice it must necessarily become periodic. Thus it suffices to prove that γ is dense.

Density of such a curve follows from results in Teichmüller theory [18], and we just sketch the proof. Essentially, as long as the direction k does not contain a "saddle connection", which is a trajectory connecting two zeroes of the holomorphic differential, it will be dense. In our case, if the slit coordinates are rational, we choose an irrational k; otherwise we choose a random k. In both cases, with probability 1 we will neither hit the lattice points Γ_k nor the end points of the slits. This guarantees density. □

Note that although proving progressive density requires more sophisticated math techniques, our simulations in [32] show that the curve we construct does indeed satisfy this property.

4 Algorithm

In this section we present a centralized algorithm for the input of a sensor network densely deployed on a surface. Note that this can be generalized to a completely distributed algorithm too; we choose not to present the distributed algorithm for the sake of simplicity in this extended abstract, and refer to [32] for details.

We assume that the sensors are densely deployed on some underlying surface S such that locally the sensors lie on a flat plane. Thus we can apply existing algorithms to first come up with a triangulation of the sensors that approximate the underlying surface S [7, 8].

The surface is approximated by a triangular mesh $M = (V, E, F)$, where V, E, F denotes the vertex, edge and face sets respectively. We use $v_i \in V$ to represent a vertex, $[v_i, v_j]$ an oriented edge from v_i to v_j, $[v_i, v_j, v_k]$ an oriented face where v_i, v_j and v_k are sorted counter-clock-wisely. We assume the mesh is closed with genus g.

All the mathematics objects we described in Section 3 (homology groups, cohomology groups, harmonic and holomorphic 1-forms, branched covering maps) have straightforward discrete analogs. We omit the definitions (they can be found in [32]). In the following, we are talking about the discrete versions of such objects.

The algorithm pipeline is as follows:

1. Compute the basis of the first homology group $H_1(M, \mathbb{Z})$.
2. Calculate the dual basis of the first cohomology group $H^1(M, \mathbb{R})$.
3. Obtain the basis of the harmonic 1-form group
4. Achieve the basis of the holomorphic 1-form group
5. Integrate a holomorphic 1-form to get the required branched covering map.
6. Use the curve described in the previous section.

Steps $1-3$ have been carried out in literature (even in the distributed setting) before, and we give their details in the appendix. Here we elaborate on the final and novel Steps 4 and 5.

Branch Covering Map We first compute the cut graph[1] G of the mesh. Then we slice the mesh along the cut group to obtain a *fundamental domain* M/G. Choose one holomorphic 1-form $\omega + \sqrt{-1}^*\omega$ and integrate the holomorphic 1-form on the fundamental domain to get a branched covering map. Fix a vertex $v_0 \in M/G$ as the base vertex, for any vertex $v_i \in M/G$,

$$\varphi(v_i) = \int_{v_0}^{v_i} \omega + \sqrt{-1}^*\omega,$$

the integration path $\gamma \subset M/G$ can be chosen arbitrarily, which consists a sequence of consecutive oriented edges, connecting v_0 to v_i, denoted as

$$\gamma = e_0 + e_1 + \cdots e_k,$$

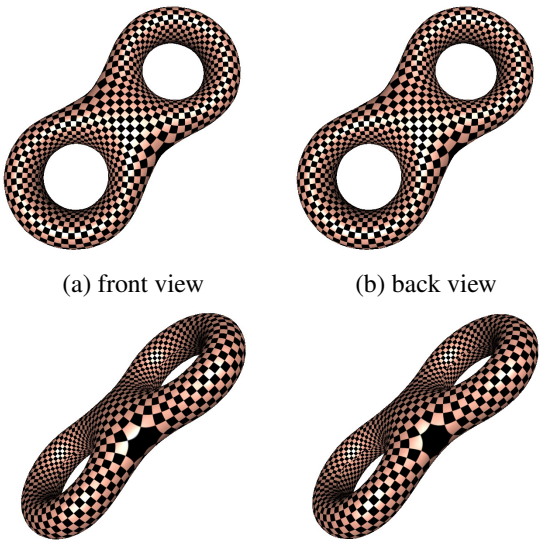
such that target vertex of e_i equals to the source vertex of e_{i+1}, the source of e_0 is v_0, the target of e_k is v_i.

$$\int_\gamma \omega = \sum_{i=0}^{k} \omega(e_i).$$

The branching points of φ are the zero points of the holomorphic 1-form. The slits are the horizontal trajectories connecting the zeros of the holomorphic 1-form.

As shown in Fig.1, there are $2g-2$ zero points of the holomorphic 1-form. The horizontal trajectories through the zeros segment the surface into handles as shown in Frame (a).

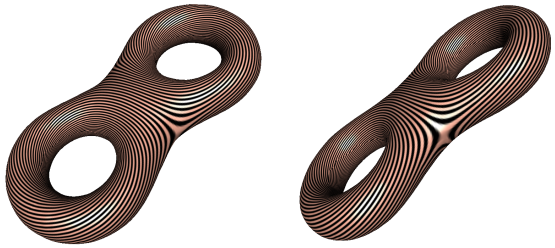
[1] Intuitively, this is the set of edges whose removal transforms the surface into a disk.



(a) front view (b) back view

(c) first zero point (d) second zero point

Figure 2: Holomorphic 1-form and zero points.



(a) dense curve (b) zero point

Figure 3: Dense curve on the surface.

Each handle is conformally mapped onto a flat torus with a slit, the end points of the slit are the zero points, as shown in Frame (b). The flat tori are glued together through slits, the top (bottom) edge of the slit on one torus is glued to the bottom (top) edge of the slit on the other torus, as shown in Frame (c). In the neighborhood of each zero point, the mapping is a branch covering similar to $z \mapsto z^2$, as illustrated in Frame (d).

The curve On the glued tori, start from any point and draw a line γ with slope k. Then γ goes across the handles via the slits; when it hits a slit it moves from one handle to another and continues with the same slope k. We take care to chose k in such a way that this line does not pass through the endpoint of any slit. This is easy to maintain since we only have finitely many of these endpoints.

5 Conclusion

We show in this paper a new construction for computing a dense curve on a 3D sensor network when the sensors are densely on a 2D manifold. The algorithm substantially generalizes over the prior work by Ban *et al.* [4] while keeping essentially the same nice properties. As future work we would like to see how to generalize the idea to truly 3D networks (volumetric 3D sensor networks).

Acknowledgment

Jie Gao would like to acknowledge NSF support through DMS-1418255, DMS-1221339, CNS-1217823 and Air Force Research AFOSR FA9550-14-1-0193.

References

[1] G. Anastasi, M. Conti, and M. Di Francesco. Data collection in sensor networks with data mules: An integrated simulation analysis. In *Computers and Communications, 2008. ISCC 2008. IEEE Symposium on*, pages 1096–1102, July 2008.

[2] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, 2007.

[3] S. Arora. Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. *J. ACM*, 45:753–782, September 1998.

[4] X. Ban, M. Goswami, W. Zeng, X. D. Gu, and J. Gao. Topology dependent space filling curves for sensor networks and applications. In *Proc. of 32nd Annual IEEE Conference on Computer Communications (INFOCOM'13)*, April 2013.

[5] A. Douady and J. Hubbard. On the density of strebel differentials. *Inventiones mathematicae*, 30(2):175–179, 1975.

[6] E. Ekici, Y. Gu, and D. Bozdag. Mobility-based communication in wireless sensor networks. *Communications Magazine, IEEE*, 44(7):56–62, July 2006.

[7] S. Funke and N. Milosavljević. Network sketching or: "how much geometry hides in connectivity? - part II". In *SODA '07: Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 958–967, 2007.

[8] J. Gao, L. J. Guibas, J. Hershberger, L. Zhang, and A. Zhu. Geometric spanner for routing in mobile networks. In *Proceedings of the 2nd ACM Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'01)*, pages 45–55, 2001.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[10] D. Jea, A. Somasundara, and M. Srivastava. Multiple controlled mobile elements (data mules) for data collection in sensor networks. In *Proceedings of the First IEEE International Conference on Distributed Computing in Sensor Systems*, DCOSS'05, pages 244–257, 2005.

[11] M. Kamat, A. Ismail, and S. Olariu. Modified hilbert space-filling curve for ellipsoidal coverage in wireless ad hoc sensor networks. In *Signal Processing and Communications, 2007. ICSPC 2007. IEEE International Conference on*, pages 1407 –1410, nov. 2007.

[12] A. Kansal, A. A. Somasundara, D. D. Jea, M. B. Srivastava, and D. Estrin. Intelligent fluid infrastructure for embedded networks. In *Proceedings of the 2Nd International Conference on Mobile Systems, Applications, and Services*, MobiSys '04, pages 111–124, 2004.

[13] M. Klann. Mobile response. chapter Tactical Navigation Support for Firefighters: The LifeNet Ad-Hoc Sensor-Network and Wearable System, pages 41–56. Springer-Verlag, Berlin, Heidelberg, 2009.

[14] M. Li and Y. Liu. Underground coal mine monitoring with wireless sensor networks. *ACM Trans. Sen. Netw.*, 5:10:1–10:29, April 2009.

[15] L. Lovasz. Random walks on graphs: A survey. *Bolyai Soc. Math. Stud.*, 2, 1996.

[16] M. Ma and Y. Yang. Sencar: An energy-efficient data gathering mechanism for large-scale multihop sensor networks. *Parallel and Distributed Systems, IEEE Transactions on*, 18(10):1476–1488, Oct 2007.

[17] D. D. L. Mascarenas, E. Flynn, K. Lin, K. Farinholt, G. Park, R. Gupta, M. Todd, and C. Farrar. Demonstration of a roving-host wireless sensor network for rapid assessment monitoring of structural health.

[18] T. S. Masur H. Rational billiards and flat structures. *Hasselblatt B., Katok, A. (eds.) Handbook of Dynamical Systems*, 1A:1015–1089, 2002.

[19] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric tsp, k-mst, and related problems. *SIAM J. Comput.*, 28:1298–1309, March 1999.

[20] S. Patil and S. R. Das. Serial data fusion using space-filling curves in wireless sensor networks. In *Proceedings of IEEE International Conference on Sensor and Ad Hoc Communications and Networks (SECON'04)*, pages 182–190, 2004.

[21] H. Sagan. *Space-Filling Curves*. Springer-Verlag, New York, 1994.

[22] R. Sokullu, M. A. Akkas, and F. Demirel. Data gathering system for watering and gas pipelines using wireless sensor networks. In *ICNS 2011, The Seventh International Conference on Networking and Services*, pages 190–195, May 2011.

[23] A. Somasundara, A. Kansal, D. Jea, D. Estrin, and M. Srivastava. Controllably mobile infrastructure for low energy embedded networks. *Mobile Computing, IEEE Transactions on*, 5(8):958–973, Aug 2006.

[24] A. Somasundara, A. Ramamoorthy, and M. Srivastava. Mobile element scheduling for efficient data collection in wireless sensor networks with dynamic deadlines. In *Real-Time Systems Symposium, 2004. Proceedings. 25th IEEE International*, pages 296–305, Dec 2004.

[25] A. A. Somasundara, A. Ramamoorthy, and M. B. Srivastava. Mobile element scheduling with dynamic deadlines. *IEEE Transactions on Mobile Computing*, 6(4):395–410, 2007.

[26] P. Toth and D. Vigo, editors. *The Vehicle Routing Problem*. 2002.

[27] I. Vasilescu, K. Kotay, D. Rus, M. Dunbabin, and P. Corke. Data collection, storage, and retrieval with an underwater sensor network. In *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems*, SenSys '05, pages 154–165, 2005.

[28] Y. Wang, J. Gao, and J. S. B. Mitchell. Boundary recognition in sensor networks by topological methods. In *Proc. of the ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom)*, pages 122–133, September 2006.

[29] D. Wu, L. Bao, and R. Li. A holistic approach to wireless sensor network routing in underground tunnel environments. *Comput. Commun.*, 33:1566–1573, August 2010.

[30] G. Xing, T. Wang, Z. Xie, and W. Jia. Rendezvous planning in mobility-assisted wireless sensor networks. In *Real-Time Systems Symposium, 2007. RTSS 2007. 28th IEEE International*, pages 311–320, Dec 2007.

[31] W. Zhao and M. Ammar. Message ferrying: proactive routing in highly-partitioned wireless ad hoc networks. In *Distributed Computing Systems, 2003. FTDCS 2003. Proceedings. The Ninth IEEE Workshop on Future Trends of*, pages 308–314, May 2003.

[32] M. Goswami, S. Li, J. Zhang, E. Saucan, X. D. Gu, and J. Gao Space Filling Curves for 3D Sensor Networks with Complex Topology. In*Technical Report*, http://www3.cs.stonybrook.edu/ jgao/paper/densecurve3d.pdf.

Appendix–The Algorithm Details

As mentioned, we choose not to present the distributed algorithm for the sake of simplicity in this extended abstract, and refer the reader to [32] for details.

We assume that the sensors are densely deployed on some underlying surface S such that locally the sensors lie on a flat plane. Thus we can apply existing algorithms to first come up with a triangulation of the sensors that approximate the underlying surface S [7, 8].

The surface is approximated by a triangular mesh $M = (V, E, F)$, where V, E, F denotes the vertex, edge and face sets respectively. We use $v_i \in V$ to represent a vertex, $[v_i, v_j]$ an oriented edge from v_i to v_j, $[v_i, v_j, v_k]$ an oriented face where v_i, v_j and v_k are sorted counter-clock-wisely. We assume the mesh is closed with genus g.

The algorithm pipeline is as follows:

1. Compute the basis of the first homology group $H_1(M, \mathbb{Z})$.
2. Calculate the dual basis of the first cohomology group $H^1(M, \mathbb{R})$.
3. Obtain the basis of the harmonic 1-form group.
4. Achieve the basis of the holomorphic 1-form group.
5. Integrate a holomorphic 1-form to get the required branched covering map.
6. Use the curve described in the Section 3.

Homology Group First, the dual mesh $\bar{M} = (\bar{V}, \bar{E}, \bar{F})$ of the input mesh M is constructed. Each vertex $v_i \in V$, face $f_j \in F$ and edge $e_k \in E$ corresponds to a face $\bar{v}_i \in \bar{F}$, a vertex $\bar{f}_j \in \bar{V}$ and and edge $\bar{e}_k \in \bar{E}$ on the dual mesh respectively.

Second, a spanning tree $\bar{T}$ of the dual mesh $\bar{M}$ is computed. The *cut graph* $G \subset M$ is the union of edges, whose dual edges are not in the spanning tree:

$$G = \{e \in E | \bar{e} \notin \bar{T}\}.$$

Intuitively, the mesh $M \setminus G$ with the cut graph removed is a topological disk. Third, a spanning tree T of the cut graph G is calculated. The complement of T in G is a union of edges:

$$G/T = \{e_1, e_2, \cdots, e_{2g}\},$$

Each edge e_i when included in the spanning tree T (thus $T \cup e_i$) gives rise to a unique loop $\gamma_i \subset T \cup e_i$. These loops

$$\{\gamma_1, \gamma_2, \cdots, \gamma_{2g}\}$$

form the basis of the first homology group $H_1(M, \mathbb{Z})$.

Cohomology Group The differential forms are approximated by discrete forms. A discrete 0-form is a function defined on vertices, $f : V \to \mathbb{R}$; a discrete 1-form is a function defined on the oriented edges, $\omega : E \to \mathbb{R}$, $\omega([v_i, v_j]) = -\omega([v_j, v_i])$; a discrete 2-form is defined on the oriented faces $\tau : F \to \mathbb{R}$. Discrete exterior differential operator d is dual to the boundary operator ∂, for example

$$\begin{aligned} d\omega([v_i, v_j, v_k]) &= \omega(\partial[v_i, v_j, v_k]) \\ &= \omega([v_i, v_j]) + \omega([v_j, v_k]) + \omega([v_k, v_i]). \end{aligned}$$

Given the first homology group $H_1(M, \mathbb{Z})$ basis $\{\gamma_1, \cdots, \gamma_{2g}\}$, the dual cohomology group basis can be obtained as follows. For each base loop γ_k, slice the mesh M to get an open mesh M_k. The boundary of M_k has two connected components, denoted them as

$$\partial M_k = \gamma_k^+ \cup \gamma_k^-.$$

Construct a function $f_k : M_k \to \mathbb{R}$,

$$f_k(v_i) = \begin{cases} 1 & v_i \in \gamma_k^+ \\ 0 & v_i \in \gamma_k^- \\ \text{rand} & v_i \notin \partial M_k \end{cases}$$

Then the 1-form df_k, $df_k([v_i, v_j]) = f_k(v_j) - f_k(v_i)$ is 0 on all boundary edges. Therefore, one can define ω_k on the original closed mesh M. Suppose e is not on the loop γ_k, then it has a unique corresponding edge $\tilde{e}$ on M_k, define $\omega_k(e) = df_k(\tilde{e})$. If $e \subset \gamma_k$, then let $\omega_k(e) = 0$. ω_k is a closed 1-form, and not exact. These non-exact closed 1-forms

$$\{\omega_1, \omega_2, \cdots, \omega_{2g}\}$$

form the basis of $H^1(M, \mathbb{R})$.

Harmonic Differential Group According to Hodge theory, each cohomological class has a unique harmonic 1-form. Given a cohomology group basis $\{\omega_1, \omega_2, \cdots, \omega_{2g}\}$, for each closed 1-form ω_k, there is a function $h_k : V \to \mathbb{R}$, such that $\omega_k + dh_k$ is harmonic. By definition, the 1-form is curl free

$$d(\omega_k + dh_k) = d\omega_k + d^2 h_k = 0.$$

The divergence is

$$^*d^*(\omega_k + dh_k) = 0,$$

this induces the linear system, for each vertex $v_i \in V$,

$$\sum_{[v_i, v_j] \in E} w_{ij} \left(h_k(v_j) - h_k(v_i) + \omega_k([v_i, v_j])\right) = 0,$$

where w_{ij} is the cotangent edge weight. Suppose edge $[v_i, v_j]$ is shared by two faces $[v_i, v_j, v_k]$ and $[v_j, v_i, v_l]$, then

$$w_{ij} = \cot \theta_k^{ij} + \cot \theta_l^{ji},$$

where θ_k^{ij} is the corner angle at vertex v_k in triangle $[v_i, v_j, v_k]$. The coefficient matrix of the linear system is

positive definite, the solution exists and is unique. The 1-forms

$$\{\omega_1 + dh_1, \omega_2 + dh_2, \cdots, \omega_{2g} + dh_{2g}\}$$

form the basis of the harmonic 1-form group.

Holomorphic Differential Group Suppose the harmonic 1-form group basis is given, still denoted as $\{\omega_1, \omega_2, \cdots, \omega_{2g}\}$. Let ω be a harmonic 1-form, its conjugate 1-form ${}^*\omega$ is harmonic as well, therefore it can be represented as linear combination of $\{\omega_k\}$,

$${}^*\omega = \lambda_1\omega_1 + \lambda_2\omega_2 + \cdots + \lambda_{2g}\omega_{2g}. \quad (1)$$

The coefficients can be obtained by solving the following the linear system

$$\int_M {}^*\omega \wedge \omega_k = \sum_{i=1}^{2g} \lambda_i \int_M \omega_i \wedge \omega_k, k = 1, 2, \cdots, 2g.$$

On one triangle $[v_i, v_j, v_k]$ embed on the plane $\mathbb{R}^2$, the closed 1-form ω_k can be represented as a constant 1-form $\omega_k = a_k dx + b_k dy$, such that

$$\omega_k([v_i, v_j]) = \int_{[v_i, v_j]} a_k dx + b_k dy,$$

same equations hold for other edges $[v_j, v_k]$ and $[v_k, v_i]$. The wedge product on the face is given by

$$\omega_i \wedge \omega_j = (a_i dx + b_i dy) \wedge (a_j dx + b_j dy) = \begin{vmatrix} a_i & b_i \\ a_j & b_j \end{vmatrix} dx \wedge dy.$$

Therefore

$$\int_{[v_i, v_j, v_k]} \omega_i \wedge \omega_j = (a_i b_j - a_j b_i)\, Area([v_i, v_j, v_k]).$$

and

$$\int_M \omega_i \wedge \omega_j = \sum_{[v_i, v_j, v_k]} \int_{[v_i, v_j, v_k]} \omega_i \wedge \omega_j.$$

Locally, the Hodge operator is given by

$${}^*\omega_k = {}^*(a_k dx + b_k dy) = a_k dy - b_k dx.$$

So the coefficients in the linear equation 1 can be easily computed. By solving the linear system, the conjugate harmonic 1-form ${}^*\omega$ is obtained.

The harmonic 1-form basis $\{\omega_k\}$, paired with its conjugate harmonic 1-form $\{{}^*\omega\}$ form the holomorphic 1-form basis

$$\{\omega_1 + \sqrt{-1}{}^*\omega_1, \omega_2 + \sqrt{-1}{}^*\omega_2, \cdots, \omega_{2g} + \sqrt{-1}{}^*\omega_{2g}\}$$

Branch Covering Map Compute the cut graph G of the mesh, slice the mesh along the cut group to obtain a *fundamental domain* M/G. Choose one holomorphic 1-form $\omega + \sqrt{-1}{}^*\omega$ and integrate the holomorphic 1-form on the fundamental domain to get the branch covering map. Fix a vertex $v_0 \in M/G$ as the base vertex, for any vertex $v_i \in M/G$,

$$\varphi(v_i) = \int_{v_0}^{v_i} \omega + \sqrt{-1}{}^*\omega,$$

the integration path $\gamma \subset M/G$ can be chosen arbitrarily, which consists a sequence of consecutive oriented edges, connecting v_0 to v_i, denoted as

$$\gamma = e_0 + e_1 + \cdots e_k,$$

such that target vertex of e_i equals to the source vertex of e_{i+1}, the source of e_0 is v_0, the target of e_k is v_i.

$$\int_\gamma \omega = \sum_{i=0}^{k} \omega(e_i).$$

The branching points of φ are the zero points of the holomorphic 1-form. The slits are the horizontal trajectories connecting the zeros of the holomorphic 1-form.

As shown in Fig.1, there are $2g-2$ zero points of the holomorphic 1-form. The horizontal trajectories through the zeros segment the surface into handles as shown in Frame (a). Each handle is conformally mapped onto a flat torus with a slit, the end points of the slit are the zero points, as shown in Frame (b). The flat tori are glued together through slits, the top (bottom) edge of the slit on one torus is glued to the bottom (top) edge of the slit on the other torus, as shown in Frame (c). In the neighborhood of each zero point, the mapping is a branch covering similar to $z \mapsto z^2$, as illustrated in Frame (d).

Inscribing $\mathcal{H}$-Polyhedra in Quadrics Using a Projective Generalization of Closed Sets*

Willem Hagemann and Eike Möhlmann†

Abstract

We present a projective generalization of closed sets, called *complete projective embeddings*, which allows us to inscribe $\mathcal{H}$-polyhedra in quadrics efficiently. Essentially, the complete projective embedding of a closed convex set $\mathbf{P} \subseteq \mathbb{K}^d$ is a double cone in $\mathbb{K}^{d+1}$. We show that complete projective embeddings of polyhedral sets are of particular interest and already occurred in the theory of linear fractional programming. Our approach works as follows: By projective principal axis transformation the quadric is converted to a hyperboloid and then approximated by an inner (right) spherical cylinder. Now, given an inscribed $\mathcal{H}$-polytope of the spherical cross section, cylindrification of the polyhedron yields an inscribed $\mathcal{H}$-polyhedron of the spherical cylinder and, hence, of the hyperboloid. After application of the inverse base transformation this approach finally yields an inscribed set of the quadric. The crucial task of this procedure is to find an appropriate generalization of closed sets, which is closed under the involved projective transformations and allows us to recover the non-projective equivalents to the inscribed sets obtained by our approach. It turns out that complete projective embeddings are the requested generalizations.

1 Introduction

During our research for an efficient method to generate inscribed $\mathcal{H}$-polyhedra of quadrics, we developed the notion of *complete projective embeddings* of closed sets. A complete projective embedding of a closed convex set $\mathbf{P} \subseteq \mathbb{K}^d$ is essentially a double cone in $\mathbb{K}^{d+1}$ that is obtained by translating $\mathbf{P}$ onto the hyperplane $\{\binom{\mathbf{x}}{\lambda} \mid \lambda = 1\} \subseteq \mathbb{K}^{d+1}$ and consists of all lines joining the origin and some point of the translated set. Certainly, we can always recover the original set by restricting the double cone to its intersection with the designated hyperplane $\{\binom{\mathbf{x}}{\lambda} \mid \lambda = 1\}$. As we shall apply linear maps on the double cones, we are interested in what happens to the intersections of the images with the designated hyperplane – a question which is very similar to the classical theory of conic sections. For the important case where $\mathbf{P}$ is an $\mathcal{H}$-polyhedron we obtain that each of this intersections can be described by a union of two polyhedra having representation matrices of certain symmetry.

In 2009 Gallier described the double cone representation of polyhedra which he calls *projective polyhedra* [5]. It turns out that projective polyhedra are complete projective embedding of polyhedral sets, and for these we shall use both notions synonymously. In his introduction he noted that "*to the best of our knowledge, this notion of projective polyhedron is new*". Although we neither found any systematic work on this notion, we discovered that projective polyhedra are closely related to the sets of feasible solutions of linear fractional programs as given by Charnes and Cooper [2] in 1962. This paper is a contribution to show the theoretical and practicable relevance of projective polyhedra and their generalization – the complete projective embeddings of closed sets – and aims for establishing them as independent objects of investigation.

In the second part of the paper we show how complete projective embeddings can be used to generate inscribed $\mathcal{H}$-polyhedra in quadrics. The research is motivated by a combination of reachability analysis and stability analysis for linear systems[1]: In reachability analysis of linear systems one typically computes tight overapproximations of the reachable states until all trajectories either leave the region of admissible states or a trajectory enters a certain set **Avoid**. The computation is based on a convex representation of flow segments over a time interval of short length. The convex set representation is highly compatible with $\mathcal{H}$-polyhedra. This holds for approaches using support functions as well as for our particular approach where we have chosen *symbolic orthogonal projections* as a representation of polyhedral sets. Symbolic orthogonal projections allow an exact and efficient representation and evaluation of typical geometric operations occurring in reachability analysis, including Minkowski sums, convex hulls, intersections, and arbitrary affine transformations [7].

On the other hand, stability analysis provides Lyapunov functions in terms of quadratic forms. For each

*This work has been partly supported by the German Research Foundation (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS, www.avacs.org).

†Department of Computer Science, Carl von Ossietzky University of Oldenburg, Germany. Correspond to {willem.hagemann, eike.moehlmann}@informatik.uni-oldenburg.de.

[1] Actually we deal with hybrid systems. However, for an explanation of our motivation it suffices to restrict to linear systems.

admissible state of the system the value of this quadratic form is positive and decreases monotonically along each trajectory where the value of the equilibrium point is 0. A level set of a quadratic form consists of all states whose value is below a certain level. Hence, its boundary is given by a quadric. Assume, there exists a lower bound b on the value of the quadratic form for all states in **Avoid**. If the current flow segment is completely located within a level set of level b, we conclude that there will no further intersection with **Avoid**, and we may stop the reachability analysis. Alas, while there exists efficient methods to check whether the current flow segment – either represented by support functions or symbolic orthogonal projections – is contained in an $\mathcal{H}$-polyhedron, we have no efficient method to test whether the current flow segment is contained in a level set. Hence, we use an inner approximation of the level set in terms of an $\mathcal{H}$-polyhedron to assess the inclusion of the current flow segment instead. For further details we point the interested reader to [8].

2 Preliminaries

By $\mathbb{K}$ we denote an arbitrary ordered field. The corresponding euclidean field is denoted by $\mathbb{E}$. A closed convex polyhedron $\mathbf{P}$ is the set $\mathbf{P} = \mathbf{P}(A, \mathbf{a}) = \{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{a}\}$. In the following the term *polyhedron* will always refer to a closed convex polyhedron. Vectors and sets of vectors are denoted by bold letters. All coefficients of the vectors $\mathbf{0}$ and $\mathbf{1}$ are 0 or 1, respectively. We use the superscript ^{T} to indicate the transpose of a vector or matrix. The notion A^{-T} denotes the transpose of the inverse of the matrix A. The coefficients of a d-dimensional vector $\mathbf{x}$ are denoted by $x_1, \ldots, x_d$.

2.1 Projective Space

We give a short introduction into projective geometry as it can be found in many textbooks like [6, 1]. The *projective space* $\mathrm{Proj}^d(\mathbb{K})$ *induced by* $\mathbb{K}$ can be identified with the set $\mathbb{K}^{d+1} \setminus \{\mathbf{0}\}$ where two vectors $\mathbf{x}$, $\mathbf{y}$ are *projectively equal* if and only if there exists some $\lambda \neq 0$ such that $\mathbf{x} = \lambda\mathbf{y}$, formally

$$\mathbf{x} =_p \mathbf{y} \iff \exists \lambda \neq 0 \colon \mathbf{x} = \lambda\mathbf{y}. \tag{1}$$

The injective mapping $\iota \colon \mathbb{K}^d \to \mathrm{Proj}^d(\mathbb{K})$, $\mathbf{x} \mapsto \binom{\mathbf{x}}{1}$ defines the *canonical embedding* of $\mathbb{K}^d$ into the projective space $\mathrm{Proj}^d(\mathbb{K})$. On the other hand, any point $\binom{\mathbf{x}}{\lambda} \in \mathrm{Proj}^d(\mathbb{K})$ either represents the point $\frac{1}{\lambda}\mathbf{x}$ in $\mathbb{K}^d$ if and only if $\lambda \neq 0$, or $\binom{\mathbf{x}}{\lambda}$ represents a point at infinity, which is a point that has no corresponding point in $\mathbb{K}^d$. Actually, $\binom{\mathbf{x}}{0}$ represents $\lim_{\lambda \to 0^+} \frac{1}{\lambda}\mathbf{x}$, which coincides with $\lim_{\lambda \to 0^-} \frac{1}{\lambda}\mathbf{x}$ by projective equality.

3 Complete Projective Embeddings

The classical approach allows us to treat projective sets as subsets of $\mathbb{K}^{d+1} \setminus \{\mathbf{0}\}$, but it has a drawback: Due to the existential quantifier in (1) it is laborious to assess projective subset relations in $\mathbb{K}^{d+1} \setminus \{\mathbf{0}\}$. We shall introduce the notion of *projective completeness*, which allows us to assess projective subset relations in a simple way. The projective vector space $\mathrm{Proj}^d(\mathbb{K})$ is canonically embedded in the vector space $\mathbb{K}^{d+1} \setminus \{\mathbf{0}\}$ and, hence, also embedded in $\mathbb{K}^{d+1}$.

Definition 1 *Let $\mathbf{P}$ be a subset of $\mathbb{K}^{d+1}$. If (i) $\mathbf{0} \in \mathbf{P}$ and (ii) $\mathbf{x} \in \mathbf{P}$ implies that $\lambda\mathbf{x} \in \mathbf{P}$ for all $\lambda \in \mathbb{K}$, then we call $\mathbf{P}$* projectively complete.

Definition 2 *Let $\mathbf{P}$ be a closed set, and let $\tilde{\mathbf{P}}$ be the least closed and projectively complete set with $\iota(\mathbf{P}) \subseteq \tilde{\mathbf{P}}$. The set $\tilde{\mathbf{P}}$ is called the* complete projective embedding *of $\mathbf{P}$.*

Hence, the projectively complete embedding $\tilde{\mathbf{P}}$ of $\mathbf{P}$ always contains $\mathbf{0}$ and all projective representations $\binom{\mathbf{x}}{\lambda}$, $\lambda \neq 0$, of a point $\mathbf{x} \in \mathbf{P}$. Furthermore, since the union of two closed sets $\mathbf{P}_1$ and $\mathbf{P}_2$ is closed again, the complete projective embedding of $\mathbf{P}_1 \cup \mathbf{P}_2$ is the union $\tilde{\mathbf{P}}_1 \cup \tilde{\mathbf{P}}_2$ of the complete projective embeddings $\tilde{\mathbf{P}}_1$ and $\tilde{\mathbf{P}}_2$.

Proposition 3 *Let $\mathbf{P}$, $\mathbf{Q}$ be two closed sets and $\tilde{\mathbf{P}}$, $\tilde{\mathbf{Q}}$ their complete projective embeddings. Then $\mathbf{P} \subseteq \mathbf{Q}$ if and only if $\tilde{\mathbf{P}} \subseteq \tilde{\mathbf{Q}}$.*

Proof. Assume $\mathbf{P} \subseteq \mathbf{Q}$ holds. Let $\binom{\mathbf{x}}{\lambda} \in \tilde{\mathbf{P}}$. In the case $\lambda \neq 0$ we have $\frac{1}{\lambda}\mathbf{x} \in \mathbf{P}$. Since $\mathbf{P} \subseteq \mathbf{Q}$, we also have $\frac{1}{\lambda}\mathbf{x} \in \mathbf{Q}$. The set $\tilde{\mathbf{Q}}$ is the complete projective embedding of $\mathbf{Q}$. Hence, $\binom{\mathbf{x}}{\lambda} \in \tilde{\mathbf{Q}}$. In the case $\lambda = 0$ we either have $\mathbf{x} = \mathbf{0}$ and $\binom{\mathbf{x}}{\lambda} = \binom{\mathbf{0}}{0}$ is trivially contained in $\tilde{\mathbf{Q}}$, or there exists a sequence $\left(\binom{\mathbf{x}_i}{\lambda_i}\right)_{i \in \mathbb{N}} \subseteq \tilde{\mathbf{P}}$ with $\lim_{i \to \infty} \binom{\mathbf{x}_i}{\lambda_i} = \binom{\mathbf{x}}{0}$ and $\lambda_i \neq 0$ for all $i \in \mathbb{N}$ since $\tilde{\mathbf{P}}$ is the least closed and projective complete set containing $\iota(\mathbf{P})$. We already have seen that any $\binom{\mathbf{x}_i}{\lambda_i} \in \tilde{\mathbf{P}}$ with $\lambda_i \neq 0$ is also in $\tilde{\mathbf{Q}}$. Furthermore, $\tilde{\mathbf{Q}}$ is closed. Hence, $\binom{\mathbf{x}}{0} \in \tilde{\mathbf{Q}}$. We have shown that $\mathbf{P} \subseteq \mathbf{Q}$ implies $\tilde{\mathbf{P}} \subseteq \tilde{\mathbf{Q}}$.

On the other hand, assume $\tilde{\mathbf{P}} \subseteq \tilde{\mathbf{Q}}$. Let $\mathbf{x} \in \mathbf{P}$. Then $\binom{\mathbf{x}}{1} \in \tilde{\mathbf{P}}$ and $\tilde{\mathbf{P}} \subseteq \tilde{\mathbf{Q}}$ implies $\binom{\mathbf{x}}{1} \in \tilde{\mathbf{Q}}$. Hence, $\mathbf{x} \in \mathbf{Q}$. We have shown that $\tilde{\mathbf{P}} \subseteq \tilde{\mathbf{Q}}$ implies $\mathbf{P} \subseteq \mathbf{Q}$. □

Proposition 4 *Let $\mathbf{P} = \mathbf{P}(A, \mathbf{a}) = \left\{\mathbf{x} \in \mathbb{K}^d \,\middle|\, A\mathbf{x} \leq \mathbf{a}\right\}$ be a polyhedron, and let $\tilde{\mathbf{P}} = \mathbf{P}_1 \cup \mathbf{P}_2$, where*

$$\mathbf{P}_1 = \mathbf{P}\left(\begin{pmatrix} A & -\mathbf{a} \\ \mathbf{0}^T & -1 \end{pmatrix}, \begin{pmatrix} \mathbf{0} \\ 0 \end{pmatrix}\right),$$
$$\mathbf{P}_2 = \mathbf{P}\left(\begin{pmatrix} -A & \mathbf{a} \\ \mathbf{0}^T & 1 \end{pmatrix}, \begin{pmatrix} \mathbf{0} \\ 0 \end{pmatrix}\right).$$

Then $\tilde{\mathbf{P}}$ is the complete projective embedding of $\mathbf{P}$.

Proof. Obviously, we have $\iota(\mathbf{P}) \subseteq \mathbf{P}_1 \subseteq \tilde{\mathbf{P}}$.

We show that $\tilde{\mathbf{P}}$ is projectively complete. Obviously, $\binom{\mathbf{0}}{0} \in \tilde{\mathbf{P}}$. Let $\binom{\mathbf{x}}{\lambda} \in \tilde{\mathbf{P}}$, i. e., $A\mathbf{x} - \lambda\mathbf{a} \leq \mathbf{0}$, $-\lambda \leq 0$ or $-A\mathbf{x} + \lambda\mathbf{a} \leq \mathbf{0}$, $\lambda \leq 0$. Let $\mu \in \mathbb{K}$. Either we have $\mu \geq 0$ or $\mu < 0$. In both cases we obtain $A\mu\mathbf{x} - \mu\lambda\mathbf{a} \leq \mathbf{0}$, $-\mu\lambda \leq 0$ or $-A\mu\mathbf{x} + \mu\lambda\mathbf{a} \leq \mathbf{0}$, $\mu\lambda \leq 0$. Hence, $\binom{\mathbf{x}}{\lambda} \in \tilde{\mathbf{P}}$ implies $\mu\binom{\mathbf{x}}{\lambda} \in \tilde{\mathbf{P}}$ for all $\mu \in \mathbb{K}$.

We show that $\tilde{\mathbf{P}}$ is a closed set. Let $\left(\binom{\mathbf{x}_i}{\lambda_i}\right)_{i\in\mathbb{N}}$ be a sequence with $\binom{\mathbf{x}_i}{\lambda_i} \in \tilde{\mathbf{P}}$ for all $i \in \mathbb{N}$ and $\lim_{i\to\infty}\binom{\mathbf{x}_i}{\lambda_i} = \binom{\mathbf{x}}{\lambda}$. We have to show that $\binom{\mathbf{x}}{\lambda} \in \tilde{\mathbf{P}}$. In the case $\lambda \neq 0$ let J be the set of all indices i where $\lambda_i \neq 0$. Then $(\frac{1}{\lambda_i}\mathbf{x}_i)_{i\in J}$ is an infinite sequence of members in $\mathbf{P}$ which converges to $\frac{1}{\lambda}\mathbf{x}$. The polyhedron $\mathbf{P}$ is closed, hence $\frac{1}{\lambda}\mathbf{x} \in \mathbf{P}$ and $\binom{\mathbf{x}}{\lambda} \in \tilde{\mathbf{P}}$. Let $\lambda = 0$. Either we have an infinite subset $J \subseteq \mathbb{N}$ such that $\lambda_i = 0$ for all $i \in J$. Then $A\mathbf{x}_i \leq \mathbf{0}$ or $-A\mathbf{x}_i \leq \mathbf{0}$ for all $i \in J$. The sets $\{\mathbf{y} \mid A\mathbf{y} \leq \mathbf{0}\}$ and $\{\mathbf{y} \mid -A\mathbf{y} \leq \mathbf{0}\}$ are closed, and so is their union. It follows $A\mathbf{x} \leq \mathbf{0}$ or $-A\mathbf{x} \leq \mathbf{0}$ and by definition of $\tilde{\mathbf{P}}$ also $\binom{\mathbf{x}}{0} \in \tilde{\mathbf{P}}$. Otherwise, there exists an index i_0 with $\lambda_i \neq 0$ for all $i \geq i_0$. The remaining sequence $\left(\binom{\mathbf{x}_i}{\lambda_i}\right)_{i\geq i_0}$ has the same limes. Hence, without loss of generality, we may assume $\lambda_i \neq 0$ for all $i \in \mathbb{N}$. Setting $\mathbf{y}_i = \mathbf{x}_i - \frac{\lambda_i}{\lambda_0}\mathbf{x}_0$ yields the equalities $\binom{\mathbf{y}_i}{0} = \binom{\mathbf{x}_i}{\lambda_i} - \frac{\lambda_i}{\lambda_0}\binom{\mathbf{x}_0}{\lambda_0}$ for all $i \in \mathbb{N}$. Since $\lim_{i\to\infty}\lambda_i = 0$ and $\lim_{i\to\infty}\mathbf{x}_i = \mathbf{x}$, we obtain $\lim_{i\to\infty}\frac{\lambda_i}{\lambda_0}\mathbf{x}_0 = \mathbf{0}$ and $\lim_{i\to\infty}\binom{\mathbf{y}_i}{0} = \lim_{i\to\infty}\binom{\mathbf{x}_i}{\lambda_i} - \lim_{i\to\infty}\frac{\lambda_i}{\lambda_0}\binom{\mathbf{x}_0}{\lambda_0} = \binom{\mathbf{x}}{0}$. Hence, $\binom{\mathbf{y}_i}{0}$ is a sequence with $\lim_{i\to\infty}\binom{\mathbf{y}_i}{0} = \binom{\mathbf{x}}{0}$ where the last coefficient of all members is equal to zero. For this case we already have shown that $\binom{\mathbf{x}}{0} \in \tilde{\mathbf{P}}$.

It remains to show that $\tilde{\mathbf{P}}$ is the least closed set which includes $\iota(\mathbf{P})$ and is projectively complete. Assume, $\tilde{\mathbf{P}}'$ is an arbitrary closed set that includes $\iota(\mathbf{P})$ and is projectively complete. We have to show that any $\binom{\mathbf{x}}{\lambda} \in \tilde{\mathbf{P}}$ is also contained in $\tilde{\mathbf{P}}'$. Therefore, let $\binom{\mathbf{x}}{\lambda} \in \tilde{\mathbf{P}}$. In the case $\lambda \neq 0$ we have $\frac{1}{\lambda}\mathbf{x} \in \mathbf{P}$. Then $\frac{1}{\lambda}\mathbf{x} \in \iota(\mathbf{P})$ and, hence, $\frac{1}{\lambda}\mathbf{x} \in \tilde{\mathbf{P}}'$. Since $\tilde{\mathbf{P}}'$ is projectively complete, it follows $\binom{\mathbf{x}}{\lambda} \in \tilde{\mathbf{P}}'$. In the case $\lambda = 0$ we have $A\mathbf{x} \leq \mathbf{0}$ or $-A\mathbf{x} \leq \mathbf{0}$. Hence, for all $\mathbf{y} \in \mathbf{P}$, i. e., $A\mathbf{y} \leq \mathbf{a}$, we have $\mathbf{y}+\mu\mathbf{x} \in \mathbf{P}$ for all $\mu \geq 0$ or $\mathbf{y}+\mu\mathbf{x} \in \mathbf{P}$ for all $\mu \leq 0$. The set $\tilde{\mathbf{P}}'$ includes $\iota(\mathbf{P})$, hence, $\binom{\mathbf{y}+\mu\mathbf{x}}{1} \in \tilde{\mathbf{P}}'$ for all $\mu \geq 0$ or $\binom{\mathbf{y}+\mu\mathbf{x}}{1} \in \tilde{\mathbf{P}}'$ for all $\mu \leq 0$. In the former case let $\mu_i = i$ and in the latter case let $\mu_i = -i$. Since $\tilde{\mathbf{P}}'$ is projectively complete, we have $\frac{1}{\mu_i}\binom{\mathbf{y}+\mu_i\mathbf{x}}{1} \in \tilde{\mathbf{P}}'$ for all $i \geq 1$. Hence, $\lim_{i\to\infty}\frac{1}{\mu_i}\binom{\mathbf{y}+\mu_i\mathbf{x}}{1} = \lim_{i\to\infty}\binom{\frac{1}{\mu_i}\mathbf{y}+\mathbf{x}}{\frac{1}{\mu_i}} = \binom{\mathbf{x}}{0}$. Since $\tilde{\mathbf{P}}'$ is closed, it follows $\binom{\mathbf{x}}{0} \in \tilde{\mathbf{P}}'$. □

Proposition 5 *Let $\tilde{\mathbf{P}}$ be the union of two cones*

$$\tilde{\mathbf{P}} = \mathbf{P}\left(\begin{pmatrix} A & -\mathbf{a}\end{pmatrix}, \mathbf{0}\right) \cup \mathbf{P}\left(\begin{pmatrix} -A & \mathbf{a}\end{pmatrix}, \mathbf{0}\right).$$

Then $\tilde{\mathbf{P}}$ is the complete projective embedding of $\mathbf{P}(A, \mathbf{a}) \cup \mathbf{P}(-A, -\mathbf{a})$.

Proof. Let $\tilde{\mathbf{R}}$ be the complete projective embedding of $\mathbf{P}(A, \mathbf{a}) \cup \mathbf{P}(-A, -\mathbf{a})$, i. e.,

$$\begin{aligned}\tilde{\mathbf{R}} &= \mathbf{P}\left(\begin{pmatrix} A & -\mathbf{a} \\ \mathbf{0}^T & -1\end{pmatrix}, \begin{pmatrix}\mathbf{0}\\0\end{pmatrix}\right) \cup \mathbf{P}\left(\begin{pmatrix} -A & \mathbf{a} \\ \mathbf{0}^T & 1\end{pmatrix}, \begin{pmatrix}\mathbf{0}\\0\end{pmatrix}\right) \cup \\ &\quad \mathbf{P}\left(\begin{pmatrix} -A & \mathbf{a} \\ \mathbf{0}^T & -1\end{pmatrix}, \begin{pmatrix}\mathbf{0}\\0\end{pmatrix}\right) \cup \mathbf{P}\left(\begin{pmatrix} A & -\mathbf{a} \\ \mathbf{0}^T & 1\end{pmatrix}, \begin{pmatrix}\mathbf{0}\\0\end{pmatrix}\right) \\ &= \mathbf{P}\left(\begin{pmatrix} A & -\mathbf{a}\end{pmatrix}, \mathbf{0}\right) \cup \mathbf{P}\left(\begin{pmatrix} -A & \mathbf{a}\end{pmatrix}, \mathbf{0}\right).\end{aligned}$$

Hence, $\tilde{\mathbf{R}} = \tilde{\mathbf{P}}$. □

Any complete projective embedding $\tilde{\mathbf{P}}$ of the form $\tilde{\mathbf{P}} = \mathbf{P}\left(\begin{pmatrix} A & -\mathbf{a}\end{pmatrix}, \mathbf{0}\right) \cup \mathbf{P}\left(\begin{pmatrix} -A & \mathbf{a}\end{pmatrix}, \mathbf{0}\right)$ is a double cone. Gallier calls these double cones *projective polyhedra* [5]. In the following we shall make use of his nomenclature. Note that the transition from the projective polyhedron $\tilde{\mathbf{P}} = \mathbf{P}\left(\begin{pmatrix} A & -\mathbf{a}\end{pmatrix}, \mathbf{0}\right) \cup \mathbf{P}\left(\begin{pmatrix} -A & \mathbf{a}\end{pmatrix}, \mathbf{0}\right) \subseteq \mathbb{K}^{d+1}$ to $\mathbf{P}(A, \mathbf{a}) \cup \mathbf{P}(-A, -\mathbf{a}) \subseteq \mathbb{K}^d$ geometrically corresponds to the intersection of $\tilde{\mathbf{P}}$ with the hyperplane $\left\{\binom{\mathbf{x}}{\lambda} \,\middle|\, \lambda = 1\right\} \subseteq \mathbb{K}^{d+1}$.

Lemma 6 *Any linear map $\phi : \mathbb{K}^{d+1} \to \mathbb{K}^{d'+1}$ is compatible with projective equality $=_p$.*

Proof. Let $\mathbf{u}, \mathbf{v} \in \mathrm{Proj}^d(\mathbb{K})$ with $\mathbf{u} =_p \mathbf{v}$, i. e., there exists some $\lambda \neq 0$ with $\mathbf{u} = \lambda\mathbf{v}$. Then $\phi(\mathbf{u}) = \phi(\lambda\mathbf{v}) = \lambda\phi(\mathbf{v})$, hence $\phi(\mathbf{u}) =_p \phi(\mathbf{v})$. □

Definition 7 *Let $\phi : \mathbb{K}^{d+1} \to \mathbb{K}^{d+1}$ be a bijective linear map. Then ϕ is called a* projectivity.

Proposition 8 *The class of projective polyhedra is closed under projectivities. It is also closed under linear maps $\phi : \mathbb{K}^{d+1} \to \mathbb{K}^{d'+1}$.*

Proof. The proof is obvious since the image of a closed convex cone is a closed convex cone again. □

In case of a projectivity ϕ let M be its transformation matrix. Then we can explicitly state the following formula for the image of a projective polyhedron $\tilde{\mathbf{P}} = \mathbf{P}\left(\begin{pmatrix} A & -\mathbf{a}\end{pmatrix}, \mathbf{0}\right) \cup \mathbf{P}\left(\begin{pmatrix} -A & \mathbf{a}\end{pmatrix}, \mathbf{0}\right)$:

$$M\tilde{\mathbf{P}} = \mathbf{P}\left(\begin{pmatrix} A & -\mathbf{a}\end{pmatrix} M^{-1}, \mathbf{0}\right) \cup \mathbf{P}\left(\begin{pmatrix} -A & \mathbf{a}\end{pmatrix} M^{-1}, \mathbf{0}\right).$$

Notes on Projective Polyhedra. During our research we found the following interesting properties of projective polyhedra:

(i) The class of projective polyhedra is not closed under intersections [5].

(ii) There is a close relationship between projective polyhedra and the set of feasible solutions of the linear fractional program

$$\text{maximize } \frac{\mathbf{c}^T\mathbf{x} + \alpha}{\mathbf{d}^T\mathbf{x} + \beta} \text{ subject to } A\mathbf{x} \leq \mathbf{a}.$$

Under the regularity conditions that $\{\mathbf{x} \mid A\mathbf{x} \leq \mathbf{a}\}$ is bounded and not empty, Charnes and Cooper [2] show that it suffices to solve the two linear programs

$$\begin{aligned}&\text{maximize } \mathbf{c}^T\mathbf{y} + \alpha\lambda \text{ subject to}\\ &\qquad A\mathbf{y} - \mathbf{a}\lambda \leq \mathbf{0}, \mathbf{d}^T\mathbf{y} + \beta\lambda = 1, \lambda \geq 0\\ &\text{maximize } \mathbf{c}^T\mathbf{y} + \alpha\lambda \text{ subject to}\\ &\qquad -A\mathbf{y} + \mathbf{a}\lambda \leq \mathbf{0}, \mathbf{d}^T\mathbf{y} + \beta\lambda = 1, \lambda \leq 0.\end{aligned}$$

It is not hard to see that the intersection of the projective embedding $\mathbf{P}\left((A, -\mathbf{a}), \mathbf{0}\right) \cup \mathbf{P}\left((-A, \mathbf{a}), \mathbf{0}\right)$ of $\mathbf{P}(A, \mathbf{a})$ with the hyperplane $\{\binom{\mathbf{x}}{\lambda} \mid \binom{\mathbf{d}}{\beta}^T \binom{\mathbf{x}}{\lambda} = 1\}$ corresponds to the set of feasible solutions of the linear programs above. Moreover, projective polyhedra allow us to drop the regularity conditions.

In the remainder of this section we return to general, non-polyhedral complete projective embeddings.

Proposition 9 *Let $\mathbf{Q} = \left\{\mathbf{x} \,\middle|\, \mathbf{x}^T Q \mathbf{x} \leq c^2\right\}$ be a quadric. Further, let*

$$\tilde{\mathbf{Q}} = \left\{ \binom{\mathbf{x}}{\lambda} \,\middle|\, \binom{\mathbf{x}}{\lambda}^T \begin{pmatrix} Q & \mathbf{0} \\ \mathbf{0}^T & -c^2 \end{pmatrix} \binom{\mathbf{x}}{\lambda} \leq 0 \right\}.$$

Then $\tilde{\mathbf{Q}}$ is the complete projective embedding of $\mathbf{Q}$.

Proof. Obviously, $\tilde{\mathbf{Q}}$ includes $\iota(\mathbf{Q})$. Furthermore, $\binom{\mathbf{x}}{\lambda}^T \left(\begin{smallmatrix} Q & \mathbf{0} \\ \mathbf{0}^T & -c^2 \end{smallmatrix}\right) \binom{\mathbf{x}}{\lambda} \leq 0$ implies $\mu \binom{\mathbf{x}}{\lambda}^T \left(\begin{smallmatrix} Q & \mathbf{0} \\ \mathbf{0}^T & -c^2 \end{smallmatrix}\right) \mu \binom{\mathbf{x}}{\lambda} \leq 0$ for all $\mu \in \mathbb{K}$. Hence, $\tilde{\mathbf{Q}}$ is projectively complete.

We show that $\tilde{\mathbf{Q}}$ is closed. Let $\left(\binom{\mathbf{x}_i}{\lambda_i}\right)_{i\in\mathbb{N}}$ be a convergent sequence with $\binom{\mathbf{x}_i}{\lambda_i} \in \tilde{\mathbf{Q}}$ for all $i \in \mathbb{N}$ and $\lim_{i\to\infty} \binom{\mathbf{x}_i}{\lambda_i} = \binom{\mathbf{x}}{\lambda}$. Assume, $\lambda \neq 0$. Let J be the set of all indices where $\lambda_i \neq 0$. Then $\left(\binom{\frac{1}{\lambda_i}\mathbf{x}_i}{1}\right)_{i\in J}$ is an infinite sequence which converges to $\binom{\frac{1}{\lambda}\mathbf{x}}{1}$. Furthermore, for all $i \in J$ we have $\frac{1}{\lambda_i} \in \mathbf{Q}$. Since $\mathbf{Q}$ is closed, we have $\frac{1}{\lambda}\mathbf{x} \in \mathbf{Q}$. Hence, $\binom{\frac{1}{\lambda}\mathbf{x}}{1} \in \iota(\mathbf{Q})$ and, by projective completeness, $\binom{\mathbf{x}}{\lambda} \in \tilde{\mathbf{Q}}$. Assume, $\lambda = 0$. Either we have an infinite subset $J \subseteq \mathbb{N}$ such that $\mathbf{x}_i^T Q \mathbf{x}_i \leq 0$ for all $i \in J$. The set $\left\{\mathbf{y} \,\middle|\, \mathbf{y}^T Q \mathbf{y} \leq 0\right\} \subseteq \mathbf{Q}$ is closed. Hence, $\mathbf{x}^T Q \mathbf{x} \leq 0$ and $\binom{\mathbf{x}}{0}^T \left(\begin{smallmatrix} Q & \mathbf{0} \\ \mathbf{0}^T & -c^2 \end{smallmatrix}\right) \binom{\mathbf{x}}{0} \leq 0$. That is, $\binom{\mathbf{x}}{0} \in \tilde{\mathbf{Q}}$. Otherwise, there exists an index i_0 with $\lambda_i \neq 0$ for all $i \geq i_0$. The remaining infinite sequence $\left(\binom{\mathbf{x}_i}{\lambda_i}\right)_{i\geq i_0}$ has the same limes and without loss of generality we may assume that $i = 0$. Setting $\mathbf{y}_i = \mathbf{x}_i - \frac{\lambda_i}{\lambda_0}\mathbf{x}_0$ yields the equalities $\binom{\mathbf{y}_i}{0} = \binom{\mathbf{x}_i}{\lambda_i} - \frac{\lambda_i}{\lambda_0}\binom{\mathbf{x}_0}{\lambda_0}$ for all $i \in \mathbb{N}$. Since $\lim_{i\to\infty} \lambda_i = 0$ and $\lim_{i\to\infty} \mathbf{x}_i = \mathbf{x}$, we obtain $\lim_{i\to\infty} \frac{\lambda_i}{\lambda_0}\mathbf{x}_0 = \mathbf{0}$ and $\lim_{i\to\infty} \binom{\mathbf{y}_i}{0} = \lim_{i\to\infty} \binom{\mathbf{x}_i}{\lambda_i} - \lim_{i\to\infty} \frac{\lambda_i}{\lambda_0}\binom{\mathbf{x}_0}{\lambda_0} = \binom{\mathbf{x}}{0}$. Hence, $\binom{\mathbf{y}_i}{0}$ is a sequence with $\lim_{i\to\infty} \binom{\mathbf{y}_i}{0} = \binom{\mathbf{x}}{\lambda}$ where the last coefficient of the members is equal to zero. For this case we already have shown that $\binom{\mathbf{x}}{0} \in \tilde{\mathbf{Q}}$.

It remains to show that $\tilde{\mathbf{Q}}$ is the least closed projectively complete set which contains $\iota(\mathbf{Q})$. Assume $\tilde{\mathbf{Q}}'$ is another closed projectively complete set that contains $\iota(\mathbf{Q})$. We have to show that any $\binom{\mathbf{x}}{\lambda} \in \tilde{\mathbf{Q}}$ is also in $\tilde{\mathbf{Q}}'$. Therefore, let $\binom{\mathbf{x}}{\lambda} \in \tilde{\mathbf{Q}}$, i. e., $\mathbf{x}^T Q \mathbf{x} \leq \lambda^2 c^2$. In the case $\lambda \neq 0$ we have $\frac{1}{\lambda}\mathbf{x} \in \mathbf{Q}$ and, hence, $\binom{\frac{1}{\lambda}\mathbf{x}}{1} \in \iota(\mathbf{Q})$. The set $\tilde{\mathbf{Q}}'$ includes $\iota(\mathbf{Q})$ and is projectively complete. Hence, $\binom{\mathbf{x}}{\lambda} \in \tilde{\mathbf{Q}}'$. Otherwise, we have $\lambda = 0$, i. e., $\mathbf{x}^T Q \mathbf{x} \leq 0$. Then for any μ we have $\mu\mathbf{x}^T Q \mu\mathbf{x} \leq 0$. Let $\mu_i = i$. Since $0 \leq c^2$, we have $\mu_i \mathbf{x} \in \mathbf{Q}$ for all $i \geq 1$. Further, since $\iota(\mathbf{Q}) \in \tilde{\mathbf{Q}}'$ and $\tilde{\mathbf{Q}}'$ is projectively complete, we also have $\frac{1}{\mu_i}\binom{\mu_i\mathbf{x}}{1} = \binom{\mathbf{x}}{\frac{1}{\mu_i}} \in \tilde{\mathbf{Q}}'$. We obtain $\lim_{i\to\infty} \binom{\mathbf{x}}{\frac{1}{\mu_i}} = \binom{\mathbf{x}}{0} \in \mathbb{K}^{d+1}$. Since $\tilde{\mathbf{Q}}'$ is closed, it follows $\binom{\mathbf{x}}{\lambda} = \binom{\mathbf{x}}{0} \in \tilde{\mathbf{Q}}'$. □

4 Inner Approximation of a Quadric

In the following we show how to find an inner polyhedral approximation of a quadric $\mathbf{Q} = \left\{\mathbf{x} \,\middle|\, \mathbf{x}^T Q \mathbf{x} \leq c^2\right\} \subseteq \mathbb{E}^d$. Without loss of generality we may assume that Q is a symmetric matrix.[2] We will show that it suffices to provide an inscribed polyhedron of the unit sphere in any dimension to find an inner approximation of the quadric. For example, for any dimension d the polyhedron

$$\mathbf{B}(d) = \mathbf{P}\left(\left(\begin{smallmatrix} \mathrm{I} \\ -\mathrm{I} \end{smallmatrix}\right), \frac{1}{\sqrt{d}}\left(\begin{smallmatrix} \mathbf{1} \\ \mathbf{1} \end{smallmatrix}\right)\right) = \left\{\mathbf{x} \,\middle|\, \mathbf{x} \leq \tfrac{1}{\sqrt{d}}\mathbf{1}, -\mathbf{x} \leq \tfrac{1}{\sqrt{d}}\mathbf{1}\right\}$$

is an inscribed hypercube of the hyperball

$$\mathbf{S}^d = \left\{\mathbf{x} \,\middle|\, \mathbf{x}^T\mathbf{x} \leq 1\right\} = \left\{\mathbf{x} \,\middle|\, x_1^2 + x_2^2 + \cdots + x_d^2 \leq 1\right\}.$$

Proposition 10 (Principal Axis Transformation) *Let $\mathbf{Q} = \left\{\mathbf{x} \,\middle|\, \mathbf{x}^T Q \mathbf{x} \leq c^2\right\}$ be a quadric in $\mathbb{E}^d$ where Q is a symmetric matrix. Further, let*

$$\tilde{\mathbf{Q}} = \left\{ \binom{\mathbf{x}}{\lambda} \,\middle|\, \binom{\mathbf{x}}{\lambda}^T \tilde{Q} \binom{\mathbf{x}}{\lambda} \leq 0 \right\} \text{ with } \tilde{Q} = \begin{pmatrix} Q & \mathbf{0} \\ \mathbf{0}^T & -c^2 \end{pmatrix}$$

be the complete projective embedding of $\mathbf{Q}$ in $\mathbb{E}^{d+1}$. Then there exists an invertible matrix L and a diagonal matrix

$$E = \left(\begin{smallmatrix} \mathrm{I} & \mathrm{O} & \mathrm{O} \\ \mathrm{O} & \mathrm{O} & \mathrm{O} \\ \mathrm{O} & \mathrm{O} & -\mathrm{I} \end{smallmatrix}\right) \quad \textit{such that} \quad \tilde{Q} = LEL^T.$$

While the matrix L is in general not unique, the matrix E is uniquely determined by Sylvester's law of inertia. If Q is positive semidefinite, then E has exactly one negative entry.

Proof. Symmetric Gaussian elimination yields an invertible diagonal matrix $D = \operatorname{diag}(d_1, \ldots, d_{d+1}) = U\tilde{Q}U^T$ where U represents the row operations and the

[2] Otherwise, let $\hat{Q} = \frac{1}{2}(Q + Q^T)$, i. e., $\mathbf{x}^T\hat{Q}\mathbf{x} = \frac{1}{2}(\mathbf{x}^T Q\mathbf{x} + \mathbf{x}^T Q^T\mathbf{x}) = \frac{1}{2}(\mathbf{x}^T Q\mathbf{x} + (\mathbf{x}^T Q^T\mathbf{x})^T) = \frac{1}{2}(\mathbf{x}^T Q\mathbf{x} + \mathbf{x}^T Q\mathbf{x}) = \mathbf{x}^T Q\mathbf{x}$ for all $\mathbf{x}$, and use $\hat{Q}$ instead of Q.

transposed matrix U^T represents the corresponding column operations. We normalize the elements of D by multiplying with the matrix $S = \mathrm{diag}(s_1, \ldots, s_{d+1}) = S^T$, where each s_i is defined as $s_i = 1$ if $d_i = 0$, or $s_i = \frac{1}{\sqrt{|d_i|}}$ if $d_i \neq 0$. Then $SDS^T = SU\tilde{Q}U^TS^T$ is a diagonal matrix whose entries are 1, -1 or 0. Finally, we sort the entries of the diagonal matrix SDS^T in descending order, yielding a row permutation matrix P and a corresponding column permutation matrix P^T and $E = \left(\begin{smallmatrix} \mathrm{I} & \mathrm{O} & \mathrm{O} \\ \mathrm{O} & \mathrm{O} & \mathrm{O} \\ \mathrm{O} & \mathrm{O} & -\mathrm{I} \end{smallmatrix}\right) = PSDS^TP^T = PSU\tilde{Q}U^TS^TP^T$. Since P, S, and U are invertible, we define $L = (PSU)^{-1}$ and obtain the factorization $\tilde{Q} = LEL^T$. □

Note that the decomposition $\tilde{Q} = LEL^T$ as given above is only possible in a projective vector space. In a non-projective setting we would obtain a richer variety of resulting forms.

Let $\mathbf{Q}$, $\tilde{\mathbf{Q}}$, Q, $\tilde{Q}$, and L, E be given as in Prop. 10. We define $\binom{\mathbf{y}}{\mu} = L^T\binom{\mathbf{x}}{\lambda}$ and obtain the identity $\mathbf{x}^TQ\mathbf{x} - \lambda^2c^2 = \binom{\mathbf{x}}{\lambda}^T\tilde{Q}\binom{\mathbf{x}}{\lambda} = \binom{\mathbf{x}}{\lambda}^TLEL^T\binom{\mathbf{x}}{\lambda} = \binom{\mathbf{y}}{\mu}^TE\binom{\mathbf{y}}{\mu}$. Hence, the base transformation matrix L transforms the projective quadric $\tilde{\mathbf{Q}}$ into a projective hyperboloid of the form $\tilde{\mathbf{H}} = \left\{\binom{\mathbf{y}}{\mu} \;\middle|\; \binom{\mathbf{y}}{\mu}^TE\binom{\mathbf{y}}{\mu}\right\}$.

We explain how to find an inner approximation of the projective hyperboloid $\tilde{\mathbf{H}}$. Application of the inverse base transformation finally yields (a union of two) inscribed polyhedra of $\mathbf{Q}$. Let k the number of 1s, l the numbers of 0s, and m the number of -1s in E, with $k + l + m = d + 1$. According to Sylvester's law of inertia, k, l, and m are uniquely determined by Q.

We characterize the solutions $\binom{\mathbf{y}}{\mu}$ of $\binom{\mathbf{y}}{\mu}^TE\binom{\mathbf{y}}{\mu} \leq 0$, or equivalently:

$$y_1^2 + \cdots + y_k^2 - y_{k+l+1}^2 - \cdots - y_d^2 - \mu^2 \leq 0. \qquad (2)$$

We have the following mutual exclusive cases:

(i) If $k = 0$, then $\tilde{\mathbf{H}} = \mathbb{E}^{d+1}$, i. e., $\tilde{\mathbf{H}}$ is the complete projective embedding of the entire vector space $\mathbb{E}^d$.

(ii) If $k = d+1$, then $\tilde{\mathbf{H}} = \{\mathbf{0}\}$, i. e., $\tilde{\mathbf{H}}$ is the complete projective embedding of the empty set.

(iii) If $0 < k \leq d+1$ and $m = 0$, then $\tilde{\mathbf{H}} = \{0\}^k \times \mathbb{E}^{d-k} \times \mathbb{E}$, i. e., $\tilde{\mathbf{H}}$ is the complete projective embedding of the subspace $\{0\}^k \times \mathbb{E}^{d-k}$.

(iv) If $0 < k < d+1$ and $m = 1$, then $\tilde{\mathbf{H}} = \{\binom{\mathbf{y}}{\mu} \mid \mathbf{y}^T(\begin{smallmatrix} \mathrm{I} & \mathrm{O} \\ \mathrm{O} & \mathrm{O} \end{smallmatrix})\mathbf{y} \leq \mu^2\}$ is the complete projective embedding of the cylinder $\mathbf{S}^k \times \mathbb{E}^{d-k}$. Let $\mathbf{T}(k) = \mathbf{P}(A, \mathbf{a})$ be an inscribed polyhedron of $\mathbf{S}^k$. The cylindrification $\mathbf{T}(k) \times \mathbb{E}^{d-k}$ of $\mathbf{T}(k)$ is given by $\mathbf{P} = \mathbf{P}\left(\begin{pmatrix} A & \mathrm{O} \end{pmatrix}, \mathbf{a}\right)$. We have $\mathbf{P} \subseteq \mathbf{S}^k \times \mathbb{E}^{d-k}$. According to Prop. 3 the subset relation carries over to the complete projective embeddings $\tilde{\mathbf{P}} \subseteq \tilde{\mathbf{H}}$, where $\tilde{\mathbf{P}} = \mathbf{P}\left(\left(\begin{smallmatrix} A & \mathrm{O} & -\mathbf{a} \\ \mathbf{0}^T & \mathbf{0}^T & -1 \end{smallmatrix}\right), \binom{\mathbf{0}}{0}\right) \cup \mathbf{P}\left(\left(\begin{smallmatrix} -A & \mathrm{O} & \mathbf{a} \\ \mathbf{0}^T & \mathbf{0}^T & 1 \end{smallmatrix}\right), \binom{\mathbf{0}}{0}\right)$. Hence, after application of the base transformation L^{-T} on $\tilde{\mathbf{P}}$ we obtain a union of polyhedra $L^T(\tilde{\mathbf{P}}) = \mathbf{P}\left(\begin{pmatrix} A_1 & -\mathbf{a}_1 \end{pmatrix}, \mathbf{0}\right) \cup \mathbf{P}\left(\begin{pmatrix} -A_1 & \mathbf{a}_1 \end{pmatrix}, \mathbf{0}\right)$ with $L^T(\tilde{\mathbf{P}}) \subseteq \tilde{\mathbf{Q}}$ and the matrices are given by $\begin{pmatrix} A_1 & -\mathbf{a}_1 \end{pmatrix} = \left(\begin{smallmatrix} A & \mathrm{O} & -\mathbf{a} \\ \mathbf{0}^T & \mathbf{0}^T & -1 \end{smallmatrix}\right)L^T$ and $\begin{pmatrix} -A_1 & \mathbf{a}_1 \end{pmatrix} = \left(\begin{smallmatrix} -A & \mathrm{O} & \mathbf{a} \\ \mathbf{0}^T & \mathbf{0}^T & 1 \end{smallmatrix}\right)L^T$. According to Prop. 5, $L^T(\tilde{\mathbf{P}})$ is the complete projective embedding of $\mathbf{R} = \mathbf{P}(A_1, \mathbf{a}_1) \cup \mathbf{P}(-A_1, -\mathbf{a}_1)$. Furthermore, by Prop. 3 we have $\mathbf{R} \subseteq \mathbf{Q}$.

(v) Otherwise, we have $0 < k < d+1$ and $m > 1$. Now, let E' be the matrix which is obtained by replacing all but the last occurrence of -1 in E by 0, i. e., $E' = \mathrm{diag}(1, \ldots, 1, 0, \ldots, 0, -1)$. Any solution of $\binom{\mathbf{y}}{\mu}^TE'\binom{\mathbf{y}}{\mu} \leq 0$ is also a solution of (2), since for all $y_1, \ldots, y_d$ it holds $y_1^2 + \cdots + y_k^2 \geq y_1^2 + \cdots + y_k^2 - y_{k+l}^2 - \cdots - y_d^2$. Hence, $\tilde{\mathbf{H}}' = \left\{\binom{\mathbf{y}}{\mu} \;\middle|\; \binom{\mathbf{y}}{\mu}^TE'\binom{\mathbf{y}}{\mu} \leq 0\right\}$ is the complete projective embedding of the cylinder $\mathbf{S}^k \times \mathbb{E}^{d-k}$ and $\tilde{\mathbf{H}}' \subseteq \tilde{\mathbf{H}}$. Now, we proceed like in the previous item, where we use E' and $\tilde{\mathbf{H}}'$ instead of E and $\tilde{\mathbf{H}}$.

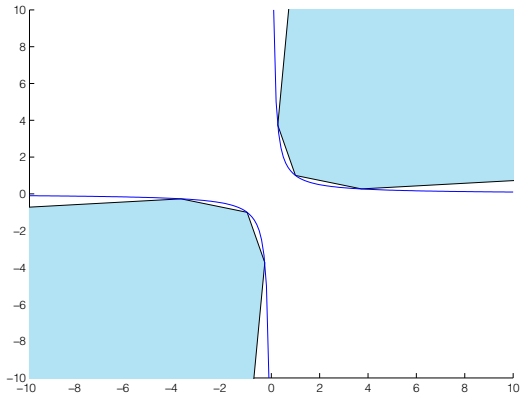


Figure 1: Hexagon Inscribed in Hyperbola $xy \geq 1$

5 Generating Polytopes With Circumsphere

In this section we discuss the problem of generating polytopes with circumsphere of arbitrary dimensions.

Obviously, the convex hull of several sampling points $\mathbf{v}_1, \ldots, \mathbf{v}_n$ on the boundary of the d-dimensional hyperball $\mathbf{S}$ results in an inscribed $\mathcal{V}$-polytope $\mathbf{P}$. Hence, to obtain the corresponding $\mathcal{H}$-representation, we had to perform a costly facet-enumeration and lose control on the number of facets of the resulting $\mathcal{H}$-representation. Instead, we are interested in methods which allow us to generate inscribed $\mathcal{H}$-polytopes directly. We abandon the idea of taking samples and provide a more regular way to generate inscribed $\mathcal{H}$-polyhedra. An interesting class of polytopes with circumsphere is the class of uniform polytopes [4, 3] which includes the class of regular

polytopes. A complete enumeration of uniform polyhedra is only known in low dimensions. In two dimensions the uniform polytopes are the infinitely many regular polygons. In three dimensions the uniform polytopes cover the five Platonic solids, the 13 Archimedean solids, and the infinite set of prisms and antiprisms. In all dimensions the class of uniform polytopes contains the regular simplex, the hypercube, and the cross polytope. A d-dimensional simplex has only $d+1$ facets and vertices, which is certainly insufficient to provide a good inner approximation of a sphere. The d-dimensional cross polytope has 2^d facets and $2d$ vertices. Hence, the resulting $\mathcal{H}$-polytopes are only feasible in lower dimensions. The dual of a d-dimensional cross polytope is the d-dimensional hypercube having $2d$ facets and 2^d vertices. Apparently, the hypercube is well suited for computational purposes, but still may have too less facets to provide a good inner approximation of the sphere.

Hence, it is desirable to have a regular method which allows us to generate a richer variety of uniform polytopes. The next proposition provides such a method.

Proposition 11 *Given a polytope* $\mathbf{P}_1 = \mathbf{P}(A_1, \mathbf{a}_1) \in \mathbb{K}^{d_1}$ *with unit circumsphere* $\mathbf{S}^{d_1}$ *and a polytope* $\mathbf{P}_2 = \mathbf{P}(A_2, \mathbf{a}_2) \in \mathbb{K}^{d_2}$ *with unit circumsphere* $\mathbf{S}^{d_2}$*. Then for any* $\alpha > 0$*,* $\beta > 0$ *with* $\alpha^2 + \beta^2 = 1$ *the weighted cross product*

$$\begin{aligned}\mathbf{P} &= \alpha\mathbf{P}_1 \times \beta\mathbf{P}_2 = \left\{\binom{\mathbf{x}}{\mathbf{y}} \,\middle|\, \mathbf{x} \in \alpha\mathbf{P}_1,\, \mathbf{y} \in \beta\mathbf{P}_2\right\} \\ &= \mathbf{P}\left(\begin{pmatrix} A_1 & \mathrm{O} \\ \mathrm{O} & A_2 \end{pmatrix}, \binom{\alpha\mathbf{a}_1}{\beta\mathbf{a}_2}\right)\end{aligned}$$

is a polytope in $\mathbb{K}^{d_1+d_2}$ *with unit circumsphere* $\mathbf{S}^{d_1+d_2}$*. Furthermore, let* f_1*,* f_2 *be the number of facets of* $\mathbf{P}_1$ *and* $\mathbf{P}_2$*, and let* v_1*,* v_2 *be the number of vertices of* $\mathbf{P}_1$ *and* $\mathbf{P}_2$*. Then* $\mathbf{P}$ *has* $f_1 + f_2$ *facets and* $v_1 v_2$ *vertices.*

Proof. The statement on the number of vertices and facets is rather obvious and belongs to mathematical folklore. We show that $\mathbf{P}$ has the circumsphere $\mathbf{S}^{d_1+d_2}$. That is, for any vertex $\binom{\alpha\mathbf{x}_i}{\beta\mathbf{y}_j}$, $i = 1, \ldots, v_1$, $j = 1, \ldots, v_2$ of $\mathbf{P}$ we have $\left|\binom{\alpha\mathbf{x}_i}{\beta\mathbf{y}_j}\right| = \binom{\alpha\mathbf{x}_i}{\beta\mathbf{y}_j}^T \binom{\alpha\mathbf{x}_i}{\beta\mathbf{y}_j} = \alpha^2\mathbf{x}_i^T\mathbf{x}_i + \beta^2\mathbf{y}_j^T\mathbf{y}_j = \alpha^2 + \beta^2 = 1$. □

Clearly, the previous proposition can be generalized to the weighted cross product of finitely many circumscribed polytopes. For example, the d-dimensional hypercube is the d-fold weighted cross product of the line segments $[-1, 1]$ with weight $\alpha = \frac{1}{\sqrt{d}}$.

6 Conclusion

We discussed a projective generalization of closed sets which coincides in the case of polyhedral closed set with Gallier's projective polyhedra and the set of feasible solutions of a linear fractional program. Within this framework we used projective base transformation to generate inscribed $\mathcal{H}$-polyhedra of arbitrary quadrics provided we have given inscribed $\mathcal{H}$-polyhedra of the unit sphere. Finally, we discussed an easy method to generate inscribed polyhedra of a higher dimensional sphere out of inscribed polyhedra of spheres of lower dimension.

We have not discussed any quantifiable predication on the quality of the approximation, like ratio of the volumes or Hausdorff distance of both sets. Although it would be interesting to have some measure for the quality of the approximation, both notions are not well-defined for unbounded sets which may occur in our setting. However, for our purpose – finding inner approximation of a level set – the presented approach turned out to work well and there was no need to extend our very limited selection of template polyhedra with circumsphere.

References

[1] M. Berger. *Geometry I*, volume 1. Springer, 1987.

[2] A. Charnes and W. W. Cooper. Programming with linear fractional functionals. *Naval Research logistics quarterly*, 9(3-4):181–186, 1962.

[3] H. S. M. Coxeter. *Regular polytopes*. Methuen, London, 1948.

[4] H. S. M. Coxeter, M. S. Longuet-Higgins, and J. Miller. Uniform polyhedra. *Philosophical Transactions of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 246(916):401–450, 1954.

[5] J. Gallier. Notes on convex sets, polytopes, polyhedra, combinatorial topology, Voronoi diagrams and Delaunay triangulations. Technical Report 650, University of Pennsylvania Department of Computer and Information Science, 2009.

[6] J. Gallier. *Geometric methods and applications: for computer science and engineering*, volume 38. Springer, 2011.

[7] W. Hagemann. Reachability analysis of hybrid systems using symbolic orthogonal projections. In A. Biere and R. Bloem, editors, *Computer Aided Verification*, volume 8559 of *Lecture Notes in Computer Science*, pages 406–422. Springer, 2014.

[8] W. Hagemann, E. Möhlmann, and O. E. Theel. Hybrid tools for hybrid systems – proving stability and safety at once. In *Formal Modeling and Analysis of Timed Systems*, Lecture Notes in Computer Science. Springer, 2015. to appear.

1-String B_1-VPG Representations of Planar Partial 3-Trees and Some Subclasses

Therese Biedl* Martin Derka†

Abstract

Planar partial 3-trees are subgraphs of those planar graphs obtained by repeatedly inserting a vertex of degree 3 into a face. In this paper, we show that planar partial 3-trees have 1-string B_1-VPG representations, i.e., representations where every vertex is represented by an orthogonal curve with at most one bend, every two curves intersect at most once, and intersections of curves correspond to edges in the graph. We also show that some subclasses of planar partial 3-trees have {L}-representations, i.e., a B_1-VPG representation where every curve has the shape of an L.

1 Introduction

A *string representation* is a representation of a graph where every vertex v is assigned a curve $\mathbf{v}$. Vertices u, v are connected by an edge if and only if curves $\mathbf{u}, \mathbf{v}$ intersect. A 1-*string representation* is a string representation where every two curves intersect at most once.

String representations of planar graphs were first investigated by Ehrlich, Even and Tarjan in 1976 [12]. They showed that every planar graph has a 1-string representation using "general" curves. In 1984, Scheinerman conjectured [18] that every planar graph has a 1-string representation, and furthermore curves are line segments (not necessarily axis-parallel). Chalopin, Gonçalves and Ochem [7, 8] proved that every planar graph has a 1-string representation in 2007. Scheinerman's conjecture itself remained open until 2009 when it was proved true by Chalopin and Gonçalves [6].

Our paper investigates string representations that use *orthogonal curves*, i.e., curves consisting of vertical and horizontal segments. If every curve has at most k bends, these are called B_k-*VPG representations*. The hierarchy of B_k-VPG representations was introduced by Asinowski et al. [1, 2]. VPG is an acronym for Vertex-Path-Grid since vertices are represented by paths in a rectangular grid.

It is easy to see that all planar graphs are VPG-graphs (e.g. by generalizing the construction of Ehrlich, Even and Tarjan). For bipartite planar graphs, curves can even be required to have no bends [17, 11]. For arbitrary planar graphs, bends in orthogonal curves are required. Chaplick and Ueckerdt showed that 2 bends per curve always suffice [10]. In a recent paper we strengthened this to give a B_2-VPG representation that is also a 1-string representation [3].

B_k-VPG representations were further studied by Chaplick, Jelínek, Kratochvíl and Vyskočil [9] who showed that recognizing B_k-VPG graphs is NP-complete even when the input graph is given by a B_{k+1}-VPG representation, and that for every k, the class of B_{k+1}-VPG graphs is strictly larger than B_k-VPG.

Our Contribution Felsner et al. [14] showed that every planar 3-tree has a B_1-VPG representation. Moreover, every vertex-curve has the shape of an L (we call this an {L}-*representation*). This implies that any two vertex-curves intersect at most once, so this is a 1-string B_1-VPG representation. In this paper, we extend the result to more graphs, and in particular, show:

Theorem 1 *Every planar partial 3-tree G has a 1-string B_1-VPG representation.*

There are 4 possible shapes of orthogonal curves with one bend. Depending on where the bend is situated, we call them L, Γ, ⅂ and ⅃ respectively. Note that a horizontal or vertical curve without bends can be turned into any of the shapes by adding one bend.

The construction of our proof of Theorem 1 uses all 4 possible shapes L, Γ, ⅂, ⅃. However, for some subclasses of planar partial 3-trees, we can show that fewer shapes suffice. We use the notation {L, ⅂}-representation for a B_1-VPG representation where all curves are either L or ⅂, and similarly for other subsets of shapes. We can show the following:

Theorem 2 *Any IO-graph has an {L}-representation.*

Theorem 3 *Any Halin-graph has an {L, ⅂}-representation, and only one vertex uses a ⅂-shape.*

We give the definitions of these graph classes and the proof of these theorems in the next three sections, and end with open problems in Section 5.

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

†David R. Cheriton School of Computer Science, University of Waterloo, mderka@uwaterloo.ca. Research supported by an NSERC Vanier CGS.

2 Planar partial 3-trees

A *planar graph* is a graph that can be drawn without edge crossings. If one such drawing Γ is fixed, then a *face* is a maximal connected region of $\mathbb{R}^2 - \Gamma$. The *outer face* corresponds to the unbounded region; the *interior faces* are all other faces. A vertex is called *exterior* if it is on the outer face and *interior* otherwise.

A 3-*tree* is a graph that is either a triangle or has a vertex order $v_1, \ldots, v_n$ such that for $i \geq 4$, vertex v_i is adjacent to exactly three predecessors and they form a triangle. A *partial* 3-*tree* is a subgraph of a 3-tree.

Our proof of Theorem 1 employs the method of "private regions" used previously for various string representation constructions [3, 8, 14]. We define the following:

Definition 1 (F-shape and rectangular shape) *An* F-shaped area *is a region bounded by a* 10*-sided polygon with CW or CCW sequence of interior angles* 90°, 270°, 90°, 90°, 270°, 270°, 90°, 90°, 90° *and* 90°. *A* rectangle-shaped area *is a region bounded by an axis-aligned rectangle.*

Definition 2 (Private region) *Given a* 1*-string representation, a* private region *of vertices* $\{a, b, c\}$ *is an F-shaped or rectangle-shaped area that intersects (up to permutation of names) curves* $\mathbf{a}, \mathbf{b}, \mathbf{c}$ *in the way depicted in Figure 1(a), and that intersects no other curves and private regions.*

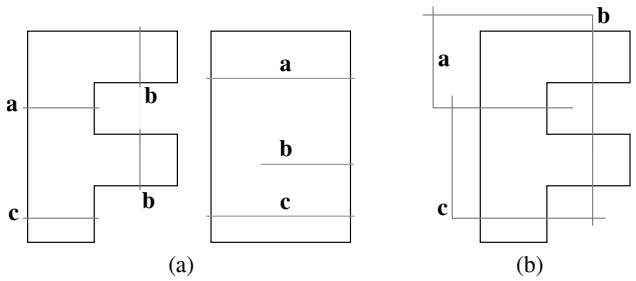


Figure 1: (a) An F-shaped (left) and rectangle-shaped (right) private region of $\{a, b, c\}$. (b) The base case. Intersections among $\{\mathbf{a}, \mathbf{b}, \mathbf{c}\}$ can be omitted as needed.

Now we are ready to prove Theorem 1. Let G be a planar partial 3-tree. By definition, there exists a 3-tree H for which G is a subgraph. One can show [4] that we may assume H to be planar. Let $v_1, \ldots, v_n$ be a vertex order of H such that for $i \geq 4$ vertex v_i is adjacent to 3 predecessors that form a triangle. In particular, v_4 is incident to a triangle formed by $\{v_1, v_2, v_3\}$. One can show (see e.g. [4]) that the vertex order can be chosen in such a way that $\{v_1, v_2, v_3\}$ is the outer face of H in some planar drawing.

For $i \geq 3$, let G_i and H_i be the subgraphs of G (respectively H) induced by vertices $v_1, \ldots, v_i$. We prove Theorem 1 by showing the following by induction on i:

> G_i has a 1-string B_1-VPG representation with a private region for every interior face of H_i.

In the base case, $i = 3$ and $G \subseteq K_3 \simeq H$. Construct a representation R and find a private region for the unique interior face of H as depicted in Figure 1(b).

Now consider $i \geq 4$. By induction, construct a representation R_0 of G_{i-1} that contains a private region for every interior face of H_{i-1}.

Let $\{a, b, c\}$ be the predecessors of v_i in H. Recall that they form a triangle. Since H is planar, this triangle must form a face in H_{i-1}. Since $\{v_1, v_2, v_3\}$ is the outer face of H (and hence also of H_{i-1}), the face into which v_i is added must be an interior face, so there exists an interior face $\{a, b, c\}$ in H_{i-1}. Let P_0 be the private region that exists for $\{a, b, c\}$ in R_0; it can have the shape of an F or a rectangle.

Observe that in G, vertex v_i may be adjacent to any possible subset of $\{a, b, c\}$. This gives 16 cases (two possible shapes, up to rotation and reflection, and 8 possible adjacencies).

In each case, the goal is to place a curve $\mathbf{v_i}$ inside P_0 such that it intersects exactly the curves of the neighbours of v_i in $\{a, b, c\}$ and no other curve. Furthermore, having placed $\mathbf{v_i}$ into P_0, we need to find a private region for the three new interior faces in H_i, that is, the three faces formed by v_i and two of $\{a, b, c\}$.

Case 1: P_0 has the shape of an F. After possible rotation / reflection of R_0 and renaming of $\{a, b, c\}$ we may assume that P_0 appears as in Figure 1(a). If (v_i, a) is an edge, then place a bend for curve $\mathbf{v_i}$ in the region above $\mathbf{a}$. Let the vertical segment of $\mathbf{v_i}$ intersect $\mathbf{a}$ and (optionally) $\mathbf{c}$. Let the horizontal segment of $\mathbf{v_i}$ intersect (optionally) the top occurrence of $\mathbf{b}$. If (v_i, a) is not an edge but (v_i, c) is an edge, then place a bend for $\mathbf{v_i}$ in the region below $\mathbf{a}$, let the vertical segment of $\mathbf{v_i}$ intersect $\mathbf{c}$ and the horizontal segment of $\mathbf{v_i}$ intersect (optionally) $\mathbf{b}$. Finally, if neither (v_i, a) nor (v_i, c) is an edge, then $\mathbf{v_i}$ is a horizontal segment in the region below $\mathbf{a}$ and above $\mathbf{c}$ that (optionally) intersects $\mathbf{b}$.

In all sub-cases, $\mathbf{v_i}$ remains inside P_0, so it cannot intersect any other curve of R_0. Private regions for the newly created faces can be found as shown in Figure 2.

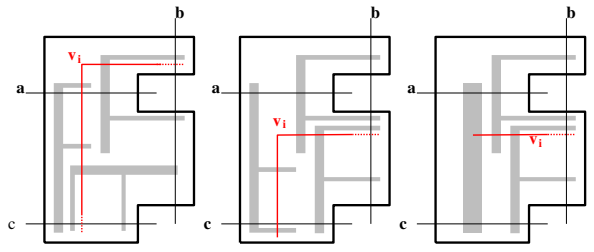


Figure 2: Inserting curve $\mathbf{v_i}$ into an F-shaped private region. (Left) (v_i, a) is an edge. (Middle) $(v_i, a) \notin E$, but $(v_i, c) \in E$. (Right) $(v_i, a), (v_i, c) \notin E$.

Case 2: P_0 **has the shape of a rectangle.** After possible rotation / reflection of R_0 and renaming of $\{a, b, c\}$ we may assume that P_0 appears as in Figure 1(a). If (v_i, a) is an edge, then $\mathbf{v}$ is a vertical segment that intersects $\mathbf{a}$ and (optionally) $\mathbf{b}$ and (optionally) $\mathbf{c}$. If (v_i, c) is an edge, then symmetrically $\mathbf{v_i}$ is a vertical segment that intersects $\mathbf{c}$ and (optionally) $\mathbf{b}$ and $\mathbf{a}$. Finally if neither (v_i, a) nor (v_i, c) is an edge, then let $\mathbf{v}$ be a horizontal segment between $\mathbf{a}$ and $\mathbf{c}$ with (optionally) a vertical segment attached to create an intersection with $\mathbf{b}$.

In all cases, $\mathbf{v_i}$ remains inside P_0, so it cannot intersect any other curve of R_0. Private regions for the newly created faces can be found as shown in Figure 3.

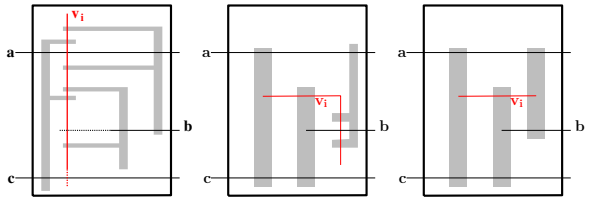


Figure 3: Inserting curve $\mathbf{v_i}$ into a rectangle-shaped private region. (Left) (v_i, a) is an edge. (Middle) $(v_i, a), (v_i, c) \notin E$, but $(v_i, b) \in E$. (Right) $(v_i, a), (v_i, b), (v_i, c) \notin E$.

Theorem 1 now holds by induction. □

We note here that in our proof-approach, both types of private regions and all four shapes with one bend are required in some cases.

3 IO-Graphs

An *IO-graph* [13] is a 2-connected planar graph with a planar embedding such that the interior vertices form a (possibly empty) independent set. One can easily show [13] that every IO-graph is a planar partial 3-tree.

We now prove Theorem 2 by constructing an $\{\mathtt{L}\}$-representation of an IO-graph G. Let O be the set of exterior vertices; by definition these induce an *outerplanar graph*, i.e., a graph that can be embedded so that all vertices are on the outer face. Moreover, since G is 2-connected, the outer face is a simple cycle, and hence the outerplanar graph $G[O]$ is also 2-connected. We first construct an $\{\mathtt{L}\}$-representation of $G[O]$, and then insert the interior vertices. To do so, we again use private regions, but we modify their definition slightly in three ways: (1) Interior vertices may have arbitrarily high degree, and so the private regions must be allowed to cross arbitrarily many curves. (2) Interior vertices may only be adjacent to exterior vertices. It therefore suffices for the private region of a face f to intersect only those curves that belong to exterior vertices on f.

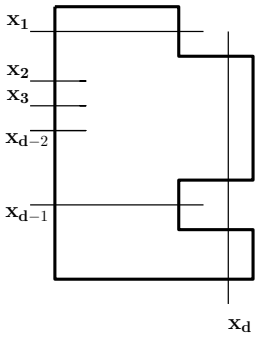


Figure 4: An IO-private region. We require that the supporting line of $\mathbf{x_i}$ (for $i = 2, \ldots, k-2$) intersects the upper segment of $\mathbf{x_d}$.

It is exactly this latter observation that allows us to find private regions more easily, therefore use fewer shapes for them, and therefore use fewer shapes for the curves. We can therefore also add: (3) The private region must be an F-shape, and it must be in the rotation $\mathtt{L}$. The formal definition is given below:

Definition 3 (IO-private region) *Given a 1-string representation of an IO-graph, an* IO-private region *of a face f is an F-shaped area P, in the rotation $\mathtt{L}$, which intersects curves $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_d}$ as shown in Figure 4. Here, $\{x_1, \ldots, x_d\}$ is a subset of the vertices of f enumerated in CCW order, and includes all exterior vertices that belong to f (it may or may not include other vertices). Lastly, P intersects no other curves and no other private regions.*

Lemma 4 *Any outer planar graph has an $\{L\}$-representation with an IO-private region for every interior face.*

Proof. We may assume that the outerplanar graph is 2-connected, otherwise we can add vertices to make it so and delete their curves later. Enumerate the vertices on the outer face as $v_1, \ldots, v_k$ in CCW order. For every vertex v_i on the outer-face, let $\mathbf{v_i}$ be an $\mathtt{L}$ with bend at $(i, -i)$. The vertical segment of $\mathbf{v_i}$ reaches until $(i, -r_i + \varepsilon)$, where $r_i = \min\{j : (v_j, v_i) \in E\}$. (Use $r_0 = 0$.) The horizontal segment of $\mathbf{v_i}$ reaches until $(s_i + \varepsilon, i)$, where $s_i = \max\{j : (v_j, v_i) \in E\}$. (Use $s_k = k$.) See also Figure 5.

It is quite easy to see that this is a 1-string representation. For every edge (v_i, v_k) with $i < k$ we have created an intersection at $(k, -i)$. Assume for contradiction that $\mathbf{v_i}$ and $\mathbf{v_k}$ intersect for some $(v_i, v_k) \notin E$ with $i < k$. Then we must have $s = \max\{j : (v_i, v_j) \in E\} > k$, else there is no intersection. Also $r = \min\{j : (v_j, v_k) \in E\} < i$, else there is no intersection. But then $\{v_i, v_k, v_s, v_r\}$, together with the outer face, form a K_4-minor; this is impossible in an outer planar graph.

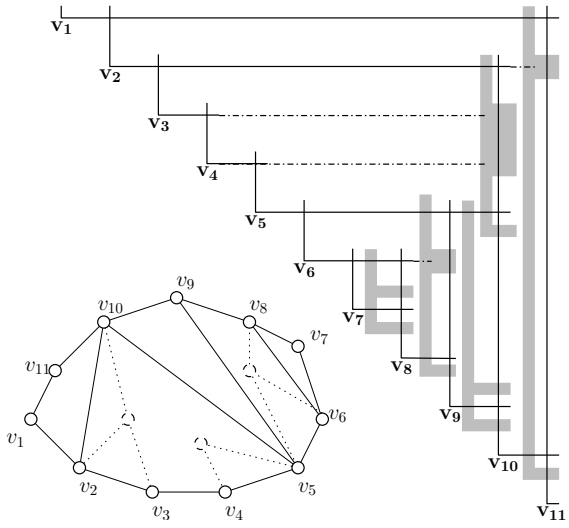


Figure 5: Example of an IO-graph and the {L}-representation of $G[O]$. The IO-private regions are shaded in grey.

Thus we found the {L}-representation. To find IO-private regions, we stretch horizontal segments of curves further as follows. For vertex v_i, set $t_i = \max\{j : v_i$ and v_j are on a common interior face$\}$. If $t_i > s_i$, then expand $\mathbf{v_i}$ horizontally until $t_i - \varepsilon$. To see that this does not introduce new crossings, observe that adding (v_i, v_{t_i}) to the graph would not destroy outerplanarity, since the edge could be routed inside the common face. The {L}-representation of such an expanded graph would contain the constructed one and also contain the added segment. Therefore the added segment cannot intersect any other curves.

After stretching all curves horizontally in this way, an IO-private region for each interior face f can then be inserted to the left of the vertical segment of $\mathbf{v_j}$, where v_j is the vertex on f with maximal index; see also Figure 5. □

Now we can prove Theorem 2, i.e., we can show that every IO-graph G has an {L}-representation. Start with the {L}-representation of $G[O]$ of Lemma 4. We add the interior vertices $v_1, \ldots, v_{n-k}$ to this in arbitrary order, maintaining the following invariant:

> For every interior face of the current graph there exists an IO-private region.

Clearly this invariant holds for the representation of $G[O]$. Let v be the next interior vertex to be added, and let f be the face where it should be inserted. By induction there exists a IO-private region P_0 for face f such that the curves $\mathbf{x_1}, \ldots, \mathbf{x_d}$ that intersect P_0 include the curves of all exterior vertices that are on f, in CCW order. We need to place an L-curve $\mathbf{v}$ into P_0, intersecting curves of neighbours of v and nothing else, and then find IO-private regions for every newly created face.

Since the interior vertices form an independent set, all neighbours of v are on the outer face, and hence belong to $\{x_1, \ldots, x_d\}$. Since G is 2-connected, v has at least two such neighbours. We have two cases.

Case 1. If (v, x_d) is not an edge, then $\mathbf{v}$ is a vertical segment that extends from the topmost to the bottommost of the curves of its neighbours, and intersects these curves after expanding them rightwards.

Since the order of $\mathbf{x_1}, \ldots, \mathbf{x_d}$ is CCW around the outer face, for every newly created face f' incident to v we have a region inside P_0 in which the curves of outer face vertices on f' appear in CCW order. IO-private regions for these faces can be found as shown in Figure 6(top). Note that some of these private regions intersect v while others do not; both are acceptable since v is on those faces, but not an exterior vertex.

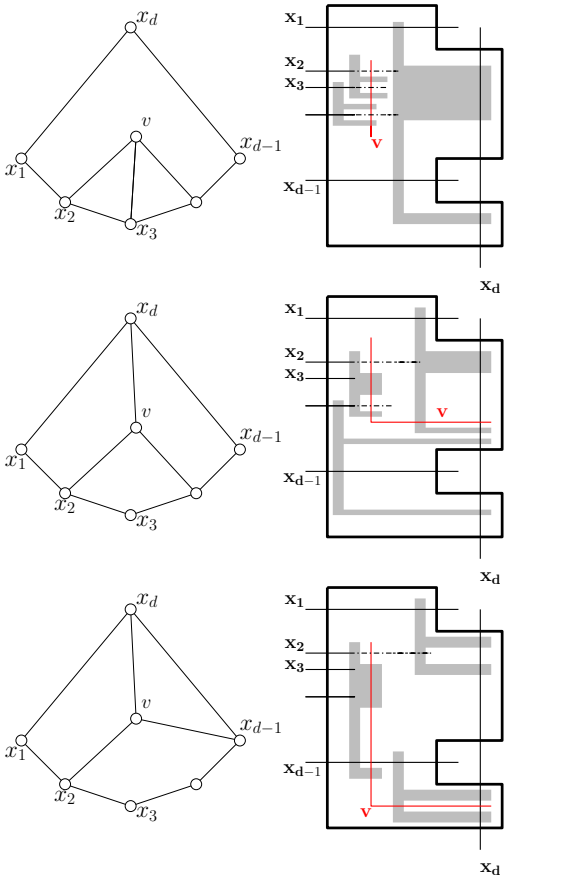


Figure 6: Inserting a vertex into a face of an IO-graph. (Top) v is not adjacent to x_d. (Middle) v is adjacent to x_d, but not x_{d-1}. (Bottom) v is adjacent to both x_d and x_{d-1}.

Case 2. If (v, x_d) is an edge, then $\mathbf{v}$ is an L, with the bend below $\mathbf{x_{d-1}}$ if (v, x_{d-1}) is an edge and above

$\mathbf{x_{d-1}}$ otherwise. The vertical segment of $\mathbf{v}$ extends from this bend to the topmost of v's neighbours in $\{\mathbf{x_1}, \ldots, \mathbf{x_{d-1}}\}$, and intersects the curves of these neighbours after expanding them rightwards. The horizontal segment extends as to intersect $\mathbf{x_d}$.

IO-private regions can again be found easily, see Figure 6(middle and bottom).

Repeating this insertion operation for all interior vertices hence gives the desired representation of G. □

4 Halin graphs

A *Halin-graph* [16] is a graph obtained by taking a tree T with $n \geq 3$ vertices that has no vertex of degree 2 and connecting the leaves in a cycle. Such graphs were originally of interest since they are minimally 3-connected, but it was later shown that they are also planar partial 3-trees [5].

We now prove Theorem 3 and show that any Halin-graph G has a {⊤,L}-representation. We note here that our construction works even if T has some vertices of degree 2. Fix an embedding of G such that the outer face is the cycle C connecting the leaves of tree T. Enumerate the outer face as $v_1, \ldots, v_k$ in CCW order. Since every exterior vertex was a leaf of T, vertex v_k has degree 3; let r be the interior vertex that is a neighbour of v_k. Root T at r and enumerate the vertices of T in post-order as $w_1, \ldots, w_n$, starting with the leaves (which are $v_1, \ldots, v_k$) and ending with r.

Let G_i be the graph induced by $w_1, \ldots, w_i$. Call vertex v_j *unfinished* in G_i if it has a neighbour in $G - G_i$. For $i = k, \ldots, n$, we create an {L}-representation of $G_i - (v_1, v_k)$ that satisfies the following:

> For any unfinished vertex v, curve $\mathbf{v}$ ends in a horizontal ray, and the top-to-bottom order of these rays corresponds to the CW order of the unfinished vertices on the outer face while walking from v_1 to v_k.

The {L}-representation of $G_k - (v_1, v_k)$ (i.e., the path $v_1, \ldots, v_k$) is obtained easily by placing the bend for $\mathbf{v_i}$ at $(i, -i)$, giving the vertical segment length $1 + \varepsilon$ and leaving the horizontal segment as a ray as desired. To add vertex w_i for $i > k$, let $x_1, \ldots, x_d$ be its children in T; their curves have been placed already. Insert a vertical segment for $\mathbf{w_i}$ with x-coordinate i, and extending from just below the lowest curve of $\mathbf{x_1}, \ldots, \mathbf{x_d}$ to just above the highest. The rays of $\mathbf{x_1}, \ldots, \mathbf{x_d}$ end at x-coordinate $i + \varepsilon$, while $\mathbf{w_i}$ appends a horizontal ray at its lower endpoint.

Since adding w_i means that $x_1, \ldots, x_d$ are now finished (no vertex has two parents), the invariant holds. Continuing until $i = n$ yields an {L}-representation of $G - (v_1, v_k)$. It remains to add an intersection for edge (v_1, v_k). To do so, we change the shape of $\mathbf{v_1}$. Observe that its vertical segment was not used for any intersection, and that its horizontal segment can be expanded until $(n + 1, -1)$ without intersecting anything except its neighbours. After this expansion, we add a vertical segment going downward at its right end. Since v_k is a neighbour of r, curve $\mathbf{v_k}$ ended when $\mathbf{r}$ was added, i.e., at x-coordinate $n + \varepsilon$, and we can extend it until x-coordinate $n + 1 + \varepsilon$. Hence $\mathbf{v_1}$ and $\mathbf{v_k}$ can meet at $(n + 1, -k)$ if we change the shape of $\mathbf{v_1}$ to ⊤. We have hence proved Theorem 3. □

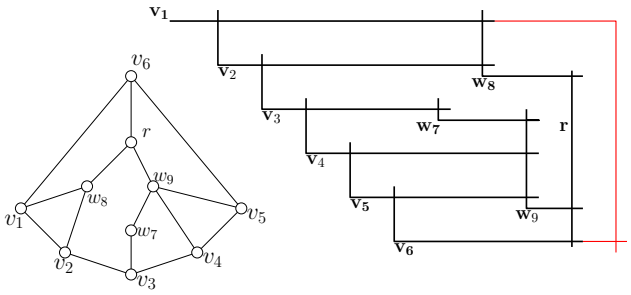


Figure 7: Example of an extended Halin-graph and its {L, ⊤}-representation, obtained by changing the curve of $\mathbf{v_1}$ so that it intersects $\mathbf{v_k}$.

Notice that in the construction for Halin-graphs, any intersection of curves occurs near the end one of the two curves. Our result therefore holds not only for Halin graphs, but also for any subgraph of a Halin graph.

The natural question to ask is whether any Halin graph has an {L}-representation, i.e., whether it is possible to avoid the single ⊤-shape that we used for $\mathbf{v_1}$. In very recent work [15] done independently from ours, Francis and Lahiri answered this question affirmatively and proved that every Halin graph has an {L}-representation.

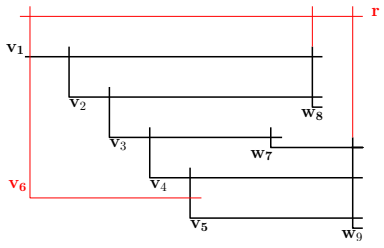


Figure 8: {L}-representation, obtained by changing the curve of $\mathbf{r}$ and $\mathbf{v_k}$, if r has no other neighbours on the outer face.

5 Conclusion

In this paper, we studied 1-string VPG-representations of planar graphs such that curves have at most one bend. It is not known whether all planar graphs have such a

representation, but curiously, also no planar graph is known that does not have an {L}-representation. Felsner et al. [14] asked whether every planar graph has a {Γ, L}-representation since (as they point out) a positive answer would provide a different proof of Scheinerman's conjecture. They proved this for planar 3-trees.

In this paper, we made another step towards their question and showed that every planar partial 3-tree has a 1-string B_1-VPG representation. We also showed that IO-graphs and Halin-graphs have {L}-representations, except that for Halin-graphs one vertex curve might be a ⊺.

The obvious direction for future work is to show that all planar partial 3-trees have {L}-representations, or at least {L, Γ}-representations. As a first step, an interesting subclass would be those 2-connected planar graphs G where deleting the vertices on the outer face leaves a forest; these encompass both IO-graphs and Halin graphs.

Note that all representations constructed in this paper are *ordered*, in the sense that the order of intersections along the curves of vertices corresponds to the order of edges around the vertex in a planar embedding. This is not the case for the 1-string B_2-VPG-representations in our earlier construction [3]. One possible avenue towards showing that planar graphs do not always have an {L}-representation is to restrict the attention to ordered representations first. Thus, is there a planar graph that has no ordered {L}-representation?

References

[1] Andrei Asinowski, Elad Cohen, Martin Charles Golumbic, Vincent Limouzy, Marina Lipshteyn, and Michal Stern. String graphs of k-bend paths on a grid. *Electronic Notes in Discrete Mathematics*, 37:141–146, 2011.

[2] Andrei Asinowski, Elad Cohen, Martin Charles Golumbic, Vincent Limouzy, Marina Lipshteyn, and Michal Stern. Vertex intersection graphs of paths on a grid. *J. Graph Algorithms Appl.*, 16(2):129–150, 2012.

[3] Therese Biedl and Martin Derka. 1-string B_2-VPG representations of planar graphs. *Symposium on Computational Geometry (SoCG'15)*, 2015. To appear.

[4] Therese Biedl and Lesvia Elena Ruiz Velázquez. Drawing planar 3-trees with given face areas. *Comput. Geom.*, 46(3):276–285, 2013.

[5] Hans Bodlaender. Planar graphs with bounded treewidth. Technical Report RUU-CS-88-14, Rijksuniversiteit Utrecht, 1988.

[6] Jérémie Chalopin and Daniel Gonçalves. Every planar graph is the intersection graph of segments in the plane: extended abstract. In *ACM Symposium on Theory of Computing, STOC 2009*, pages 631–638. ACM, 2009.

[7] Jérémie Chalopin, Daniel Gonçalves, and Pascal Ochem. Planar graphs are in 1-string. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, pages 609–617, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.

[8] Jérémie Chalopin, Daniel Gonçalves, and Pascal Ochem. Planar graphs have 1-string representations. *Discrete & Computational Geometry*, 43(3):626–647, 2010.

[9] Steven Chaplick, Vít Jelínek, Jan Kratochvíl, and Tomás Vyskocil. Bend-bounded path intersection graphs: Sausages, noodles, and waffles on a grill. In *Graph-Theoretic Concepts in Computer Science - WG 2012*, volume 7551 of *Lecture Notes in Computer Science*, pages 274–285. Springer, 2012.

[10] Steven Chaplick and Torsten Ueckerdt. Planar graphs as VPG-graphs. *J. Graph Alg. Appl.*, 17(4):475–494, 2013.

[11] Hubert de Fraysseix, Patrice Ossona de Mendez, and János Pach. Representation of planar graphs by segments. *Intuitive Geometry*, 63:109–117, 1991.

[12] Gideon Ehrlich, Shimon Even, and Robert Endre Tarjan. Intersection graphs of curves in the plane. *J. Comb. Theory, Ser. B*, 21(1):8–20, 1976.

[13] Ehab S. El-Mallah and Charles J. Colbourn. Partitioning the edges of a planar graph into two partial k-trees. *Congr. Numerantium 66*, page 69–80, 1988.

[14] Stefan Felsner, Kolja B. Knauer, George B. Mertzios, and Torsten Ueckerdt. Intersection graphs of l-shapes and segments in the plane. In *Mathematical Foundations of Computer Science - MFCS 2014*, volume 8635 of *Lecture Notes in Computer Science*, pages 299–310. Springer, 2014.

[15] Mathew C. Francis and Abhiruk Lahiri. VPG and EPG bend-numbers of Halin graphs. *arXiv:1505.06036*, 2015.

[16] Rudof Halin. Studies on minimally n-connected graphs. In *Combinatorial Mathematics and its Applications*, pages 129–136. Academic Press, London, 1971.

[17] Irith Ben-Arroyo Hartman, Ilan Newman, and Ran Ziv. On grid intersection graphs. *Discrete Mathematics*, 87(1):41–52, 1991.

[18] Edward R. Scheinerman. *Intersection Classes and Multiple Intersection Parameters of Graphs*. PhD thesis, Princeton University, 1984.

Diversity Maximization via Composable Coresets

Sepideh Aghamolaei* Majid Farhadi* Hamid Zarrabi-Zadeh*

Abstract

Given a set S of points in a metric space, and a diversity measure $\mathrm{div}(\cdot)$ defined over subsets of S, the goal of the *diversity maximization* problem is to find a subset $T \subseteq S$ of size k that maximizes $\mathrm{div}(T)$. Motivated by applications in massive data processing, we consider the composable coreset framework in which a coreset for a diversity measure is called α-composable, if for any collection of sets and their corresponding coresets, the maximum diversity of the union of the coresets α-approximates the maximum diversity of the union of the sets. We present composable coresets with near-optimal approximation factors for several notions of diversity, including remote-clique, remote-cycle, and remote-tree. We also prove a general lower bound on the approximation factor of composable coresets for a large class of diversity maximization problems.

1 Introduction

The *diversity maximization* problem—finding a subset of k points to maximize some function of the inter-point distances—is a fundamental problem in location theory [20, 21] and has received considerable attention over the past few years, due to its application to search result diversification [5, 6, 14]. Various notions of diversity have been studied in the literature, most of which are proved to be NP-hard in both metric and geometric settings, and hence, the focus has been on providing efficient approximation algorithms. Among the most well-studied diversity problems are *remote-edge*, whose objective is to maximize the minimum distance in the k-subset [7, 11, 22], and the *remote-clique* problem, whose aim is to maximize the average distance [8, 12, 13, 18]. There are also some results on maximizing other combinatorial structures such as minimum spanning trees and minimum-weight tours [10, 16].

Motivated by applications in massive data processing, we consider the coreset framework, which is a fundamental tool for designing approximation algorithms, especially for large data sets [4]. In this framework, a small subset of input data set, called a "coreset", is extracted in such a way that solving the optimization problem on the coreset yields a solution to the whole data set with a guaranteed approximation factor. Many coresets considered in the literature are "decomposable" in the sense that taking the union of two coresets computed for two given sets yields a coreset for the union of those two sets with the same approximation guarantee. This property is essentially useful for designing streaming algorithms [9, 17], as it allows to maintain a coreset for the points recently inserted, and merge it to the coreset maintained for the rest of the points.

In [24], Zarrabi-Zadeh introduced a special class of decomposable coresets, called "core-preserving", having an additional property that taking a coreset of a coreset yields a coreset with the same size and approximation factor. Such coresets are in particular useful for obtaining streaming algorithms whose working space is independent of the size of input. The idea was used to obtain efficient streaming algorithms for problems such as k-center [24] and maintaining ε-kernels of fat point sets [25]. A similar idea was coined as "mergeable coresets" by Agarwal *et al.* [3], and was used to obtain better algorithms for maintaining statistical data summaries in the data stream model.

Very recently, Indyk *et al.* [19] introduced the notion of "composable coresets" in which the union of a collection of coresets gives a coreset for the points in the union of the sets within a guaranteed approximation factor. All decomposable coresets (and hence, core-preserving and mergeable coresets) are composable by definition. However, in composable coresets, the approximation factor may be increased after taking union, though it is still guaranteed to be within a certain factor. Composable coresets are in particular useful for distributed settings and MapReduce computation, in which a massive point set is partitioned among a set of machines/mappers, and each machine maps its input data into a composable coreset. A single reducer then takes the union of all the coresets received from the mappers, and computes a solution to the union, which is guaranteed to be within a good approximation factor.

Our contributions. In this paper, we revisit the composable coresets framework of Indyk *et al.* [19], and further refine it to the notion of "disjoint composable coresets", in which input data sets are assumed to be disjoint. We present improved composable coresets for several diversity maximization problems in both disjoint and non-disjoint settings. The problems studied in this

*Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. Email: {aghamolaei, m_farhadi}@ce.sharif.edu, zarrabi@sharif.edu.

Problem	Diversity Measure	Approximation Factor		
		Previous [19]	New	
			Disjoint	General
Remote-edge	$\min_{p,q\in S} d(p,q)$	3	$3^\dagger$	$3^\dagger$
Remote-clique	$\sum_{p,q\in S} d(p,q)$	51	$6+\varepsilon$	$7+4\sqrt{2}+\varepsilon$
Remote-star	$\min_{p\in S} \sum_{q\in S\setminus\{p\}} d(p,q)$	102	12	26
Remote-bipartition	$\min_{Q\subset S,\lvert Q\rvert=k/2} \sum_{p\in Q,q\in S\setminus Q} d(p,q)$	255	18	38
Remote-tree	$w(\mathrm{MST}(S))$	6	4	4
Remote-cycle	$w(\mathrm{TSP}(S))$	12	$3^\dagger$	$3^\dagger$
Remote t-trees	$\min_{S=S_1\rvert\cdots\rvert S_t} \sum_{i=1}^{t} w(\mathrm{MST}(S_i))$	6	4	4
Remote t-cycles	$\min_{S=S_1\rvert\cdots\rvert S_t} \sum_{i=1}^{t} w(\mathrm{TSP}(S_i))$	12	5	5

Table 1: Summary of the new results. In this table, S denotes the input set, d is the distance function in the underlying metric space, $\varepsilon > 0$ is an arbitrarily small constant, and $S = S_1|\cdots|S_t$ denotes a partition of S into t subsets. Tight factors are marked with † sign.

paper are listed and formally defined in Table 1. Here is a brief summary of our results.

- For the remote-clique problem, a factor-51 composable coreset was presented in [19]. When input sets are disjoint, we show that a much better approximation factor of $6+\varepsilon$ (for any $\varepsilon > 0$) is achievable for the problem. In general non-disjoint case, we provide an approximation factor of $7+4\sqrt{2}+\varepsilon \approx 12.66+\varepsilon$, greatly improving over the best previous factor of 51.

- For the remote-edge problem, a factor-3 composable coreset was presented in [1, 19]. Indyk *et al.* [19] left this question open whether a better approximation factor is possible. We settle this question in negative by showing that 3 is the best factor possible for the remote-edge problem. Our proof is indeed very general, and implies a lower bound of 3 for all notions of diversity listed in Table 1.

- We show that for any point set, the weight of its clique approximates the weight of its minimum partition to within a factor of $3-\frac{4}{k}$, improving upon the previous bound of 5 proved in [19]. Combined with our new factor for the remote-clique problem, this yields improved factors of 18 and 38 for the remote partition problem in the disjoint and non-disjoint settings, respectively, substantially improving over the previous bound of 255 available for the problem.

- We prove a tight upper bound of $2-\frac{2}{k}$ on the ratio of the weight of the minimum star of a point set and the weight of its clique. This yields improved factors of 12 and 26 for the remote-star problem in the disjoint and non-disjoint settings, respectively, greatly improving over the previous bound of 102 available for the problem.

- For the remote-cycle problem, we present a factor-3 composable coreset, improving the best previous bound of 12 available for the problem. Our coreset is indeed optimal, considering the general lower bound of 3 that we have presented in this paper.

- For the remote-tree and remote t-trees problems, we provide an approximation factor of 4, improving over the best previous factor of 6 obtained in [19]. We also improve the approximation factor of the remote t-cycles problem from 12 to 5.

As with many other approximation algorithms, our algorithms for extracting the coresets are simple, and are based on two known off-line algorithms, namely the Gonzalez's algorithm and the local search. However, the analyses of the approximation factors are non-trivial, and are based on finding a careful mapping from the points in the optimal solution to the points in the coreset, while keeping the error incurred as small as possible.

2 Preliminaries

Let (X, d) be a metric space, and f be a measure defined over subsets of X. A function $c(\cdot)$ that maps a set $S \subseteq X$ into one of its subsets is called an *α-composable coreset* for f, if for any collection of sets $S_1, \ldots, S_\ell$, with $S = \cup_{i=1}^{\ell} S_i$ and $T = \cup_{i=1}^{\ell} c(S_i)$,

$$\max\left\{\frac{f(S)}{f(T)}, \frac{f(T)}{f(S)}\right\} \leqslant \alpha.$$

The value $\alpha \geqslant 1$ is called the *approximation factor* of the coreset. A *disjoint α-composable coreset* is analogously defined, with an additional property that the input sets S_i are assumed to be disjoint.

Given a point set S in a metric space (X, d), we denote by $G[S]$ a complete graph over vertex set S, with edge weights specified by the metric distance d. Let Π denote a specific graph structure (e.g., a clique or a spanning

Algorithm 1 GMM(S, k)

1: $T \leftarrow \{\text{an arbitrary point } p \in S\}$
2: **for** $i = 2, \ldots, k$ **do**
3: find a point $p \in S \setminus T$ maximizing $d(p, T)$
4: $T \leftarrow T \cup \{p\}$
5: **return** T

Algorithm 2 LocalSearch(S, k)

1: $T \leftarrow$ a k-subset of S containing the two farthest pts
2: **while** $\exists\ p \in T,\ q \in S \setminus T$ s.t.
$\operatorname{div}(T \setminus \{p\} \cup \{q\}) > (1 + \frac{\varepsilon}{k}) \operatorname{div}(T)$ **do**
3: $T \leftarrow T \setminus \{p\} \cup \{q\}$
4: **return** T

tree). Following the terminology of [10], we define the remote-Π problem as follows. For a point set $S \subseteq X$, the *diversity* of S (with respect to Π), denoted by $\operatorname{div}(S)$, is the weight of a Π structure in $G[S]$ whose total edge weight is minimum. The *k-diversity* of S, denoted by $\operatorname{div}_k(S)$, is the maximum diversity over all k-subsets of S, i.e., $\operatorname{div}_k(S) = \max_{P \subseteq S, |P|=k} \operatorname{div}(P)$. The *remote-*Π problem is then to compute, for a given point set S and a parameter k, the k-diversity of S with respect to Π. For example, the remote-tree problem involves finding a k-subset of S whose minimum spanning tree has maximum weight. Note that $\operatorname{div}_k(S)$ is undefined when $|S| < k$.

For a weighted graph G, we denote by $w(G)$ the total weight of the edges in G. Given a set S, we denote by $S = S_1 | \cdots | S_t$ the partition of S into t disjoint subsets $S_1, \ldots, S_t$.

2.1 Algorithms

The two offline algorithms that we will use for computing the coresets are the Gonzalez's algorithm and the local search. The Gonzalez's algorithm [15], presented in Algorithm 1, starts from an arbitrary point, and iteratively adds a point whose distance to the points already chosen is maximized. If r denotes the minimum pairwise distance in the set $T = \text{GMM}(S, k)$, then the following two properties, known as *anti-cover* properties, hold:

- $\forall p \in T : d(p, T \setminus \{p\}) \geqslant r$
- $\forall p \in S : d(p, T) \leqslant r$

The local search algorithm [2], presented in Algorithm 2, starts with an arbitrary subset of size k containing the two farthest points, and then, at each iteration tries to locally improve its current solution by exchanging a single point. The total number of iterations of this algorithm is at most $\log_{1+\frac{\varepsilon}{k}}(k^2) = O(\frac{k}{\varepsilon} \log k)$.

3 Composable Coresets for Diversity Problems

Consider a collection of sets $S_1, \ldots, S_\ell$, and let $S = \cup_{i=1}^{\ell} S_i$. For each set S_i, we compute a coreset $T_i = c(S_i)$, and set $T = \cup_{i=1}^{\ell} T_i$. Let O be an optimal solution for S, i.e. a k-subset of S for which $\operatorname{div}(O) = \operatorname{div}_k(S)$. We denote by O_i the portion of O lying inside S_i, but not in any other S_j $(j < i)$, i.e., $O_i = O \cap S_i \setminus \cup_{j<i} S_j$. This partitions O into ℓ disjoint subsets O_i.

In the following, we obtain upper bounds on the approximation factor of composable coresets designed for various notions of diversity. More precisely, we show how to compute coresets T_i such that their union T is a good representation of S, i.e., its diversity is within a guaranteed factor of $\operatorname{div}_k(S)$. We accomplish this by comparing the k-diversity of T with that of O, which is in turn equal to the k-diversity of S.

3.1 Remote Clique

In this section, we show that the local search algorithm computes a factor $6 + \varepsilon$ composable coreset for the remote-clique problem when input sets are disjoint. Throughout this subsection, $\operatorname{div}(\cdot)$ refers to the remote-clique diversity.

Let $T_i = \text{LocalSearch}(S_i, k)$. We denote by r_i the average weight of edges in T_i, i.e., $r_i = \operatorname{div}(T_i)/\binom{k}{2}$, and set $r = \max_i \{r_i\}$. Note that, for $i = \arg\max_i \{r_i\}$, $\operatorname{div}_k(T) \geqslant \operatorname{div}_k(T_i) = \binom{k}{2} r_i = \binom{k}{2} r$. We first prove the following lemma.

Lemma 1 *For any point $o \in O_i \setminus T_i$,*

$$\sum_{t \in T_i} d(o, t) \leqslant (1 + \varepsilon) k r.$$

Proof. For any $a \in T_i$, the termination condition of local search implies that

$$\operatorname{div}(T_i \setminus \{a\} \cup \{o\}) \leqslant (1 + \frac{\varepsilon}{k}) \operatorname{div}(T_i).$$

By the definition of remote-clique diversity we have

$$\sum_{p,q \in T_i} d(p, q) - \sum_{t \in T_i} d(a, t) + \sum_{t \in T_i} d(o, t) - d(o, a) \leqslant (1 + \frac{\varepsilon}{k}) \operatorname{div}(T_i).$$

Summing over all points $a \in T_i$, we get

$$k \operatorname{div}(T_i) - 2 \operatorname{div}(T_i) + k \sum_{t \in T_i} d(o, t) - \sum_{t \in T_i} d(o, t) \leqslant (k + \varepsilon) \operatorname{div}(T_i),$$

which simplifies to

$$(k - 1) \sum_{t \in T_i} d(o, t) \leqslant (2 + \varepsilon) \operatorname{div}(T_i).$$

Replacing $r_i = \text{div}(T_i)/\binom{k}{2}$, we get

$$\sum_{t \in T_i} d(o,t) \leqslant (1+\frac{\varepsilon}{2}) \times kr_i \leqslant (1+\varepsilon)kr.$$

Hence, the proof. □

Lemma 2 *Let $Q_i = O_i \setminus T_i$. There exists a bipartite matching between Q_i and T_i that covers Q_i and has weight at most $(1+\varepsilon)|Q_i|r$.*

Proof. Let M be the set of all maximal bipartite matchings between Q_i and T_i. Any maximal matching in M covers Q_i, because $|Q_i| \leqslant |T_i|$. There are $P(k,|Q_i|) = \frac{k!}{(k-|Q_i|)!}$ matchings in M. Each edge $(q,t) \in Q_i \times T_i$ appears in exactly $P(k-1,|Q_i|-1)$ of such matchings. Therefore, the sum of the weights of all matchings in M is:

$$\begin{aligned} &P(k-1,|Q_i|-1) \sum_{q \in Q_i} \sum_{t \in T_i} d(q,t) \\ &\quad \leqslant P(k-1,|Q_i|-1) \sum_{q \in Q_i} (1+\varepsilon)kr \\ &\quad = P(k-1,|Q_i|-1)(1+\varepsilon)|Q_i|kr \\ &\quad = P(k,|Q_i|)(1+\varepsilon)|Q_i|r, \end{aligned}$$

where the first inequality holds by Theorem 1. Therefore, the expected weight of the matchings in M is at most $(1+\varepsilon)|Q_i|r$, and hence, there must exist a matching in M whose weight does not exceed this expectation. □

Theorem 3 *The local search algorithm computes a factor-$(6+\varepsilon)$ disjoint composable coreset for the remote-clique problem.*

Proof. Let M_i be a maximal bipartite matching between Q_i and T_i, obtained by Lemma 2. Let M be the union of M_i's. Since all T_i's are disjoint, M forms a matching between $Q = O \setminus T$ and T that covers all vertices of Q and has weight at most $(1+\varepsilon)|Q|r$.

Let $f : O \to T$ be a function that maps each vertex $o \in O \cap T$ to o itself, and each vertex $o \in O \setminus T$ to the vertex matched to o by M. The weight of this mapping is equal to the weight of M, and hence, is at most $(1+\varepsilon)|Q|r$. Moreover, for each vertex in range(f), there are at most two vertices of O mapped to it. Now, we can use triangle inequality to get:

$$\begin{aligned} \text{div}(O) &= \sum_{o_1,o_2 \in O} d(o_1,o_2) \\ &\leqslant \sum_{o_1,o_2 \in O} [d(o_1,f(o_1)) + d(f(o_1),f(o_2)) + d(f(o_2),o_2)] \\ &= (|O|-1) \sum_{o \in O} d(o,f(o)) + \sum_{o_1,o_2 \in O} d(f(o_1),f(o_2)) \\ &\leqslant (|O|-1)(1+\varepsilon)(|Q|r) + 4\,\text{div}(\text{range}(f)) \\ &\leqslant 2(1+\varepsilon)\binom{k}{2}r + 4\,\text{div}_k(T) \leqslant (6+2\varepsilon)\,\text{div}_k(T), \end{aligned}$$

where in the last two inequalities we used $|Q| \leqslant |O| = k$, and $\text{div}_k(T) \geqslant \binom{k}{2}r$. □

Remark. When input sets are not necessarily disjoint, we prove that the local search algorithm computes a factor $7+4\sqrt{2}+\varepsilon$ composable coreset for the remote-clique problem. Details will be provided in the full version.

3.2 Remote Bipartition and Remote Star

In order to provide improved composable coresets for the remote-bipartition and remote-star problems, we first show that the weight of the clique of a point set approximates the weight of the minimum bipartition and the minimum star of that point set to within factors $3-\frac{4}{k}$ and $2-\frac{2}{k}$, respectively. These improve the previous bounds of 5 and 2, respectively, proved in [19]. Both our new bounds are indeed tight.

Lemma 4 *For any point set of size $k \geqslant 2$, the weight of its clique is a $(3-\frac{4}{k})$-approximation of the weight of its minimum bipartition. This bound is tight.*

Proof. Recall that a bipartition of a point set P of size k is obtained by dividing P into two subsets L and R of equal size $k/2$, and the weight of such bipartition is the total weight of edges between L and R. It is clear from the definition that $w(\text{bipartition}(L,R)) \leqslant w(\text{clique}(P))$. By triangle inequality, for any two vertices $u,v \in R$ and any $w \in L$, we have $d(u,v) \leqslant d(u,w)+d(w,v)$. Summing this inequality over all $u,v \in R$ and $w \in L$ yields:

$$\frac{k}{2} \times w(\text{clique}(R)) \leqslant \left(\frac{k}{2}-1\right) w(\text{bipartition}(L,R)).$$

The same inequality holds for $w(\text{clique}(L))$. Therefore, $w(\text{clique}(P)) = w(\text{clique}(L)) + w(\text{clique}(R)) + w(\text{bipartition}(L,R)) \leqslant \left(3-\frac{4}{k}\right) w(\text{bipartition}(L,R))$. To see tightness, consider an example in which all edges inside L and R have weight 2, and the edges between L and R have weight 1. The approximation factor in this case is $\left((\frac{k}{2})^2 + 4\binom{k/2}{2}\right)/(\frac{k}{2})^2 = 3-\frac{4}{k}$. □

The proof for the remote-star is similar, and is omitted here. Combined with the factor-$(6+\varepsilon)$ composable coreset for the remote-clique problem presented in Theorem 3, and by setting $\varepsilon = O(1/k)$, we get the following result.

Theorem 5 *The local search algorithm computes a factor-12 composable coreset for the remote-star problem, and a factor-18 composable coreset for the remote-bipartition problem, when input sets are disjoint.*

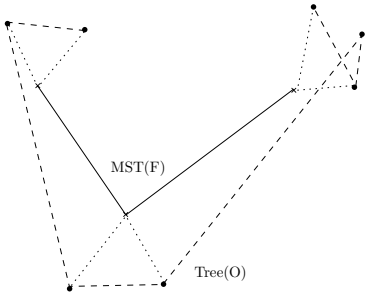


Figure 1: Tree(O) built from MST(F). Dotted lines show the mapping from O to F.

Remark. When input sets are not disjoint, our improved composable coreset for the remote-clique problem which has an approximation factor of $7+4\sqrt{2}+\varepsilon \approx 12.32+\varepsilon$ yields a factor-26 composable coreset for the remote-star problem, and a factor-38 composable coreset for the remote-bipartition problem.

3.3 Remote Tree and Remote Cycle

In this section, we provide a factor-4 composable coreset for the remote-tree problem, and a factor-3 composable coreset for the remote-cycle problem. For both problems, we first run GMM on each S_i to obtain $T_i = \text{GMM}(S_i, k)$. We then obtain the union of the coresets $T = \cup_{i=1}^{\ell} T_i$, and set $r = \max_i \min_{p,q \in T_i} d(p,q)$.

Theorem 6 *The GMM algorithm computes a factor-4 composable coreset for the remote-tree problem.*

Proof. Let $\text{div}(S) = w(\text{MST}(S))$ denote the remote-tree diversity, and let O be a k-subset of S maximizing $\text{div}(O)$. We show that $\text{div}(O) \leqslant 4\,\text{div}_k(T)$.

Consider a mapping $f : O \to T$ that maps each point $o \in O$ to its closest point in T. Let $F = \{f(o) : o \in O\} \subseteq T$ be the range of f, and fix a minimum spanning tree MST(F) of F.

We partition O into subsets $Q_1, \ldots, Q_m$ such that $p, q \in Q_i$ if and only if $f(p) = f(q)$. We now build a spanning tree Tree(O) on O by first building an arbitrary tree on each subset Q_i, and then connecting two components Q_i and Q_j if there are $o_i \in Q_i$ and $o_j \in Q_j$ such that $f(o_i)$ and $f(o_j)$ are connected in MST(F). (See Figure 1.)

By the anticover property of GMM, the length of edges between each o_i and $f(o_i)$ is at most r. So, by triangle inequality, the total cost of edges corresponding to the trees Q_i is at most $(k - |F|) \times 2r$. For each edge $e_f \in \text{MST}(F)$, there is an edge $e_o \in \text{Tree}(O)$ such that $e_o \leqslant e_f + 2r$. There are $|F| - 1$ such edges in total. Therefore,

$$\begin{aligned} w(\text{Tree}(O)) &\leqslant w(\text{MST}(F)) + 2r(k - |F|) + 2r(|F| - 1) \\ &= w(\text{MST}(F)) + 2r(k-1). \end{aligned}$$

Now, let $R \subseteq T$ be an arbitrary superset of F of size k, and let ST(R) be a minimum Steiner tree of R that connects the vertices of F. It is well-known that $w(\text{MST}(F)) \leqslant 2 \cdot w(\text{ST}(R))$ (see, e.g., [23]). Moreover, it is obvious that $w(\text{ST}(R)) \leqslant w(\text{MST}(R))$, because ST($R$) is a minimum-weight tree that only connects a subset of R, as opposed to MST(R) that connects all points in R. Therefore, we have $w(\text{MST}(F)) \leqslant 2 \cdot w(\text{MST}(R))$, and hence,

$$\begin{aligned} w(\text{MST}(O)) &\leqslant w(\text{Tree}(O)) \\ &\leqslant w(\text{MST}(F)) + 2r(k-1) \\ &\leqslant 2 \cdot w(\text{MST}(R)) + 2\,\text{div}_k(T) \leqslant 4\,\text{div}_k(T), \end{aligned}$$

where, the inequality $(k-1)r \leqslant \text{div}_k(T)$ follows from the fact that by the definition of r, there is a set T_i with k points whose pairwise distance is at least r. □

Theorem 7 *The GMM algorithm computes a factor-3 composable coreset for the remote-cycle problem.*

Proof. Let $\text{div}(S) = w(\text{TSP}(S))$ denote the remote-cycle diversity, and let O be a k-subset of S maximizing $\text{div}(O)$. We show that $\text{div}(O) \leqslant 3\,\text{div}_k(T)$.

Consider a function $f : O \to T$ that maps each vertex $o \in O$ to its closest point in T. By the anticover property of GMM, we have $d(o, f(o)) \leqslant r$. Let $R = \{f(o) : o \in O\} \subseteq T$ be the range of f, and let TSP(R) be an optimal tour on R.

We build a graph G on the vertex set $O \cup R$, by first adding to G the edges of TSP(R), and then, adding for each $o \in O$, two copies of the edge $(o, f(o))$ to G. Obviously, G is connected and all its vertices are even. Therefore, G contains an Eulerian tour E. Let C be a cycle obtained from E by short-cutting the vertices not in O. Then,

$$\begin{aligned} w(\text{TSP}(O)) &\leqslant w(C) \leqslant w(E) \\ &\leqslant w(\text{TSP}(R)) + 2kr \\ &\leqslant w(\text{TSP}(R)) + 2\,\text{div}_k(T) \leqslant 3\,\text{div}_k(T), \end{aligned}$$

where, the inequality $w(\text{TSP}(R)) \leqslant \text{div}_k(T)$ holds because TSP($\cdot$) is a monotone increasing function—i.e., for any $A \subseteq B$, we have $w(\text{TSP}(A)) \leqslant w(\text{TSP}(B))$. Moreover, the inequality $kr \leqslant \text{div}_k(T)$ holds because by the definition of r, there is a set T_i with k points whose pairwise distance is at least r. □

Using similar arguments, we can obtain a factor-4 composable coreset for the remote t-trees problem, and a factor-5 composable coreset for the remote t-cycles problem. Details are omitted in this version.

4 Lower Bound

In this section, we prove a general lower bound of 3 on the approximation factor of composable coresets for various notions of diversity in a metric space. This implies

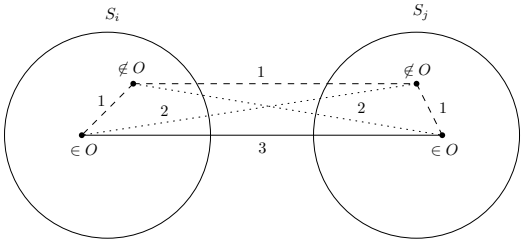


Figure 2: A lower bound example

the optimality of the composable coreset presented in Section 3.3 for the remote-cycle problem. This also settles an open problem posed by Indyk *et al.* [19] on the existence of better composable coresets for the remote-edge problem.

Theorem 8 *Let (X, d) be a metric space, and Π be a graph structure defined over induced subsets of X, such that all graphs with Π structure on a k-point set have the same number of edges. Then, the remote-Π problem admits no α-composable coreset, for any $\alpha < 3$.*

Proof. Consider k sets $S_i \subseteq X$, where each set has at least $k+1$ points. Suppose that the optimal solution O has exactly one point from each set S_i. Let the edges inside each S_i, as well as the edges between non-optimal points from different S_i's have weight 1, the edges connecting points in O have weight 3, and the remaining edges have weight 2. (See Figure 2.) It is easy to verify that this weight function is metric.

Let c be any function that computes a composable coreset $T_i = c(S_i)$ for the remote-Π problem. Due to edge weight symmetry inside each S_i, we can assume that T_i is a k-subset of $S_i \setminus O$. Therefore, the resulting set $T = \cup_i T_i$ will be a subset of $S \setminus O$, and hence, includes only edges of weight 1. Since all edges between the vertices of O have weight 3, the k-diversity of O will be 3 times the k-diversity of T with respect to Π. □

References

[1] S. Abbar, S. Amer-Yahia, P. Indyk, S. Mahabadi, and K. R. Varadarajan. Diverse near neighbor problem. In *Proc. 29th Annu. ACM Sympos. Comput. Geom.*, pages 207–214, 2013.

[2] Z. Abbassi, V. S. Mirrokni, and M. Thakur. Diversity maximization under matroid constraints. In *Proc. 19th ACM SIGKDD Internat. Conf. Knowledge Discovery and Data Mining*, pages 32–40, 2013.

[3] P. K. Agarwal, G. Cormode, Z. Huang, J. M. Phillips, Z. Wei, and K. Yi. Mergeable summaries. *ACM Transactions on Database Systems*, 38(4):26, 2013.

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

[5] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *Proc. 2nd ACM Internat. Conf. Web Search and Data Mining*, pages 5–14, 2009.

[6] A. Angel and N. Koudas. Efficient diversity-aware search. In *Proc. 2011 ACM SIGMOD Internat. Conf. Management of Data*, pages 781–792, 2011.

[7] C. Baur and S. P. Fekete. Approximation of geometric dispersion problems. *Algorithmica*, 30(3):451–470, 2001.

[8] B. Birnbaum and K. J. Goldman. An improved analysis for a greedy remote-clique algorithm using factor-revealing lps. *Algorithmica*, 55(1):42–59, 2009.

[9] T. M. Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Comput. Geom. Theory Appl.*, 35(1):20–35, 2006.

[10] B. Chandra and M. M. Halldórsson. Approximation algorithms for dispersion problems. *J. Algorithms*, 38(2):438–465, 2001.

[11] A. Czygrinow. Maximum dispersion problem in dense graphs. *Oper. Res. Lett.*, 27(5):223–227, 2000.

[12] U. Feige, D. Peleg, and G. Kortsarz. The dense k-subgraph problem. *Algorithmica*, 29(3):410–421, 2001.

[13] S. P. Fekete and H. Meijer. Maximum dispersion and geometric maximum weight cliques. *Algorithmica*, 38(3):501–511, 2004.

[14] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *Proc. 18th Internat. Conf. World Wide Web*, pages 381–390, 2009.

[15] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoret. Comput. Sci.*, 38:293–306, 1985.

[16] M. M. Halldórsson, K. Iwano, N. Katoh, and T. Tokuyama. Finding subsets maximizing minimum structures. *SIAM Journal on Discrete Mathematics*, 12(3):342–359, 1999.

[17] S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *Proc. 36th Annu. ACM Sympos. Theory Comput.*, pages 291–300, 2004.

[18] R. Hassin, S. Rubinstein, and A. Tamir. Approximation algorithms for maximum dispersion. *Oper. Res. Lett.*, 21(3):133–137, 1997.

[19] P. Indyk, S. Mahabadi, M. Mahdian, and V. S. Mirrokni. Composable core-sets for diversity and coverage maximization. In *Proc. 33rd ACM Sympos. Principles of Database Systems*, pages 100–108, 2014.

[20] M. J. Kuby. Programming models for facility dispersion: The ρ-dispersion and maxisum dispersion problems. *Geographical Analysis*, 19(4):315–329, 1987.

[21] I. D. Moon and S. S. Chaudhry. An analysis of network location problems with distance constraints. *Management Science*, 30(3):290–307, 1984.

[22] S. S. Ravi, D. J. Rosenkrantz, and G. K. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994.

[23] V. V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[24] H. Zarrabi-Zadeh. Core-preserving algorithms. In *Proc. 20th Canad. Conf. Computat. Geom.*, pages 159–162, 2008.

[25] H. Zarrabi-Zadeh. An almost space-optimal streaming algorithm for coresets in fixed dimensions. *Algorithmica*, 60(1):46–59, 2011.

An Upper Bound on Trilaterating Simple Polygons

Matthew Dippel* Ravi Sundaram †

Abstract

In this work, we introduce the Minimum Trilateration Problem, the problem of placing distance measuring guards in a polygon in order to locate points in the interior. We provide the first non-trivial bounds on trilaterating simple polygons, by showing that $\lfloor \frac{8N}{9} \rfloor$ guards suffice for any non-degenerate polygon of N sides, and present an $O(N \log N)$ algorithm for the corresponding placement. We also show how this mapping can be efficiently inverted, in order to determine a point's location given its distances to the guards which can see it.

1 Introduction

Trilateration is the technique of determining absolute locations of points using distances and the geometry of circles and spheres. For example, in 2-D if the two distances of a point from two fixed centers are known then there are only two possible candidate locations for the point. If three distances of a point from three fixed, non collinear points are known, there is only one possible location for the unknown point.

Formally, suppose we have a set of known points $\{r_i\}$, and their corresponding distances $\{d_i\}$ to an unknown point p. Then, to trilaterate p, we must solve the system:

$$\|r_i - p\| = d_i \tag{1}$$

for all possible solutions p. If there exists a unique solution, then we say that the set $\{r_i\}$ trilaterates p.

Observation 1 *If r_1, r_2, r_3 are non collinear points, and the d_i's are valid distances, then the system in Equation 1 is always solvable for a unique p.*

We define distance measuring guards as points which can measure the distance to other points in their visibility region. Given a polygon P, we wish to find a set of distance measuring guards R such that for every point p, when we consider only those $\{r_i\}$ which can view p, the system in Equation 1 has a unique solution in P. We will formalize this problem in Section 2.

We call the problem of finding the smallest such guard set for a polygon as the **Minimum Trilateration Problem**. Our results show upper bounds for this problem analogous to the upper bounds for the Art Gallery Problem [12]. As the Art Gallery Problem is analogous to dominating sets in visibility graphs, our problem also has an analogous graph theory problem known as the metric dimension of the graph. [8]. There has also been work on guarding polygons where each point must be viewed by multiple guards [3], but although they mention trilateration as an application, k-guarding a polygon is not sufficient for unambiguous trilateration.

First, we show that when constrained to certain properties, partitioning a polygon and finding a trilaterating guard set for each individual piece can result in a valid trilateration of the original polygon. We demonstrate that every polygon admits such a partition, and show how to efficiently map these partitions into trilaterating guard sets of size no more than $8N/9$ when the polygon has N sides.

We also show how these algorithms and bounds can be extended to the case where the polygon is not in general position. This is an important case because of the role that collinearity plays in our problem. We show that even in this case, collinear guards can locate points by taking advantage of the visibility geometry of the polygon.

2 Simple Trilateration

2.1 Definitions

Let P be a simple polygon, which may or may not be in general position (both cases will be addressed in this paper). For any two points $a, b \in P$, we say that a and b are mutually visible if $\overline{ab} \subset P$. For a specific point p, we define the visibility region $V(p)$, as the set of all points visible from p. The kernel of P, $K(P)$ is the set of points $k \in P$ such that $V(k) = P$. A polygon is star-shaped if $K(P)$ is non-empty.

For a specific point r, we can define a "vision-masked distance" function, $d_r : P \to \mathbb{R}$, such that $d_r(p) = \|p - r\|$ if r can view p, and -1 otherwise. For

*College of Computer Science, Northeastern University, mdippel@ccs.neu.edu

†College of Computer Science, Northeastern University, koods@ccs.neu.edu

a group of k points R, we similarly define $D_R : P \to \mathbb{R}^k$ as the vector of vision-masked distances from p to the various $r \in R$.

Our goal is to find a guard set such that, for all pairs of points $p, q \in P$, we have $D_R(p) \neq D_R(q)$. We formalize this with the following definition:

Definition 1 *For a simple polygon P, and a finite set of points $R \subset P$, we say that R trilaterates P without ambiguity if the vision-masked distance vector function $D_R(p)$ is injective over the domain P.*

2.2 Simple cases

There are several simple cases to consider. The most obvious is when P is the unbounded plane, and R is a set of three, non-collinear guards. Then, by solving a system of equations describing the intersection of three circles, we can uniquely locate any point p. If P is a star shaped polygon with at least 3 non-collinear points in the kernel, then we can trilaterate P with these three points.

When the kernel of P includes two points on the same edge, we can trilaterate P with only those two points. To see why, consider when R is two such points a and b, and we attempt to solve System 1. Solving the system of equations yields two points, p and p', reflected across $\overline{ab}$. WLOG p is on the same side of $\overline{ab}$ as the polygon P. Then, since a and b are in the kernel of P and are on an edge of it, this edge alone blocks their view of the other half of the plane induced by $\overline{ab}$. Thus we know that p' is not in the domain polygon P, and return p as the correct point. We give a generalized observation:

Observation 2 *Let R be a guard set in P, and consider two $a, b \in R$. If $V(a) \cap V(b)$ is entirely on one side of the closed half plane induced by $\overline{ab}$, then a and b can trilaterate points in the set $V(a) \cap V(b)$.*

The logic is the same as above. Solving System 1 with the distances to a and b yields two points, in opposite half planes induced by $\overline{ab}$. Thus we can rule out one of them from our domain, based on which one is on the same side of $\overline{ab}$ as the region $V(a) \cap V(b)$.

2.3 Partition and cover

One possible approach for finding a trilaterating set is to partition P into polygons which are simple to trilaterate (star-shaped or otherwise), individually trilaterate each one, and take the union of all guard sets. However, this doesn't always work. Consider the simple case where we wish to trilaterate a square, and we split it into two triangles via a diagonal. If we trilaterate each piece with two guards on the shared diagonal, the square cannot be trilaterated, as all guards are collinear. If we partition the polygon and use two guards on an edge to cover a piece, we need to be sure that the piece sharing that edge does not also put guards on it. We address this issue with the following lemma:

Lemma 1 *Let P be a simple polygon, partitioned into $P_1, P_2, ..., P_k$. Let $R_i \subset K(P_i)$ be sets of disjoint guards that trilaterate each respective polygon in the partition. Then, if $R_i \cup R_j$ contains three non-collinear guards for all i, j, R trilaterates P.*

Proof. We show that given any $p \in P$, we can derive p from the given distance vector $D_R(p)$. We will split it into parts $D_{R_i}(p)$, projected onto the respective R_i components. Note that since $R_i \subset K(P_i)$, all points in R_i can see all points in P_i. Thus if an entry of $D_{R_i}(p)$ is -1, we can conclude that $p \notin P_i$. If this is the case, we will say that P_i is an impossible location for p, else we consider it plausible. Observe that there must be at least one plausible P_i, since we have that p is in some P_i, we just do not know which yet.

Suppose there is only one plausible P_i. Then, the same process used to trilaterate P_i with R_i can be reused.

Suppose instead that there are at least two plausible P_i, P_j. Then all guards in $R_i \cup R_j$ can see p and yield distance values. Since $R_i \cup R_j$ contains three non-collinear guards, we can solve for the location of p. □

We will show how to find such a partition for a general polygon P which satisfies Lemma 1. In particular, we will partition using only diagonals of P. For each piece and corresponding guard placement (P_i, R_i), we will have either R_i being three non-collinear guards, or two guards on an edge of P_i, which is a diagonal or edge of P. If we make sure not to reuse the same diagonal for different placements, this will limit our placement to distinct diagonals. Our first argument will assume that P is in general position, meaning the above placement satisfies Lemma 1. Our second argument will address the case when diagonals of P may be collinear, and show that, as long as we are still using distinct diagonals for placement, an application of Observation 2 will let us locate all points in P.

Observation 3 *If P is in general position, and a, b and c, d are on distinct diagonals, then a, b, c, d are not collinear.*

3 Upper bound for general position polygons

Before we state our main theorem, we first show a trivial upper bound and lower bound. Suppose we put a distance measuring guard on each vertex of a polygon with N vertices. If we consider its triangulation, then

we can see that any point in P can always view three non-collinear vertices of P. Hence it can see three non-collinear guards, and can be trilaterated. Thus N guards always suffices for any polygon.

To show a lower bound, we can reuse the comb polygon lower bound for the art gallery problem. No point in the comb can view into two different comb spikes. Thus we need at least two guards per spike to trilaterate all of the points in its interior. Thus $2N/3$ is a lower bound for minimum trilateration.

We now present the main theorem of this paper, which is giving an upper bound better than N guards:

Theorem 2 *Any simple polygon P with N vertices can be trilaterated by a guard set of size no more than $8N/9$.*

To prove Theorem 2, we give both a method of constructing a guard set, and a method for inverting distance vectors back to points. Our method has several steps. First, we use a generic fan partition to divide P into $N/3$ pieces, each piece being star shaped, and some having several prospective edges (which are either diagonals or edges of P) on which we could place guards. Second, we use a bipartite graph connecting pieces of the partition to possible diagonals / edges for placement of guards, preventing separate pieces from using the same diagonal for placement. After finding a maximum matching in this graph, we return, along with each piece of the partition, the set of guards which trilaterate that piece. The pseudocode for this algorithm is presented later on as Algorithm 1. For inverting these distances, we still use the procedure presented in Lemma 1, the pseudocode for which is presented as Algorithm 2.

The partition that we use is Chvátal's fan partition from the original proof of the art gallery theorem [5]. Although it is classic, it has largely been overshadowed by the Fisk proof using coloration [7]. Thus we will restate it here:

Definition 2 *A fan is a polygon P with at least one vertex u, such that for all other vertexes v not adjacent to u, $\overline{uv}$ is a proper diagonal of P. We call u the center of the fan.*

Theorem 3 *Every N-triangulation can be partitioned into m fans where $m \leq \lfloor n/3 \rfloor$. Furthermore, this can be done in $O(N \log N)$ time.*

This theorem is useful as it allows us to use any triangulation method we wish in order to find a fan partition. Thus our algorithm is agnostic to the method used to find the triangulation. Our run time is essentially dominated by the $O(N \log N)$ time needed to convert the triangulation into a fan partition. Thus, we can use the standard $O(N \log N)$ triangulation method. See Figure 1 for an example of a polygon partitioned into fans.

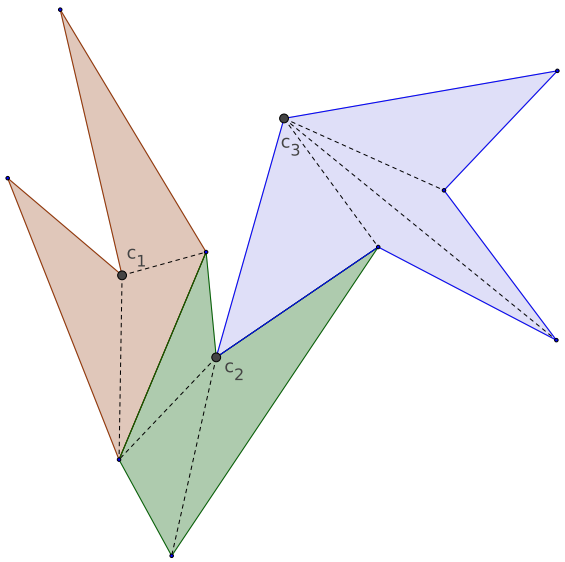


Figure 1: A polygon partitioned into three fans, with centers labeled c_1, c_2, c_3.

3.1 Trilaterating fans

We show several important structural lemmas. Mainly, we show that every fan with 4 or more triangles can be trilaterated by 3 guards, while every fan with fewer than 4 triangles can be trilaterated by 2 guards on an edge.

We refer to a fan with k triangles in its triangulation as a k-fan.

Definition 3 *For an edge e of a polygon P, we say that e is a prospective edge if there are two points $a, b \in e$ in the kernel of P. As such, any polygon with a prospective edge can be trilaterated with 2 guards.*

Lemma 4 *Every fan of k edges can be trilaterated with 3 guards, which can be found in $O(k)$ time.*

Lemma 5 *Every 1-fan has 3 prospective edges, every 2-fan has at least 2 prospective edges, and every 3-fan has at least 1 prospective edge. Further, these edges can be found in $O(1)$ time.*

We defer the proof of these structural lemmas to the appendix.

3.2 Finding a diagonal disjoint placement

We have shown that every fan can be trilaterated, either with three guards in its kernel, or two guards on

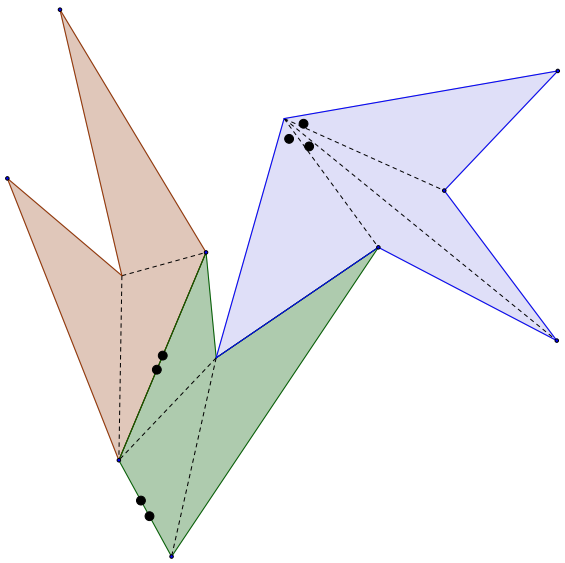


Figure 2: The fan partition from Figure 1 with guards covering each fan.

its boundary. Such a boundary placement corresponds to a placement on a diagonal or edge of P. In order to maintain the requirement of Lemma 1, we must find a placement such that no two fans put their guards on the same diagonal.

To do this, our algorithm uses the following steps:

- Partition P into a set F of no more than $N/3$ fans.
- For each fan f, associate with it all diagonals or edges which intersect $K(f)$ in at least two points. We do not bother attempting this step if f has 4 or more triangles.
- Given each fans' prospective edges, find a max matching between fans and edges.
- For each fan, report an appropriate pair of guards if it was matched to an edge, or triple of guards inside its kernel otherwise.

To do this, we create a bipartite graph $G = ((F \cup D), E)$, where F is the set of fans, and D is the set of diagonals and edges of the partition. We add the edge (f, d) if d is a viable boundary edge for fan f. We note a few properties of the graph. First, all 3-fans have degree at least 1, 2-fans at least 2, 1-fans exactly 3, and 4+-fans exactly 0. Second, if $d \in D$, then $deg(d) \leq 2$. Lastly, Because our graph is defined by the partition, it is a tree. Note that in finding a matching on this graph and reporting approprite guards, we do not guarantee that a fan which could have been trilaterated with 2 guards will still only use 2 in our placement. However, this is necessary in order to satisfy Lemma 1. See Figure 2 for an example of a valid placement of guards in a fan partition which could result from this algorithm.

Consider the number of fans using i triangles, for $i = 1, 2, 3, 4, ...$ Call these sets F_i. As previously shown in Lemma 5, all fans in F_1 have at least 3 prospective edges, F_2 at least 2, and F_3 at least 1. Hence these nodes in the bipartite graph will have degrees at least 3, 2, and 1 respectively. Fans with 4 or more triangles we will simply cover with 3 guards and not pair with any diagonals. Any diagonal can border at most 2 distinct fans, so every diagonal node in the graph will have degree no more than 2.

Suppose that we had k fans, so that $|F| = k$, and we found a matching of size j. Then, for j of the fans, we could use 2 guards, and for the remaining $k - j$ fans we would use 3 guards. This makes our total guard usage $2j + 3(k - j) = 3k - j$. Since Theorem 3 implies $k \leq \lfloor \frac{N}{3} \rfloor$, we are using no more than $N - j$ guards. We thus minimize our guard count by maximizing j.

We now present the main lemma regarding our graph construction, which we present as a generalized result on bipartite graphs with certain degree constraints:

Lemma 6 *Let $G = (A \cup B, E)$ be a bipartite graph, satisfying that for all nodes $v \in B$, $deg(v) \leq 2$. Let c_0, c_1, c_{2+} be the sets of nodes in A with degrees 0, 1, and ≥ 2, respectively, as well as the sizes of these sets. Then a matching of size $\frac{c_1}{2} + c_{2+}$ is always possible.*

Proof. We prove the above by an application of the deficit version of Hall's Theorem [10]. Let $S \subset A$. Then the defect of S is defined as $df(S) = |S| - |N(S)|$, the size of S minus the number of unique neighbors of S in B. Then the maximum matching M satisfies $|M| = \min_{S \subset A}\{|A| - df(S)\}$.

We will upper bound $df(S)$, which gives us a lower bound on $|M|$. Consider an arbitrary such S. Let m be the number of edges leaving S, and let $d(k)$ be the number of nodes of degree k in S. Then $|N(S)| \geq m/2$, as each node in B has degree at most 2. Also note that $m = \sum kd(k)$, and $|A| = c_0 + c_1 + c_{2+}$.

$$\begin{aligned} |S| - |N(S)| &\leq |S| - m/2 \\ &= \sum d(k) - 1/2 \sum kd(k) \\ &= \sum (1 - k/2)d(k) \\ &\leq d(0) + d(1)/2 \\ &\leq c_0 + c_1/2 \end{aligned}$$

Hence $df(S) \leq c_0 + c_1/2$, and $|M| \geq A - (c_0 + c_1/2) = c_1/2 + c_{2+}$, as desired. □

We can now prove the main theorem of the paper, restated below:

Theorem 2 *Any simple polygon P with N vertices can be trilaterated by a guard set of size no more than $8N/9$.*

Proof. We can partition P into $k \leq N/3$ fans, associate with each fan its plausible diagonal placements, and find a matching on the graph G as above. For any set $R_i \cup R_j$, either R_i contains three non collinear guards, or R_i and R_j are on distinct, non-collinear diagonals. Thus the placement satisfies Lemma 1 and trilaterates P.

The matching of G will have size at least $c_1/2 + c_{2+} = F_3/2 + F_2 + F_1$. Thus, the number of guards we will use is no more than $3k - F_3/2 - F_2 - F_1$. Thus we can bound the number of guards returned by analyzing the linear program:

$$\begin{aligned} \max \quad & 3k - F_3/2 - F_2 - F_1 \\ \text{s.t.} \quad & F_1 + F_2 + ... = k \\ & k \leq \lfloor N/3 \rfloor \\ & F_1 + 2F_2 + 3F_3 + ... = N - 2 \\ & F_i \geq 0 \end{aligned}$$

It can be shown via standard dual arguments that for all N the objective function is bounded above by $8N/9$. See the appendix for a derivation of this bound. Thus we can always achieve less than $8N/9$ guards. □

We now present the algorithm pseudocode for placement and for location. Although the lemma for locating points provides an algorithm, we will explicitly give it as pseudocode, in order to generalize our results to arbitrary simple polygons in the next section.

Lemma 7 *Algorithm 1 runs in $O(N \log N)$ time.*

We defer the proof to the appendix, as it is a routine examination of the algorithm step by step.

4 Extending the upper bound to polygons that are not in general position

Consider now a polygon which may not be in general position. The argument that guards on distinct diagonals can trilaterate points is no longer valid, as distinct diagonals may be collinear. We show how to augment the the location step with an additional method, in order to still locate points, even if the only guards that can see them are all collinear. To do this, we use the following lemma:

Algorithm 1: guard Locations Algorithm

```
Input  : Polygon P
Output: Partition of P with corresponding guard
        placements
T = Any triangulation of P
F = FanPartition(T)
D = Edges of F
f = Faces of F
G = (f ∪ D, {})
for f_i ∈ f do
    d_i ← viable diagonal edges for f_i
    Add edge (f_i, d) to G for all d ∈ d_i
end
M = MaxMatching(G)
for (f_i, d_i) ∈ M do
    r_1, r_2 ← two points ∈ d_i ∩ K(f_i)
    Yield (f_i, {r_1, r_2})
end
for f_i ∉ M do
    r_1, r_2, r_3 ← three non-collinear points ∈ K(f_i)
    Yield (f_i, {r_1, r_2, r_3})
end
```

Algorithm 2: Distance Vector Reversing

```
Input  : Polygon P, Partition {P_i, R_i}, Distance
         Vector D
Output: Unique point p that generates D
for {P_i, R_i} in partition do
    D_i ← D projected onto R_i
    if D_i has a −1 entry then
        Disregard {P_i, R_i}
    end
end
if At least two (P_i, R_i), (P_j, R_j) remain then
    locatePoint(R_i ∪ R_j, D_i ∪ D_j)
end
else
    locateWithinP(P_i, R_i)
end
```

Lemma 8 *Let $r_1 \neq r_2$ be points in P such that the line segment $e = \overline{r_1 r_2}$ intersects $\delta(P)$, the boundary of P. Then, the intersection of visibility regions $V(r_1) \cap V(r_2)$ is entirely in one closed half plane induced by $\overline{r_1 r_2}$.*

We defer the proof of the above to the appendix. The application of this lemma is that, if r_1 and r_2 are on distinct diagonals, at least one point directly between them is on the border of P. Thus if $p \in V(r_1) \cap V(r_2)$, we can narrow down its location to a specific half plane defined by r_1, r_2.

Suppose we had a method which took queries of pairs of collinear diagonals d_l, d_r, and returned which of their sides cannot have common visibility to both. Then, up-

dating our location method involves only updating the method **locatePoint**. First check if the input guards are collinear. If they are not, solve the system as before. If they are collinear, then since they are on distinct diagonals, we determine on which side of these diagonals p cannot be in. Once we know this side, we can solve the system for two potential points, and return the point that is on the correct side.

Our proposed method will require us to maintain the triangulation structure from the placement part of the algorithm, and perform path finding in the dual graph. Let d_1, d_2 be collinear diagonals in a triangulation T, and consider how the graph dual of T is partitioned by the edge corresponding to d_1. Then the piece of the partition which has d_2 contains the piece of the polygon which can have common visibility to d_1 and d_2. We can make a similar claim on d_2. To determine which side of d_1 this piece is on, use the triangulation structure to determine the triangle which uses d_1 and is in the same piece of the partition as d_2. Use a clockwise test to return whether the third point of this triangle is to the right or the left of d_1. The piece of P with common visibility must also be to the right or to the left of d_1. See Algorithm 3 for the pseudocode for this method.

Algorithm 3: General Position locatePoint

Input : Polygon Triangulation T, Guard positions R, Distance Vector d
Output: Unique point p that generates d
P = All solutions to system $\|r_i - d_i\| = p$
if $|P| = 1$ **then**
  Return the unique $p \in P$
end
else
  $P = \{p_1, p_2\}$
  Take $r_i, r_j \in R$ on distinct diagonals D_i, D_j
  Let t be the first triangle in the unique path between D_i and D_j in T
  Let x be the vertex of T not on D_i
  if $ccw(r_i, r_j, x) == ccw(r_i, r_j, p_1)$ **then**
    return p_1
  end
  else
    return p_2
  end
end

5 Conclusion

We introduced the minimum trilateration problem, and showed that it has an upper bound of $8N/9$ guards. We gave an $O(N \log N)$ algorithm for achieving this bound, as well as an algorithm for using the given placement to invert distance vectors to locate points in the polygon.

Having introduced the trilateration problem and derived an upper bound similar to that for the art gallery problem, there are several questions left unanswered. The ones we are most interested in are finding a tight upper bound, giving an algorithm to verify proposed guard sets, and showing improved bounds for the usual variants of art gallery, such as orthogonal polygons, guards which can move along edges, or guards which can see through k walls.

References

[1] D. Avis and G. T. Toussaint. An optimal algorithm for determining the visibility of a polygon from an edge. *IEEE Trans. Comput.*, 30(12):910–914, Dec. 1981.

[2] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, Santa Clara, CA, USA, 3rd ed. edition, 2008.

[3] Daniel Busto, William S. Evans, and David G. Kirkpatrick. On k-guarding polygons. In *Proceedings of the 25th Canadian Conference on Computational Geometry, CCCG 2013, Waterloo, Ontario, Canada, August 8-10, 2013*, 2013.

[4] J. Cáceres, M. C. Hernando, M. Mora, I. M. Pelayo, M. L. Puertas, C. Seara, and D. R. Wood. On the metric dimension of cartesian products of graphs. *SIAM J. Discrete Math.*, 21(2):423–441, 2007.

[5] V. Chvátal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory Series B*, 18:39–41, 1975.

[6] K. Curran, E. Furey, T. Lunney, J. Santos, D. Woods, and A. McCaughey. An evaluation of indoor location determination technologies. *J. Location Based Services*, 5(2):61–78, 2011.

[7] S. Fisk. A short proof of Chvátal's watchman theorem. *J. Comb. Theory, Ser. B*, 24(3):374, 1978.
W. H. Freeman & Co., New York, NY, USA, 1979.

[8] W. Goddard and O. R. Oellermann. Distance in graphs. In M. Dehmer, editor, *Structural Analysis of Complex Networks*, pages 49–72. Birkhuser Boston, 2011.

[9] D. T. Lee and F. P. Preparata. An optimal algorithm for finding the kernel of a polygon. *J. ACM*, 26(3):415–421, July 1979.

[10] Oystein Ore. Graphs and matching theorems. *Duke Math. J.*, 22(4):625–639, 12 1955.

[11] J. O'Rourke. *Art Gallery Theorems and Algorithms*. The International Series of Monographs on Computer Science. Oxford University Press, New York, NY, 1987.

[12] J. Urrutia, U. Nacional, and A. Mxico. Art gallery and illumination problems. In *In Handbook on Computational Geometry, Elsevier Science Publishers, J.R. Sack and*, page 1026, 2000.

Appendix

We provide the omitted proofs for several lemmas.

Lemma 4 *Every fan of k edges can be trilaterated with 3 guards, which can be found in $O(k)$ time.*

Proof. First, note that if a vertex can view all other vertices, it can view all other points in the polygon [1]. This asserts that all fans have a non-null kernel containing the center vertex v. We will now examine the kernel as the intersection of k half-planes.

Each edge of the polygon defines a half plane, the intersection of which is the kernel of the polygon. The two edges adjacent to v will map to half planes which intersect v. Since these half planes cannot be parallel or anti-parallel (defining opposite half-spaces of the plane), their intersection is a region which v is on the border of. Observe that every other half plane must contain v, since v is in the kernel, and that every other half plane cannot intersect v on its boundary. To see why, consider the half plane defined by the edge $\overline{ab}$. We have that both $\overline{av}$ and $\overline{bv}$ are non-intersecting valid diagonals of P. Since they are not intersecting, v is not collinear with $\overline{ab}$. Hence the edge of that half plane is at least some distance ϵ away from v.

Hence if we take the intersection of all these half planes, the result is a region around v with an infinite number of points. Taking any 3 of them as the guard set will suffice.

To find these guards, the linear time kernel algorithm can be used to determine the kernel [9], from which we return several random non-collinear points. If we wished for a slightly simpler algorithm, we do not need the explicit kernel, but just a subset of it. We calculate the distance of v from each half plane, and take the min of these as r. Then any point which is in the intersection of the $\overline{av}$ and $\overline{bv}$ half planes and within distance r of v will be in the kernel. It suffices to pick one and move it $\pm\epsilon$ to get three non-collinear points.

□

Lemma 5 *Every 1-fan has 3 prospective edges, every 2-fan has at least 2 prospective edges, and every 3-fan has at least 1 prospective edge. Further, these edges can be found in $O(1)$ time.*

Proof. The lemma is clearly true of a 1-fan since it is a triangle which is convex. Thus all edges are prospective edges.

A 2-fan is a quadrilateral. Let v be the center of the fan, a and b the vertices adjacent to v, and c the last vertex. Note that the angles at a and b must be convex. Consider the visibility regions $V(a)$ and $V(b)$. First note they must intersect an edge not adjacent to a and b respectively. Hence $V(a)$ contains $\overline{ac}$ and intersects part of $\overline{bc}$, while $V(b)$ contains $\overline{bc}$ and intersects part of $\overline{ac}$. Thus some sections of $\overline{bc}$ and $\overline{ac}$ are in the kernel, making them prospective edges.

A 3-fan is a pentagon. Consider the vertices of the fan labeled in CW order a, v, b, c, d, so that a and b are adjacent to v. Note that the angles at a and b must be convex. If the angle at v is convex, then following Lemma 4, the kernel intersects both $\overline{va}$ and $\overline{vb}$. Else, we consider the case where v is a reflex angle. Since a pentagon can have at most two reflex angles, at least one of c or d is a convex angle. WLOG assume c is convex.

Suppose that d is a reflex angle. Then, we have that we can extend both $\overline{ad}$ and $\overline{av}$ until they hit the boundary of the fan. They must both end at $\overline{bc}$, creating a subsegment which can see every vertex. Thus $\overline{bc}$ is a prospective edge.

Suppose instead that d is a convex angle. Then instead consider extending $\overline{av}$ and $\overline{bv}$ until they hit the boundary of the fan. If $\overline{av}$ hits $\overline{bc}$, then because d is convex, c can see a. Thus c could also be the vertex center, and we can reduce to the case where it is a fan with convex angle at the center. A similar argument applies to if $\overline{bv}$ hits $\overline{ad}$. Thus, we must have that both $\overline{av}$ and $\overline{bv}$ extend to meet $\overline{cd}$. Then similar to the previous case, they create a subsegment which can see every vertex. Thus $\overline{cd}$ is a prospective edge. Thus in all cases, a pentagon has a prospective edge.

To find these edges, we can explicitly compute the kernel, and determine which edge of the polygon it intersects with in a continuous region. Since our fan size is no more than 5 edges, this takes $O(1)$ time.

□

Lemma 1 *Algorithm 1 runs in $O(N \log N)$ time.*

Proof. Triangulating the polygon can be done in $O(N \log N)$ time. 3-coloring this triangulation and determining the resulting fan partition can be done in $O(N \log N)$ time, by computing the DCEL representation of the partition [2], and considering the bounded faces to be the fans.

For each fan of k edges, either determining the prospective edges or finding three points in the kernel takes $O(k)$ time. Thus summing over all fans in the partition, the total time from these operations is $O(N)$.

To find the maximum matching in our bipartite graph, it is known that a maximum matching on trees can be found in $O(N)$ time by using dynamic programming. Since our graph was induced by the edges of the polygon and diagonals of the fan partition, it is a tree.

Thus the dominating factor in the run time is using the diagonals of the triangulation to retrieve the DCEL representation of the partition, which takes $O(N \log N)$ time.

□

Theorem 2 *Any simple polygon P with N vertices can be trilaterated by a guard set of size no more than $8N/9$.*

Proof. We derive an upper bound for the linear program, which we restate here:

$$\begin{aligned} \max \quad & 3k - F_3/2 - F_2 - F_1 \\ \text{s.t.} \quad & F_1 + F_2 + ... = k \\ & k \leq \lfloor N/3 \rfloor \\ F_1 + 2F_2 + 3F_3 + ... &= N - 2 \\ & F_i \geq 0 \end{aligned}$$

First note that our objective function is equivalent to $2F_1 + 2F_2 + 2.5F_3 + 3F_4 + 3F_5 + ...$ We show that a linear combination of the constraints bounds our objective function.

Take 5/3 of the first inequality, and add it to 1/3 of the second equality, yielding:

$$\begin{aligned} (5/3)(F_1 + F_2 + F_3 + ...) &\leq (5/3)\lfloor N/3 \rfloor \leq 5N/9 \\ (1/3)(F_1 + 2F_2 + 3F_3 + ...) &= (N-2)/3 \\ \hline 2F_1 + (7/3)F_2 + (8/3)F_3 + 3F_4 + ... &\leq (8N-6)9 \end{aligned}$$

Where the LHS of the final inequality is greater than our objective function. Thus our objective function is bounded above by $8N/9$. □

Lemma 2 *Let $r_1 \neq r_2$ be points in P such that the line segment $e = \overline{r_1 r_2}$ intersects $\delta(P)$, the boundary of P. Then, the intersection of visibility regions $V(r_1) \cap V(r_2)$ is entirely in one closed half plane induced by $\overline{r_1 r_2}$.*

Proof. We will consider a coordinate frame of axis where r_1 and r_2 are both on the x-axis and have opposite signs for their x coordinate. First, we consider the case where either r_1 or r_2 is on an edge e of P. In our coordinate frame, e is coincident with the x-axis. In this case, it should be clear that, depending on which side of e P is on, WLOG r_1 can only see above or below the x axis, but not both. Hence the intersection of visibility regions must be above or below the x-axis, as desired.

Let $X \neq r_1, r_2$ be a point in $\delta(P)$ which is on e. Take a point with arbitrarily small x coordinate $Z = (-\infty, 0)$, and consider any closed curve path from Z to X which does not intersect P before touching X. Let Y be the first point on e which P meets, which may end up being Z. Then, we now consider the close curve path P', which is P from Z to Y. Consider the side of e that P'approaches Y from. Since there is a path from Z to Y that lies completely outside the polygon, $V(r_1)$ and $V(r_2)$ cannot intersect on that side of e, as this would mean that they also intersect P', which lays entirely outside the polygon.

□

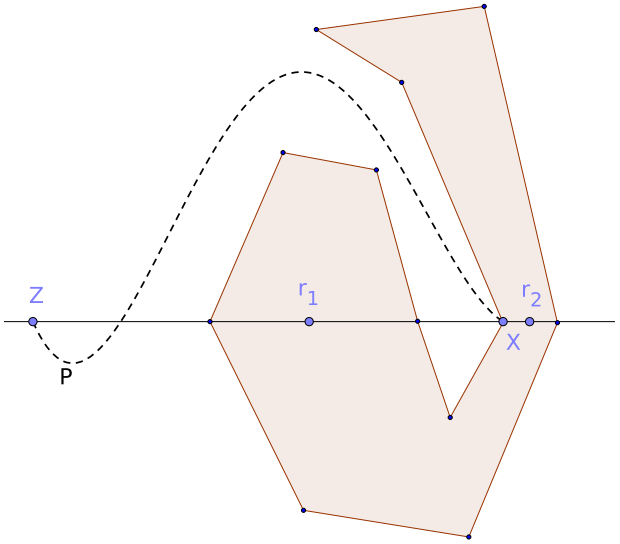


Figure 3: An illustration of Lemma 2. The existence of the path P guarantees us that $V(r_1)$ and $V(r_2)$ cannot intersect above the x-axis.

Constrained Empty-Rectangle Delaunay Graphs*

Prosenjit Bose‡ Jean-Lou De Carufel‡ André van Renssen§¶

Abstract

Given an arbitrary convex shape C, a set P of points in the plane and a set S of line segments whose endpoints are in P, a constrained generalized Delaunay graph of P with respect to C denoted $CDG_C(P)$ is constructed by adding an edge between two points p and q if and only if there exists a homothet of C with p and q on its boundary and no point of P in the interior visible to both p and q. We study the case where the empty convex shape is an arbitrary rectangle and show that the constrained generalized Delaunay graph has spanning ratio at most $\sqrt{2}\cdot(2l/s+1)$, where l and s are the length of the long and short side of the rectangle.

1 Introduction

A geometric graph G is a graph whose vertices are points in the Euclidean plane and whose edges are line segments between pairs of points. Every edge is weighted by the Euclidean distance between its endpoints. A geometric graph G is called plane if no two edges intersect properly. The distance between two vertices u and v in G, denoted by $\delta_G(u,v)$, is defined as the sum of the weights of the edges along the shortest path between u and v in G. A subgraph H of G is a t-spanner of G (for $t \geq 1$) if for each pair of vertices u and v, $\delta_H(u,v) \leq t \cdot \delta_G(u,v)$. The smallest value t for which H is a t-spanner is the *spanning ratio* or *stretch factor* of H. The spanning properties of various geometric graphs have been studied extensively in the literature (see [5, 9] for an overview of the topic).

We study this problem in the presence of line segment *constraints*. Specifically, let P be a set of points in the plane and let S be a set of line segments with endpoints in P, with no two line segments intersecting properly. The line segments of S are called *constraints*. Two vertices u and v can *see each other* or *are visible to each other* if and only if either the line segment uv does not properly intersect any constraint or uv is itself a constraint. If two vertices u and v can see each other, the line segment uv is a *visibility edge*. The *visibility graph* of P with respect to a set of constraints S, denoted $Vis(P,S)$, has P as vertex set and all visibility edges as edge set. In other words, it is the complete graph on P minus all edges that properly intersect one or more constraints.

This setting has been studied extensively within the context of motion planning amid obstacles. Clarkson [7] was one of the first to study this problem and showed how to construct a linear-sized $(1+\epsilon)$-spanner of $Vis(P,S)$. Subsequently, Das [8] showed how to construct a spanner of $Vis(P,S)$ with constant spanning ratio and constant degree. Bose and Keil [4] showed that the Constrained Delaunay Triangulation is a $4\pi\sqrt{3}/9 \approx 2.419$-spanner of $Vis(P,S)$. The constrained Delaunay graph where the empty convex shape is an equilateral triangle was shown to be a 2-spanner [3]. Recently, it was shown that regardless of the empty convex shape C used, the constrained generalized Delaunay graph is a plane spanner with constant spanning ratio, where the spanning ratio depends on the perimeter and the width of C [2].

In this paper, we improve the spanning ratio for the case where the empty convex shape is a rectangle. In the unconstrained setting, Chew [6] showed that the spanning ratio for squares is at most $\sqrt{10} \approx 3.16$. This was later improved by Bonichon *et al.* [1], who showed a tight spanning ratio of $\sqrt{4+2\sqrt{2}} \approx 2.61$. We show that in the constrained setting the spanning ratio is at most $\sqrt{2}\cdot(2l/s+1)$, where l and s are the length of the long and short side of C. For squares (the rectangles that minimize l/s), this implies a ratio of $3\sqrt{2} \approx 4.25$.

2 Preliminaries

Throughout this paper, we fix a convex shape C. We assume without loss of generality that the origin lies in the interior of C. A *homothet* of C is obtained by scaling C with respect to the origin, followed by a translation. Thus, a homothet of C can be written as

$$x + \lambda C = \{x + \lambda z : z \in C\},$$

for some scaling factor $\lambda > 0$ and some point x in the interior of C after translation. We refer to x as the *center* of the homothet $x + \lambda C$.

For a given set of vertices P and a set of constraints S, we now define the constrained generalized Delaunay graph. Given any two visible vertices p and q, let $C(p,q)$ be any homothet of C with p and q on its boundary. The

*Research supported in part by FQRNT, NSERC, and Carleton University's President's 2010 Doctoral Fellowship.

‡School of Computer Science, Carleton University, Ottawa, Canada. jit@scs.carleton.ca, jdecaruf@cg.scs.carleton.ca

§National Institute of Informatics (NII), Tokyo, Japan. andre@nii.ac.jp

¶JST, ERATO, Kawarabayashi Large Graph Project.

constrained generalized Delaunay graph contains an edge between p and q if and only if there exists a $C(p,q)$ such that there are no vertices of P in the interior of $C(p,q)$ visible to both p and q. Note that this implies that constraints are *not* necessarily edges of the constrained generalized Delaunay graph. We assume that no four points lie on the boundary of any homothet of C.

2.1 Auxiliary Lemmas

Next, we present three auxiliary lemmas that are needed to prove our main results. First, we reformulate a lemma that appears in [10].

Lemma 1 *Let C be a closed convex curve in the plane. The intersection of two distinct homothets of C is the union of two sets, each of which is either a segment, a single point, or empty.*

We say that a region R *contains* a vertex v if v lies in the interior or on the boundary of R. We call a region *empty* if it does not contain any vertex of P. Though the following lemma was applied to constrained θ-graphs in [3], the property holds for any visibility graph.

Lemma 2 *Let u, v, and w be three arbitrary points in the plane such that uw and vw are visibility edges and w is not the endpoint of a constraint intersecting the interior of triangle uvw. Then there exists a convex chain of visibility edges from u to v in triangle uvw, such that the polygon defined by uw, wv and the convex chain is empty and does not contain any constraints.*

Finally, we re-introduce a definition and lemma from [2]. Let p and q be two vertices that can see each other and let $C(p,q)$ be a convex polygon with p and q on its boundary. We look at the constraints that have p as an endpoint and the edge(s) of $C(p,q)$ on which p lies, and extend them to half-lines that have p as an endpoint (see Figure 1a). Given the cyclic order of these half-lines around p and the line segment pq, we define the clockwise neighbor of pq to be the half-line that minimizes the strictly positive clockwise angle with pq. Analogously, we define the counterclockwise neighbor of pq to be the half-line that minimizes the strictly positive counterclockwise angle with pq. We define the *cone* C_q^p that contains q to be the region between the clockwise and counterclockwise neighbor of pq. Finally, let $C(p,q)_q^p$, the *region of $C(p,q)$ that contains q with respect to p*, be the intersection of $C(p,q)$ and C_q^p (see Figure 1b).

Lemma 3 *Let p and q be two vertices that can see each other and let $C(p,q)$ be any convex polygon with p and q on its boundary. If there is a vertex x in $C(p,q)_q^p$ (other than p and q) that is visible to p, then there is a vertex y (other than p and q) in $C(p,q)$ that is visible to both p and q and triangle pyq is empty.*

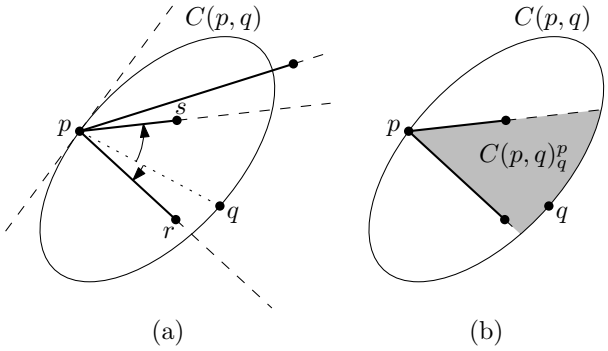


Figure 1: Defining the region of $C(p,q)$ that contains q with respect to p: (a) The clockwise and counterclockwise neighbor of pq are the half-lines through pr and ps, (b) $C(p,q)_q^p$ is marked in gray.

3 The Constrained Empty-Rectangle Delaunay Graph

We look at the case where the empty convex shape is an arbitrary rectangle. We assume without loss of generality that the rectangle is axis-aligned. We do not, however, assume anything about the ratio between the height and width of the rectangle. We first show that if two visible vertices cannot see any vertices in $C(p,q)$ on one side of pq, then no vertex in $C(p,q)$ on the opposite side of pq can see any vertices beyond pq either.

Lemma 4 *Let p and q be two vertices that can see each other, such that pq is not vertical, and let $C(p,q)$ be any convex polygon with p and q on its boundary. If the region of $C(p,q)$ below pq does not contain any vertices visible to p and q, then no point x in $C(p,q)$ above pq can see any vertices in $C(p,q)$ below pq.*

Proof. We prove the lemma by contradiction, so assume that there exists a vertex y in $C(p,q)$ below pq that is visible to x, but not to p and q. Since $C(p,q)$ is a convex polygon and x and y lie on opposite sides of pq, the visibility edge xy intersects pq. Let z be this intersection (see Figure 2).

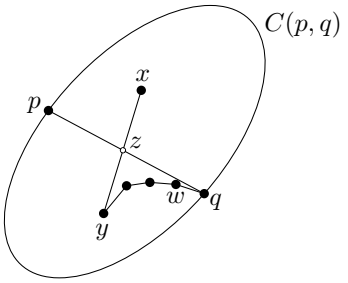


Figure 2: If x can see a vertex below pq, then so can q.

Hence, zy and zq are visibility edges. Since z is not a vertex, it is not the endpoint of any constraints in-

tersecting the interior of triangle yzq. It follows from Lemma 2 that there exists a convex chain of visibility edges between y and q and this chain is contained in yzq. However, this implies that w, the neighbor of q along this chain, is visible to q and lies in $C(p,q)$ below pq. Next, we apply Lemma 2 on triangle pqw and find that the neighbor of p along the chain from p to w is visible to both p and q and lies in $C(p,q)$ below pq, contradicting that this region does not contain any vertices visible to p and q. □

Next, we introduce some notation for the following lemma. Let p and q be two vertices of the constrained generalized Delaunay graph that can see each other. Let R be a rectangle with p and q on its West and East boundary and let a, b, and r be the Northwest, Northeast, and Southwest corner of R. Let $m_1, ..., m_{k-1}$ be any $k-1$ points on pq in the order they are visited when walking from p to q (see Figure 3). Let $m_0 = p$ and $m_k = q$. Consider the homothets S_i of R with m_i and m_{i+1} on their respective boundaries, for $0 \leq i < k$, such that $|pa|/|ra| = |m_i a_i|/|r_i a_i|$, where a_i, b_i, r_i are the Northwest, Northeast, and Southwest corner of S_i.

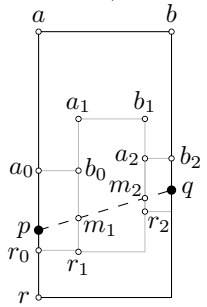


Figure 3: The total length of the sides of the rectangles S_i equals that of $C(p,q)$.

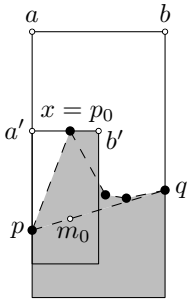


Figure 4: An inductive path from p to q.

Lemma 5 *We have*

$$\sum_{i=0}^{k-1} \big(|m_i a_i| + |a_i b_i| + |b_i m_{i+1}|\big) = |pa| + |ab| + |bq|.$$

Proof. Let $c = (|pa|+|ab|+|bq|)/|pq|$. Since for every S_i we have that $|pa|/|ra| = |m_i a_i|/|r_i a_i|$, we have $(|m_i a_i| + |a_i b_i| + |b_i m_{i+1}|)/|m_i m_{i+1}| = c$, for $0 \leq i < k$. Hence, we get

$$\begin{aligned}\sum_{i=0}^{k-1} \big(|m_i a_i| + |a_i b_i| + |b_i m_{i+1}|\big) &= \sum_{i=0}^{k-1} \big(c \cdot |m_i m_{i+1}|\big) \\ &= c \cdot |pq| \\ &= |pa| + |ab| + |bq|,\end{aligned}$$

proving the lemma. □

Before we prove the bound on the spanning ratio of the constrained generalized Delaunay graph, we first bound the length of the spanning path between vertices p and q for the case where the rectangle $C(p,q)$ is partially empty. We call a rectangle $C(p,q)$ *half-empty* when $C(p,q)$ contains no vertices in $C(p,q)^p_q$ below pq that are visible to p and $C(p,q)$ contains no vertices in $C(p,q)^q_p$ below pq that are visible to q. We denote the x- and y-coordinate of a point p by p_x and p_y.

Lemma 6 *Let p and q be two vertices that can see each other. Let $C(p,q)$ be a rectangle with p and q on its boundary, such that it is half-empty. Let a and b be the corners of $C(p,q)$ on the non-half-empty side. The constrained generalized Delaunay graph contains a path between p and q of length at most $|pa| + |ab| + |bq|$.*

Proof. We prove the lemma by induction on the rank of $C(x,y)$ when ordered by size, for any two visible vertices x and y, such that $C(x,y)$ is half-empty. We assume without loss of generality that p lies on the West boundary, q lies on the East boundary and that $C(p,q)$ is half-empty below pq. This implies that a and b are the Northwest and Northeast corner of $C(p,q)$. We also assume without loss of generality that the slope of pq is non-negative, i.e. $p_x < q_x$ and $p_y \leq q_y$ (see Figure 4).

We note that the case where p lies on the West boundary, q lies on the North boundary and $C(p,q)$ is half-empty below pq can be viewed as a special case of the one above: We shrink $C(p,q)$ until one of p and q lies in a corner. This point can now be viewed as being on both sides defining the corner and hence p and q are on opposite sides. An analogous statement holds for the case where p lies on the West boundary, q lies on the North boundary and $C(p,q)$ is half-empty above pq.

Let r be the Southwest corner of $C(p,q)$. Let R be a homothet of $C(p,q)$ that is contained in $C(p,q)$ and whose West boundary is intersected by pq. Let a', b', r' be the Northwest, Northeast, and Southwest corner of R and let m be the intersection of $a'r'$ and pq. We call homothet R *similar* to $C(p,q)$ if and only if $|pa|/|ra| = |ma'|/|r'a'|$.

Base case: If $C(p,q)$ is a rectangle of smallest area, then $C(p,q)$ does not contain any vertices visible to both p and q: Assume this is not the case and grow a rectangle R similar to $C(p,q)$ from p to q. Let x be the first vertex hit by R that is visible to p and lies in $C(p,q)^p_q$. Note that this implies that R is contained in $C(p,q)$. Therefore, R is smaller than $C(p,q)$. Furthermore, R is half-empty: By Lemma 4, the part below the line through p and q does not contain any vertices visible to p or x in $C(p,q)^p_q$, and the part between the line through p and x and the line through p and q does not contain any vertices visible to p or x since x is the first visible vertex hit while growing R. However, this contradicts that $C(p,q)$ is the smallest half-empty rectangle.

Hence, $C(p,q)$ does not contain any vertices visible to both p and q, which implies that pq is an edge of

the constrained generalized Delaunay graph. Therefore the length of the shortest path from p to q is at most $|pq| \leq |pa| + |ab| + |bq|$.

Induction step: We assume that for all half-empty rectangles $C(x, y)$ smaller than $C(p, q)$ the lemma holds. If pq is an edge of the constrained generalized Delaunay graph, the length of the shortest path from p to q is at most $|pq| \leq |pa| + |ab| + |bq|$.

If pq is not an edge of the constrained generalized Delaunay graph, there exists a vertex in $C(p, q)$ that is visible from both p and q. We grow a rectangle R similar to $C(p, q)$ from p to q. Let x be the first vertex hit by R that is visible to p and lies in $C(p, q)^p_q$ and let a' and b' be the Northwest and Northeast corner of R (see Figure 4). Note that this implies that R is contained in $C(p, q)$. We also note that px is not necessarily an edge in the constrained generalized Delaunay graph, since if it is a constraint, there can be vertices visible to both p and x above px. However, since R is half-empty and smaller than $C(p, q)$, we can apply induction on it and we obtain that the path from p to x has length at most $|pa'| + |a'b'| + |b'x|$ when x lies on the East boundary of R, and that the path from p to x has length at most $|pa'| + |a'x|$ when x lies on the North boundary of R.

Let m_0 be the projection of x along the vertical axis onto pq. Since m_0 is contained in R, x can see m_0. Since xm_0 and m_0q are visibility edges and m_0 is not the endpoint of a constraint intersecting the interior of triangle xm_0q, we can apply Lemma 2 and obtain a convex chain $x = p_0, p_1, ..., p_k = q$ of visibility edges (see Figure 4). For each of these visibility edges p_ip_{i+1}, there is a homothet R_i of $C(p, q)$ that falls in one of the following three types (see Figure 5): (i) p_i lies on the North boundary and p_{i+1} lies in the Southeast corner, (ii) p_i lies on the West boundary and p_{i+1} lies on the East boundary and the slope of p_ip_{i+1} is negative, (iii) p_i lies on the West boundary and p_{i+1} lies on the East boundary and the slope of p_ip_{i+1} is not negative. Let a_i and b_i be the Northwest and Northeast corner of R_i. We note that by convexity, these three types occur in the order Type (i), Type (ii), and Type (iii).

Let m_i be the projection of p_i along the vertical axis onto pq, let C_i be the homothet of $C(p, q)$ with m_i and m_{i+1} on its boundary that is similar to $C(p, q)$, and let a'_i and b'_i be the Northwest and Northeast corner of C_i. Using these C_i, we shift Type (ii) and Type (iii) rectangles down as far as possible: We shift R_i down until either p_i or p_{i+1} lies in one of the North corners or the South boundary corresponds to the South boundary of C_i. In the latter case, R_i and C_i are the same rectangle.

Since all rectangles R_i are smaller than $C(p, q)$, we can apply induction, provided that we can show that R_i is half-empty. For Type (i) visibility edges, the part of the rectangle that lies below the line through p_i and p_{i+1} is contained in R, which does not contain any visible vertices, and the region of $C(p, q)^p_q$ below the convex chain, which is empty. For Type (ii) and Type (iii) visibility edges, the part of the rectangle that lies below the line through p_i and p_{i+1} is contained in the region of $C(p, q)^p_q$ below the convex chain, which is empty, and the region of $C(p, q)$ below the line through p and q, which does not contain any visible vertices by Lemma 4. Hence, all R_i are half-empty and we obtain an inductive path of length at most: (i) $|p_ib_i| + |b_ip_{i+1}|$, (ii) $|p_ia_i| + |a_ib_i| + |b_ip_{i+1}|$, (iii) $|p_ia_i| + |a_ib_i| + |b_ip_{i+1}|$.

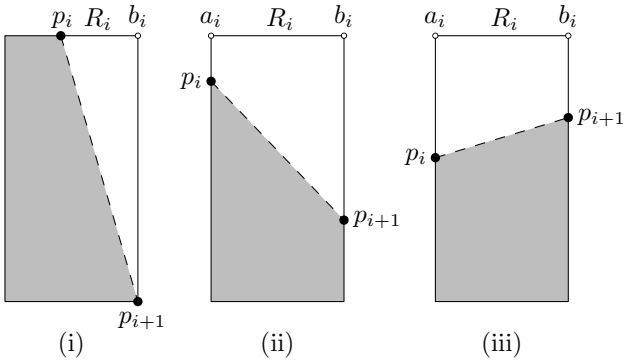


Figure 5: The three types of rectangles along the convex chain.

To bound the total path length, we perform case distinction on the location of x on R and whether the convex path from x to q goes down: (a) x lies on the East boundary of R and the convex path does not go down, (b) x lies on the East boundary of R and the convex path goes down, (c) x lies on the North boundary of R and the convex path does not go down, (d) x lies on the North boundary of R and the convex path goes down.

Case (a): The vertex x lies on the East boundary of R and the convex path does not go down. Recall that the length of the path from p to x is at most $|pa'| + |a'b'| + |b'x|$, which is at most $|pa'| + |a'b'| + |b'm_0|$. Since the convex chain does not go down, it cannot contain any Type (i) or Type (ii) visibility edges. Furthermore, since x lies on the East boundary of R, R and all C_i are disjoint. Thus, Lemma 5 implies that the boundaries above pq of R and all C_i sum up to $|pa| + |ab| + |bq|$. Hence, if we can show that, for all R_i, $|p_ia_i| + |a_ib_i| + |b_ip_{i+1}| \leq |m_ia'_i| + |a'_ib'_i| + |b'_im_{i+1}|$, the proof of this case is complete.

By convexity, the slope of p_ip_{i+1} is at most that of pq and m_im_{i+1}. Hence, when p_{i+1} lies in the Northeast corner of R_i, we have $p_{i+1} = b_i$ and $|p_ia_i| + |a_ip_{i+1}| \leq |m_ia'_i| + |a'_ib'_i| + |b'_im_{i+1}|$. If p_{i+1} does not lie in the Northeast corner, $R_i = C_i$. Hence, since p_i and p_{i+1} lie above pq, we have that $|p_ia_i| + |a_ib_i| + |b_ip_{i+1}| \leq |m_ia'_i| + |a'_ib'_i| + |b'_im_{i+1}|$.

Case (b): The vertex x lies on the East boundary of R and the convex path goes down. Recall that the length of the path from p to x is at most $|pa'|+|a'b'|+|b'x|$. Let

p_j be the lowest vertex along the convex chain. Since p_j lies above pq and pq has non-negative slope, the descent of the convex path is at most $|xm_0|$. Hence, when we charge this to R, we used $|pa'| + |a'b'| + |b'm_0|$ of its boundary (see Figure 6).

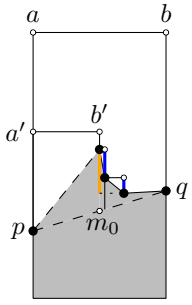


Figure 6: Going down along the convex chain (blue) is charged to R (orange).

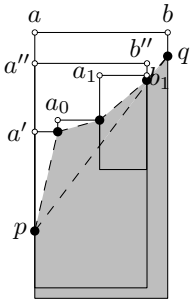


Figure 7: Charging the path from p to p_j to $C(p, p_j)$.

Like in the Case (a), since x lies on the East boundary of R, R and all C_i are disjoint. Thus, Lemma 5 implies that the boundaries above pq of R and all C_i sum up to $|pa|+|ab|+|bq|$. Hence, if we can show that, for all R_i, the inductive path length is at most $|m_ia'_i|+|a'_ib'_i|+|b'_im_{i+1}|$, the proof of this case is complete.

For Type (i) visibility edges, we have already charged $|b_ip_{i+1}|$ to R, so it remains to show that $|p_ib_i| \leq |m_ia'_i|+|a'_ib'_i|+|b'_im_{i+1}|$. This follows, since m_i and m_{i+1} are the vertical projections of p_i and p_{i+1}, which implies that $|p_ib_i| = |a'_ib'_i|$.

For Type (ii) visibility edges, we already charged $|b_ip_{i+1}| - |p_ia_i|$ to R, so we can consider p_ip_{i+1} to be horizontal and it remains to charge the remaining $2 \cdot |p_ia_i| + |a_ib_i|$. If p_i lies in the Northwest corner of R_i, it follows that $|p_ia_i| = 0$ and we have that $|p_ib_i| = |a'_ib'_i| \leq |m_ia'_i| + |a'_ib'_i| + |b'_im_{i+1}|$. If p_i does not lie in the Northwest corner, R_i is the same as C_i. Hence, since we can consider p_ip_{i+1} to be horizontal and p_i and p_{i+1} lie above pq, it follows that $2 \cdot |p_ia_i| + |a_ib_i| \leq |m_ia'_i| + |a'_ib'_i| + |b'_im_{i+1}|$.

Finally, Type (iii) visibility edges are charged as in Case (a), hence we have that $|p_ia_i| + |a_ib_i| + |b_ip_{i+1}| \leq |m_ia'_i| + |a'_ib'_i| + |b'_im_{i+1}|$, completing the proof of this case.

Case (c): Vertex x lies on the North boundary of R and the convex path does not go down. Recall that the length of the path from p to x is at most $|pa'| + |a'x|$. Since the convex chain does not go down, it cannot contain any Type (i) or Type (ii) visibility edges. Let p_j be the first vertex along the chain, such that R_{j-1} is the same as C_{j-1}. Since q lies on the East boundary of $C(p, q)$, this condition is satisfied for the last visibility edge along the convex chain, hence p_j exists.

Let $C(p, p_j)$ be the homothet of $C(p, q)$ that has p and p_j on its boundary and is similar $C(p, q)$. Let a'' and b'' be the Northwest and Northeast corners of $C(p, p_j)$ (see Figure 7). Since p_j is first vertex along the convex chain that does not lie in the Northeast corner of R_{j-1}, we have that along the path from p to p_j the projections of $a'x$, all a_ip_{i+1}, and $a_{j-1}b_{j-1}$ onto $a''b''$ are disjoint and the projections of pa', all p_ia_i, and $p_{j-1}a_{j-1}$ onto pa'' are disjoint. Hence, their lengths sum up to at most $|pa''| + |a''b''|$. Finally, since $|b_{j-1}p_j| \leq |b''p_j|$, the total length of the path from p to p_j is at most $|pa''|+|a''b''|+|b''p_j|$, which is at most $|pa''| + |a''b''| + |b''m_j|$.

All Type (iii) visibility edges following p_j are charged as in Case (a), hence we have that $|p_ia_i| + |a_ib_i| + |b_ip_{i+1}| \leq |m_ia'_i| + |a'_ib'_i| + |b'_im_{i+1}|$. We now apply Lemma 5 to $C(p, p_j)$ and all C_i following p_j and obtain that the total length of the path from p to q is at most $|pa| + |ab| + |bq|$.

Case (d): Vertex x lies on the North boundary of R and the convex path goes down. Recall that the length of the path from p to x is at most $|pa'|+|a'x|$ and that p_1 is the neighbor of x along the convex chain. Let $C(p, p_1)$ be the homothet of $C(p, q)$ that has p and p_1 on its boundary and is similar to $C(p, q)$. Let a'' and b'' be the Northwest and Northeast corners of $C(p, p_1)$. Since p_1 lies to the right of R and lower than x, it lies on the East boundary of $C(p, p_1)$. We first show that the length of the path from p to p_1 is at most $|pa''| + |a''b''| + |b''p_1|$.

If xp_1 is a Type (i) visibility edge, the length of the path from x to p_1 is at most $|xb_0|+|b_0p_1|$. Hence we have a path from p to p_1 of length at most $|pa'|+|a'x|+|xb_0|+|b_0p_1| = |pa'| + |a''b''| + |b_0p_1|$. Since $|pa'| \leq |pa''|$ and $|b_0p_1| \leq |b''p_1|$, this implies that the path has length at most $|pa''|+|a''b''|+|b''p_1|$. If xp_1 is a Type (ii) visibility edge and x lies in the Northwest corner an analogous argument shows that the path from p to p_1 is at most $|pa''| + |a''b''| + |b''p_1|$. If xp_1 is a Type (ii) visibility edge and $R_0 = C_0$, we have that the projections of $a'x$ and a_0b_0 onto $a''b''$ are disjoint and the projections of pa' and xa_0 onto pa'' are disjoint. Hence, their total lengths sum up to at most $|pa''| + |a''b''|$. Finally, since $|b_0p_1| \leq |b''p_1|$, the total length of the path from p to p_1 is at most $|pa''| + |a''b''| + |b''p_1|$.

Next, we observe, like in Case (b), that starting from p_1 the convex path cannot go down more than $|p_1m_1|$. Hence, when we charge this to $C(p, p_1)$, we used $|pa''| + |a''b''| + |b''m_1|$ of its boundary. Finally, we use arguments analogous to the ones in Case (b) to show that each inductive path after p_1 has length at most $|m_ia'_i| + |a'_ib'_i| + |b'_im_{i+1}|$. We now apply Lemma 5 to $C(p, p_1)$ and all C_i following p_1 and obtain that the total length of the path from p to q is at most $|pa| + |ab| + |bq|$. □

Lemma 7 *Let p and q be two vertices that can see each other. Let $C(p, q)$ be the rectangle with p and q on its boundary, such that p lies in a corner of $C(p, q)$.*

Let l and s be the length of the long and short side of $C(p,q)$. The constrained generalized Delaunay graph contains a path between p and q of length at most $\left(\frac{2l}{s}+1\right)\cdot(|p_x-q_x|+|p_y-q_y|)$.

Proof. We assume without loss of generality that p lies on the Southwest corner and q lies on the East boundary. Note that this implies that the slope of pq is non-negative, i.e. $p_x < q_x$ and $p_y \leq q_y$. We prove the lemma by induction on the rank of $C(x,y)$ when ordered by size, for any two visible vertices x and y, such that x lies in a corner of $C(x,y)$. In fact, we show that the constrained generalized Delaunay graph contains a path between x and y of length at most $c\cdot(q_x-p_x)+d\cdot(q_y-p_y)$ and derive bounds on c and d.

Base case: If $C(p,q)$ is the smallest rectangle with p in a corner, then $C(p,q)$ does not contain any vertices visible to both p and q: Let u be a vertex in $C(p,q)$ that is visible to both p and q. Let $C(p,u)$ be the rectangle with p in a corner and u on its boundary. Since u lies in $C(p,q)$, $C(p,u)$ is smaller than $C(p,q)$, contradicting that $C(p,q)$ is the smallest rectangle with p in a corner. Hence, $C(p,q)$ does not contain any vertices visible to both p and q, which implies that pq is an edge of the constrained generalized Delaunay graph. Hence, the constrained generalized Delaunay graph contains a path between p and q of length at most $|pq| \leq (q_x-p_x)+(q_y-p_y) \leq c\cdot(q_x-p_x)+d\cdot(q_y-p_y)$, provided that $c \geq 1$ and $d \geq 1$.

Induction step: We assume that for all rectangles $C(x,y)$, with x in some corner of $C(p,q)$, smaller than $C(p,q)$ the lemma holds. If pq is an edge of the constrained generalized Delaunay graph, by the triangle inequality, the length of the shortest path from p to q is at most $|pq| \leq |p_x-q_x|+|p_y-q_y|$.

If there is no edge between p and q, there exists a vertex u in $C(p,q)$ that is visible from both p and q. We first look at the case where u lies below pq. Let g be the intersection of the South boundary of $C(p,q)$ and the line though q parallel to the diagonal of $C(p,q)$ through p, and let h be the Southeast corner of $C(p,q)$ (see Figure 8). If u lies in triangle pgq, by induction we have that the path from p to u has length at most $c\cdot(u_x-p_x)+d\cdot(u_y-p_y)$ and the path from u to q has length at most $c\cdot(q_x-u_x)+d\cdot(q_y-u_y)$. Hence, there exists a path from p to q via u of length at most $c\cdot(q_x-p_x)+d\cdot(q_y-p_y)$.

If u lies in triangle ghq, by induction we have that the path from p to u has length at most $c\cdot(u_x-p_x)+d\cdot(u_y-p_y)$ and the path from q to u has length at most $d\cdot(q_x-u_x)+c\cdot(q_y-u_y)$. When we take c and d to be equal, this implies that there exists a path from p to q via u of length at most $c\cdot(q_x-p_x)+d\cdot(q_y-p_y)$.

If there does not exist a vertex below pq that is visible to both p and q, than Lemma 3 implies that there are no vertices in $C(p,q)^p_q$ below pq that are visible to p and

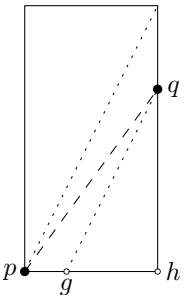


Figure 8: Rectangle $C(p,q)$ with points g and h.

that there are no vertices in $C(p,q)^q_p$ below pq that are visible to q. Hence, we can apply Lemma 6 and obtain that there exists a path between p and q of length at most $|pa|+|ab|+|bq|$, where a and b are the Northwest and Northeast corner of $C(p,q)$. Since $|ab|$ is (q_x-p_x) and $|bq| \leq |pa| \leq \frac{l}{s}\cdot(q_x-p_x)$, we can upper bound $|pa|+|ab|+|bq|$ by $c\cdot(q_x-p_x)$ when c is at least $\left(\frac{2l}{s}+1\right)$. Hence, since c and d need to be equal, we obtain that all cases work out when $c=d=\left(\frac{2l}{s}+1\right)$. □

Finally, since $(|p_x-q_x|+|p_y-q_y|)/|pq|$ is at most $\sqrt{2}$, we obtain the following theorem.

Theorem 8 *The constrained generalized Delaunay graph using an empty rectangle as empty convex shape has spanning ratio at most $\sqrt{2}\cdot\left(\frac{2l}{s}+1\right)$.*

References

[1] N. Bonichon, C. Gavoille, N. Hanusse, and L. Perković. The stretch factor of L_1- and L_∞-Delaunay triangulations. In *ESA*, pages 205–216, 2012.

[2] P. Bose, J.-L. De Carufel, and A. van Renssen. Constrained generalized Delaunay graphs are plane spanners. *EuroCG*, pages 176–179, 2015.

[3] P. Bose, R. Fagerberg, A. van Renssen, and S. Verdonschot. On plane constrained bounded-degree spanners. In *LATIN*, pages 85–96, 2012.

[4] P. Bose and J. M. Keil. On the stretch factor of the constrained Delaunay triangulation. In *ISVD*, pages 25–31, 2006.

[5] P. Bose and M. Smid. On plane geometric spanners: A survey and open problems. *CGTA*, 46(7):818–830, 2013.

[6] L. P. Chew. There is a planar graph almost as good as the complete graph. In *SoCG*, pages 169–177, 1986.

[7] K. Clarkson. Approximation algorithms for shortest path motion planning. In *STOC*, pages 56–65, 1987.

[8] G. Das. The visibility graph contains a bounded-degree spanner. In *CCCG*, pages 70–75, 1997.

[9] G. Narasimhan and M. Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.

[10] K. Swanepoel. Helly-type theorems for homothets of planar convex curves. *AMS*, 131(3):921–932, 2003.

Flips in Edge-Labelled Pseudo-Triangulations*

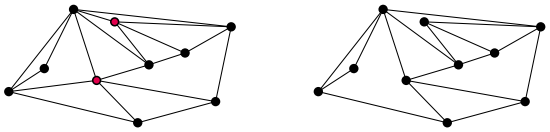
Prosenjit Bose§ Sander Verdonschot§

Abstract

We show that $O(n^2)$ exchanging flips suffice to transform any edge-labelled pointed pseudo-triangulation into any other with the same set of labels. By using insertion, deletion and exchanging flips, we can transform any edge-labelled pseudo-triangulation into any other with $O(n \log c + h \log h)$ flips, where c is the number of convex layers and h is the number of points on the convex hull.

1 Introduction

A *pseudo-triangle* is a simple polygon with three convex interior angles, called *corners*, that are connected by reflex chains. Given a set P of n points in the plane, a *pseudo-triangulation* of P is a subdivision of its convex hull into pseudo-triangles, using all points of P as vertices (see Figure 1a). A pseudo-triangulation is *pointed* if all vertices are incident to a reflex angle in some face (including the outer face; see Figure 1b for an example). Pseudo-triangulations find applications in areas such as kinetic data structures [6] and rigidity theory [9]. More information on pseudo-triangulations can be found in a survey by Rote, Santos, and Streinu [8].



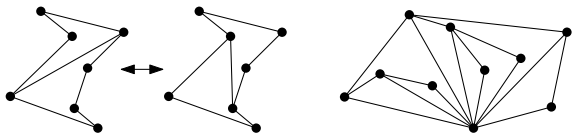
(a) (b)

Figure 1: (a) A pseudo-triangulation with two non-pointed vertices. (b) A pointed pseudo-triangulation.

Since a regular triangle is also a pseudo-triangle, pseudo-triangulations generalize triangulations (subdivisions of the convex hull into triangles). Triangulations have numerous applications and are extremely well-studied; the particular topic we are interested in is that of flips. In a triangulation, a flip is a local transformation that removes one edge, leaving an empty quadrilateral, and inserts the other diagonal of that quadrilateral. Note that this is only possible if the quadrilateral is convex. Lawson [7] showed that any triangulation with n vertices can be transformed into any other with $O(n^2)$ flips, and Hurtado, Noy, and Urrutia [5] gave a matching $\Omega(n^2)$ lower bound.

Pointed pseudo-triangulations support a similar type of flip, but before we can introduce this, we need to generalize the concept of pseudo-triangles to *pseudo-k-gons*: weakly simple polygons with k convex interior angles. A diagonal of a pseudo-k-gon is called a *bitangent* if the pseudo-k-gon remains pointed after insertion of the diagonal. In a pointed pseudo-triangulation, *flipping* an edge removes the edge, leaving a pseudo-quadrilateral (a pseudo-4-gon), and inserts the unique other bitangent of the pseudo-quadrilateral (see Figure 2a). In contrast with triangulations, all internal edges of a pointed pseudo-triangulation are flippable. Bereg [3] showed that $O(n \log n)$ flips suffice to transform any pseudo-triangulation into any other.

Aichholzer et al. [2] showed that the same result holds for all pseudo-triangulations (including triangulations) if we allow two more types of flips: *insertion* and *deletion* flips. As the name implies, these either insert or delete one edge, provided that the result is still a pseudo-triangulation. To disambiguate, they call the other flips *exchanging* flips. In a later paper, this bound was refined to $O(n \log c)$ [1], where c is the number of convex layers of the point set.



(a) (b)

Figure 2: (a) A flip in a pseudo-quadrilateral. (b) A left-shelling pseudo-triangulation.

In this paper, we investigate flips in *edge-labelled pseudo-triangulations*: pseudo-triangulations where each internal edge has a unique label in $\{1, \ldots, 3n-3-2h\}$, where h is the number of vertices on the convex hull ($3n-3-2h$ is the number of internal edges in a triangulation). In the case of an exchanging flip, the new edge receives the label of the old edge. For a deletion flip, the edge and its label are simply removed, and for an insertion flip, the new edge receives an unused label from the set of all possible labels. The edge-labelled version is more difficult than the unlabelled version, since we no longer have the freedom to choose the mapping between

*This work was partially supported by NSERC.

§School of Computer Science, Carleton University, Ottawa, jit@scs.carleton.ca, sander@cg.scs.carleton.ca

edges in the initial and final pseudo-triangulation.

Bose et al. [4] initiated the study of flips in edge-labelled triangulations. They gave a tight $\Theta(n \log n)$ bound on the worst-case number of flips required for triangulations of points in convex position. However, in general, they show that it is not always possible to flip between two given edge-labelled triangulations. In contrast, we show that it is always possible to transform two given edge-labelled pseudo-triangulations into each other using flips.

Our results are the following: using only exchanging flips, we show that $O(n^2)$ flips suffice to transform any edge-labelled pointed pseudo-triangulation into any other with the same set of labels. By using insertion, deletion and exchanging flips, we can transform any edge-labelled pseudo-triangulation into any other with $O(n \log c + h \log h)$ flips.

Before we prove our results, we need a few more definitions. Given a set of points in the plane, let v_0 be the point with the lowest y-coordinate, and let $v_1, \ldots, v_n$ be the other points in clockwise order around v_0. The *left-shelling* pseudo-triangulation is the union of the convex hulls of $v_0, \ldots, v_i$, for all $2 \leq i \leq n$ (see Figure 2b). Thus, every vertex after v_1 is associated with two edges: a *bottom* edge connecting it to v_0 and a *top* edge that is tangent to the convex hull of the earlier vertices. The *right-shelling* pseudo-triangulation is similar, with the vertices added in counter-clockwise order instead.

2 Transforming pointed pseudo-triangulations

In this section, we show that every edge-labelled pointed pseudo-triangulation can be transformed into any other with the same set of labels by $O(n^2)$ exchanging flips. We do this by showing how to transform a given edge-labelled pointed pseudo-triangulation into a *canonical* one. The result then follows by the reversibility of flips. We use the left-shelling pseudo-triangulation as canonical pseudo-triangulation, with the bottom edges labelled in clockwise order around v_0, followed by the internal top edges in the same order (based on their associated vertex).

Since we can transform any pointed pseudo-triangulation into the left-shelling pseudo-triangulation with $O(n \log n)$ flips [3], the main part of the proof lies in reordering the labels of a left-shelling pseudo-triangulation. We use two tools for this, called a *sweep* and a *shuffle*, that are implemented by a sequence of flips. A sweep interchanges the labels of some internal top edges with their respective bottom edges, while a shuffle permutes the labels on all bottom edges.

Lemma 1 *We can transform any left-shelling pseudo-triangulation into the canonical one with $O(1)$ shuffle and sweep operations.*

Proof. In the canonical pseudo-triangulation, we call the labels assigned to bottom edges *low*, and the labels assigned to top edges *high*. In the first step, we use a shuffle to line up every bottom edge with a high label to a top edge with a low label. Then we exchange these pairs of labels with a sweep. Now all bottom edges have low labels and all top edges have high labels, so all that is left is to sort the labels. We can sort the low labels with a second shuffle. To sort the high labels, we sweep them to the bottom edges, shuffle to sort them there, then sweep them back. □

The remainder of this section describes how to perform a sweep and a shuffle with flips.

Lemma 2 *We can interchange the labels of the edges incident to an internal vertex v of degree two with three exchanging flips.*

Proof. Consider what happens when we remove v. Deleting one of its edges leaves a pseudo-quadrilateral. Removing the second edge then either merges two corners into one, or removes one corner, leaving a pseudo-triangle T. There are three bitangents that connect v to T, each corresponding to the geodesic between v and a corner of T. Any choice of two of these bitangents results in a pointed pseudo-triangulation. When one of them is flipped, the only new edge that can be inserted so that the result is still a pointed pseudo-triangulation is the bitangent that was not there before the flip. Thus, we can interchange the labels with three flips (see Figure 3). □

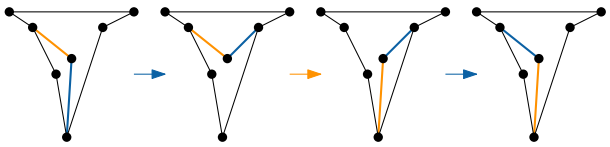


Figure 3: Interchanging the labels of the edges incident to a vertex of degree two.

Lemma 3 (Sweep) *In the left-shelling pseudo-triangulation, we can interchange the labels of any number of internal top edges and their corresponding bottom edges with $O(n)$ exchanging flips.*

Proof. Let S be the set of vertices whose internal top edge should have its label swapped with the corresponding bottom edge. Consider a ray L from v_0 that starts at the positive x-axis and sweeps through the point set to the negative x-axis. We will maintain the following invariant: the graph induced by the vertices to the left of L is their left-shelling pseudo-triangulation and the graph induced by the vertices to the right of L is their right-shelling pseudo-triangulation (both groups include

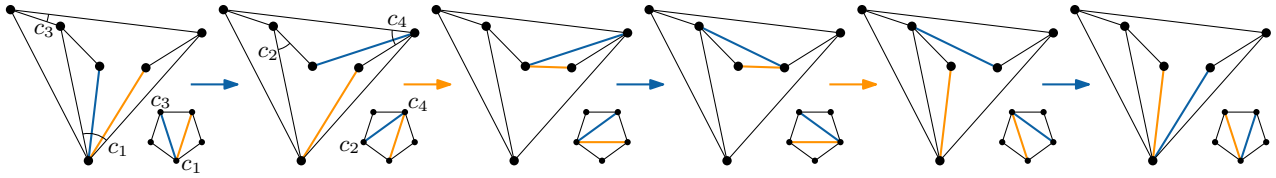


Figure 4: Interchanging the labels of two bitangents of a pseudo-pentagon with five bitangents. An edge in the pentagon corresponds to a geodesic between two corners of the pseudo-pentagon.

v_0). Furthermore, the labels of the top edges of the vertices in S to the right of L have been interchanged with their respective bottom edges. This invariant is satisfied at the start.

Suppose that L is about to pass a vertex v_k. If v_k is on the convex hull, its top edge is not internal and no action is required for the invariant to hold after passing v_k. So assume that v_k is not on the convex hull and consider its incident edges. It is currently part of the left-shelling pseudo-triangulation of points to the left of L, where it is the last vertex. Thus, v_k is connected to v_0 and to one vertex to its left. It is not connected to any vertex to its right, since there are $2n-3$ edges in total, and the left- and right-shelling pseudo-triangulations to each side of L contribute $2(k+1)-3+2(n-k)-3 = 2n-4$ edges. So the only edge that crosses L is an edge of the convex hull. Therefore v_k has degree two, which means that we can use Lemma 2 to swap the labels of its top and bottom edge with three flips if $v_k \in S$.

Furthermore, the sides of the pseudo-triangle that remains if we were to remove v_k, form part of the convex hull of the points to either side of L. Thus, flipping the top edge of v_k results in the tangent from v_k to the convex hull of the points to the right of L – exactly the edge needed to add v_k to their right-shelling pseudo-triangulation. Therefore we only need $O(1)$ flips to maintain the invariant when passing v_k.

At the end, we have constructed the right-shelling pseudo-triangulation and swapped the desired edges. An analogous transformation without any swapping can transform the graph back into the left-shelling pseudo-triangulation with $O(n)$ flips in total. □

Lemma 4 *In the left-shelling pseudo-triangulation, we can interchange the labels of two consecutive bottom edges with $O(1)$ exchanging flips.*

Proof. When we remove the two consecutive bottom edges (say a and b), we are left with a pseudo-pentagon X. A pseudo-pentagon can have up to five bitangents, as each bitangent corresponds to a geodesic between two corners. If X has exactly five bitangents, this correspondence is a bijection. This implies that the bitangents of X can be swapped just like diagonals of a convex pentagon (see Figure 4). On the other hand, if X has only four bitangents, it is impossible to swap a and b without flipping an edge of X.

Fortunately, we can always transform X into a pseudo-pentagon with five bitangents. If the pseudo-triangle to the right of b is a triangle, X already has five bitangents (see Lemma 19 in the Appendix). Otherwise, the top endpoint of b is an internal vertex of degree two and we can flip its top edge to obtain a new pseudo-pentagon that does have five bitangents (see Lemma 20 in the Appendix). After swapping the labels of a and b, we can flip this top edge back. Thus, in either case we can interchange the labels of a and b with $O(1)$ flips. □

We can use Lemma 4 to reorder the labels of the bottom edges with insertion or bubble sort, as these algorithms only swap adjacent values.

Corollary 5 (Shuffle) *In the left-shelling pseudo-triangulation, we can reorder the labels of all bottom edges with $O(n^2)$ exchanging flips.*

Combining this with Lemmas 1 and 3, and the fact that we can transform any pointed pseudo-triangulation into the left-shelling one with $O(n \log n)$ flips [3], gives the main result.

Theorem 6 *We can transform any edge-labelled pointed pseudo-triangulation with n vertices into any other with $O(n^2)$ exchanging flips.*

The following lower bound follows from the $\Omega(n \log n)$ lower bound on the flip distance between edge-labelled triangulations of a convex polygon [4].

Theorem 7 *There are pairs of edge-labelled pointed pseudo-triangulations with n vertices that require $\Omega(n \log n)$ exchanging flips to transform one into the other.*

3 Transforming general pseudo-triangulations

In this section, we extend our results for edge-labelled pointed pseudo-triangulations to all edge-labelled pseudo-triangulations. Since not all pseudo-triangulations have the same number of edges, we need to allow flips that change the number of edges. In particular, we allow a single edge to be deleted or inserted, provided that the result is still a pseudo-triangulation.

Since we are dealing with edge-labelled pseudo-triangulations, we need to determine what happens to the edge labels. It is useful to first review the properties we would like these flips to have. First, a flip should be a local operation – it should affect only one edge. Second, a labelled edge should be flippable if and only if the edge is flippable in the unlabelled setting. This allows us to re-use the existing results on flips in pseudo-triangulations. Third, flips should be reversible. Like most proofs about flips, our proof in the previous section crucially relies on the reversibility of flips.

With these properties in mind, the edge-deletion flip is rather straightforward – the labelled edge is removed, and other edges are not affected. Since the edge-insertion flip needs to be the inverse of this, it should insert the edge and assign it a *free label* – an unused label in $\{1, \ldots, 3n-3-2h\}$, where h is the number of vertices on the convex hull ($3n-3-2h$ is the number of internal edges in a triangulation).

With the definitions out of the way, we turn our attention to the number of flips required to transform any edge-labelled pseudo-triangulation into any other. In this section, we show that by using insertion and deletion flips, we can shuffle (permute the labels on bottom edges) with $O(n + h \log h)$ flips. Combined with the unlabelled bound of $O(n \log c)$ flips by Aichholzer et al. [1], this brings the total number of flips down to $O(n \log c + h \log h)$. Note that, by the results of Bose et al. [4], this holds for a set of points in convex position ($h = n$). In the remainder of this section we assume that $h < n$. As before, we first build a collection of simple tools that help prove the main result.

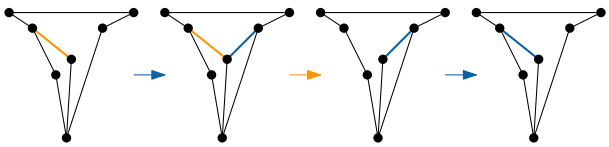


Figure 5: Interchanging the label of an edge incident to a vertex of degree two with a free label.

Lemma 8 *With $O(1)$ flips, we can interchange the label of an edge incident to an internal vertex of degree two with a free label.*

Proof. Let v be a vertex of degree two and let e be an edge incident to v. Since v has degree two, its removal leaves an empty pseudo-triangle T. There are three bitangents that connect v to T, one for each corner. Thus, we can insert the third bitangent f with the desired free label, making v non-pointed (see Figure 5). Flipping e now removes it and frees its label. Finally, flipping f moves it into e's starting position, completing the exchange. □

This implies that, using an arbitrary free label as placeholder, we can swap any two edges incident to internal degree-two vertices – no matter where they are in the pseudo-triangulation.

Corollary 9 *We can interchange the labels of two edges, each incident to some internal vertex of degree two, with $O(1)$ flips.*

Recall that during a sweep (Lemma 3), each internal vertex has degree two at some point. Since the number of free labels for a pointed pseudo-triangulation is equal to the number of internal vertices, this means that we can use Lemma 8 to swap every label on a bottom edge incident to an internal vertex with a free label by performing a single sweep. Afterwards, a second sweep can replace these labels on the bottom edges in any desired order. Thus, permuting the labels on bottom edges incident to internal vertices can be done with $O(n)$ flips. Therefore, the difficulty in permuting the labels on all bottom edges lies in bottom edges that are not incident to an internal vertex, that is, chords of the convex hull. If there are few such chords, a similar strategy (free them all and replace them in the desired order) might work. Unfortunately, the number of free labels can be far less than the number of chords.

We now consider operations on maximal groups of consecutive chords, which we call *fans*. As the vertices of a fan are in convex position, fans behave in many ways like triangulations of a convex polygon, which can be rearranged with $O(n \log n)$ flips [4]. The problem now becomes getting the right set of labels on the edges of a fan.

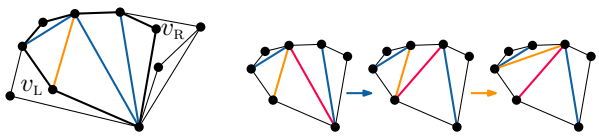


Figure 6: (a) An indexed fan. (b) Shifting the index.

Consider the internal vertices directly to the left (v_L) and right (v_R) of a fan F, supposing both exist. Vertex v_L has degree two and forms part of the reflex chain of the first pseudo-triangle to the left of F. Thus, flipping v_L's top edge connects it to the leftmost vertex of F (excluding v_0). Vertex v_R is already connected to the rightmost vertex of F, so we just ensure that it has degree two. To do this, we flip all incident edges from vertices further to the right, from the bottom to the top. Now the diagonals of F form a triangulation of a convex polygon whose boundary consists of v_0, v_L, the top endpoints of the chords, and v_R (see Figure 6a). It is possible that there is no internal vertex to one side of F. In that case, there is only one vertex on that side of

F, which is part of the convex hull, and we can simply use that vertex in place of v_{L} or v_{R} without flipping any of its edges. Since there is at least one internal vertex by assumption, either v_{L} or v_{R} is an internal vertex. This vertex is called the *index* of F. If a vertex is the index of two fans, it is called a *shared index*.

A triangulated fan is called an *indexed fan* if there is one edge incident to the index, the *indexed edge*, and the remaining edges are incident to one of the neighbours of the index on the boundary. Initially, all diagonals of F are incident to v_0, so we transform it into an indexed fan by flipping the diagonal of F closest to the index. Next, we investigate several operations on indexed fans that help us move labels between fans.

Lemma 10 (Shift) *In an indexed fan, we can shift the indexed edge to the next diagonal with $O(1)$ flips.*

Proof. Suppose that v_{L} is the index (the proof for v_{R} is analogous). Let e be the current indexed edge, and f be the leftmost diagonal incident to v_0. Then flipping f followed by e makes f the only edge incident to the index and e incident to the neighbour of the index (see Figure 6b). Since flips are reversible, we can shift the index the other way too. □

Lemma 11 *We can switch which fan a shared index currently indexes with $O(1)$ flips.*

Proof. Flipping the current indexed edge "parks" it by connecting it to the two neighbours of the index, and reduces the degree of the index to two (see Figure 7). Now, flipping the top edge of the index connects it to the other fan, where we parked the previously indexed edge. Flipping that edge connects it to the index again. □

Lemma 12 *In a pointed pseudo-triangulation, we can always decrease the degree of a vertex v of degree three by flipping one of the edges incident to its reflex angle.*

Proof. Consider the geodesic from v to the opposite corner c of the pseudo-triangle v is pointed in. The line supporting the part of the geodesic when it reaches v splits the edges incident to v into two groups. As there are three edges, one of these groups must contain multiple edges. Flipping the edge incident to its reflex angle in the group with multiple edges results in a geodesic to c. If this geodesic passed through v, it would insert the missing edges along the geodesic from v to c (otherwise we could find a shorter path). But inserting this geodesic would make v non-pointed. Thus, v cannot be on this geodesic. Therefore the new edge is not incident to v and the flip reduces the degree of v. □

Since the index always has degree three, this allows us to extend the results from Lemma 8 and Corollary 9 regarding vertices of degree two to indexed edges.

Corollary 13 *In an indexed fan, we can interchange the label of the indexed edge with a free label in $O(1)$ flips.*

Corollary 14 *Given two indexed fans, we can interchange the labels of the two indexed edges with $O(1)$ flips.*

Now we have enough tools to shuffle the bottom edges.

Lemma 15 (Shuffle) *In the left-shelling pseudo-triangulation, we can reorder the labels of all bottom edges with $O(n + h \log h)$ flips, where h is the number of vertices on the convex hull.*

Proof. In the initial pseudo-triangulation, let B and $\mathcal{F}$ be the sets of labels on bottom edges and free labels, respectively. Let F_i be the set of labels on the i-th fan (in some fixed order), and let $\overline{F}$ be the set of labels on non-fan bottom edges. Let F_i' and $\overline{F}'$ be these same sets in the target pseudo-triangulation. As we are only rearranging the bottom labels, we have that $B = F_1 \cup \ldots \cup F_k \cup \overline{F} = F_1' \cup \ldots \cup F_k' \cup \overline{F}'$, where k is the number of fans.

We say that a label ℓ *belongs to* fan i if $\ell \in F_i'$. At a high level, the reordering proceeds in four stages. In stage one, we free all labels in $\overline{F}$. In stage two, we place each label from $B \setminus \overline{F}'$ in the fan it belongs to, leaving the labels in $\overline{F}'$ free. Then, in stage three, we correct the order of the labels within each fan. Finally, we place the labels in $\overline{F}'$ correctly.

Since each internal vertex contributes exactly one top edge, one bottom edge, and one free label, we have that $|\overline{F}| = |\mathcal{F}|$. To free all labels in $\overline{F}$, we perform a sweep (see Lemma 3). As every internal vertex has degree two at some point during the sweep, we can exchange the label on its bottom edge with a free label at that point, using Lemma 8. This requires $O(n)$ flips. The labels in $\mathcal{F}$ remain on the bottom edges incident to internal vertices throughout stage two and three, as placeholders.

To begin stage two, we index all fans with $O(n)$ flips and shift these indices to the first 'foreign' edge: the first edge whose label does not belong to the current fan. If no such edge exists, we can ignore this fan for the remainder of stage two, as it already has the right set of labels. Now suppose that there is a fan F_i whose indexed edge e is foreign: $\ell_e \notin F_i'$. Then either $\ell_e \in F_j'$ for some $j \neq i$, or $\ell_e \in \overline{F}'$. In the first case, we exchange ℓ_e with the label on the indexed edge of F_j, and shift the index of F_j to the next foreign edge. In the second case, we exchange ℓ_e with a free label in $B \setminus \overline{F}'$. If this label belongs to F_i, we shift its index to the next foreign edge. In either case, we increased the number of correctly placed labels by at least one. Thus $n - 1$ repetitions suffice to place all labels in the

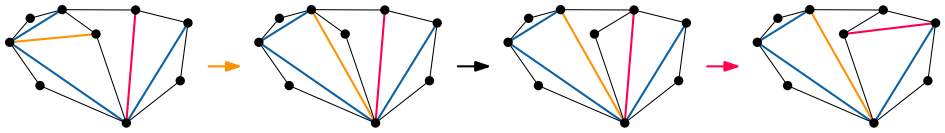


Figure 7: Changing which side a shared index indexes.

fan they belong to, wrapping up stage two. Since we perform a linear number of swaps and shifts, and each takes a constant number of flips, the total number of flips required for stage two is $O(n)$.

For stage three, we note that each indexed fan corresponds to a triangulation of a convex polygon. As such, we can rearrange the labelled diagonals of a fan F_i into their desired final position with $O(|F_i| \log |F_i|)$ flips [4]. Thus, if we let h be the number of vertices on the convex hull, the total number of flips for this step is bounded by

$$\sum_i O(|F_i| \log |F_i|) \leq \sum_i O(|F_i| \log h) = O(h \log h).$$

For stage four, we first return to a left-shelling pseudo-triangulation by un-indexing each fan, using $O(n)$ flips. After stage two, the labels in $\overline{F}'$ are all free, so all that is left is to place these on the correct bottom edges, which we can do with a final sweep. Thus, we can reorder all bottom labels with $O(n + h \log h)$. □

This leads to the following bound.

Theorem 16 *We can transform any edge-labelled pseudo-triangulation with n vertices into any other with $O(n \log c + h \log h)$ flips, where c is the number of convex layers and h is the number of vertices on the convex hull.*

Proof. Using the technique by Aichholzer et al. [1], we first transform the pseudo-triangulation into the left-shelling pseudo-triangulation T with $O(n \log c)$ flips. Our canonical pseudo-triangulation contains the labels $\{1, \ldots, 2n - h - 3\}$, but it is possible for T to contain a different set of labels. Since all labels are drawn from $\{1, \ldots, 3n - 2h - 3\}$, at most $n - h$ labels differ. This is exactly the number of internal vertices. Thus, we can use $O(n + h \log h)$ flips to shuffle (Lemma 15) all non-canonical labels on fan edges to bottom edges incident to an internal vertex. Once there, we use a sweep (Lemma 3) to ensure that every internal vertex has degree two at some point, at which time we replace its incident non-canonical labels with canonical ones with a constant number of flips (Lemma 8). Once our left-shelling pseudo-triangulation has the correct set of labels, we use a constant number of shuffles and sweeps to sort the labels (Lemma 1). Since we can shuffle and sweep with $O(n + h \log h)$ and $O(n)$ flips, respectively, the total number of flips reduces to $O(n \log c + n + h \log h) = O(n \log c + h \log h)$. □

The correspondence between triangulations of a convex polygon and pseudo-triangulations gives us the following lower bound.

Theorem 17 *There are pairs of edge-labelled pseudo-triangulations with n vertices such that any sequence of flips that transforms one into the other has length $\Omega(n \log n)$.*

Acknowledgements

We would like to thank Anna Lubiw and Vinayak Pathak for helpful discussions.

References

[1] O. Aichholzer, F. Aurenhammer, C. Huemer, and H. Krasser. Transforming spanning trees and pseudo-triangulations. *Information Processing Letters*, 97(1):19–22, 2006.

[2] O. Aichholzer, F. Aurenhammer, H. Krasser, and P. Brass. Pseudotriangulations from surfaces and a novel type of edge flip. *SIAM Journal on Computing*, 32(6):1621–1653 (electronic), 2003.

[3] S. Bereg. Transforming pseudo-triangulations. *Information Processing Letters*, 90(3):141–145, 2004.

[4] P. Bose, A. Lubiw, V. Pathak, and S. Verdonschot. Flipping edge-labelled triangulations. *ArXiv e-prints*, 2013. arXiv:1310.1166 [cs.CG].

[5] F. Hurtado, M. Noy, and J. Urrutia. Flipping edges in triangulations. *Discrete & Computational Geometry.*, 22(3):333–346, 1999.

[6] D. Kirkpatrick, J. Snoeyink, and B. Speckmann. Kinetic collision detection for simple polygons. *International Journal of Computational Geometry & Applications*, 12(1-2):3–27, 2002.

[7] C. L. Lawson. Transforming triangulations. *Discrete Mathematics*, 3(4):365–372, 1972.

[8] G. Rote, F. Santos, and I. Streinu. Pseudo-triangulations—a survey. In *Surveys on discrete and computational geometry*, volume 453 of *Contemporary Mathematics*, pages 343–410. 2008.

[9] I. Streinu. Pseudo-triangulations, rigidity and motion planning. *Discrete & Computational Geometry.*, 34(4):587–635, 2005.

Appendix

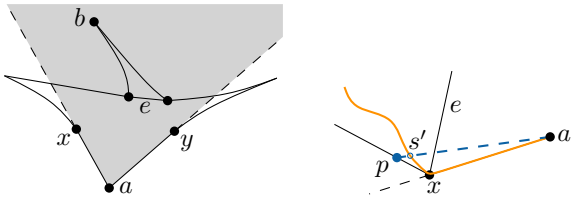


Figure 8: (a) A corner of a pseudo-triangle and an edge such that the entire pseudo-triangle on the other side of the edge lies inside the corner's wedge. (b) If a can see a point past x, then the geodesic does not contain x.

Lemma 18 *Let a be a corner of a pseudo-triangle with neighbours x and y, and let e be an edge on the chain opposite a. If all vertices of the other pseudo-triangle containing e lie in the wedge formed by extending the edges ax and ay into half-lines (see Figure 8a), then flipping e will result in an edge incident on a.*

Proof. Let T be the pseudo-triangle on the other side of e, and let b be the corner of T opposite e. Then flipping e inserts the geodesic between a and b. This geodesic must intersect e in a point s and then follow the shortest path from s to a. If s lies strictly inside the wedge, nothing can block as, thus the new edge will contain as and be incident on a.

Now, if all of e lies strictly inside the wedge, our result follows. But suppose that e has x as an endpoint and the geodesic between a and b intersects e in x. As a can see x and all of T lies inside the wedge, there is an $\varepsilon > 0$ such that a can see the point X on the boundary of T at distance ε from x (see Figure 8b). The line segment ap intersects the geodesic at a point s'. By the triangle inequality, $s'a$ is shorter than following the geodesic from s' via x to a. But then this would give a shorter path between a and b, by following the geodesic to s' and then cutting directly to a. As the geodesic is the shortest path by definition, this is impossible. Thus, the geodesic cannot intersect e at x and the new edge must be incident to a. □

Lemma 19 *Let a and b be two consecutive internal bottom edges in the left-shelling pseudo-triangulation, such that the pseudo-triangle to the right of b is a triangle. Then the pseudo-pentagon X formed by removing a and b has five bitangents.*

Proof. Let $c_0, \ldots, c_4$ be the corners of X in counterclockwise order around the boundary. By Lemma 18, flipping b results in an edge b' that intersects b and is incident on c_1. This edge is part of the geodesic between c_1 and c_3, and as such it is tangent to the convex chain $v_0, v_a, \ldots, c_3$, where v_a is the top endpoint of a (v_a could be c_3). Therefore it is also the tangent from c_1 to the convex hull of $\{v_0, \ldots, v_a\}$. This means that the newly created pseudo-triangle with c_1 as corner and a on the opposite pseudo-edge also meets the conditions of Lemma 18. Thus, flipping a results in another edge, a', also incident on c_1. As b separates c_1 from all vertices in $\{v_0, \ldots, v_a\}$, a' must also intersect b. This gives us four bitangents, of which two are incident on v_0 (a and b), and two on c_1 (a' and b'). Finally, flipping a before flipping b results in a bitangent that is not incident on v_0 (as v_0 is a corner and cannot be on the new geodesic), nor on c_1 (as b separates a from c_1). Thus, X has five bitangents. □

Lemma 20 *Let a and b be two consecutive internal bottom edges in the left-shelling pseudo-triangulation, such that the pseudo-triangle to the right of b is not a triangle. Then the pseudo-pentagon X formed by flipping the corresponding top edge of b and removing a and b has five bitangents.*

Proof. Let v_a and v_b be the top endpoints of a and b. By Lemma 18 and since b had degree two, flipping the top edge of b results in the edge v_bc_1. We get three bitangents for free: a, b, and b' – the old top edge of b and the result of flipping b.

X consists of a reflex chain C that is part of the convex hull of the points to the left of a, followed by three successive tangents to C, v_a, or v_b. Since C lies completely to the left of a, it cannot significantly alter any of the geodesics or bitangents inside the polygon, so we can reduce it to a single edge. Now, X consists either of a triangle with two internal vertices, or a convex quadrilateral with one internal vertex.

If X is a triangle with two internal vertices, the internal vertices are v_a and v_b. Let its exterior vertices be v_0, x, and y. Then there are seven possible bitangents: $a = v_0v_a, b = v_0v_b, xv_a, xv_b, yv_a, yv_b$, and v_av_b. We know that xv_a and yv_b are edges, so there are five possible bitangents left. As all vertices involved are either corners or have degree one in X, the only condition for an edge to be a bitangent is that it does not cross the boundary of X. Since the exterior boundary is a triangle, this reduces to it not crossing xv_a and yv_b. Two line segments incident to the same vertex cannot cross. Thus, xv_b, yv_a, and v_av_b cannot cross xv_a and yv_b, and X has five bitangents.

If X's convex hull has four vertices, the internal vertex is v_b (otherwise the pseudo-triangle to the right of b would be a triangle). Let its exterior vertices be v_0, x, v_a, and y. Then there are six possible bitangents: $a = v_0v_a, b = v_0v_b, xy, xv_b, yv_b$, and v_av_b, of which one (yv_b) is an edge of X. Since a and b are guaranteed to be bitangents, and xy, xv_b, and v_av_b all share an endpoint with yv_b, the arguments from the previous case apply and we again have five bitangents. □

The Shadows of a Cycle Cannot All Be Paths

Prosenjit Bose* Jean-Lou De Carufel* Michael G. Dobbins † Heuna Kim ‡ Giovanni Viglietta§

Abstract

A *shadow* of a subset S of Euclidean space is an orthogonal projection of S into one of the coordinate hyperplanes. In this paper we show that it is not possible for all three shadows of a cycle (i.e., a simple closed curve) in $\mathbb{R}^3$ to be paths (i.e., simple open curves).

We also show two contrasting results: the three shadows of a path in $\mathbb{R}^3$ can all be cycles (although not all convex) and, for every $d \geq 1$, there exists a d-sphere embedded in $\mathbb{R}^{d+2}$ whose $d+2$ shadows have no holes (i.e., they deformation-retract onto a point).

1 Introduction

Oskar's maze, named after the Dutch puzzle designer Oskar van Deventer, who invented it in 1983, is a mechanical puzzle consisting of a hollow cube and three mutually orthogonal rods joined at their centers (see Figure 1). Each face of the cube has slits forming a maze, and the mazes on opposite faces are identical. Each rod is orthogonal to a pair of opposite faces, and it is able to slide in the slits, tracing out the maze. Hence, in order to move the rods around, one has to solve three mazes simultaneously.

In 1994, Hendrik W. Lenstra asked if the mazes could be chosen so that the common point of the three rods could trace a simple closed curve. Observe that none of the mazes may contain any cycles, or some pieces of the cube would fall out of the puzzle. So, what Lenstra was really asking for is a simple closed curve whose projections onto three pairwise orthogonal planes contain no cycles. In other words, he wanted the three shadows of a simple closed 3D curve to all be trees.

As Peter Winkler reported in his book *Mathematical mind-benders* [4], a solution had already been found some years before by John R. Rickard, who discovered the curve illustrated in Figure 2, also appearing on the front cover of Winkler's book.

Figure 1: Oskar's maze, produced by Bits and Pieces.

Several other, more complex solutions to Lenstra's problem are known. Notably, in 2012 Adam P. Goucher constructed a simple closed curve having shadows that are all trees, which also happens to be a trefoil knot [3]. The curve was therefore named *Treefoil*. Goucher also constructed a pair of linked cycles whose union has shadows that are all trees.

Figure 2: Rickard's curve, illustrated by Afra Zomorodian, and appearing on the front cover of Peter Winkler's book *Mathematical mind-benders*.

Our research is motivated by the following two questions. Is it possible for the three shadows of a simple closed curve to be paths, i.e., have neither cycles nor branch points? Can the three shadows of a simple open curve be simple closed curves? Both these questions are related to Lenstra's question, whose history is outlined

*School of Computer Science, Carleton University, Ottawa ON, Canada, jit@scs.carleton.ca, jdecaruf@cg.scs.carleton.ca.

†Department of Mathematical Sciences, Binghamton University, Binghamton, NY, USA. michaelgenedobbins@gmail.com

‡Department of Mathematics, Arbeitsgruppe Theoretische Informatik, Freie Universität Berlin, Berlin Germany. heunak@mi.fu-berlin.de

§School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa ON, Canada, viglietta@gmail.com.

in [4]. These questions have also been posed independently (see [1, 2]).

Our contribution. In Section 2 we answer the first question in the negative: the three shadows of a simple closed curve in $\mathbb{R}^3$ cannot all be paths.

In Section 3 we answer the second question in the affirmative: there exist simple open curves in $\mathbb{R}^3$ whose three shadows are simple closed curves, although the shadows cannot all be convex. Furthermore, we exhibit a polygonal chain with this property having only six vertices, and we prove that six is the minimum.

In Section 4 we extend Rickard's curve to higher dimensions, giving an inductive construction of a d-sphere embedded in $\mathbb{R}^{d+2}$, for every $d \geq 1$, whose $d+2$ shadows are all contractible. (A contractible set is one that can be continuously shrunk to a point, and hence it has no holes.)

Section 5 concludes the paper with some remarks and suggestions for further work.

This research has obvious applications in computer vision and 3D object reconstruction, where the goal is to deduce properties of an unknown 3-dimensional object given its three projections. Specifically, we may want to study the topology of an object that projects to three given paths. It is easy to see that such an object may not be unique, and hence it makes sense to study the set of 3-dimensional objects that are *compatible* with three given projections. Observe that any such set is closed under taking unions, and therefore it has a unique "largest" object, which is the union of all the objects in the set.

It is interesting to note that there are triplets of paths that are not compatible with any connected set, such as the one in Figure 3. This means that an Oskar's-maze-like puzzle could be "unsolvable" even if it had no crossroads on any face. By "unsolvable" we mean that the set of locations that are reachable by the central point of the three rods depends on where the rods are located. Therefore, if we assign two points in the 3-dimensional maze determined by the three 2-dimensional mazes, it may be impossible to go from one to the other by moving the rods around.

2 The shadows of a cycle

In this section we prove that the shadows of a simple closed curve in $\mathbb{R}^3$ cannot all be simple open curves. We start with some notation and definitions.

For a point $p \in \mathbb{R}^n$ and $1 \leq i \leq n$, we denote by p_i the i-th coordinate of p. The *x_i-projection*, or *x_i-shadow*, of a set $A \subseteq \mathbb{R}^n$, denoted by $\pi_i(A)$, is the orthogonal projection of A into the i-th coordinate hyperplane, e.g., $\pi_1(A) = \{(p_2, p_3, \cdots, p_n) \mid p \in A\}$. If $A = \{p\}$, we may simply write $\pi_i(p)$ instead of $\pi_i(\{p\})$.

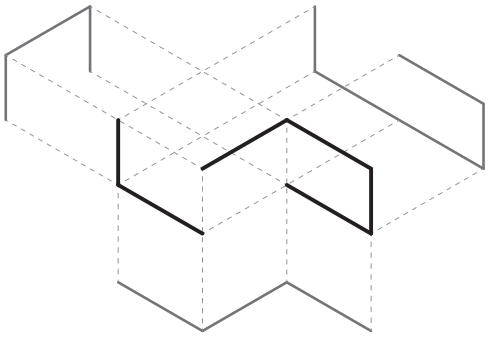


Figure 3: Connected shadows whose unique compatible set is disconnected.

A *path* is a (non-degenerate) simple open curve, and the *interior* γ° of a path γ is a copy of the path with its endpoints removed. A *cycle* is a (non-degenerate) simple closed curve.

An *x_i-strand* of a simple curve is a minimal path between the x_i-extremes of the curve. That is, an x_i-strand of a simple curve γ with x_i-minimum $a_i = \min_{x \in \gamma} x_i$ and x_i-maximum $b_i = \max_{x \in \gamma} x_i$ is a path $\sigma \subseteq \gamma$ whose endpoints s and t are such that $s_i = a_i$ and $t_i = b_i$, and every internal point $x \in \sigma^\circ$ is such that $x_i \neq a_i, b_i$.

Observation 1 *The interiors of any two distinct x_i-strands of a simple curve are disjoint. Hence any two distinct x_i-strands of a path intersect at most at one common endpoint.*

Observation 2 *If σ is an x_i-strand of a simple curve γ, then $\pi_j(\sigma)$ is an x_i-strand of $\pi_j(\gamma)$, for $j \neq i$.*

If $\pi_j(\gamma)$ is a path, the converse of Observation 2 is also true, as stated in the next lemma.

Lemma 1 *If σ is an x_i-strand of the x_j-projection of a simple curve γ, with $i \neq j$, and $\pi_j(\gamma)$ is a path, then there exists an x_i-strand of γ whose x_j-projection is σ.*

Proof. Let a and b be the endpoints of $\pi_j(\gamma)$, let $a', b' \in \gamma$ such that $\pi_j(a') = a$ and $\pi_j(b') = b$, and let $\gamma' \subseteq \gamma$ be a path with endpoints a' and b'. Since $\pi_j(\gamma)$ is a path, $\pi_j(\gamma) = \pi_j(\gamma')$. Let c and d be the endpoints of σ, such that a and c belong to the same connected component of $\pi_j(\gamma') \setminus \sigma^\circ$. Parameterizing γ' from a' to b', let c' be the last point of γ' such that $\pi_j(c') = c$. Because d separates c and b in $\pi_j(\gamma')$, there are points of γ' after c' whose x_j-projection is d. Letting d' be the first of such points, the sub-path of γ' with endpoints c' and d' is an x_i-strand of γ whose x_j-projection is σ. □

In the following lemma we show that, if two shadows of a non-degenerate cycle are paths, then each of the two shadows has at least two similarly-oriented strands.

Lemma 2 *If γ is a cycle in $\mathbb{R}^3$ that is not contained in any x_1-orthogonal plane, and $\pi_2(\gamma)$ and $\pi_3(\gamma)$ are paths, then $\pi_3(\gamma)$ has at least two distinct x_1-strands.*

Proof. Since γ is not in an x_1-orthogonal plane, γ and $\pi_3(\gamma)$ both have at least one x_1-strand. Let σ be an x_1-strand of γ, let $\tau_2 = \pi_2(\sigma)$, and assume for contradiction that $\pi_3(\gamma)$ has a unique x_1-strand τ_3, as sketched in Figure 4.

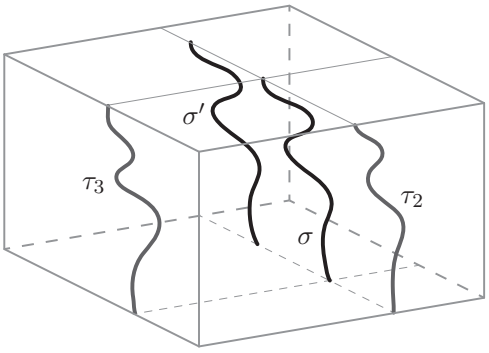


Figure 4: Some x_1-strands of the curve in Lemma 2.

Since τ_2 is an x_1-strand of $\pi_2(\gamma)$ by Observation 2, and the endpoints of a path cannot be in the interior of one of its strands, $\pi_2(\gamma) \setminus \tau_2^\circ$ contains the endpoints of $\pi_2(\gamma)$. Also, the x_2-shadow of $\gamma \setminus \sigma^\circ$ is a superset of $\pi_2(\gamma) \setminus \tau_2^\circ$, and hence it contains the endpoints of $\pi_2(\gamma)$, as well. Moreover, $\gamma \setminus \sigma^\circ$ is connected (because γ is a cycle), hence $\pi_2(\gamma \setminus \sigma^\circ)$ is a connected subset of the path $\pi_2(\gamma)$ containing its endpoints, and therefore it must be all of $\pi_2(\gamma)$. By Lemma 1, γ has an x_1-strand $\sigma' \in \gamma \setminus \sigma^\circ$ such that $\pi_2(\sigma') = \pi_2(\sigma) = \tau_2$. Since $\pi_3(\gamma)$ has a unique x_1-strand τ_3, we have $\pi_3(\sigma') = \pi_3(\sigma) = \tau_3$, again by Observation 2.

Respectively parameterize σ, σ', τ_2, and τ_3 each from the x_1-minimum a_1 to the x_1-maximum b_1 of γ. Choose some value c_1 strictly between these extremes, $a_1 < c_1 < b_1$. Let $s \in \sigma$, $s' \in \sigma'$, $t_2 \in \tau_2$, $t_3 \in \tau_3$ respectively be the first point of each strand where the x_1 coordinate attains the value c_1. With this we have $\pi_2(s) = \pi_2(s') = t_2$ and $\pi_3(s) = \pi_3(s') = t_3$, which implies $s = s'$. So the interiors of the strands σ and σ' intersect, contradicting Observation 1. Thus our assumption must be wrong: $\pi_3(\gamma)$ must have at least two distinct x_1-strands. □

Next we prove that an x_1-strand and an x_2-strand of a planar path must intersect each other, and therefore their union must be a sub-path.

Lemma 3 *If σ_1 and σ_2 are respectively an x_1-strand and an x_2-strand of a path γ in $\mathbb{R}^2$, then $\sigma_1 \cup \sigma_2$ is a path.*

Proof. Let $B = [a_1, b_1] \times [a_2, b_2]$ be the bounding box of γ, and let s_1 and t_1 be the leftmost and rightmost points of σ_1, respectively. Consider the polygonal chain τ with vertices s_1, $(a_1 - 1, a_2 - 1)$, $(b_1 + 1, a_2 - 1)$, t_1, in this order. Then $\sigma_1 \cup \tau$ is a cycle which, by the Jordan Curve Theorem, disconnects the plane into two components: an interior I and an exterior E.

Let s_2 and t_2 be the lowest and highest points of σ_2, respectively. Note that s_2 lies on the bottom edge of B, and hence it lies either in I or on the curve $\sigma_1 \cup \tau$. Similarly, t_2 lies on the top edge of B, and hence it lies either in E or on the curve $\sigma_1 \cup \tau$. Thus, $s_2 \notin E$ and $t_2 \notin I$. It follows that σ_2 must intersect $\mathbb{R}^2 \setminus (I \cup E) = \sigma_1 \cup \tau$. Since $\sigma_2 \subset B$ and $\tau^\circ \cap B = \varnothing$, σ_2 must intersect σ_1.

Thus, $\sigma_1 \cup \sigma_2$ is a connected subset of the path γ, and is therefore a path. □

In our final lemma we show that a planar path cannot have two distinct x_1-strands and two distinct x_2-strands.

Lemma 4 *A path in $\mathbb{R}^2$ has either a unique x_1-strand or a unique x_2-strand.*

Proof. Assume for a contradiction that γ is a path in $\mathbb{R}^2$ with distinct x_1-strands σ_1, σ_2 and distinct x_2-strands τ_1, τ_2. By Observation 1, σ_1 and σ_2 are either disjoint, or their intersection is precisely a common endpoint. Suppose for a contradiction that they are disjoint, and let $\sigma' \subset \gamma$ be the minimal path connecting them. By Lemma 3, $\sigma_1 \cup \tau_1$ is a path, as well as $\sigma_2 \cup \tau_1$, which implies that $\sigma' \subseteq \tau_1$. Similarly, $\sigma' \subseteq \tau_2$, and therefore $\sigma' \subseteq \tau_1 \cap \tau_2$, contradicting Observation 1. Thus $\sigma_1 \cap \sigma_2$ is a single point p, and by a symmetric argument $\tau_1 \cap \tau_2 = p$, as well. Let $B = [a_1, b_1] \times [a_2, b_2]$ be the bounding box of γ. Then p must be a vertex of B, and we may assume that $p = (b_1, b_2)$. Also, by symmetry, we may assume that $\tau_1 \subseteq \sigma_1$. It follows that $\sigma_1 \cap \tau_2 = \sigma_2 \cap \tau_1 = p$, and either $\tau_2 \subseteq \sigma_2$ or $\sigma_2 \subseteq \tau_2$.

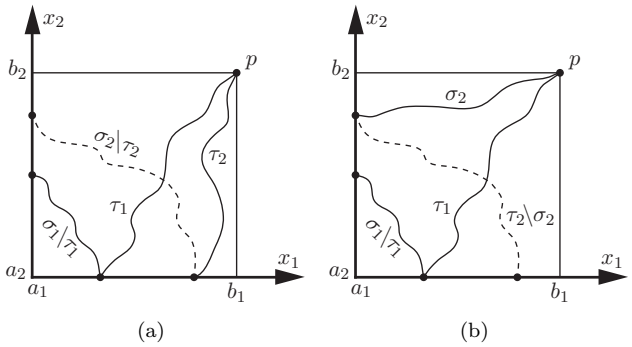


Figure 5: Cases of Lemma 4.

Suppose that $\tau_2 \subseteq \sigma_2$, as in Figure 5(a). Then τ_2° is in the same connected component of $B \setminus \tau_1$ as the edge $\{b_1\} \times [a_2, b_2)$. This implies that $\sigma_2 \setminus \tau_2$ intersects τ_1, contradicting the fact that $\sigma_2 \cap \tau_1 = p$.

Suppose that $\sigma_2 \subseteq \tau_2$, as in Figure 5(b). Then σ_2° is in the same connected component of $B \setminus \sigma_1$ as the edge $[a_1, b_1) \times \{b_2\}$. This implies that $\tau_2 \setminus \sigma_2$ intersects σ_1, contradicting the fact that $\sigma_1 \cap \tau_2 = p$.

Thus our assumption fails: γ has either a unique x_1-strand or a unique x_2-strand. □

We are now able to prove the main result of this section.

Theorem 5 *There is no cycle in $\mathbb{R}^3$ whose shadows are all paths.*

Proof. Assume for a contradiction that the three shadows of a cycle γ in $\mathbb{R}^3$ are all paths. Note that γ cannot lie in any x_i-orthogonal plane, or $\pi_i(\gamma)$ would not be a path. By Lemma 2, since the x_2-shadow and the x_3-shadow are both paths, the x_3-shadow must have at least two distinct x_1-strands. Likewise, since the x_1-shadow and the x_3-shadow are both paths, the x_3-shadow must also have at least two distinct x_2-strands. But by Lemma 4, a path in the (x_1, x_2)-plane cannot have two distinct x_1-strands and two distinct x_2-strands, which is a contradiction. □

3 The shadows of a path

Here we study the simple open curves in $\mathbb{R}^3$ whose shadows are simple closed curves. In contrast with the similarly-defined curves of the previous section, in this case we can construct a wealth of such curves. An example is illustrated in Figure 6.

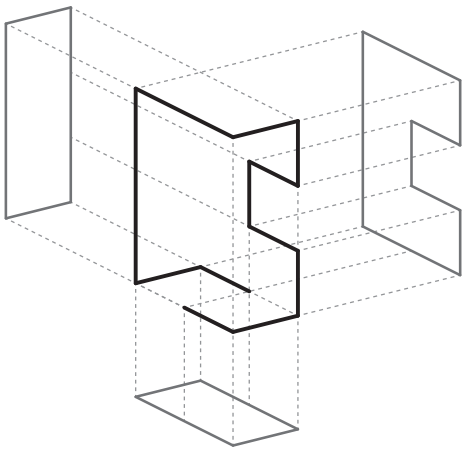


Figure 6: Axis-aligned polygonal path whose shadows are all cycles.

Note that the curve in Figure 6 is a *polygonal path* (i.e., a simple open polygonal chain) consisting of axis-parallel segments. If we allow arbitrarily oriented segments, we can find an example with only six vertices, which is the minimum possible.

Theorem 6 *There exists a polygonal path in $\mathbb{R}^3$ with six vertices whose shadows are cycles. No such polygonal path exists with fewer than six vertices.*

Proof. An example of such a polygonal path is $(1,0,1)\,(0,0,0)\,(1,1,0)\,(0,3,0)\,(2,0,2)\,(1,0,0)$, which is shown in Figure 7.

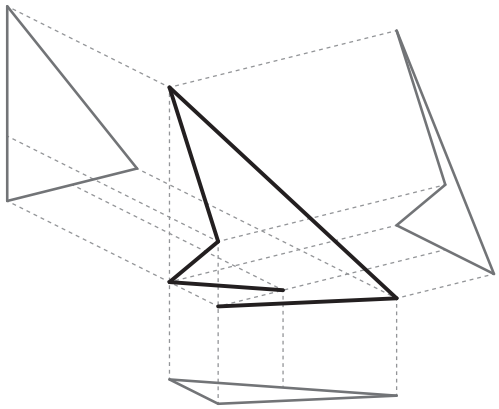


Figure 7: Minimal polygonal path whose shadows are all cycles.

Suppose for a contradiction that a polygonal path in $\mathbb{R}^3$ with $n < 6$ vertices exists such that its shadows are cycles. If $n \leq 3$, then clearly no shadow can be a cycle. Suppose that $n = 4$, and let the polygonal path be $v_1\, v_2\, v_3\, v_4$. Then each shadow must be a triangle, and hence $\pi_i(v_1) = \pi_i(v_4)$ for every $i \in \{1, 2, 3\}$. It follows that $v_1 = v_4$, which contradicts the fact that a polygonal path is an open curve.

Assume now that $n = 5$, and the polygonal path is $v_1\, v_2\, v_3\, v_4\, v_5$. For every $i \in \{1, 2, 3\}$, the x_1-projection of the polygonal path is either a triangle or a quadrilateral. In both cases, the x_i-shadows of the segments $v_1\, v_2$ and $v_4\, v_5$ have a non-empty intersection. Since $v_1 \neq v_5$, the x_i-shadows of v_1 and v_5 do not coincide for at least two i's, say, $i = 1$ and $i = 2$. Then the x_1-shadow of the polygonal path must be a triangle, $\pi_i(v_1\, v_2)$ and $\pi_i(v_4\, v_5)$ are collinear, and hence the segments $v_1\, v_2$ and $v_4\, v_5$ lie on a plane that is orthogonal to the (x_2, x_3)-plane. Similarly, the segments $v_1\, v_2$ and $v_4\, v_5$ lie on a plane that is orthogonal to the (x_1, x_3)-plane, too. Hence $v_1\, v_2$ and $v_4\, v_5$ are either collinear or they lie on a common x_3-orthogonal plane. If $v_1\, v_2$ and $v_4\, v_5$ are collinear (and disjoint), then their x_i-shadows are disjoint for some $i \in \{1, 2, 3\}$, contradicting the fact that their intersection must be non-empty. If $v_1\, v_2$ and $v_4\, v_5$ lie on a common x_3-orthogonal plane, then $\pi_3(v_1\, v_2)$ and $\pi_3(v_4\, v_5)$ are disjoint, which is again a contradiction. □

Note that, in all the above examples, one of the shadows is a non-convex cycle. It is natural to ask whether a

path exists whose shadows are all convex cycles. In the following theorem, we answer in the negative. (Due to space constraints, we only give a sketch of the proof.)

Theorem 7 *There is no path in $\mathbb{R}^3$ whose shadows are convex cycles.*

Proof (sketch). Suppose for contradiction that there exists a path γ in $\mathbb{R}^3$ whose shadows are convex cycles. For every $i \in \{1, 2, 3\}$, γ lies on the surface Γ_i of a cylinder with section $\pi_i(\gamma)$ and x_i-parallel axis.

The intersection of Γ_1 and Γ_2 is sketched in Figure 8. It consists of two horizontal axis-aligned rectangles R_1 and R_2 (assuming that the vertical direction is x_3-parallel) whose vertices are joined by four paths σ_1, σ_2, σ_3, and σ_4. The rectangles R_1 and R_2 may be degenerate, i.e., they may be x_1-parallel or x_2-parallel segments, or points. Let a horizontal plane intersect the interior of the path σ_i in the point s_i, for each $i \in \{1, 2, 3, 4\}$. Then, for all $i \in \{1, 2, 3, 4\}$, either $s_i \in \gamma$ or $s_{i+1} \in \gamma$, where indices are taken modulo 4.

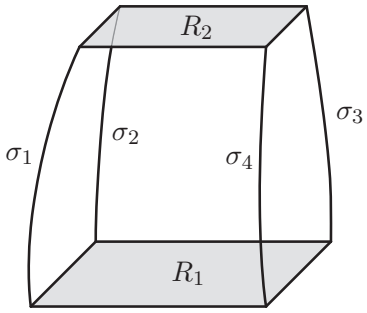


Figure 8: Intersection of Γ_1 and Γ_2.

Further intersecting $\Gamma_1 \cap \Gamma_2$ with Γ_3, we reduce R_1 and R_2 to at most four horizontal curves each. Therefore, in total we have $n \leq 12$ curves, whose union is an embedding in $\mathbb{R}^3$ of a graph G with n edges. Also, we may assume without loss of generality that each endpoint of γ lies at a vertex of the embedding of G, or at the midpoint of one of the n edges. Hence there are only finitely many possible graphs G to consider, and only finitely many choices of γ in each graph embedding. By exhaustively examining all the possible choices of γ, we conclude that none of them has shadows that are all convex cycles. □

4 Shadows in higher dimensions

In this section we generalize Rickard's curve to higher dimensions. We inductively construct an embedding of a d-sphere in $\mathbb{R}^{d+2}$ whose $d + 2$ shadows are all contractible, i.e., they deformation-retract to a point.

An *x_i-slice* of a set $A \subseteq \mathbb{R}^n$, with $1 \leq i \leq n$, is a non-empty intersection between A and an x_i-orthogonal hyperplane.

Theorem 8 *For every $d \geq 1$, there exists an embedding of a d-sphere in $\mathbb{R}^{d+2}$ whose shadows are all contractible.*

Proof. Let S_1 be Rickard's curve, introduced in Section 1. Then, for all $d \geq 1$, we inductively define

$$S_{d+1} = \bigcup_{\lambda \in [-1,1]} (1 - |\lambda|) \cdot S_d \times \{\lambda\}.$$

It is easy to see that S_d is an embedding of a d-sphere in $\mathbb{R}^{d+2}$ for every $d \geq 1$. We claim that all the shadows of S_d deformation-retract to the point $\{0\}^{d+1}$. This is true for $d = 1$, as suggested by Figure 2. Assume now the inductive hypothesis that the claim is true for S_d, and therefore there exists a continuous map

$$F_{d,i} \colon \pi_i(S_d) \times [0, 1] \to \pi_i(S_d)$$

with $F_{d,i}(x, 0) = x$ and $F_{d,i}(x, 1) = \{0\}^{d+1}$, for every $1 \leq i \leq d + 2$. Now, for each $1 \leq i \leq d + 3$, we can construct a continuous map

$$F_{d+1,i} \colon \pi_i(S_{d+1}) \times [0, 1] \to \pi_i(S_{d+1})$$

with $F_{d+1,i}(x, 0) = x$ and $F_{d+1,i}(x, 1) = \{0\}^{d+2}$.

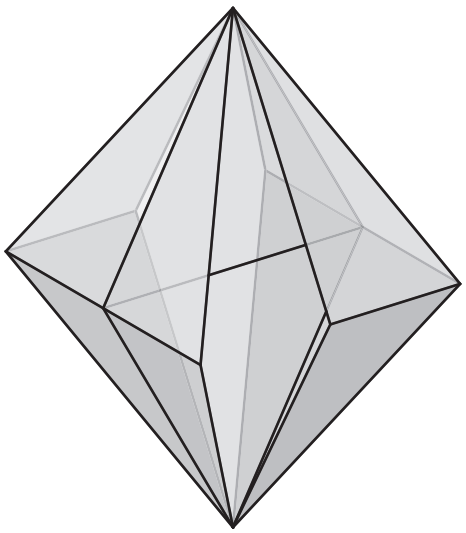


Figure 9: x_i-shadow of S_2, for $1 \leq i \leq 3$.

If $1 \leq i \leq d + 2$, we first define the auxiliary map

$$F' \colon \pi_i(S_{d+1}) \times [0, 1] \to \pi_i(S_{d+1})$$

as follows. For every $x \in \pi_i(S_{d+1})$ such that $|x_{d+2}| \neq 1$ and $\lambda \in [0, 1]$, we let

$$F'(x, \lambda) = (1 - |x_{d+2}|) \cdot F_{d,i}\left(\frac{\pi_{d+2}(x)}{1 - |x_{d+2}|}, \lambda\right) \times \{x_{d+2}\}.$$

If $x \in \pi_i(S_{d+1})$ with $|x_{d+2}| = 1$ and $\lambda \in [0, 1]$, we let $F'(x, \lambda) = x$. Observe that every x_{d+2}-slice of $\pi_i(S_{d+1})$

is a scaled copy of $\pi_i(S_d)$. (Figure 9 shows $\pi_i(S_{d+1})$ for $d = 1$.) Informally, F' applies $F_{d,i}$ with parameter λ to a suitably scaled copy of each x_{d+2}-slice, and then it rescales it back. Therefore, since $F_{d,i}$ is a deformation retraction of $\pi_i(S_d)$ to the point $\{0\}^{d+1}$, F' is a deformation retraction of $\pi_i(S_{d+1})$ to the segment $\{0\}^{d+1} \times [-1, 1]$. To obtain $F_{d+1,i}$, one just has to compose F' with a deformation retraction of $\{0\}^{d+1} \times [-1, 1]$ to the point $\{0\}^{d+2}$. In formulas, for $x \in \pi_i(S_{d+1})$ and $\lambda \in [0, 1]$,

$$F_{d+1,i}(x, \lambda) = \begin{cases} F'(x, 2\lambda) & \text{if } \lambda < 1/2 \\ (2 - 2\lambda) \cdot F'(x, 1) & \text{if } \lambda \geq 1/2. \end{cases}$$

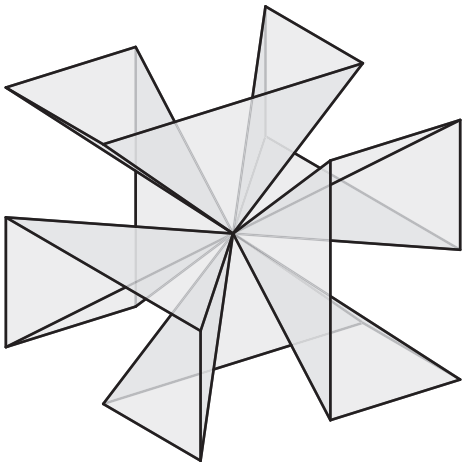


Figure 10: x_4-shadow of S_2.

If $i = d + 3$, we can simply set

$$F_{d+1,i}(x, \lambda) = (1 - \lambda) \cdot x$$

for every $x \in \pi_i(S_{d+1})$ and $\lambda \in [-1, 1]$. This is easily seen to be a deformation retraction to $\{0\}^{d+2}$. (Figure 10 shows $\pi_i(S_{d+1})$ for $d = 1$.)

Hence all the shadows of the d-sphere S_d deformation-retract to a point for every $d \geq 1$, meaning that they are contractible. □

5 Concluding remarks

In this paper we studied the shadows of curves in $\mathbb{R}^3$ (a shadow being an axis-parallel projection), also settling some long-standing open problems posed in [1, 2].

In Section 2 we proved that there is no cycle in $\mathbb{R}^3$ whose shadows are all paths. Note that by applying a projective transformation, we may equivalently define shadows to be perspective projections, provided that the three viewpoints are not collinear, and the plane through them does not intersect the curve.

In Section 3 we proved that there exist paths in $\mathbb{R}^3$ whose shadows are all cycles. We also showed that, if such a path is a polygonal chain, it must have at least six vertices, and we found an example with exactly six vertices. Then we proved that there is no path in $\mathbb{R}^3$ whose shadows are all convex cycles.

Finally, in Section 4 we showed that there exists an embedding of a d-sphere in $\mathbb{R}^{d+2}$ whose shadows are all contractible, for every $d \geq 1$. This generalizes Rickard's curve (see Figure 2), which is a cycle in $\mathbb{R}^3$ whose shadows contain no cycles.

Our results can be expanded in several directions. A natural goal would be to minimize the total number of branch points of the shadows of a cycle in $\mathbb{R}^3$, assuming that all shadows are cycle-free. Because each shadow of Rickard's curve has two branch points, such a minimum is at most six. On the other hand, by Theorem 5, the minimum is at least one. With the same proof technique employed in Section 2, we can prove the following generalized version of Theorem 5, which implies that the minimum number of branch points of the shadows must be at least three.

Theorem 9 *There is no cycle γ in $\mathbb{R}^3$ with cycle-free shadows such that $\pi_1(\gamma)$ is a path, and $\pi_2(\gamma)$ and $\pi_3(\gamma)$ have at most one branch point each.* □

We conjecture Rickard's curve to be an optimal example in terms of branch points of its shadows.

Conjecture 1 *If the shadows of a cycle in $\mathbb{R}^3$ are all cycle-free, then each shadow has at least two branch points.*

Acknowledgments. The authors are indebted to Giuseppe Antonio Di Luna, Pat Morin, and Joseph O'Rourke for stimulating discussions.

Prosenjit Bose and Jean-Lou De Carufel were supported in part by NSERC.

Michael G. Dobbins was supported by NRF grant 2011-0030044 (SRC-GAIA) funded by the government of South Korea.

Heuna Kim was supported by the Deutsche Forschungsgemeinschaft within the research training group "Methods for Discrete Structures" (GRK 1408).

References

[1] Erik D. Demaine and Joseph O'Rourke. Open Problems from CCCG 2007. In *Proceedings of CCCG 2008*, pages 183–186, 2008. `http://cccg.ca/proceedings/2008/paper45.pdf`

[2] Mathematics Research Communities, workshop held at the Snowbird Resort, Utah, USA, 2012. `http://www.ams.org/programs/research-communities/mrc-12`

[3] Adam P. Goucher. Treefoil. *Complex Projective 4-Space*, December 12, 2012. `http://cp4space.wordpress.com/2012/12/12/treefoil/`

[4] Peter Winkler. *Mathematical mind-benders.* A K Peters, Ltd., 2007.

A Fault Tolerant Data Structure for Peer-to-Peer Range Query Processing

Zahra Mirikharaji* Bradford G. Nickerson†

Abstract

We present a fault tolerant dynamic data structure based on a constant-degree Distributed Hash Table (DHT) called FissionE that supports orthogonal range search in d-dimensional space. A publication algorithm, which distributes data objects among all nodes in the network is described, along with a search algorithm that processes range queries and reports all objects in range to the query issuer. Routing messages between two nodes is performed by the FissionE routing algorithm. The worst case orthogonal range search cost in our data structure with n nodes is $O(\log n+m)$ messages plus reporting cost, where m is the minimum number of nodes intersecting the query. Storing d complete copies of each data object on d different nodes provides redundancy for our scheme. This redundancy permits completely answering a query in the case of simultaneous failure of $d-1$ nodes.

1 Introduction

In structured peer-to-peer networks, the P2P overlay network topology is tightly controlled to place contents at specified locations that will make data discovery more efficient. Many structured P2P systems like Chord [15], Tapestry [19], Pastry [11], CAN [9] and FissionE [8] use a DHT [10] to distribute data objects deterministically among the peers and retrieve them with the data object's unique key. DHT-based systems employ hashing to assign IDs to the peers; each peer is responsible for a small specific subset of the data. The number of required messages exchanged between nodes to answer a query defines search cost in these networks.

DHT schemes are normally capable of processing exact match searches, but not more complex searches such as range search . A number of recent papers have investigated DHTs to process range queries. DHT-based techniques for range query answering are classified into two groups [18]; layered indexing and customized indexing. In layered indexing techniques, the underlying topology and routing algorithm of DHTs are used to answer range queries. Our work is in the layered category. Customized indexing uses a custom-designed P2P overlay or modifies an existing P2P overlay network to support range search.

In layered indexing, Gupta et al. [5] use a probabilistic scheme that relies on locality sensitive hashing. However, this method can only report approximate answers for one dimensional range queries. Squid [12] and DCF-CAN [1] use space-filling curves (SFC) to map multi-dimensional keys to the peers. Space-filling curves are locality preserving, but they lead to a less efficient search cost, because a single range query may cover several parts of the curve, each of which requires a separate query. In customized indexing, the skip graph [2] and SkipNet [6] are P2P networks having $O(\log n)$ degree that support one dimensional orthogonal indexing. Family trees [17] and the rainbow skip graph [4] are both constant-degree and support one dimensional range queries. Mercury [3], Znet [13] and MIDAS [16] provide indexing schemes for multi-dimensional space. Mercury [3] provides multiple attribute range queries by indexing the data set along each attribute. The latency of the message routing algorithm in Mercury [3] is $\log^2 \frac{n}{k}$ when each node maintains k links to other nodes. MIDAS [16] resolves the request in $O(\log n)$ hops when each peer's degree is $O(\log n)$. In Znet [13], SFCs (Space Filling Curves) are used and skip graphs [2] are extended for query routing, with each node maintaining $O(\log n)$ states.

Most distributed indexing structures supporting range search don't work on a constant-degree graph. Among the existing constant-degree schemes supporting range search, Armada [7] provides a higher efficiency in terms of query delay and number of required messages. It has been proven in [7] that the lower bound on the message cost of general range query schemes on constant-degree distributed hash tables (DHTs) is $\Omega(\log n)+m-1$, where m is the number of nodes intersecting queries. Armada uses the FissionE P2P network topology DHT scheme based on Kautz graphs [8]. Li et al. [7] have proven that the average message cost of one dimensional queries on uniformly distributed data in PIRA (PrunIng Routing Algorithm) is close to the lower bound on message cost of range queries on constant-degree DHTs. For multi-dimensional indexing, Li et al. [7] have not presented any guarantee on the number of messages required to answer a d-dimensional orthogonal range query. They used simulation to show that the average message cost of MIRA (Multiple attribute prunIng Routing Algorithm) is about $\log n + 4m - 1$ messages.

*Faculty of Computer Science, University of New Brunswick, zahra.miri@unb.ca

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

Armada uses the failure recovery mechanisms of the underlying DHT structure of FissionE [8] to accommodate routing recovery, but they don't provide data recovery. Our work improves on Armada's MIRA algorithm by efficiently providing support for orthogonal range search even with simultaneous failure of up to $d-1$ nodes in a network of n nodes.

2 Results

Our paper presents a peer-to-peer distributed dynamic data structure employing FissionE [8] as a constant-degree DHT to route the messages. We give a data publication algorithm to assign d copies of each object to d different nodes. An orthogonal rage search algorithm for each node in an n-node peer-to-peer network is given that can answer d-dimensional range queries Q issued from any network node. The worst case cost for a d-dimensional range search on our data structure with n nodes is $O(\log n + m)$ messages, for m the minimum number of nodes intersecting the query. To support dynamic joining and departure of nodes and failure recovery, we use split large and merge small policies [8]. To the best of our knowledge, our data structure is the first distributed dynamic spatial data structure to fully support orthogonal range search with simultaneous failure of $d-1$ nodes.

3 Data Structure

3.1 Introduction to FissionE

FissionE [8] is a constant-degree distributed hash table based on the Kautz graph. A Kautz graph is a directed graph with static topology that uses Kautz strings as node identifiers. In the following, we present related definitions explaining Kautz graph topology on which the FissionE DHT is built.

The string $u_1u_2...u_k$ of length k and base d is a Kautz string where $u_i \in \{0,1,2,...,d\}$ and $u_i \neq u_{i+1}$ $(1 \leq i \leq k-1)$. All Kautz strings of length k and base d create the $KautzSpace(d,k)$ of size d^k+d^{k-1}. To show the size of $KautzSpace(d,k)$, we know that the first symbol in a Kautz string has $d+1$ possibilities. Two consecutive symbols in a Kautz string must be different, so all other symbols have d possibilities.

The Kautz graph $K(d,k)$ is a directed graph of degree d with $d^k + d^{k-1}$ nodes labelled by strings in $KautzSpace(d,k)$. Each node $U = u_1u_2...u_k$ of a Kautz graph has the same out-degree and in-degree d. There is an outgoing edge from U to V if and only if $V = u_2u_3...u_k\alpha$ where $\alpha \in \{0,1,...,d\}$ and $\alpha \neq u_k$. Figure 1 shows Kautz graph $K(2,3)$ with out-degree 2 and 12 nodes.

The Kautz graph has desirable properties like optimal diameter that are important in peer-to-peer networks.

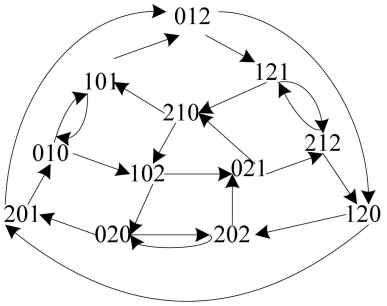


Figure 1: Kautz graph $K(2,3)$ (from [8]).

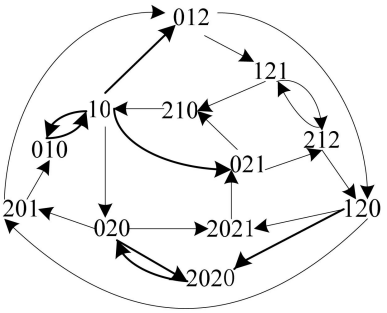


Figure 2: An example of FissionE topology (from [8]).

Diameter is the longest shortest path between any two vertices of a graph and is always in trade-off with the degree of a graph. For a graph with $n = d^k + d^{k-1}$ nodes and degree d, the Kautz graph has the smallest diameter of any possible directed graph. In addition, in Kautz graph $K(d,k)$, there are d disjoint paths between any two nodes and failure of $d-1$ nodes is tolerable.

FissionE uses a $K(2,k)$ Kautz graph. A Kautz graph is a static topology, so it needs some adjustment to be used for dynamic peer-to-peer networks. Li et al. in [8] propose a new topology called approximate Kautz graph. To achieve an approximate Kautz graph, the network topology is first initiated with a Kautz graph, and then in dynamic operations (addition and deletion of nodes) a topological rule called the *neighbourhood invariant* rule is adopted. Based on this rule, the length of identifiers may be different for different peers and the difference in length of node identifiers of any two neighbours must be one or zero. Figure 2 shows an example of neighbourhoods in FissionE topology. This topology is first initiated with Kautz graph $K(2,3)$. Node 202 is split to permit node 2021 to join the network with existing node 202 becoming node 2020. Data in nodes 101 and 102 are merged to provide one less node which results in node 101 being relabelled to node 10, and node 102 departing from the network.

To distribute objects among nodes in the FissionE scheme, the *Kautz_hash* algorithm is proposed in [8]. The *Kautz_hash* algorithm maps an object's unique key

(of any length) to the destination Kautz string of length m consisting of digits $v_i \in 0, 1, 2$, where consecutive digits must be different. Li et al. [8] show that when $m = 100$, the *Kautz_hash* algorithm uniformly distributes the Kautz strings it generates in Kautz namespace $KautzSpace(2, 100)$. As mentioned in Algorithm 1.2.2 of [8], this namespace has size $2^{100}+2^{99} \simeq 1.9\times10^{30}$. To publish an object O in a FissionE topology from node p, the Kautz string V of the object is first computed. Next, node p (the original node) routes the generated Kautz string V to place O in the node whose identifier is a prefix of V. To locate an object in the network, the same process is performed, with node p being the query issuer.

The long path routing algorithm in a Kautz graph is chosen as the Routing algorithm in FissionE. In this algorithm, routing from node U to the node where destination Kautz string V resides is performed by left shifting the symbols of U and adding the symbols of V from left to right at the end of U. For example, if $U = 021$ and $V = 12010$, the longest common prefix of V and suffix of U is equal to 1. So the length of path from node U to the node whose identifier is a prefix of V is 2, and the routing path is $021 \rightarrow 212 \rightarrow 120$.

It is proven in [8] that the average degree of vertices in a FissionE network is 4 and its Kautz graph diameter is less than $2 \log n$. These desirable characteristics motivate us to use FissionE as our overlay network to route messages between nodes and provide dynamic operations of node arrival and departure.

3.2 Data Distribution

DHT-based peer-to-peer networks usually use consistent hashing functions to map data objects and peers to a namespace. In the namespace, each node takes the responsibility of storing values with IDs close to its own ID. In the case of range queries, a peer-to-peer network requires data ordering, so the hash function used to map values into the namespace is replaced by a locality preserving mapping function. Although the FissionE scheme is a high performance distributed peer-to-peer network and achieves optimal diameter on a constant degree graph, it supports only processing of exact match queries (point queries). In this work, we present a general range query scheme that uses FissionE for routing messages. Two main components of our work are the data distribution and the range search algorithms. Our data distribution algorithm publishes d copies of each object on d different nodes in a way that preserves data locality. Our range search algorithm efficiently forwards queries to the appropriate nodes in range. We first give a formal definition of a total order relation, and then explain our data distribution algorithm. To efficiently answer a range query over a peer-to-peer network, it is required to define a total order relation on the dataset to keep the order of data in each dimension. A total order relation is a binary relation on set X denoted by $\leq$ which has the following properties for all a, b and $c \in X$:

1. Antisymmetry: If $a \leq b$ and $b \leq a$ then $a = b$.
2. Transitivity: if $a \leq b$ and $b \leq c$ then $a \leq c$.
3. Totality: $a \leq b$ or $b \leq a$.

A total order relation on data provides propagation of objects on FissionE nodes in such a way that objects with close values are placed on the same or neighbouring nodes. In our data structure, we define a total order relation for each dimension i as follows:
For two points $P(p_1, p_2, ..., p_d)$ and $Q(q_1, q_2, ..., q_d)$, $P \leq Q$ in dimension i if $p_i \leq q_i$.

As explained in section 3.1, in FissionE the identifier of nodes are Kautz strings and network nodes are initiated to a Kautz graph. All Kautz graphs have a Hamiltonian path. A Hamiltonian path in a graph is a path that visits each node of a graph exactly one time. In our work, we use the Hamiltonian path in the underlying Kautz graph of FissionE, and assign the index of each node in the Hamiltonian path as the key to each node.

To preserve data locality along all dimensions, we distribute data objects among nodes by partitioning the space based on point coordinates. In d-dimensional space, we assign d sets of points to each node i on the network, each set corresponding to one dimension. Set S_{ji} is the data stored on node i based on the total order relation in dimension j. For example, in 2-dimensional space, if we denote dimension 0 with x coordinates, and dimension 1 with y coordinates, we assign two sets of points S_{xi} and S_{yi} to every FissionE node i.

The distribution of points based on each dimension over n nodes is a noncrossing partition $NC(\mathcal{S}) = \{S_{j1}, S_{j2}, ..., S_{jn}\}$ [14]. A partition over set $\mathcal{S}$ on dimension j has the following properties:

- The union of the sets of $NC(\mathcal{S}) = \{S_{j1}, S_{j2}, ..., S_{jn}\}$ is equal to $\mathcal{S}$. The elements of $NC(\mathcal{S})$ are said to cover $\mathcal{S}$; i.e. for any j, $0 \leq j \leq d-1$, $\cup_{i=1}^{n} S_{ji} = \mathcal{S}$ where d is the number of dimensions in the data structure.
- The intersection of any two distinct sets of $NC(\mathcal{S})$ is empty; i.e. the elements of $NC(\mathcal{S})$ are pairwise disjoint. Thus $S_{ji} \cap S_{jk} = \emptyset$ if $S_{ji} \in NC(\mathcal{S})$, $S_{jk} \in NC(\mathcal{S})$, $i \neq k$.

In 2-dimensional space, our data structure provides one backup copy of data published on all nodes to achieve search cost near the lower bound, in addition to providing data recovery. A copy of data stored in

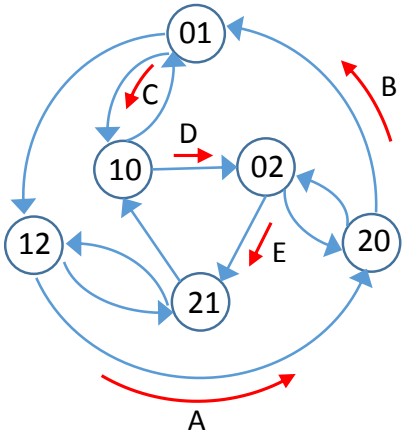


Figure 3: A Hamiltonian path on Kautz graph $K(2,2)$.

node i is stored in all $n-1$ other nodes except node i. Figure 4 shows an overview of the data distribution in a 2-dimensional space over the Kautz graph $K(2,2)$ shown in Figure 3. If the data is distributed in a uniform random fashion in space, a balanced load for each node results. The horizontal colour bar in each cell indicates the place of the first copy of data in that cell based on dimension 0 (X), and the vertical colour bar indicates the place of the second copy of data in that cell based on dimension 1 (Y). For example, assume $L = [0,0]$ and $U = [12,12]$ are the lower and upper bound of the entire 2-dimensional space, and $P = [0.8, 1.2]$ is a point. By uniformly partitioning the space among the six nodes in Figure 3, point P is placed in the lower left cell with red and orange bars. The red and orange bars show that the first and second copies of point P are stored on nodes 12 and 20, respectively.

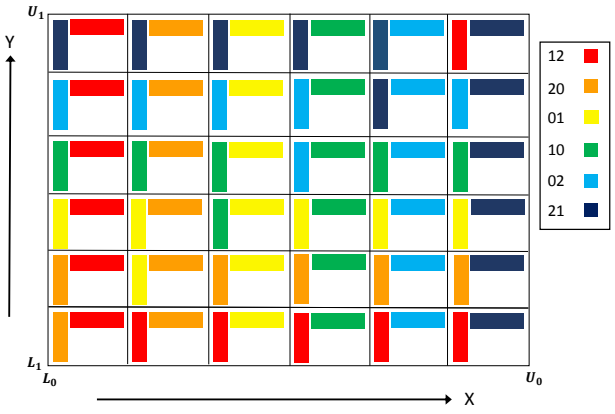


Figure 4: An overview of a 2-dimensional data distribution in our data structure.

Algorithm 1 shows how d-dimensional data objects are distributed on a network of n nodes. This algorithm publishes d copies of object O on d different nodes. The place of the ith copy of object O depends on O_i and the place of the other $i-1$ copies of O that are already specified. $[L_i, U_i]$ in this algorithm is the entire interval of object values in dimension i. When a network is initiated with a Kautz graph, a Hamiltonian path of the graph is found and stored on all nodes. In the case of join and departure of a node, this Hamiltonian path P is updated on all nodes.

Algorithm 1 Publish data object O on a network with n nodes.

1: **procedure** DATADISTRIBUTION(ObjectVal O, NumofNodes n, NumofDims d, LowerBound L, UpperBound U, HamiltonianPath P)
// $O = [O_0, O_1, ..., O_{d-1}]$ is the coordinate of a point that should be published on d nodes.
// $L = [L_0, L_1, ..., L_{d-1}]$ and $U = [U_0, U_1, ..., U_{d-1}]$ are the lower and upper bounds, respectively, of the entire space.
2: **if** $(O < L \| O > H)$ **then**
3: **return** (O is not in range.)
4: **end if**
5: $NodeIndex \leftarrow null$ // $NodeIndex$ is an array of size d showing the indices of nodes that O will be published on.
6: **for** $i \leftarrow 0, d-1$ **do**
7: $node = \lceil \frac{O_i - L_i}{U_i - L_i}(n) \rceil - 1$
8: **for** $j \leftarrow 0, i$ **do**
9: **if** $NodeIndex[j] == node$ **then**
10: $node = (node + 1) \mod n$
11: $j \leftarrow 0$
12: **end if**
13: **end for**
14: $NodeIndex[i] \leftarrow node$
15: **end for**
16: $NodeID \leftarrow null$ // $NodeID$ is an array of size d showing the Kautz string of nodes where O is stored.
17: **for** $i \leftarrow 0, d-1$ **do**
18: $NodeID[i] \leftarrow P[NodeIndex[i]]$
19: **end for**
20: **return**$(NodeID)$
21: **end procedure**

3.3 Range Search

We initially assume that the query Q is searching for one specific point $P(x,y)$ in our data structure. The query can be issued at any one of the nodes. Routing starts at the query issuer node. The query issuer uses Algorithm 1 to find the $NodeID$ list corresponding to point P which contains d distinct addresses of P. The query issuer then determines which of the point's addresses to use. To do that, the longest Kautz string S_i which is a suffix of the query issuer ID and a prefix of $NodeID[i]$ is calculated for each i, $0 \leq i \leq d-1$. Then the query issuer uses the FissionE routing algorithm [8] to pass the query to the $NodeID[j]$, where S_j has the

maximum length among all S_is.

A Hamiltonian path A, B, C, D, E for the Kautz graph $K(2,2)$ is shown in Figure 3. Algorithm 2 answers a d-dimensional range query Q in a network of n nodes. Our range search algorithm has two main parts. First, it determines which dimension of Q intersects the minimum number of nodes. Second, the first node i in range is found and the first portion of data in range is reported. Node i checks if the query upper bound is greater than the node i upper bound; if so then the rest of the search result might be in the next node and an updated query is sent to the next node in range. The new query rectangle is the result of subtracting the query range of node i from the old query rectangle. The same process continues until the last node intersecting the query reports the last part of the result to the query issuer node.

For example, assume that j is the dimension intersecting the fewest nodes. Node i is the first node that sends the result back to the query issuer if $ThisNode_{jL} < Q_{jL} < ThisNode_{jU}$ where $ThisNode_{jL}$ and $ThisNode_{jU}$ are node i's lower bound and upper bound respectively, in dimension j and Q_{jL} is the lower bound of the query in dimension j. If the upper bound Q_{jU} of the query in dimension j is greater than $ThisNode_{jU}$, the query is passed to the next node in range by the call at line 17. The next node on the Hamiltonian path now receives the query, which was updated as shown on line 16.

Theorem 1 *The worst case orthogonal range search cost in our fault tolerant data structure for any data distribution in a d-dimensional space with n nodes is* $O(\log n + m)$ *messages plus reporting cost, where m is the minimum number of nodes intersecting the query on d dimensions.*

4 Fault Tolerance

When failure of one node occurs, problems arise due to an outdated routing table and the fact that the data set assigned to the failed node will be unavailable. To enhance fault tolerance, most distributed data indexing schemes use replication based mechanisms. Data redundancy is part of our distributed spatial data structure as explained in section 3.2. In d-dimensional space, our data structure stores d copies of data in such a way that one copy of each object resides on d different nodes. If one network node fails, we use the involuntary departure of nodes methods in FissionE (Figure 7 in [8]). Each node periodically checks whether its neighbours are alive. When the failure of node k is detected by its neighbour, two neighbour nodes y_1 and y_2 in the network are found which have no neighbour node with less data. They are merged into a new node ℓ and the neighbour lists of ℓ and related nodes are updated. We now have one extra node (y_1 or y_2) to get the $NodeID$ of the failed node k and be responsible for its data. If a query requests data from a failed node, all queries can be processed completely. Our data structure can retrieve data of failed nodes whenever failure of one or more (up to $d-1$) nodes occurs. Thus, our data structure can answer orthogonal range search queries after simultaneous failure of $d-1$ nodes.

Theorem 2 *In our fault tolerant data structure with n nodes storing N points, the cost of recovering network topology and data after failure of one node in d-*

Algorithm 2 Report data objects in a range query to its issuer node.

```
1: procedure IssueQuery(rangeQuery Q)
   // Find the dimension j with minimum range
   Q_jU − Q_jL in query Q
   // L_j and U_j are the lower bound and upper bound,
   respectively, of all possible values for dimension j.
2:     j ← MinRangeDim(Q) // Proper dimension for
   query processing
3:     dstIndex ← ⌈((Q_jL − L_j)/(U_j − L_j))(n)⌉ − 1
4:     dstID ← HamiltonianPath[dstIndex]
5:     FissionERouting(thisNode, dstID, Q, j)
6: end procedure
7: procedure FissionERouting(srcNode src, dstNode
   dst, Rangequery Q, properD j)
   // Assume that src = src_1 src_2 ... src_k and
   dst = dst_1 dst_2 ... dst_m.
8:     SP ←SuffixPrefix(src, dst)
   // SP = SP_1 SP_2 ... SP_t the longest Kautz string that
   is a prefix of dst and a suffix of src.
9:     src.Routing(dst, k − t, SP, Q, j)
10: end procedure
11: procedure U.Routing(dstNode dst, pathLen L,
   sufPre SP, rangeQuery Q, properD j)
12:     if (L = 0) then
13:         if (Q_jL > ThisNode_jL) and (Q_jL <
   ThisNode_jU) then
   // Report all objects in range where O_j < ThisNode_jU.
14:             ReportAnswer(LocalSearch(Q), Q.issuer)
   // If not the last node in range
15:             if (Q_jU > ThisNode_jU) then
16:                 Q_jL ← ThisNode_jU
   // Route updated query to the next node in
   Hamiltonian path
17:
   FissionERouting(thisNode, thisNode.next, Q, j)
18:             end if
19:         end if
20:     else if ∃Q ∈ Outneighbors(U) & Q = U_2...U_k X &
   IsPrefix(SX, dst) then
   // Routing method has been called i times.
21:         S ← SX
22:         Q.Routing(dst, L − 1, S, Q, j)
23:     end if
24: end procedure
```

dimensional space is $O(\frac{dN}{nB}\log n)$ *messages, where* B *is the number of points that fit in one message.*

5 Conclusion

We have designed a dynamic peer-to-peer data structure for d-dimensional data that is capable of processing orthogonal range search on a set of N points. The constant degree FissionE topology was used to coordinate message passing among nodes. The worst case range search cost is $O(\log n + m)$ messages plus reporting cost, where n is the number of nodes in the peer-to-peer network, and m is the minimum number of peers intersecting a query. A failure recovery method was introduced that permits our data structure to support d-dimensional orthogonal range search when up to $d-1$ nodes fail simultaneously. It remains an open question how to provide load balancing of nodes in our data structure if the distribution of data is changed due to dynamic updates.

6 Acknowledgements

The authors would like to acknowledge the support of the Natural Sciences and Engineering Research Council (NSERC) of Canada and the UNB Faculty of Computer Science.

References

[1] A. Andrzejak and Z. Xu. Scalable, efficient range queries for grid information services. In *Peer-to-Peer Computing, 2002.(P2P 2002). Proceedings. Second International Conference on*, pages 33–40. IEEE, 2002.

[2] J. Aspnes and G. Shah. Skip graphs. *ACM Transactions on Algorithms (TALG)*, 3(4):37, 2007.

[3] A. R. Bharambe, M. Agrawal, and S. Seshan. Mercury: supporting scalable multi-attribute range queries. *ACM SIGCOMM Computer Communication Review*, 34(4):353–366, 2004.

[4] M. T. Goodrich, M. J. Nelson, and J. Z. Sun. The rainbow skip graph: a fault-tolerant constant-degree distributed data structure. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 384–393. ACM, 2006.

[5] A. Gupta, D. Agrawal, and A. El Abbadi. Approximate range selection queries in peer-to-peer systems. In *CIDR*, volume 3, pages 141–151, 2003.

[6] N. J. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman. Skipnet: A scalable overlay network with practical locality properties. In *USENIX Symposium on Internet Technologies and Systems*, volume 274. Seattle, WA, USA, 2003.

[7] D. Li, J. Cao, X. Lu, and K. Chan. Efficient range query processing in peer-to-peer systems. *Knowledge and Data Engineering, IEEE Transactions on*, 21(1):78–91, 2009.

[8] D. Li, X. Lu, and J. Wu. Fissione: A scalable constant degree and low congestion dht scheme based on kautz graphs. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, volume 3, pages 1677–1688. IEEE, 2005.

[9] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. *A scalable content-addressable network*, volume 31. ACM, 2001.

[10] S. Ratnasamy, I. Stoica, and S. Shenker. Routing algorithms for dhts: Some open questions. In *Peer-to-peer systems*, pages 45–52. Springer, 2002.

[11] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, pages 329–350. Springer, 2001.

[12] C. Schmidt and M. Parashar. Enabling flexible queries with guarantees in p2p systems. *IEEE Internet Computing*, 8(3):19–26, 2004.

[13] Y. Shu, B. C. Ooi, K.-L. Tan, and A. Zhou. Supporting multi-dimensional range queries in peer-to-peer systems. In *Peer-to-Peer Computing, 2005. P2P 2005. Fifth IEEE International Conference on*, pages 173–180. IEEE, 2005.

[14] R. Simion. Noncrossing partitions. *Discrete Mathematics*, 217(1):367–409, 2000.

[15] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[16] G. Tsatsanifos, D. Sacharidis, and T. Sellis. Midas: multi-attribute indexing for distributed architecture systems. In *Advances in Spatial and Temporal Databases*, pages 168–185. Springer, 2011.

[17] K. C. Zatloukal and N. J. Harvey. Family trees: an ordered dictionary with optimal congestion, locality, degree, and search time. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 308–317. Society for Industrial and Applied Mathematics, 2004.

[18] Y. Zhang, X. Lu, and D. Li. Survey of dht topology construction techniques in virtual computing environments. *Science China Information sciences*, 54(11):2221–2235, 2011.

[19] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *Selected Areas in Communications, IEEE Journal on*, 22(1):41–53, 2004.

Appendix

Proof of Theorem 1

Proof. Since the most efficient dimension j for the issued query is selected at the beginning of the range search algorithm, the worst case search cost occurs when the query is an equal-sided box. Li et al. have proven in [8] that the diameter of FissionE is $O(\log n)$. So, in the worst case, the cost of finding the node containing the lower bound of the orthogonal range query Q_{jL} is $O(\log n)$. After that we need $O(m)$ messages to pass the updated query to the following nodes in range using the current Hamiltonian path to find data objects intersecting the query. Adding the two costs gives the worst case orthogonal range search cost for d-dimensional points distributed on a peer-to-peer network of n nodes as $O(\log n + m)$ messages plus reporting cost. □

Corollary 3 *Assuming B points fit in one message, the cost of reporting K points found in range back to the query issuer node is $O((\frac{K}{B} + m)\log n)$ messages.*

Proof. The cost to report the points in range is $\sum_{i=1}^{m} \lceil \frac{K_i}{B} \rceil O(\log n)$ messages, where K_i is the number of points in range on node i and $\sum_{i=1}^{m} K_i = K$. As $\sum_{i=1}^{m} \lceil \frac{K_i}{B} \rceil O(\log n) \in O((\frac{K}{B} + m)\log n)$, we have the claimed reporting cost. □

Proof of Theorem 2

Proof. It has been proven in [8] that when one node fails, depart messages are propagated less than $\log n$ hops. So, the cost of merging two nodes and maintenance of the overlay network is $O(\log n)$. After that, each node finds which parts of its own data were stored on the failed node, and sends this data to the replacement node. If we assume that B points can fit in one message, the data recovery process requires $O(\frac{dN}{nB}\log n)$ messages since when one node fails, $\frac{dN}{n}$ points residing on the failed node are lost. So, $O(\frac{dN}{nB})$ messages are forwarded at most $O(\log n)$ hops to send back the lost data to the replacement node. The overall cost is thus $O(\log n + \frac{dN}{nB}\log n) = O(\frac{dN}{nB}\log n)$ messages. □

Range Counting with Distinct Constraints

J. Ian Munro* Yakov Nekrich† Sharma V. Thankachan‡

Abstract

In this paper we consider a special case of orthogonal point counting queries, called queries with distinct constraints. A d-dimensional orthogonal query range $Q = [b_1, b_2] \times [b_3, b_4] \times \ldots \times [b_{2d-1}, b_{2d}]$ is a range with r distinct constraints if there are r distinct values among b_1, b_2, ..., b_{2d}. We describe a data structure that supports orthogonal range counting queries with r distinct constraints. We show that the space and query time complexity of such queries depend only on the number of distinct constraints r even if r is much smaller than d. An application of queries with r distinct constraints to persistent range counting is also considered.

1 Introduction

In the orthogonal range counting problem we keep a set S of d-dimensional points in a data structure; for any orthogonal query range $Q = [b_1, b_2] \times [b_3, b_4] \times \ldots \times [b_{2d-1}, b_{2d}]$ we must be able to compute the number of points from S that are inside Q. Henceforth $[s, e]$ denotes a closed interval that contains all real values x satisfying $s \leq x \leq e$; $(-\infty, a]$ (or $[b, +\infty)$) denotes a half-open interval that contains all real values x satisfying $x \leq a$ (resp. $x \geq b$). Two-dimensional orthogonal range counting queries can be supported in $O(\log n / \log\log n)$ using an $O(n)$-space data structure [7]. For $d > 2$, the query time and space usage grow by a factor $O(\log n / \log\log n)$ with every further dimension; thus d-dimensional orthogonal range counting queries can be answered in $O((\log n / \log\log n)^{d-1})$ time by a data structure that needs $O(n(\log n / \log\log n)^{d-2})$ space [7]. In this paper we consider a special case of orthogonal range counting queries that can be supported in less time and using less space. For a query range $Q = [b_1, b_2] \times [b_3, b_4] \times \ldots \times [b_{2d-1}, b_{2d}]$ let r denote the number of distinct values b_i in the multiset $\{ b_1 \ldots, b_{2d} \}$ such that $b_i \neq \pm\infty$. We will say that r is the number of *distinct constraints* of a query Q. A query Q such that $r < d$ will be called a distinct-constraint query. We show that distinct-constraint queries can be answered faster and using less space than the general orthogonal range counting queries.

We describe our data structure in Sections 2 and 3. Potential applications are discussed in Section 4 and Section 5. In Section 4 we describe a data structure that supports persistent range counting queries. In Section 5 we describe data structures for some special cases of the orthogonal color counting problem; our solution for the color counting problem has the same complexity as the best previously known data structure.

2 Stabbing Counting Queries

As a warm-up we describe a folklore data structure for counting one-dimensional intervals that are stabbed by a query point.

Lemma 1 *Suppose that there exists an $s(n)$-space data structure that counts the number of points in a one-dimensional range $(-\infty, a]$ in time $q(n)$. Then there exists a $2s(n)$-space data structure that counts the number of intervals that are stabbed by a query point q in time $2q(n)$.*

Proof: Let S_s be the set that contains the starting points of all intervals and let S_e be the set that contains the endpoints of all intervals. An interval $[s, e]$ is stabbed by a query point q if and only if $s \leq q.x_1 \leq e$.

Let $c_q = |\{ [s, e] \in S \,|\, s \leq q \leq e \}|$, $c^+ = |\{ s \in S_s \,|\, s \leq q \}|$ and $c^- = |\{ e \in S_e \,|\, e < q \}|$. That is, c_q is the answer to a query q, c^+ is the number of intervals with starting point at most q, and c^- is the number of intervals with endpoint before q. If the endpoint of an interval is smaller than q, then its starting point is also smaller than q. If the starting point of an interval s is smaller than q, then either its endpoint is smaller than q or q stabs s. Hence $c_q = c^+ - c^-$. We can thus compute c_q by answering two range counting queries on sets of one-dimensional points. □

3 Counting with Distinct Constraints

In this section we generalize the result of Section 2 to $d > 1$ dimensions. We consider queries that ask for the number of points in a set $\{ p \in S \,|\, p.x_1 \gtrless a_1, p.x_2 \gtrless a_2, \ldots, p.x_d \gtrless a_d \}$ where $\gtrless$ denotes either "greater than or equal" or "smaller than or equal". We show that the complexity of such queries depends only on the number

*Cheriton School of CS, University of Waterloo, Waterloo, Canada. imunro@uwaterloo.ca

†Cheriton School of CS, University of Waterloo, Waterloo, Canada. ynekrich@uwaterloo.ca

‡School of CSE, Georgia Institute of Technology, Atlanta, USA. sharma.thankachan@gatech.edu

of distinct values in the sequence $a_1, \ldots, a_d$ and does not depend on d itself.

Lemma 2 *Suppose that there exists a $(d+1)$-dimensional $s(n)$-space data structure that counts the number of points in a range $(-\infty, a] \times Q_d$, where Q_d is an arbitrary d-dimensional range and a is an arbitrary real value, in time $q(n)$. Then there exists a $(d+2)$-dimensional data structure that uses space $3s(n)$ and counts the number of points in a range $(-\infty, a] \times [a, +\infty) \times Q_d$ in time $3q(n)$.*

Proof: Let $Q_m = (-\infty, a] \times [a, +\infty) \times Q_d$. We define $Q^+ = (-\infty, a] \times (-\infty, +\infty) \times Q_d$, $Q^- = (-\infty, a] \times (-\infty, a] \times Q_d$, and $Q^a = (-\infty, a] \times [a, a] \times Q_d$. Then $Q_m = (Q^+ \setminus Q^-) \cup Q^a$. We keep two $(d+1)$-dimensional sets. The set S^+ contains a point $plus(p) = (p.x_1, p.x_3, \ldots, p.x_{d+2})$ for every point $p = (p.x_1, p.x_2, p.x_3, \ldots, p.x_{d+2})$ in S. Whereas the set S^- contains a point $\max(p) = (p.x'_1, p.x_3, \ldots, p.x_{d+2})$ for every $p \in S$, where the new coordinate x'_1 is defined as $p.x'_1 = \max(p.x_1, p.x_2)$. We also keep a set S^v that contains the point $plus(p)$ for all $p \in S$, such that $p.x_2 = v$. We keep a set S^v for every value v, such that $p.x_2 = v$ for at least one $p \in S$. All auxiliary sets are kept in data structures that support $(d+1)$-dimensional range counting queries. A point $p \in S$ is in Q^+ if and only if $plus(p)$ is in $(-\infty, a] \times Q_d$. A point $p \in S$ is in Q^- if and only if $p.x'_1 = \max(p.x_1, p.x_2) \leq a$ and $(p.x_3, \ldots, p.x_{d+2}) \in Q_d$. Hence p is in Q^- if and only if $\max(p)$ is in $(-\infty, a] \times Q_d$. Finally $p \in S$ is in Q^a if and only if $p \in S^a$ and $plus(p)$ is in $(-\infty, a] \times Q_d$. Hence the numbers of points in Q^+, Q^-, and Q^a can be found by answering range counting queries on S^+, S^- and S^a respectively. Thus a query $(-\infty, a] \times [a, +\infty) \times Q_d$ is answered by answering three $(d+1)$-dimensional counting queries. □

The following Theorem is a direct corollary of Lemma 2 for the case when a $(d+2r)$-dimensional query contains at most $d+r$ distinct constraints.

Theorem 3 *Suppose that there exists a $(d+r)$-dimensional $s(n)$-space data structure that counts the number of points in a range $(-\infty, a_1] \times (-\infty, a_2] \times \ldots \times (-\infty, a_r] \times Q_d$, where Q_d is an arbitrary d-dimensional range and $a_1, \ldots, a_r$ are arbitrary real values, in time $q(n)$. Then there exists a $(d+2r)$-dimensional data structure that uses space $3^r s(n)$ and counts the number of points in a range $(-\infty, a_1] \times [a_1, +\infty) \times (-\infty, a_2] \times [a_2, +\infty) \times \ldots \times (-\infty, a_r] \times [a_r, +\infty) \times Q_d$ in time $3^r q(n)$.*

Proof: Lemma 2 is applied r times. □

We can also generalize our result for the case when the same constraint value occurs more than twice.

Lemma 4 *Suppose that there exists a $(d+1)$-dimensional $s(n)$-space data structure that counts the number of points in a range $(-\infty, a] \times Q_d$, where Q_d is an arbitrary d-dimensional range and a is an arbitrary real value, in time $q(n)$. Then there exists a $(d+d_1+d_2)$-dimensional data structure that uses space $3s(n)$ and counts the number of points in any range $Q = (-\infty, a_1] \times \ldots \times (-\infty, a_{d_1}] \times [a_{d_1+1}, +\infty) \times \ldots \times [a_{d_1+d_2}, +\infty) \times Q_d$, where $a_1 = a_2 = \ldots = a_{d_1+d_2} = a$, in time $3q(n)$.*

Proof: Let S be a set of $(d+d_1+d_2)$-dimensional points. We replace the first d_1 coordinates of each point by their maximum and the following d_2 coordinates by their minimum. The resulting set S_{new} contains a $(d+2)$-dimensional point $p_{new} = (\mu_1, \mu_2, p.x_{d_1+d_2+1}, \ldots, p.x_{d_1+d_2+d})$ for every point $p \in S$ where $\mu_1 = \max(p.x_1, \ldots, p.x_{d_1})$ and $\mu_2 = \min(p.x_{d_1+1}, \ldots, p.x_{d_1+d_2})$. Clearly, p_{new} is in $Q_{new} = (-\infty, a] \times [a, +\infty) \times Q_d$ if and only if the corresponding point p is in Q. By Lemma 2 we can count the number of points in $S_{new} \cap Q_{new}$ in time $3q(n)$ using $3s(n)$ space. □

Theorem 5 *The problem of answering d-dimensional range counting queries with r distinct constraints has the same asymptotic space and query time complexity as the general r-dimensional range counting, when r is constant.*

Proof: For each point $p = (p.x_1, \ldots, p.x_d)$ in S we create a $2d$-dimensional point $\overline{p} = (x_1, x_1, x_2, x_2, \ldots, x_d, x_d)$. That is, $\overline{p}$ contains two copies of each p's coordinate. Let $\overline{S}$ be the set of such points $\overline{p}$. A query $Q = [a_1, b_1] \times [a_2, b_2] \times \ldots \times [a_d, b_d]$ on S is equivalent to a $2d$-dimensional query $[a_1, +\infty) \times (-\infty, b_1] \times [a_2, +\infty) \times (-\infty, b_2] \times \ldots \times [a_d, +\infty) \times (-\infty, b_d]$ on $\overline{S}$. We re-order the coordinates of points in $\overline{S}$ so that half-open intervals with the same constraint value are grouped together and intervals $(-\infty, a]$ precede intervals $[a, +\infty)$ for the same value a. The transformed query is of the form $Q' = (-\infty, a_1] \times (-\infty, a_2] \times \ldots \times (-\infty, a_{2d}]$ and only $Q' = Q_1 \times Q_2 \times \ldots \times Q_r$ where each Q_i is a query range with one distinct constraint: for $1 \leq i \leq r$, $Q_i = (-\infty, a_1] \times \ldots \times (-\infty, a_{f_i}] \times [a_{f_i+1}, +\infty) \times \ldots \times [a_{g_i}, +\infty)$ where $a_j = v_i$ for some v and for all j such that $f_i \leq j \leq g_i$. Lemma 4 is applied r times to query range Q'. In this way the query is reduced to 3^r r-dimensional queries. The total space usage of our data structure is $3^r s(n)$, where $s(n)$ is the space needed by a data structure for r-dimensional counting queries.

□

4 Persistent Counting

Now we turn to applications of our approach. Consider a dynamic set of points S. A data structure on S is called partially persistent if every update (insertion or deletion of a point) creates a new version and queries on any version of the data structure are supported. A partially persistent range counting query (Q, t_q) asks for the number of points $p \in Q \cap S$ that were stored in D at time t. A data structure is called offline partially persistent if the sequence of updates is known in advance (that is, all updates of S are known when the data structure is constructed). We refer to the seminal paper of Driscoll *et al.* [5] and to a survey of Kaplan [9] for an extensive description of persistence.

In this section we describe a general method of designing persistent data structures for counting problems. Let Q_d denote an arbitrary d-dimensional range. Our approach enables us to transform any data structure that answers $(d+1)$-dimensional queries of the form $Q_d \times (-\infty, a]$ into a partially persistent data structure that counts the number of points in Q_d and supports both insertions and deletions. The same method can be also applied to other geometric objects (segments, rectangles etc.) We show that d-dimensional offline partially persistent range counting is equivalent to $(d+1)$-dimensional static orthogonal range counting. For instance, one-dimensional partially persistent counting queries can be answered in $O(\log n / \log\log n)$ time using an $O(n)$ space data structure. We remark that a straightforward application of techniques for obtaining partially persistent data structures from dynamic data structures [5] does not lead to a linear space data structure for one-dimensional persistent range counting: the data structure of Driscoll et al [5] can be used to turn a balanced tree into a persistent data structure. In order to support one-dimensional counting queries, we have to keep information about the number of leaves stored below every tree node. Every insertion or deletion changes this information for $O(\log n)$ nodes. Hence a straightforward algorithm for making a data structure persistent would result in an $O(n \log n)$-space data structure.

Lemma 6 *Suppose that there exists an $s(n)$-space data structure that counts the number of points in a range $Q_d \times (-\infty, a]$, where Q_d is an arbitrary d-dimensional range, in time $q(n)$. Then there exists an offline partially persistent data structure that uses space $3s(n)$ and counts the number of points in a range Q_d in time $3q(n)$.*

Proof: We associate a *lifetime* interval $[t_s(p), t_e(p)]$ with each point p, where $t_s(p)$ and $t_e(p)$ denote the times when p was inserted into S and deleted from S. We associate a point $temp(p) = (t_s(p), t_e(p), p.x_1, \ldots, p.x_d)$ to each $p \in S$. Let $S_{temp} = \{\, temp(p) \mid p \in S \,\}$. Given a query (Q_d, t), we must count points p such that $p \in Q_d$ and $t_s(p) \le t \le t_e(p)$. Counting all points that are in Q_d and are stored in a data structure at time t is equivalent to answering $(d+2)$-dimensional query $(-\infty, t] \times [t, +\infty) \times Q_d$ on S_{temp}. Such queries have at most $d+1$ constraint. By Lemma 2, such queries can be answered in time $3q(n)$ using $3s(n)$ space. □

5 Color Counting

Colored or categorical orthogonal range searching is an important variant of the range searching problem. The set of points S of size n, such that each point is assigned a color, is pre-processed and stored in a data structure. For any rectangular query range Q, we must be able to find some information about colors of points in $S \cap Q$. In the case of color counting queries, we want to compute the number of distinct point colors in $S \cap Q$. In the case of color reporting queries, we want to enumerate all distinct point colors in $S \cap Q$. In this section we describe a data structure for color counting. Our solution, based on counting with distinct constraints, matches the best previously known bounds. Thus we show that distinct-constraint counting provides an alternative solution for this problem.

Color searching problems arise naturally in many database applications when the input data objects are distributed into categories. We may want to enumerate (or count the number of) categories of objects whose attribute values are in a certain range. For instance, suppose that a geographic database contains data about locations. Given a query area, we may be interested in listing (or counting) types of soil in that area. Other applications of this problem include document retrieval [12, 13], computational geometry [10], and VLSI layout [6].

Color range searching problem were studied extensively during the last two decades, see e.g., [8, 6, 3, 4, 2, 12, 10, 14, 15, 11]. In spite of significant efforts, space-efficient data structures (i.e., using $n \log^{O(1)} n$ space) are known only for color reporting in $d \le 3$ dimensions. Space-efficient color counting in $d \ge 2$ dimensions is possible only in some special cases. Thus counting or reporting distinct point colors appears to be significantly harder than counting or reporting all points in an orthogonal range.

Gupta et al. [6] describe a data structure for one-dimensional color counting queries that uses $O(n \log n)$ space and supports queries in $O(\log n)$ time. This result is obtained by reducing one-dimensional color counting to two-dimensional point counting. Using the reduction from [6] and a linear size data structure for point-counting, described by JaJa et al. [7], we can obtain an $O(n)$-space data structure that answers one-dimensional color counting queries in $O(\log n / \log\log n)$ time. Space-efficient data structures for some special

cases of two-dimensional queries are also known. A two-dimensional dominance query is a product of two half-open intervals, e.g., $(-\infty, b] \times (-\infty, h]$. A three-sided query range is a product of a closed interval and a half-open interval, e.g., $[a, b] \times (-\infty, h]$. In [6] the authors describe an $O(n \log n)$-space data structure that supports dominance color counting in $O(\log n)$ time and an $O(n \log^2 n)$-space structure that supports three-sided color counting in $O(\log^2 n)$ time; they also describe an $O(n^2 \log^2 n)$ data structure that answers general queries in 2-D in $O(\log^2 n)$ time. Kaplan et al [10] describe a general method that reduces the problem of counting colors in d-dimensional dominance range to counting d-dimensional rectangles that are stabbed by a point q. The set of rectangles used in [10] consists of $O(n)$ rectangles for $d = 2$ or $d = 3$. Kaplan et al [10] describe an $O(n \log n)$-space data structure that answers two-dimensional dominance color counting queries in $O(\log n)$ time and an $O(n \log^2 n)$ space data structure that answers three-dimensional dominance color counting queries in $O(\log^2 n)$ time.

However if we combine the best currently known data structures for rectangle stabbing counting with the reduction from [10], then both query time and space usage can be reduced. There is a data structure that answers two-dimensional dominance color counting queries in $O(\log n / \log\log n)$ time and uses space $O(n)$. There is also a data structure that answers three-dimensional dominance color counting queries in $O((\log n / \log\log n)^2)$ time and uses space $O(n(\log n / \log\log n)$.

Below we provide an alternative solution for color dominance in two and three dimensions. Although our data structures have the same complexity as previous best solutions, we believe that our alternative solution is also of interest.

Dominance Color Counting in 2-D. In the two-dimensional dominance query (aka 2-sided two-dimensional query), the query range Q is a product of two half-open intervals. We will consider queries $(-\infty, a] \times (-\infty, b]$. A two-dimensional point q dominates a point p if both coordinates of q are not smaller than p, $q.x_1 \geq p.x_1$ and $q.x_2 \geq p.x_2$. The skyline M of a set S consists of all points in S that do not dominate any other point in S. If we arrange the points on a skyline M in increasing order of their first coordinates, then the second coordinates of points in M will form a decreasing sequence. For a point $p \in M$, let $next(p) = p'.x_1$ where p' is the right neighbor of p on M.

Let the set S_c contain all points of color c in S. Let M_c denote the skyline of S_c. The set S_1 contains a three-dimensional point $p_1 = (p.x_1, next(p), p.x_2)$ for each $p \in M_c$ and for all colors c. It was shown in [6] that $Q = (-\infty, a] \times (-\infty, b]$ contains a point of color c if and only if there is exactly one point $p \in M_c$ such that $p.x_1 \leq a$, $next(p) \geq a$, and $p.x_2 \leq b$. See Fig. 1 for an example.

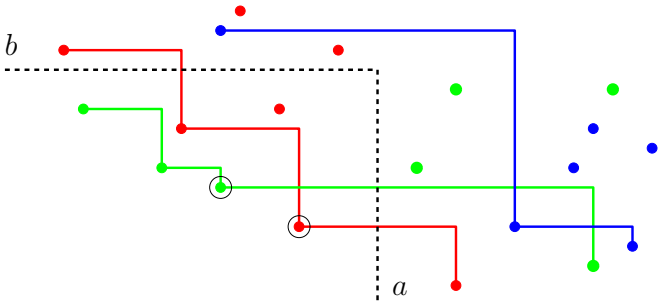


Figure 1: Answering a two-dimensional dominance color counting query on a set of red, blue, and green points. Skyline points are connected by straight lines. For a query $Q = (-\infty, a] \times (-\infty, b]$, we count the number of circled points. Exactly one point for each color that occurs in $(-\infty, a] \times (-\infty, b]$ is considered.

Thus we can count the number of colors in Q by answering a query $(-\infty, a] \times [a, +\infty) \times (-\infty, b]$ with 2 distinct constraints on S_1.

Theorem 7 *Two-dimensional dominance color counting has the same space and query time complexity as two-dimensional point counting.*

Optimal data structures in the RAM and external memory models follow immediately. We plug the data structures from [7] and [1] into Theorem 7.

Corollary 1 *There exists an $O(n)$-space data structure that answers two-dimensional dominance color counting queries in optimal $O(\log n / \log\log n)$ time.*

Corollary 2 *There exists an external-memory data structure that uses $O(n)$ words of space and answers two-dimensional dominance color counting queries in $O(\log_B n)$ I/Os.*

Insertion-Only Dominance Color Counting in 2-D. Let D_1 denote the data structure that supports two-dimensional dominance queries. The data structure D_1 can also support insertions. Suppose that a new point p_{new} of color c is inserted. If p_1 dominates some $p \in M_c$, then we do not have to change M_c and no updates of D_1 are necessary. Otherwise, we insert p_{new} into M_c. In this case we also may have to remove a number of other points from M_c. Data structure D_1 is updated accordingly. An insertion of a single point into M_c can lead to a large number of updates. But Gupta et al. [6] have shown that n insertions into an initially empty data structure require $O(n)$ updates of skylines M_c. Hence D_1 is also updated $O(n)$ times. The key observation is

that each point in inserted and removed from some M_c at most once: if p is removed from M_c, it will not be re-inserted into M_c in the future. We refer to [6] for details.

Dominance Counting in 3-D. Following the approach of [6], we can transform a 2-D dominance query into a 3-D dominance using a persistent version of the two-dimensional data structure described above. While in [6] this technique was applied to range reporting, we use it to obtain a range counting data structure. We sort points of a three-dimensional set S in increasing order by their z-coordinates. These points are then inserted in the same order into a partially persistent variant of the data structure D_1 which we will denote by D_2. We use the approach outlined in Section 4 for adding persistence. Each point in D_2 is associated with two additional coordinates. For every point $p \in D_1$ that was inserted at time t_s and removed at time t_e, D_2 contains a point $p = (p.x, next(p), p.y, t_s, t_e)$. In order to answer a query $(-\infty, a] \times (-\infty, b] \times (-\infty, h]$,we find the version t_h that corresponds to the largest z-coordinate that does not exceed h. Then we count the number of colors in a two-dimensional range $(-\infty, a] \times (-\infty, b]$ at time t_h. That is, we answer a counting query $(-\infty, a] \times [a, +\infty) \times (-\infty, b]$ at time t_h. As shown in Section 4, this is equivalent to answering a query $(-\infty, a] \times [a, +\infty) \times (-\infty, b] \times (-\infty, t_h] \times [t_h, +\infty)$. Although this is a five-dimensional query, it has three distinct constraints. Hence, it has the same complexity as three-dimensional point counting.

Theorem 8 *Three-dimensional dominance color counting has the same space and query time complexity as three-dimensional point counting.*

Again we plug the data structures from [7] and [1] into Theorem 8.

Corollary 3 *There exists an $O(n(\log n/\log\log n))$-space data structure that answers three-dimensional dominance color counting queries in $O((\log n/\log\log n)^2)$ time.*

Corollary 4 *There exists an external-memory data structure that uses $O(n \log_B n)$ words of space and answers three-dimensional dominance color counting queries in $O((\log_B n)^2)$ I/Os.*

References

[1] Pankaj K. Agarwal, Lars Arge, Sathish Govindarajan, Jun Yang, and Ke Yi. Efficient external memory structures for range-aggregate queries. *Comput. Geom.*, 46(3):358–370, 2013.

[2] Pankaj K. Agarwal, Sathish Govindarajan, and S. Muthukrishnan. Range searching in categorical data: Colored range searching on grid. In *Proc. 10th Annual European Symposium on Algorithms (ESA 2002)*, pages 17–28, 2002.

[3] Panayiotis Bozanis, Nectarios Kitsios, Christos Makris, and Athanasios K. Tsakalidis. New upper bounds for generalized intersection searching problems. In *Proc. 22nd International Colloquium on Automata, Languages and Programming (ICALP 95)*, pages 464–474, 1995.

[4] Panayiotis Bozanis, Nectarios Kitsios, Christos Makris, and Athanasios K. Tsakalidis. New results on intersection query problems. *Computer Journal*, 40(1):22–29, 1997.

[5] James R. Driscoll, Neil Sarnak, Daniel Dominic Sleator, and Robert Endre Tarjan. Making data structures persistent. *J. Comput. Syst. Sci.*, 38(1):86–124, 1989.

[6] Prosenjit Gupta, Ravi Janardan, and Michiel H. M. Smid. Further results on generalized intersection searching problems: Counting, reporting, and dynamization. *Journal of Algorithms*, 19(2):282–317, 1995.

[7] Joseph JáJá, Christian Worm Mortensen, and Qingmin Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Proc. 15th International Symposium on Algorithms and Computation (ISAAC 2004)*, pages 558–568, 2004.

[8] Ravi Janardan and Mario A. Lopez. Generalized intersection searching problems. *International Journal of Computational Geometry and Applications*, 3(1):39–69, 1993.

[9] Haim Kaplan. Persistent data structures. In *Handbook on Data Structures and Applications, D. Mehta and S. Sahni (Editors)*, pages 241–246. CRC Press 2001, 2005.

[10] Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. Efficient colored orthogonal range counting. *SIAM J. Comput.*, 38(3):982–1011, 2008.

[11] Marek Karpinski and Yakov Nekrich. Searching for frequent colors in rectangles. In *Proc. 20th Annual Canadian Conference on Computational Geometry (CCCG)*, 2008.

[12] S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 657–666, 2002.

[13] Gonzalo Navarro. Spaces, trees, and colors: The algorithmic landscape of document retrieval on sequences. *ACM Comput. Surv.*, 46(4):52, 2013.

[14] Yakov Nekrich. Efficient range searching for categorical and plain data. *ACM Trans. Database Syst.*, 39(1):9, 2014.

[15] Matthew Skala. Array range queries. In *Space-Efficient Data Structures, Streams, and Algorithms*, pages 333–350, 2013.

Bottleneck Segment Matching

Arita Banik* Matthew J. Katz Marina Simakov*

Department of Computer Science, Ben-Gurion University, Israel
aritrabanik@gmail.com {matya,simakov}@cs.bgu.ac.il

Abstract

Let R be a set of n red segments and B a set of n blue segments, we wish to find the minimum value d^*, such that there exists a perfect matching between R and B with bottleneck d^*, i.e., the maximum distance between a matched red-blue pair is d^*. We first solve the corresponding decision problem: Given R, B and a distance $d > 0$, find a maximum matching between R and B with bottleneck at most d. We begin with the simpler case where $d = 0$ and then extend our solution to the case where $d > 0$. We focus on the settings for which we are able to solve the decision problem efficiently, i.e., in roughly $O(n^{1.5})$ time. The most general of these, is when one of the sets consists of disjoint arbitrary segments and the other of vertical segments. We apply similar ideas to find a matching in the setting in which the vertical segments are replaced by points in the plane.

After solving the decision problem, we explain how to find the minimum value d^*. Finally, we show how to compute a shortest path tree for a given set of n orthogonal segments and a designated root segment in $O(n \log^2 n)$ time.

1 Introduction

The *maximum matching* problem is a fundamental problem in graph theory. Given a graph $G = (V, E)$, find a matching in G of maximum cardinality, where $M \subseteq E$ is a *matching* in G if each vertex of V is adjacent to at most one edge of M. If $|V|$ is even and $|M| = |V|/2$, then M is *perfect*. The maximum matching problem has received a lot of attention, see below for some of the known algorithmic results. When G is a weighted graph, i.e., when each edge of E is assigned some weight, then one often is interested in a *minimum weight matching* or *bottleneck matching* in G, i.e., in a perfect matching that minimizes the sum of the edge weights or the weight of the heaviest edge, respectively. In this context, bipartite graphs received special attention, since many of the motivating problems are naturally modeled using bipartite graphs.

*Work by A. Banik and M. Simakov was partially supported by the Lynn and William Frankel Center for Computer Sciences.

Bottleneck matching and minimum weight matching in geometric graphs, i.e., in graphs induced by geometric settings, are well-studied topics. The most common geometric setting is of course points in the plane. For a set P of points in the plane, the bottleneck matching problem (alternatively, the minimum weight matching problem) is to compute a bottleneck matching (resp., a minimum weight matching) in the complete graph induced by P, where the weight of an edge $e = \{p, q\}$ is the Euclidean distance between points p and q. In the bipartite version of this problem, one is given two sets of points, a red set R and a blue set B, each of size n, and the induced bipartite graph consists of all red-blue edges.

In this paper we study maximum matching problems in bipartite graphs induced by a pair of sets of line segments, i.e., a red set R and a blue set B, each consisting of n line segments. We are not aware of any previous results dealing with these problems. We are mainly interested in the variants for which a maximum matching can be found efficiently, i.e., in time roughly $O(n^{1.5})$.

In the first variant that we study, R is a set of vertical segments and B is a set of arbitrary disjoint segments, and there is an edge between $r \in R$ and $b \in B$ if and only if the two segments intersect. The goal is to compute a maximum matching in this graph. One could do this by applying one of the known graph algorithms for maximum matching in bipartite graphs, however, the running time of these algorithms is in general superquadratic; it is either $O(\sqrt{n}m)$ [4], or $O(n^{2.376})$ [8], or $\tilde{O}(m^{10/7})$ [7], where m is the number of edges in the graph. We present an $O(n^{1.5} \log^2 n)$-time algorithm for this variant.

In the second variant, we are also given a distance $d > 0$, such that there is an edge between $r \in R$ and $b \in B$ if and only if the distance between the two segments is at most d; we denote this graph by $G(R, B, d)$. We would like to compute a maximum matching in this graph. This variant is more difficult than the former, and we present an $O(n^{1.5+\varepsilon})$-time algorithm for it.

When R is also a set of arbitrary disjoint segments, the time bound increases to roughly $O(n^{11/6})$ (which is still subquadratic), for both variants above.

Our algorithms are based on the algorithm of Efrat et al. [2] for computing a maximum matching in a bi-

partite graph induced by a set of n red points and a set of n blue points in the plane, where there is an edge between a red and a blue point if and only if the distance between them is at most d, for a given parameter d. For each of the variants above, we need to replace the "oracle" component in their algorithm with a different oracle that meets our needs, see below for more details.

Using similar ideas, we also obtain an efficient algorithm for the following "mixed" problem. Given a set of n arbitrary disjoint segments and a set of n points, compute a maximum matching in the induced bipartite graph, where there is an edge between a segment and a point if and only if the distance between them is at most d, for a given parameter d. This problem can be viewed as a special case of the second variant above.

The second variant above is actually the decision version of the following optimization problem: Given R and B, find the minimum distance for which there exists a perfect matching in the graph $G(R, B, d)$, or, in other words, find a bottleneck matching in the graph $G(R, B, \infty)$. We show that this optimization problem can be solved within the same time bound, i.e., in $O(n^{1.5+\varepsilon})$ time.

Finally, we present an efficient algorithm for computing a shortest path tree from a designated segment s. Consider a scene consisting of n horizontal and vertical segments and let s be one of the segments. Let G be the bipartite graph induced by the scene (i.e., there is an edge between a horizontal and a vertical segment if and only if they intersect). The distance in G between two segments is the length (in terms of number of edges) of a shortest path between them. We wish to compute a shortest path tree T from s, that is, a tree rooted at s, such that, for any other input segment s', the length of the path in T between s and s' is the distance between them in G. We show how to construct a shortest path tree from s in $O(n \log^2 n)$ time.

2 Solving the decision problem

2.1 The basic segment matching problem

The basic segment matching problem is defined as follows: let R be a set of n vertical segments and B a set of n disjoint arbitrary segments, find a maximum matching between R and B, where two segments may be matched only if they intersect. This is the decision problem of the extended problem for the case $d = 0$.

Consider the corresponding bipartite intersection graph $G = (V, E)$, where each segment in $R \cup B$ corresponds to a vertex in V, and the edge (r, b) between a red vertex r and a blue vertex b is in E if and only if the segments intersect. Computing G explicitly requires $O(n^2)$ time, and the best known graph-theoretic bipartite matching algorithm runs in $O(n^{2.376})$ time [8] (or $O(\sqrt{n}m)$, where $m = |E|$ [4]). We seek a subquadratic solution.

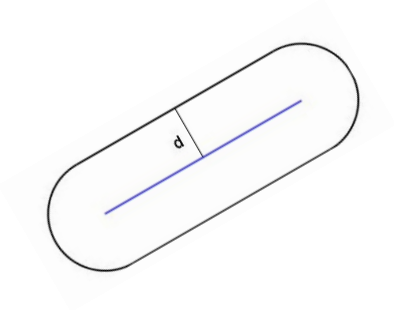


Figure 1: A segment's arena (race track) of radius d.

Theorem 1 *The basic segment matching problem can be solved in time $O(n^{1.5} \log^2 n)$.*

Proof. Efrat et al. [2] show how to find a maximum matching without constructing the entire graph, by implicitly computing augmenting paths until a maximum matching is obtained. Their solution relies on an oracle, which is actually a data structure supporting a certain type of queries and an update operation. They show that if each of these operations (i.e., handling a query and deletion) can be performed within $T(n)$ time, then the maximum matching problem can be solved in $O(n^{1.5} \cdot T(n))$.

In our case, the oracle data structure, $D(S)$, would be a segment tree for a set of segments $S \subseteq B$, which consists of disjoint arbitrary segments. The data structure requires $O(n \log n)$ space (see, e.g., [1]). The required operations are defined as follows:

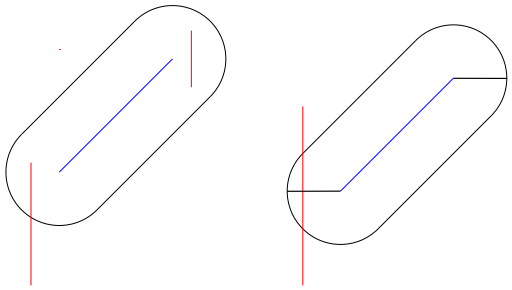
- **match(D(S), q)** — For a query segment $q \in R$, return a segment $s \in S$ such that q intersects s, or *null* if no such segment exists.
- **delete(D(S), s)** — Delete the segment s from S.

Since we are using a segment tree and the the query segments are vertical, each operation can be completed in $T(n) = O(\log^2 n)$ time [1]. Thus, by using the oracle we can compute a maximum matching in $O(n^{1.5} \cdot T(n)) = O(n^{1.5} \log^2 n)$. □

2.2 An extended segment matching problem

After solving the basic matching problem, we consider the case where $d > 0$. We define the extended segment matching problem: let R be a set of n vertical segments and B a set of n disjoint arbitrary segments, find a maximum matching between R and B, where two segments may be matched if the distance between them is at most d.

Given a segment $b \in B$, denote the *arena* (or race track) of radius d induced by it by $A_d(b)$ (see Figure 1).



(a) Case 2 (b) Case 3

Figure 2: Matching segments.

In order to solve the given problem, we require a data structure which supports the following operations:

- **match(D(S), q)** — For a query segment $q \in R$, return a segment $s \in S$, such that $q \cap A_d(s) \neq \emptyset$, or *null* if no such segment exists.
- **delete(D(S), s)** — For a segment $s \in S$, delete s from $D(S)$ to prevent s from being returned again.

If we can solve these queries efficiently, we can obtain an efficient maximum matching algorithm, by applying the scheme of Efrat et al. [2], as we did for the basic matching problem.

Let us distinguish between the different cases in which we can match a vertical segment r to a segment b (notice that there is no restriction on b):

1. The segments r and b intersect.
2. At least one of r's endpoints lies inside $A_d(b)$ (see Figure 2(a)).
3. The most difficult case we have to consider is the one in which the segments do not intersect, and none of r's endpoints lies inside $A_d(b)$. Let us draw two horizontal segments of length d starting at each of b's endpoints and extending away from b (see Figure 2(b)). If r intersects one of the horizontal segments we have added, then it can be matched with segment b.

Lemma 2 *If b and r are at distance at most d from each other, then they satisfy at least one of the conditions described above.*

Proof. Given segments b and r at distance at most d from each other, we will prove that r, b satisfy at least one of the conditions described above. Assume the segments do not satisfy the first and second conditions, we will show that they must satisfy the third one. Note that this means that the segments do not intersect, and none of r's endpoints lies inside $A_d(b)$. Since the distance between r and b is at most d, segment r must intersect $A_d(b)$ in at least one point. Also, since r is a vertical segment which does not intersect segment b, we infer that it must intersect one of the horizontal segments we have added. We conclude that the segments satisfy the third condition. □

We conclude that by detecting each of the three cases, we can determine whether two segments can be matched. We will maintain three data structures, each one allowing the detection of one of the cases. Our goal is to use the oracle once again, so each data structure must support the *match, delete* operations. The following data structures will be required:

1. Data structure D_1: A segment tree for the segments in B, as in the previous section. This data structure allows us to detect segments which satisfy the first condition. *match, delete* operations are performed in $O(\log^2 n)$.
2. Dynamic data structure D_2: A structure for the arenas $A_d(S)$, induced by a set of segments $S \subseteq B$. Let us define the required operations for using the oracle:
   - $match(D_2(A_d(S)), q)$ — Given point q, return a segment s, such that $q \in A_d(s)$. For a vertical segment $r \in R$ we will perform two queries using the segment's endpoints, this way we can match segments which satisfy the second condition.
   - $delete(D_2(A_d(S)), s)$ — Given segment $s \in S$, remove the arena $A_d(s)$ from the data structure. Note that this operation requires D_2 to be a dynamic data structure.

   Lemma 3 *The complexity of the union of the arenas induced by S is linear in $|S|$.*

   Proof. First we observe that any two arenas can intersect in at most two points, this is due to the assumption that the segments $b \in B$ are pairwise disjoint, and that the arenas are all of the same radius. By a result of Kedem et al. [6], we conclude that the complexity of the union of the arenas is linear in $|S|$. □

   Lemma 3 enables us to use the dynamic data structure of Efrat et al. [3], which is able to perform queries in $O(\log^2 n)$ time and deletions in $O(n^\varepsilon)$ time, for any constant $\varepsilon > 0$. Thus, the running time of a single *match, delete* operation in D_2 is bounded by $O(n^\varepsilon)$. The size of the data structure is near linear in $|S|$.
3. Data structure D_3: A segment tree for the horizontal segments of length d, starting at the endpoints of the segments in B and extending away

from them. For each segment $b \in B$, we add two segments, thus the size of the segment tree remains $O(n \log n)$. This data structure allows the detection of segments which satisfy the third condition. Let us define the required operations:

- $match(D_3(S), q)$ — Given a vertical segment q, return a segment $s \in S$ such that q intersects s.
- $delete(D_3(S), s)$ — Delete segment s from S. When deleting s we must also delete its 'twin' segment, so that the corresponding segment in B would not be matched twice.

These operations can be implemented in time $O(\log^2 n)$ per operation.

Now, given a vertical segment $r \in R$, we will conduct at most four queries for each *match* operation of the oracle, and exactly four removals for each *delete* operation of the oracle. The running time of each of the oracle's operations is determined by the maximum running time in the three data structures, which is bounded by $O(n^\varepsilon)$. Thus, using the oracle, we conclude that the extended segment matching problem can be solved in $O(n^{1.5+\varepsilon})$ time. The following theorem summarizes the main result of this section.

Theorem 4 *The extended segment matching problem can be solved in time $O(n^{1.5+\varepsilon})$.*

2.3 Maximum matching between segments and points

Let $d > 0$. An important special case of the extended segment matching problem is the problem of computing a maximum matching between a set of n disjoint arbitrary segments and a set of n points, both in the plane, where we match match a point to a segment if the Euclidean distance between them is at most d.

Theorem 5 *The segments and points matching problem can be solved in time $O(n^{1.5+\varepsilon})$.*

Proof. We observe that the distance between point p and segment s is at most d if and only if p lies inside $A_d(s)$. Thus, this is the only case in which a matching between p and s is valid. By maintaining a single data structure, similar to D_2 in the previous section, we can detect all relevant matchings satisfying this condition. Since each operation is bounded by $O(n^\varepsilon)$, the overall matching problem can be solved in $O(n^{1.5+\varepsilon})$. □

2.4 The general case

The most general setting of the segment matching problem is when both sets consist of disjoint arbitrary segments. This case requires more sophisticated data structures, and balancing between the space allocation and the total time required for query processing in one round of the matching algorithm. This case can be solved in roughly $O(n^{11/6})$ time using $O(n^{4/3})$ space, which is still subquadratic, but significantly worse than our goal of roughly $O(n^{1.5})$ time.

2.5 Optimization

After solving the decision problem, we focus on the optimization problem which is defined as follows: let R be a set of n vertical segments and B a set of n disjoint arbitrary segments, find d^*, which is the smallest value d for which there exists a perfect matching with bottleneck d.

Theorem 6 *The segment matching optimization problem can be solved in time $O(n^{1.5+\varepsilon})$.*

Proof. We perform a binary search in the set of potential values. This set consists of all the distances between a segment in R and a segment in B. Such a distance, if not 0, is determined by the distance between an endpoint of one of the segments and the other segment. The size of the set is $O(n^2)$, so we cannot afford to compute it explicitly. Instead, we slightly adapt the distance selection algorithm of Katz and Sharir [5], so that given k, it returns the k'th smallest distance in roughly $O(n^{4/3})$ time. For each potential value we run the algorithm for the decision problem described in section 2.2, each run requires $O(n^{1.5+\varepsilon})$, and there would be at most $O(\log n)$ potential values examined until d^* is found. Thus, we reach the total running time of $(O(n^{4/3}) + O(n^{1.5+\varepsilon})) \cdot \log n = O(n^{1.5+\varepsilon})$. □

3 Computing a shortest path tree

The shortest path tree problem in our setting is defined as follows: Given S, a set of n orthogonal segments and a segment $s \in S$, compute a shortest path tree T rooted at s; (see Figure 3). We say that there is a *path* from u to v if there exist segments $u = s_1, s_2, \ldots, s_n = v \in S$, such that any two consecutive segments intersect and have different orientations. We show how to construct T efficiently.

Theorem 7 *Given S, a set of n orthogonal segments, and a segment $s \in S$, we can compute a shortest path tree rooted at s in $O(n \log^2 n)$ time.*

Proof. Our algorithm is based on the well-known BFS algorithm. We maintain two segment trees: one for

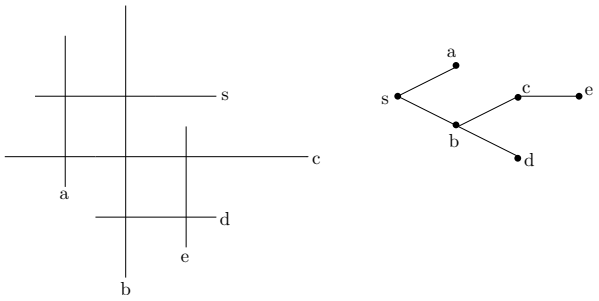


Figure 3: A shortest path tree for a set of orthogonal segments and root segment s.

the horizontal segments, D_1, and one for the vertical segments, D_2. Assume s is a vertical segment, we can find all the segments at distance 1 from s by performing a query in D_1, let us denote all these segments by S_1. In a similar manner, we can find all the segments at distance 2 from s by performing a query in D_2 with each of the segments in S_1. A segment that is found during a query, is deleted from the appropriate data structure in $O(\log^2 n)$ time. We repeat the previous step, until no new intersections are found, implying that our shortest path tree is complete.

Running time: for a given segment t, a query requires $O(\log^2 n + k)$ time, where k is the number of segments (remaining in the data structure) intersecting t. We perform at most n queries for each data structure and each segment is returned at most once, thus overall the construction takes $O(n \log^2 n)$ time. □

Acknowledgments The authors would like to thank Alon Efrat for helpful discussions.

References

[1] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry — Algorithms and Applications*. Springer-Verlag, 3rd edition, 2008.

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

[3] A. Efrat, M. J. Katz, F. Nielsen, and M. Sharir. Dynamic data structures for fat objects and their applications. *Computational Geometry* 15 (2000), 215–227.

[4] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing* 2(4) (1973), 225-231.

[5] M. J. Katz and M. Sharir. An expander-based approach to geometric optimization. *SIAM Journal on Computing* 26(5) (1997), 1384–1408.

[6] K. Kedem, R. Livne, J. Pach, and M. Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discrete Comput. Geom.* 1 (1986), 59–71.

[7] A. Madry. Navigating central path with electrical flows: From flows to matchings, and back. In *Proc. 54th IEEE Symp. Foundations of Computer Science* pp. 253-262, 2013.

[8] M. Mucha and P. Sankowski. Maximum matchings via Gaussian elimination. In *Proc. 45th IEEE Symp. Foundations of Computer Science*, pp. 248-255, 2004.

Reconfiguring a Chain of Cubes*

Laurie Heyer † Anna Lubiw ‡ Debajyoti Mondal § Ulrike Stege ¶ Sue Whitesides§

Abstract

Any configuration of a chain of cubes can be transformed into any other while maintaining contact between adjacent cubes in the chain using a quadratic number of moves. We show that this result is also true in the more constrained setting of a "Kibble chain" where the cubes are threaded on an elastic string, and slits in the cubes allow limited turns at joints.

1 Introduction

This paper is about a puzzle shown in Figure 1 that consists of a sequence of n cubes threaded on an elastic string. The string forces cubes that are adjacent in the sequence to remain in contact, but slits in the cubes allow limited rotations. We show that any configuration of such a puzzle can be transformed to any other using $O(n^2)$ steps. A main tool is to show the same result in a more abstract setting without the elastic string or the slits: to reconfigure a chain of cubes while maintaining contact between adjacent cubes.

More generally, reconfiguring a set of disjoint cubes or modules is of interest both for puzzles and for modular robotics. Different constraints on the reconfiguration process arise in different scenarios. In modular robotics, one well-studied constraint is that the set of modules should remain connected. More precisely, the graph with vertices representing modules and edges representing contact between pairs of modules must remain connected. A common model of allowable motions for cube modules is that one cube may "slide" along another cube [3, 5, 6]. Dumitrescu and Pach [11] showed how to reconfigure a set of squares in this model, and Abel and Kominers [2] extended to cubes in three and higher dimensions. More general "mover" problems were surveyed by Dumitrescu [10]. Hurtado et al. [12] considered distributed algorithms. Alternative "pivoting" motions were considered in [15] and can be physically realized. Reconfiguration of rectilinear chains is also relevant for protein folding [13].

Our setting is more constrained in that an ordering $c_1, c_2, \ldots, c_n$ of the cubes is specified, and each cube must remain in contact with its neighbours in the sequence. We show in Section 3 that any configuration of a chain of n cubes can be straightened using $O(n^2)$ slide motions, and hence that any configuration can be transformed into any other with the same bound. This is an easy generalization of a result known in 2D[1]. The idea (similar to those in [7, 13]) is to pull the chain out at a cube that lies on the boundary.

There are a number of physical puzzles made of cubes strung together in a chain. See Figure 2. In a "snake puzzle" each cube has a hole drilled through it, either straight through or making a right angle turn, and the face contacts between successive cubes are determined by the placement of the holes. A snake puzzle cannot be straightened to have all the cubes in a line and deciding whether one configuration can be transformed to another is of open complexity [1].

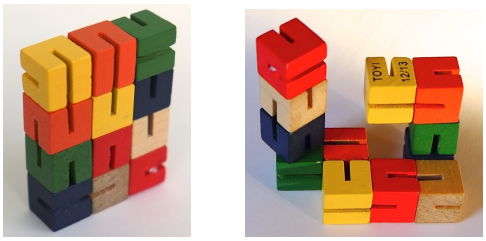


Figure 1: The Wooden Fidget puzzle.

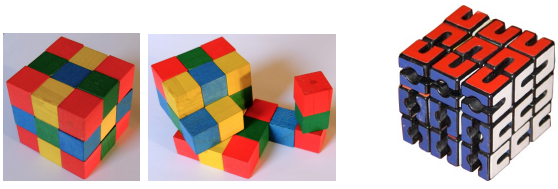


Figure 2: A snake puzzle (left) and a Kibble puzzle (right).

A "Kibble chain" is different from a snake puzzle in that there are slits in each cube that allow the face contacts to change as the elastic cord moves in the slits. It is crucial to have some stretch in the cord, otherwise face contacts are fixed and it becomes a snake puzzle. The commercial Kibble puzzle also involves face colours, but we will not be concerned with colours. A

*INRIA-UVic-McGill 2015 Geometry Workshop at Bellairs
†Davidson College, USA, laheyer@davidson.edu
‡University of Waterloo, Canada, alubiw@uwaterloo.ca
§University of Manitoba, Canada, jyoti@cs.umanitoba.ca
¶University of Victoria, Canada, {ustege,sue}@uvic.ca

[1] Aloupis, Demaine, Lubiw, private communication, from a 2009 Carleton workshop

commercial version of the puzzle we deal with is called "Shapeshifter Creativity blocks" or "Wooden Fidget Puzzle"—see Figure 1. Details of the slits are shown in Figure 3. We show in Section 4 that any configuration of a Kibble chain can be straightened in $O(n^2)$ moves, and hence that any configuration can be transformed into any other. We need rotations in addition to slides. We also show in Section 5 that in some special cases, pivot moves (to be defined) suffice to straighten a chain.

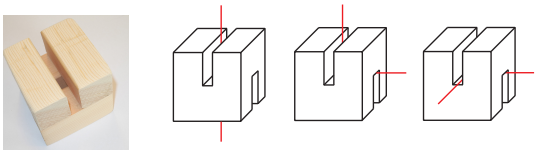


Figure 3: The slits in a single Kibble cube and the ways the string can go through it. (Model and photo: Tom Whitesides.)

Note that we assume we are given the initial and final configurations of the chain. Even in 2D the problem of deciding whether some collection of squares has a chain through it is the known NP-hard problem of finding a Hamiltonian path in a grid graph. For the snake puzzle, even the special case of deciding whether a large cube can be formed from a given snake (ignoring the actual reconfiguration process) is NP-hard [1].

Since we are dealing with a chain, our work is related to linkage reconfiguration [9, 14], and to the version where one joint is straightened at a time [4]. There has also been work on reconfiguring a chain of polygons that must remain connected at fixed points of contact between successive pairs [8]. Our version is different in that two adjacent cubes must remain in contact, but the point of contact may change.

2 Models of Motion

Slides and *pivots* are two basic motions used to reconfigure cubes. The faces of contact between two cubes A and B can be changed with either motion: the face of contact for both A and B can be changed using two slides (see Figure 4); and the face of contact for just one of them can be changed using a pivot (see Figure 5). Both motions can be carried out for some orientations of Kibble cubes.

To define the motions more exactly, we must specify what happens to the rest of the chain. A *slide* translates some cubes that lie along a line of consecutive grid positions (in one of the axis directions) by at most one grid position along the line. All other cubes remain fixed. Of course there must be some clear space for the leading cube to move into, and the constraint of maintaining contact along the chain further restricts the slides

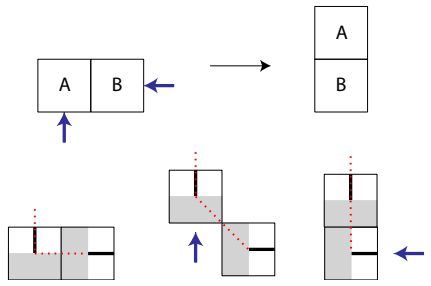


Figure 4: Changing face of contact with slides in the contact model (top)—block A slides up, then block B slides left. These slides can be performed for some orientations of Kibble cubes (bottom).

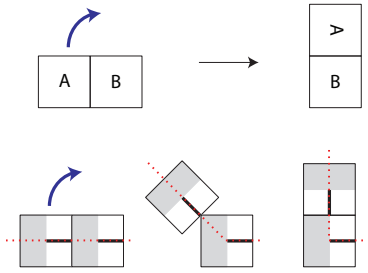


Figure 5: Changing face of contact with a pivot in the contact model (top)—block A pivots around block B. A pivot can be performed for some orientations of Kibble cubes (bottom).

that can be performed. A *rotation* rotates one cube, while all others stay fixed. Rotation may be around an axis through the center or an edge of the rotating cube. Some clear space is needed around a cube before it can be rotated. Finally, we will use the term *pivot* for the operation shown in Figure 5 that translates and rotates a cube and the subchain attached to it. In Figure 6 we show how a chain with one bend can be straightened using one pivot or using a sequence of slides.

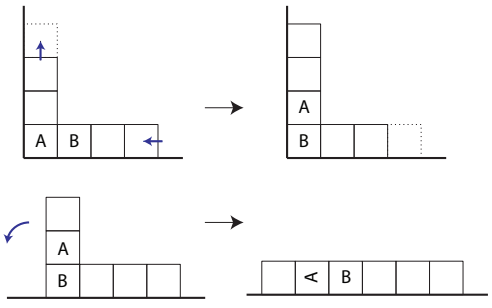


Figure 6: Straightening a chain with one bend using a sequence of slides (top) or a single pivot (bottom).

Note that the pivot requires a lot of free space, but the

sequence of slides only requires that one endpoint of the chain extends to a line in free space. Because of this, we will use pivots only to straighten chains that are initially monotone (Section 5). More generally, we use slides. Slides maintain the axis alignment of each cube. However, it is not possible to straighten every Kibble chain while maintaining axis alignment, for example see Figure 7. We need to use rotations as well. (To straighten this example see Figure 19 in the Appendix.) To rotate a cube about an axis through its center, we must clear some space around it. See Figure 8 for one example of how to do this, and observe that there are alternatives that keep any particular row/column fixed. We also need to clear some space in order to rotate a cube about an axis through an edge. See Figures 14 and 15. We perform these operations so that all required contacts are maintained.

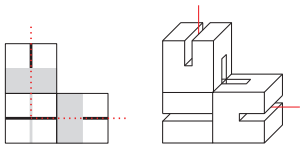


Figure 7: This chain (drawn in schematic and 3D form) cannot be straightened while keeping the cubes aligned with the axes, nor by keeping the cubes in the current plane, nor by a single pivot.

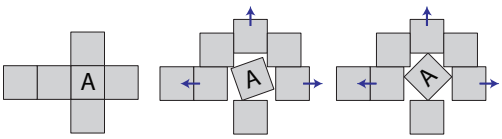


Figure 8: When cube A is connected to a cube in front of the page and to a cube below or behind, A can be rotated after we slide three rows/columns out of the way.

For a Kibble chain we note that sliding a subsequence of the chain increases the length of the string by an additive constant because the two end cubes may change their contact types—when two cubes are in face-to-face contact the length of the piece of string between their centers is 1, but when they are in edge-to-edge contact the length is $\sqrt{2}$. However, sliding a row/column (as may be needed for rotations) may increase the length by a multiplicative constant because a linear number of cube contacts may deviate from full face-to-face contact.

3 Reconfiguring in the Contact Model

In this section we deal with the "contact" model in which adjacent cubes must remain in contact, but we make no further restrictions on their motions. In the initial and final configurations, we assume complete face-to-face contact between cubes that are adjacent in the chain. During reconfiguration, most contacts will still be face-to-face contacts, but we allow edge-to-face or edge-to-edge contact.

The idea is to "pull" the chain out at a cube (or pair of cubes) on the boundary. The rest of the chain moves along the paths to these boundary cubes. We call this the "snake in a tunnel" method. See Figure 9. This idea was used by Cheung et al. [7] who showed how to reconfigure any 3D shape to any other by subdividing it into "micropixels" (spheres) joined in a Hamiltonian path with an endpoint on the boundary, and then pulling the chain out at this endpoint. The same idea was used earlier in work by Lesh et al. [13] for protein folding (with a slightly different set of elementary moves), where it was called a "reptation" method in keeping with a basic principle of polymer science.

Theorem 1 *Any configuration of a chain of n cubes with face-to-face contacts between cubes adjacent in the chain can be transformed to any other such configuration using $O(n^2)$ slides, while maintaining contact between adjacent cubes in the chain.*

Proof. As noted in the Introduction, this proof was already known for 2D. It suffices to show that slides can transform any configuration into a straight chain which then acts as an intermediate "canonical" configuration.

Consider a cube c in the topmost plane of the configuration that is connected to a cube below this plane. (If no such c exists then all the cubes lie on one plane, and we consider the rightmost column instead of the topmost plane.) Either c is an end cube of the chain, or else there is a cube c' that is adjacent to c in the chain and also lies in the topmost plane. If c is an end cube, the idea is to "pull" the chain out at c until it is straight. If c is not an end cube then we pull c and c' out until the two subchains from c and c' to the ends of the chain are straight.

Let $\chi(c)$ and $\chi(c')$ be the disjoint subchains from c and c', respectively, to the ends of the chain. A *phase* moves $\chi(c)$ or $\chi(c')$ along its tunnel by one grid position. If c' exists, the phases alternate between c' and c. At the end of each phase all contacts are face-to-face. Each phase is implemented as a sequence of slides, one for each bend in the subchain. Each slide moves a maximal subchain of collinear cubes (a "line" of cubes) along by one grid position, effectively transferring an empty position from the front of the line to the end of the line. We begin each phase by moving c or c' upwards, along with a maximal vertical subchain below it. This leaves a vacant grid position where the last cube of the subchain was. In general suppose that cube c_i has vacated a position at the front of line ℓ that consists of cubes

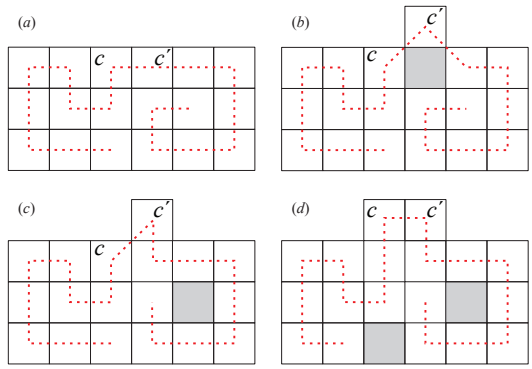


Figure 9: The snake in a tunnel method in 2D: (a) initial configuration; (b) in the first step cube c' moves up (in this example, c' is alone in its line); (c) at the end of the first phase $\chi(c')$ has moved one position further in its tunnel; (d) in the second phase c and its line move up and $\chi(c)$ moves one position further in its tunnel.

$c_{i+1} \ldots, c_j$, $j \geq i$. Cube c_{i+1} is in edge-to-edge contact with c_i. We move the cubes in line ℓ so that c_{i+1} enters the empty position, regaining face-to face-contact with c_i. Since the chain turns at c_j, the contact between c_j and c_{j+1} becomes edge-to-edge.

At the end of $O(n)$ phases $\chi(c)$ will be in one line. If $\chi(c')$ exists, it will also be in one line, and we can do a sequence of $O(n)$ slides to move the two chains into one line. Each phase takes $O(b)$ slides, where b is the number of bends in the chain, so the total number of slides is $O(nb)$ which is in $O(n^2)$. □

4 Reconfiguring a Kibble Chain

The theorem in the previous section does not apply to Kibble chains because motions are restricted by the positions of the slits. Reconfiguration is possible if we allow rotations as well as slides:

Theorem 2 *Any configuration of a Kibble chain of n cubes with face-to-face contacts between cubes adjacent in the chain can be transformed to any other such configuration using $O(n^2)$ rotations/slides, while maintaining contact between adjacent cubes in the chain.*

Proof. We follow the same plan as above, but must rotate cubes in order to re-orient the slits. Recall that a *phase* means moving c (or c') and its subchain along by one position, and each phase is composed of *steps*, where each step involves sliding a line of cubes along by one grid position.

We first claim that before each phase, we can rotate each cube in $\chi(c)$ and $\chi(c')$ (except the initial cubes c and c') so that: (1) the string enters the middle of a slit; and (2) if the string turns 90° in the cube then the string exits at the end of a slit (Figure 3, middle, with the string entering from the top). To justify this, observe that the other orientations in Figure 3 can be rotated to satisfy this.

We now show how to implement each step in the general situation. The first two phases when c and c' move up the first time need some extra care, and we discuss them later. The general situation is that we have an empty grid position at the front of a line ℓ of cubes. Suppose that X is the cube that just vacated the empty position, Y is the first cube of line ℓ, and Z is the cube after Y. Then Z is part of ℓ unless the string turns in Y. After appropriately rotating our frame of reference, the configuration of X and Y is as shown in Figure 10.

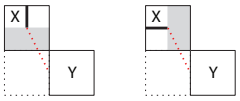


Figure 10: The two proper configurations of cubes X and Y before a step. The dotted position is empty.

For the rest of our argument, we will assume this configuration and speak of "above", "in front", etc. We will implement steps in such a way as to ensure that X is in one of two orientations, as shown in Figure 10. We call these *proper* configurations. In particular, note that if X moved upward via a slide, then we have the first proper configuration.

Because the string enters the middle of a slit of cube Y, there are two possibilities for Y—the string enters a vertical slit or a horizontal slit—see Figure 11. We consider each of them.

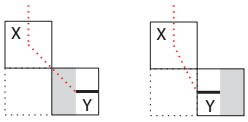


Figure 11: The slit in Y may be vertical (left) or horizontal (right).

Case 1. The string enters Y on a vertical slit. Cube Z can lie in one of three possible positions: (a) in line ℓ; (b) behind Y; (c) in front of Y. See Figure 12. In all cases, we slide the cube(s) of line ℓ along by one position, which moves Y into the empty position. Observe that the resulting configuration is proper in all cases.

Case 2. The string enters Y on a horizontal slit. The next cube, Z, can lie in one of three possible positions as shown in Figure 13. We consider these in turn.

(a) Z lies in line ℓ. Rotate cube Y around the axis of line ℓ. This yields Case 1(a).

(b) Z lies below Y. As shown in Figure 14, rotate cube Y into the empty position maintaining contact with X

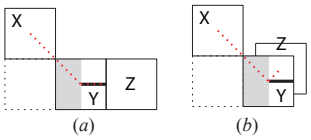


Figure 12: Positions for Z in Cases 1(a) and (b).

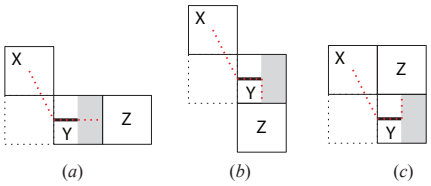


Figure 13: Positions for Z in Case 2.

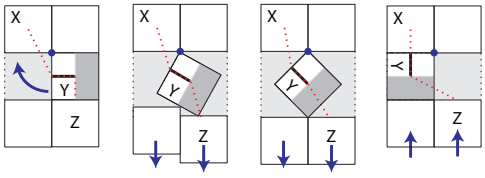


Figure 14: Handling Case 2(b) by rotating Y. Empty space is lightly shaded. Observe that instead of making room for the rotation by pushing the lower cubes downward, we could push the upper ones upward.

and Z. In order to do this, we pull two columns of cubes downward. Observe that the resulting configuration is proper.

(c) Z lies above Y. This "U-turn" is the trickiest case. We may find it necessary to do two steps at once. Note that we cannot have two U-turns in a row.

Rotate cube Y into the empty position as in the previous case (see Figure 15(a)). The resulting configuration is not proper. The bottom of cube Z may have a side-to-side slit or a front-to-back slit. In the first case (see Figure 15(b)), we slide Z (and any cubes in its line) downward, observing that the resulting configuration is proper. In the second case (see Figure 15(c)), the next cube Z' may lie above Z or to the right of Z, but it cannot lie to the left of Z because X is already there. If Z' lies above then rotate cube Z so it has a side-to-side slit, which we just saw how to handle. Finally, if Z' lies to the right of Z then rotate Z into the empty position. Observe that the resulting configuration is proper.

There are two issues outstanding. One is the orientation of c and c' in the initial phases. We rotate them into the orientations shown in Figure 16. In the first phase c' moves up and in the second phase c moves up. At the beginning of each phase we have a proper configuration.

The other issue is that during odd-numbered phases, c

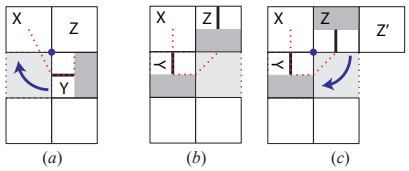


Figure 15: Handling Case 2(c) by rotating Y (a) and then either moving Z down (b) or rotating it (c).

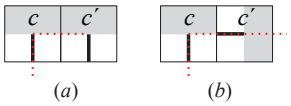


Figure 16: The initial orientations of c and c': (a) if the string turns in c'; (b) if the string goes straight through c'.

and c' will only be in edge-to-edge contact, which poses some difficulties for sliding an axis-parallel "column" of cubes to accommodate rotations. During phase 1 if we want to slide a non-vertical column that contains c, we simply slide c' as well, thus maintaining the edge-to-edge contact. The other difficulty, which may arise in any odd-numbered phase, is sliding the vertical column that contains c' upward. We note that in all cases of rotations we have the freedom to avoid sliding one particular column.

Our method uses $O(nb)$ steps in total, and requires the string to stretch by a small constant factor. □

5 Straightening a Chain with Minimum Moves

The methods in the previous sections used a quadratic number of moves to straighten a chain of cubes. One would hope that a linear number of moves would suffice, although we have not been able to prove this, nor to find a lower bound larger than b, the number of bends in the chain. In this section we consider configurations where b moves or $O(b)$ moves suffice.

The only way to straighten a chain with b moves is to use one pivot per bend. The pivot can happen on either face of the cube where the bend occurs—for example, in Figure 6(bottom) the pivot could instead be done between cube B and its right neighbour. Except for this choice, the straight sections of the chain act as rigid fixed-length segments. There is relevant work by Arkin et al. [4] on straightening a chain of line segments by straightening one joint at a time. They showed that the decision problem is weakly NP-hard even for a 2D rectilinear chain. This does not carry over to our situation because weak NP-hardness is incompatible with representing segments by unit cubes. Is it NP-hard to decide if a chain of cubes can be straightened in b moves?

Arkin et al. also considered the special case where

joints must be opened in order along the chain. It is easy to test if this works because the motion of the chain is completely determined. Similarly, for a chain of cubes we can test in polynomial time if the bends can be straightened in order along the chain via pivots.

We can make the same test for Kibble chains. In this case not every bend can be straightened with one pivot. Consider a bend in a Kibble chain, and change the frame of reference so that the bend is an L-shape as in the figure below. There are six possible orientations of the slits in the central cube: two of them cannot occur in this L-shaped bend; the two *full-slit* orientations shown in Figure 17 are impossible to straighten with one pivot; and the remaining two orientations, and the pivots that straighten them, are shown in Figure 18.

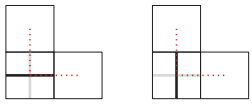


Figure 17: These *full-slit* orientations cannot be straightened with one pivot.

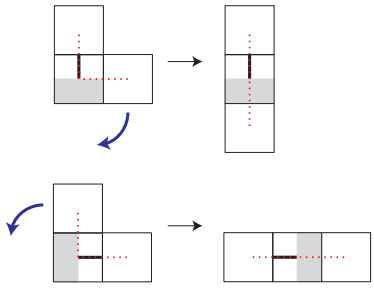


Figure 18: These two orientations of a bend in a Kibble chain can be straightened with one pivot each.

By straightening bends in order along the chain, we can prove that some chains with monotonicity properties can be straightened (details in Appendix):

Lemma 3 *Any 2D or 3D Kibble chain that is monotone in all but one of the axis directions can be straightened with $b + f$ pivots, where b is the number of bends and f is the number of full-slit bends. In the case of a 2D chain with full-slit bends, the height must expand from 1 to $\sqrt{2}$ in the third dimension.*

References

[1] Z. Abel, E. D. Demaine, M. L. Demaine, S. Eisenstat, J. Lynch, and T. B. Schardl. Finding a Hamiltonian path in a cube with specified turns is hard. *Journal of Information Processing* 21(3):368–377, 2013, doi:10.2197/ipsjjip.21.368.

[2] Z. Abel and S. D. Kominers. Universal reconfiguration of (hyper-) cubic robots, http://arxiv.org/abs/0802.3414. arxiv preprint, 2008.

[3] G. Aloupis, N. Benbernou, M. Damian, E. D. Demaine, R. Flatland, J. Iacono, and S. Wuhrer. Efficient reconfiguration of lattice-based modular robots. *Computational Geometry* 46(8):917–928, 2013, doi:10.1016/j.comgeo.2013.03.004.

[4] E. M. Arkin, S. P. Fekete, and J. S. Mitchell. An algorithmic study of manufacturing paperclips and other folded structures. *Computational Geometry* 25(1):117–138, 2003, doi:10.1016/S0925-7721(02)00133-5.

[5] N. M. Benbernou. *Geometric algorithms for reconfigurable structures*. Ph.D. thesis, MIT, 2011.

[6] Z. Butler, K. Kotay, D. Rus, and K. Tomita. Generic decentralized control for lattice-based self-reconfigurable robots. *The International Journal of Robotics Research* 23(9):919–937, 2004, doi:10.1177/0278364904044409.

[7] K. C. Cheung, E. D. Demaine, J. R. Bachrach, and S. Griffith. Programmable assembly with universally foldable strings (moteins). *IEEE Transactions on Robotics* 27(4):718–729, 2011, doi:10.1109/TRO.2011.2132951.

[8] R. Connelly, E. D. Demaine, M. L. Demaine, S. P. Fekete, S. Langerman, J. S. Mitchell, A. Ribó, and G. Rote. Locked and unlocked chains of planar shapes. *Discrete & Computational Geometry* 44(2):439–462, 2010, doi:10.1007/s00454-010-9262-3.

[9] R. Connelly, E. D. Demaine, and G. Rote. Straightening polygonal arcs and convexifying polygonal cycles. *Discrete & Computational Geometry* 30:205–239, 2003, doi:10.1007/s00454-003-0006-7.

[10] A. Dumitrescu. Mover problems. *Thirty Essays on Geometric Graph Theory*, pp. 185–211. Springer, 2013, doi:10.1007/978-1-4614-0110-0_11.

[11] A. Dumitrescu and J. Pach. Pushing squares around. *Graphs and Combinatorics* 22(1):37–50, 2006, doi:10.1007/s00373-005-0640-1.

[12] F. Hurtado, E. Molina, S. Ramaswami, and V. Sacristán. Distributed reconfiguration of 2D lattice-based modular robotic systems. *Autonomous Robots* pp. 1–31, 2015, doi:10.1007/s10514-015-9421-8.

[13] N. Lesh, M. Mitzenmacher, and S. Whitesides. A complete and effective move set for simplified protein folding. *Proceedings International Conference on Research in Computational Molecular Biology*, pp. 188–195, 2003, doi:10.1145/640075.640099.

[14] I. Streinu. A combinatorial approach to planar non-colliding robot arm motion planning. *Symposium on Foundations of Computer Science (FOCS)*, pp. 443–453, 2000, doi:10.1109/SFCS.2000.892132.

[15] C. Sung, J. Bern, J. Romanishin, and D. Rus. Reconfiguration planning for pivoting cube modular robots. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2015. to appear.

Appendix

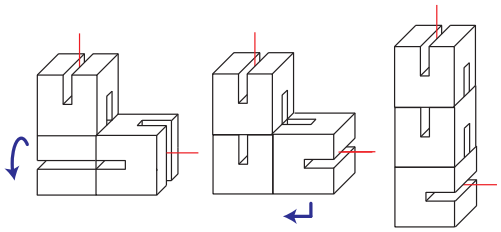


Figure 19: To straighten the chain in Figure 7, rotate the bottom two cubes around the horizontal (side-to-side) axis, and then slide (or pivot) the right-hand cube to the bottom.

Proof. [of Lemma 3] Suppose that the chain is monotone in the x-direction and, in the case of a 3D chain, also monotone in the y-direction. Direct the chain so that a sequence of cubes in a line increases in the x (and y) directions. We will straighten bends one at a time in order along the chain. In the general step, let ℓ be the initial straight portion of the chain (up to the first bend at block b) and let R be the rest of the chain. Ignoring slits for the moment, observe that if we straighten bends one at a time by keeping R fixed and moving ℓ, then R always lies in certain quadrants/octants relative to placing the origin at block b. In 2D, R will lie in the two quadrants where x is positive. In 3D, R will lie in the two octants where x and y are positive.

Bends such as those in Figure 18 can be straightened with one pivot. To deal with a full-slit bend we first rotate the cubes of ℓ around the axis of ℓ. See Figure 19 for an example (where ℓ is horizontal). Note that this motion is actually a pivot (by changing our frame of reference). Observe that this can be done without entering the quadrants/octants that R lies in. This converts the full-slit bend to a standard one, which requires one pivot. □

Folding Polyominoes into (Poly)Cubes*

Oswin Aichholzer† Michael Biro‡ Erik Demaine§ Martin Demaine§ David Eppstein¶
Sándor P. Fekete‖ Adam Hesterberg** Irina Kostitsyna†† Christiane Schmidt‡‡

Abstract

We study the problem of folding a given polyomino S into a polycube C under different folding models, allowing faces of C to be covered multiple times.

1 Introduction

When can a polyomino S be folded into a polycube C? This problem has been considered by Abel et al. [1] and Aloupis et al. [2], but with the restriction that there must be a one-to-one mapping between the unit squares of S and the faces of C. We allow polycube faces to be covered multiple times, only requiring C to be covered by S. We show that different sets of allowed folding angles give distinct variations from each other. We characterize polyominoes that can fold to a single cube, count foldings of polyominoes of different orders into cubes, and investigate the complexity of finding foldings into higher-order polycubes.

2 Notation

A *polyomino* S is a $2D$ polygon formed by a union of $|S| = n$ *unit squares* on the square lattice connected edge-to-edge. Not all edge-to-edge connections of the n unit squares must be used for the polyomino, that is we allow "cuts" on the lattice. A polyomino is a *tree shape* if the dual graph of its unit squares is a tree. A *polycube* C is a connected $3D$ polyhedron formed by a union of *unit cubes* on the cubic lattice connected face-to-face. If C is a rectangular parallelipiped, we refer to its size by its exterior dimensions, e.g., a 2×2×1-polycube.

We study the problem of folding a given S into a given C, allowing *axis-aligned* +90° and +180° mountain folds, −90° and −180° valley folds, folds of *any* degree, *diagonal folds* through opposite corners of a square, and half-grid folds that bisect a unit square in an axis-parallel fashion.

A face of S is an *interior face* of C if it is not flat folded on any of the outer faces of C; see Fig. 3(a) and (b) for examples. A folding model $\mathcal{F}$ specifies a subset of $\mathcal{F} = \{$grid: +90°,−90°, +180°, −180°, any°; interior faces; diagonal; half-grid$\}$ as allowable folds.

3 Folding hierarchy

We say that model $\mathcal{F}_x$ is stronger than $\mathcal{F}_y$ ($\mathcal{F}_x \geq \mathcal{F}_y$) if for all polyomino-polycube pairs $\{S,C\}$ such that S folds into C in $\mathcal{F}_y$, S also folds into C in $\mathcal{F}_x$. If there also exists a pair $\{S',C'\}$ such that S' folds into C' in $\mathcal{F}_x$, but not in $\mathcal{F}_y$, then $\mathcal{F}_x$ is strictly stronger than $\mathcal{F}_y$ ($\mathcal{F}_x > \mathcal{F}_y$). The relation '$\geq$' satisfies the properties of reflexivity, transitivity and antisymmetry, therefore it defines a partial order on the set of folding models. Fig. 1 shows the resulting hierarchy of the folding models that consist of combinations of the following folds: {grid: +90°,−90°, +180°, −180°, any°; interior faces}.

Integrating diagonal and half-grid folds (which are omitted from this section) can result in stronger models: a 1×7 polyomino can be folded into a unit cube C in model {grid: $\pm$ 90°/180°; diagonal}, but not in {grid: any°; interior} (the strongest model from Fig. 1); the example from Fig. 4(b)–(c) shows that $\mathcal{F}_{all} = \{$grid: any°; interior faces; diagonal; half-grid$\}$ is strictly stronger than $\mathcal{F} = \{$grid: any°; interior faces; diagonal$\}$. In addition, Lemma 3 still holds for $\mathcal{F} = \{$grid: $\pm$ 90°/180°; interior faces; diagonal; half-grid$\}$.

The following establishes the relationships between models presented in Fig. 1.

Theorem 1 *The folding models consisting of combinations of the following folds {+90°, −90°, +180°, −180°, arbitrary degree folds, interior folds} have the mutual relations presented in Fig. 1. In particular, these mutual relations hold for polyominoes without holes.*

*This research was performed in part at the 29th Bellairs Winter Workshop on Computational Geometry.

†Institute for Software Technology, Graz University of Technology, oaich@ist.tugraz.at. Partially supported by the ESF EUROCORES programme EuroGIGA – CRP ComPoSe, Austrian Science Fund (FWF): I648-N18.

‡Swarthmore College, mbiro1@swarthmore.edu.

§Massachusetts Institute of Technology, edemaine@mit.edu, mdemaine@mit.edu

¶University of California, Irvine, eppstein@ics.uci.edu. Supported by NSF Grant CCF-1228639 and ONR/MURI Grant No. N00014-08-1-1015.

‖TU Braunschweig, Germany. s.fekete@tu-bs.de

**Massachusetts Institute of Technology, achester@mit.edu

††TU Eindhoven, i.kostitsyna@tue.nl. Supported in part by NWO project no. 612.001.106

‡‡The Hebrew University of Jerusalem, cschmidt@cs.huji.ac.il, Supported by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11).

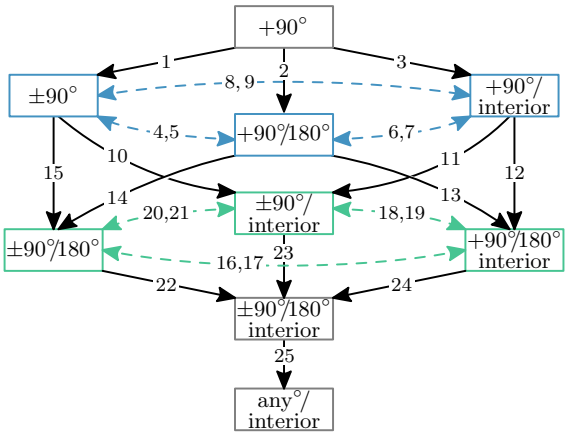


Figure 1: Hierarchy of fold operations. A black arrow from $\mathcal{F}_x$ to $\mathcal{F}_y$ indicates that $\mathcal{F}_y > \mathcal{F}_x$. Blue and green arrows indicate incomparable models.

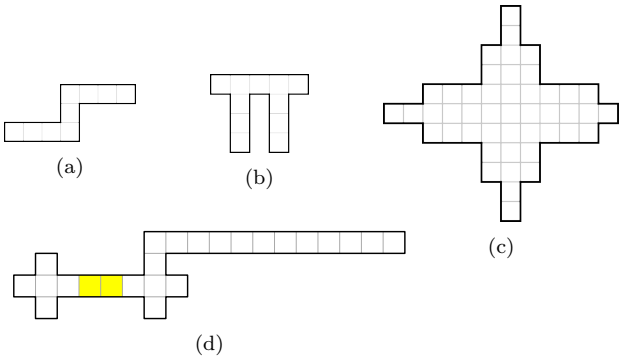


Figure 2: Examples for Thm. 1, where C is (a) a unit cube, (b) a 2×1×1 polycube, (c) a 3×3×2 polycube with a 1-cube hole centered in a 3×3 face, and (d) a 5-cube cross.

Proof. All cases can be shown using the polyomino-polycube pairs from Fig. 2. They fold with: (a) +90°/180° folds, (b) +90° and interior faces (see Lemma 3), (c) ±90°folds, (d) +90°, 180° and interior faces.

Ex. (a) shows relations 2, 4, 7, 12, 15, 19, 20, 23.
Ex. (b) shows relations 3, 6, 9, 10, 13, 16, 21, 22.
Ex. (c) shows relations 1, 5, 8, 11, 14, 17, 18, 24.
Ex. (d) shows relation 25.

We will detail as an example the proof for relation 25; the others are left to the reader. The claim is that $\mathcal{F}_9 = \{$grid: any°; interior faces$\}$ is strictly stronger than $\mathcal{F}_8 = \{$grid: ±90°/180°; interior faces$\}$. Any polyomino S that folds into polycube C in $\mathcal{F}_8$ also folds into C in $\mathcal{F}_9$. To prove a strict relation, we are left to show that there exists some polyomino S' that folds into some polycube C' in $\mathcal{F}_9$, but that does not fold into C' in $\mathcal{F}_8$. Let C' consist of five cubes forming a cross, and let S' be as in Fig. 2(d). Assume that S' can be folded into C' in folding model $\mathcal{F}_8$. $|S'| = 24$, while C' has 22 square faces on its surface. Therefore, 22 out of the 24 squares of S' will be the faces of C' when folded. Consider the 12×1 sub-polyomino of S'. When folded, it has to form the "walls" of the cross C', otherwise this strip can form not more than eight faces of the cross (looping around one of the 3×1×1 sub-polycubes). It is straightforward to see now, that in $\mathcal{F}_8$ the two yellow squares prevent S' from folding into C'. While in $\mathcal{F}_9$ the two yellow squares can form interior faces with a 60° interior fold. Therefore, $\mathcal{F}_8 < \mathcal{F}_9$. □

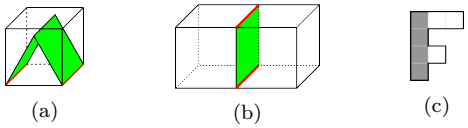


Figure 3: (a)/(b) Examples for interior faces shown in green. They connect to the red cube edges. (c) Shape S from the proof of Lemma 2.

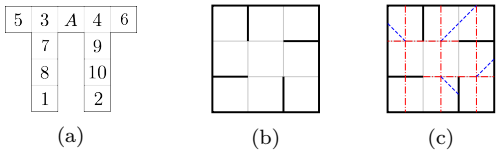


Figure 4: (a) Shape S needs interior faces to be folded into a 2×1×1-cube. (b) Shape S does not fold to a unit cube C, if we allow ±90°/180° and diagonal folds. However, if we allow half-grid folds, S does fold into C, the mountain (red) and valley folds (blue) are shown in (c).

Conjecture 1 *Theorem 1 holds for tree shapes.*

Lemma 2 *There exist tree shapes S that need both mountain and valley folds to cover a unit cube.*

Proof. The shape S from Fig. 3(c) does not fold into a unit cube C with only valley folds: the four unit squares in the left column (gray) fold to a ring of size four, and the two flaps can only cover one of the two remaining cube faces. But with both mountain and valley folds S can be folded into C, by using a 180° fold between the first column and the longer flap. □

Lemma 3 *Folding the tree shape S from Fig. 2(b) into a $2 \times 1 \times 1$-cube C with $\mathcal{F} = \{$grid: ±90°/180°; interior faces$\}$ requires interior faces. (Any of the faces A, 8 or 10 from Fig. 4(a) can be the interior face.)*

Proof. We label the faces of S as shown in Fig. 4(a). The case analysis below shows that a folding without interior faces does not exist:

If A covers one of the 1×1 faces of C, face 1 or 2 is needed for the opposite 1×1 face. By symmetry suppose face 1 covers that side. Thus, 3 needs to be folded on top of A. Face 4 folds either on 5 or A. So, we doubled two

faces. But as $|S| = 11$ and C has 10 faces, the remaining faces are not enough.

If 3, A, and 4 all cover parts of 1×2 faces of C, then the row of 5, 3, A, 4, 6 maps to only four 1×2 faces with at least one overlap. But then the 1×1 face adjacent to A cannot be covered by 7, 8, 9, 10, 1 or 2 without doubling on 3 or 4—again more than one overlap.

If A covers part of a 1×2 face of C and one of 3 or 4 (without loss of generality 3) covers the adjacent 1×1 face, then the only way to cover the two squares that are adjacent to A in the polycube but not in the polyomino is to double 4 with A and wrap column 4–9–10–2 around the polycube; but then 6 must also be doubled, again leading to more than one overlapping pair. □

4 Polyominoes that fold into a cube

In this section we characterize all polyominoes that can be folded into a unit cube and all tree shaped polyominoes that are a subset of a 2×n or 3×n strip that can be folded into a unit cube using arbitrary grid folds.

Theorem 4 *Consider a polyomino S of size $|S| = n$ and a unit cube C under a folding model $\mathcal{F} = \{$grid; any°; diagonal; half-grid$\}$, such that each face of C has to be covered by a full unit square of S. Then $n \geq 10$ is the best possible universal bound, i.e., there is a polyomino of size $n = 9$ that* cannot *be folded into C, while all polyominoes with $n \geq 10$* can *be folded.*

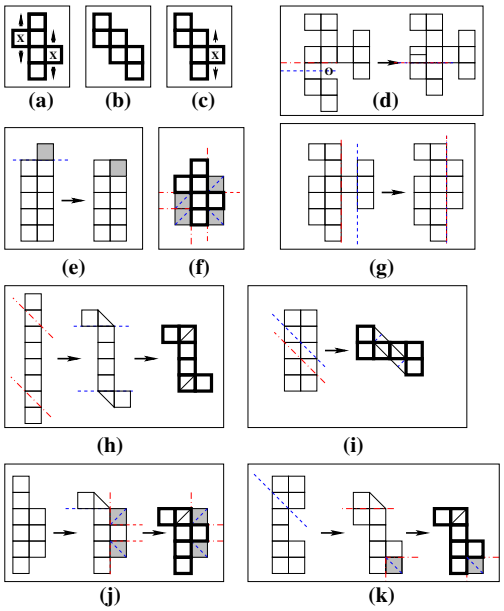


Figure 5: Proof details for Theorem 4. Unit squares marked with "X" can be located at an arbitrary height. For the two adjacent corner reductions in the left of (f), fold away the upper shaded corner before folding away the lower one.

Proof. We consider a bounding box of S of size X×Y, with $X \leq Y$. The unit squares are arranged in *columns* and *rows*, indexed $1,\ldots,X$ and $1,\ldots,Y$, with n_i unit squares S in column i, and m_j unit squares in row j.

For the lower bound see Fig. 4(b). Note that if we allow half-grid folds without requiring faces of C to be covered by full unit squares of S, we *can* turn this shape S into a unit cube; see Fig. 4(c).

For the upper bound $n \geq 10$, we start by identifying several *target polyominoes* shown in Fig. 5(a)–(c). Each can be folded into a cube using only grid folds:

(a) A *1+4+1 polyomino*, composed of one contiguous column of four unit square, with one more unit square on either side, at an arbitrary height.

(b) A *2–2–2 polyomino*, composed of three (vertical) pairs attached in the specific manner shown.

(c) A *2–3+1 polyomino*, composed of a (vertical) pair and triple attached in the specific manner shown, with one more unit square at an arbitrary height.

See Fig. 5(d)–(g) for the following. If n_i is the maximum number of unit squares in any column (say, in i), we can apply a *connectivity reduction* (d) by using (horizontal) half-grid folds to convert S into a polyomino S' in which these n_i unit squares form a contiguous set, while leaving at least one unit square in each previously occupied column. A *number reduction* (e) lets us fold away extra unit squares for $n > 10$. *Corner reductions* (f) fold away unneccessary unit squares (or half-squares) in target shapes by using diagonal folds when turning them into a unit cube. Finally, *width* and *height reductions* (g) fold over whole columns or rows of S onto each other, producing a connected polyomino with a smaller total number of columns.

Now consider a case distinction over X; see Fig. 5(h)–Fig. 6(t). For $X = 1$, the claim is obvious, as we can reduce S to a 1+4+1 target; see Fig. 5(h). For $X = 2$, note that $Y \geq 5$ and assume that $n_1 \geq n_2$. If $n_1 \geq 8$, a width reduction yields the case $X = 1$, so assume $n_1 \leq 7$, and therefore $n_2 \geq 3$. By a number reduction, we can assume $n_2 \leq 5$. If S is a 2×5 polyomino, we can make use of a 1+4+1 polyomino with corner reductions; see Fig. 5(i). If $n_1 > n_2$, we have $n_1 \geq 6$. Because S is connected, any two units squares in column 1 must be connected via column 2, requiring at least three unit squares; because of $n_2 \leq 5$, we conclude that column 1 contains at most two connected components of unit squares. Thus, at most one connectivity reduction makes column 1 connected, with $n_2' \in \{n_2 - 1, n_2\}$ unit squares in column 2. Possibly using height reduction, we get a connected polyomino S'' with vertical size six, six unit squares in column 1, and $n_2'' \in \{1,\ldots,4\}$ unit squares in column 2. For $n_2'' \in \{1,\ldots,3\}$, there is a reduction to target shape 1+4+1; see Fig. 5(j). For $n_2'' = 4$, a similar reduction exists; see Fig. 5(k). This leaves $n_1 = 5$,

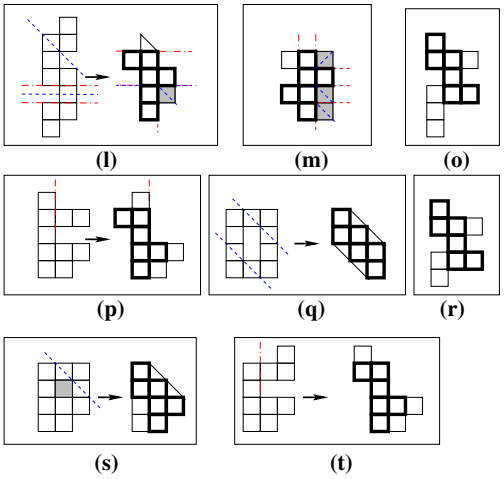


Figure 6: Proof details for Theorem 4.

and thus $n_2 = 5$, without S being a 2×5 polyomino. As shown in Fig. 6(l), this maps to a 1+4+1 target shape, by folding a unit square from column 2 that extends beyond the vertical range of the unit squares in column 1 over to column 0.

For $X = 3$ and $n_2 \geq 4$, we can use connectivity, height and width reductions to obtain a new polyomino S' that has height four, a connected set of $n'_2 = 4$ unit squares in column 2 and $1 \leq n'_1 \leq 4$, as well as $1 \leq n'_3 \leq 4$ unit squares in columns 1 and 3. This easily converts to a 1+4+1 shape, possibly with corner reductions; see Fig. 6(m). Therefore, assume $n_2 \leq 3$ and (w.l.o.g.) $n_1 \geq n_3$, implying $n_1 \geq 4$. If $n_1 \geq 5$ and column 2 is connected, then S contains a 2–3+1 target shape, see Fig. 6(o); if column 2 is disconnected, we can use a vertical fold to flip one unit square from column 3 to column 0, obtaining a 1+4+1 target shape, see Fig. 6(p). As a consequence, we are left with $n_1 = 4$, $2 \leq n_2 \leq 3$, $3 \leq n_3 \leq 4$, $n_2 + n_3 = 6$, possibly after folding away an extra unit square in column 3 in case of $n_2 = 3, n_3 = 4$. If $n_2 = 2$, the unit squares in columns 1 and 3 must be connected. For this it is straightforward to check that we can convert S into a 2–2–2 target polyomino; see Fig. 6(q). Therefore, consider $n_2 = 3$, $n_3 = 3$. This implies that column 1 contains at most two connected sets of unit squares. If there are two, then the unit squares in column 2 must be connected, implying that we can convert S into a 2–3+1 target shape; see Fig. 6(r). Thus, the four unit squares in column 1 must be connected. If the three unit squares in column 1 are connected, we get a 2–3+1 target shape; see Fig. 6(r). If the unit squares in column 2 are disconnected, but connected by the three unit squares in column 3, we convert this to a 2–2–2 target shape; see Fig. 6(s). This leaves the scenario in which there is a single unit square in column 1 whose removal disconnects the shape; for this we can flip one unit square from column 3 to column 0 in order to create a 1+4+1 target shape; see Fig. 6(t).

For $X \geq 4$, we proceed along similar lines. If there is a row or column that contains four unit squares, we can create a 1+4+1; otherwise, a row or column with three unit squares allows generating a 2–3+1. If there is no such row or column, we immediately get a 2–2–2. □

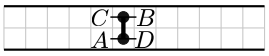


Figure 7: A vertical edge in a subset of a 2×n strip with possible adjacent vertices for subtrees.

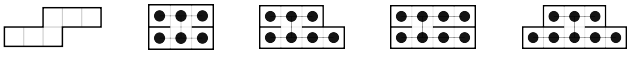


Figure 8: (a) A shape S that folds into a unit cube. (b)–(e) Shapes S, only (d) folds into a unit cube.

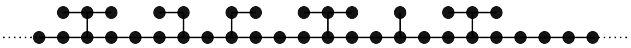


Figure 9: Infinite families that cannot fold into a unit cube.

Theorem 5 *Given a tree shape S, a unit cube C and $\mathcal{F} = \{$grid: any°$\}$.*

(a) If S is a subset of a 2×n strip, then only the infinite families defined by Fig. 9 cannot fold into C.

(b) If S is a subset of a 3×n strip, then only the infinite families defined by Fig. 10 cannot fold into C.

Proof. For the subset of a 2×n strip consider one vertical edge, as shown in Fig. 7, and the possible subtrees attached at A, B, C and D. One such vertical edge has to exist, otherwise the strip is a 1×n strip and never folds to a cube. We consider the length of subtrees attached at A, B, C and D when folded to the same row as this "docking" unit square to the vertical edge. With slight abuse of notation we refer to these lengths by A, B, C and D again.

The first observation is: If $(A \geq 2$ AND $B \geq 2)$ OR $(C \geq 2$ AND $D \geq 2)$, S folds to a cube; see Fig. 8(a). Not included in this categorization are the 4 shapes shown in Fig. 8(b)–(e). Of those only (d) folds into a cube.

Thus, more precisely, we obtain a cube for:

$$\begin{aligned}
&\{(A \geq 2 \text{ OR } D \geq 3) \text{ AND } (B \geq 2 \text{ OR } C \geq 3)\} \text{ OR}\\
&\{(A \geq 3 \text{ OR } D \geq 2) \text{ AND } (B \geq 3 \text{ OR } C \geq 2)\} \text{ OR}\\
&\{(A \geq 2 \text{ AND } C \geq 3) \text{ OR } (A \geq 3 \text{ AND } C \geq 2)\} \text{ OR}\\
&\{(B \geq 2 \text{ AND } D \geq 3) \text{ OR } (B \geq 3 \text{ AND } D \geq 2)\} \text{ OR}\\
&\{(A \geq 1 \text{ AND } C \geq 1 \text{ AND } B \geq 2 \text{ AND } D \geq 2)\} \text{ OR}\\
&\{(A \geq 2 \text{ AND } C \geq 2 \text{ AND } B \geq 1 \text{ AND } D \geq 1)\}.
\end{aligned}$$

For the subset of a 3×n strip: If there are vertical edges adjacent to a 1×n strip to two different sides

n	free polyominoes	dual trees	foldable with $\pm 90°$	and $\pm 180°$	and diagonal	not
2	1	1				
3	2	2				
4	5	5				
5	12	15				
6	35	54	11	0	0	43
7	108	212	90	24	39	59
8	369	908	571	175	126	36
9	1285	4011	3071	697	233	10
10	4655	18260	15645	2230	385	0
11	17073	84320	77029	6673	618	0
12	63600	394462	374066	19337	1059	0
13	238591	1860872	1803568	55477	1827	0
14	901971	8843896	8682390	158208	3298	0

Table 1: Different ways of folding small polyominoes into a cube.

(above and below), this folds to a cube. Consequently, if we have height three, a long 1×n strip cannot be located in the center row. W.l.o.g. let the 1×n strip be located in the lowest row. As we have height three, there is at least a height two part which is a subset of the red part in Fig. 10. If there exists another vertical edge of length at least one that is only adjacent to the 1×n strip (but not directly to the height two part) we can fold over the height two part and obtain a case from above which can easily be folded to a unit cube. Consequently, only a single vertical subset of length two can be attached, as shown in red in Fig. 10. □

4.1 Enumeration of cube-foldable polyominoes

In this section we present results on folding polyominoes of constant size—consisting of up to 14 unit squares—into a cube. These results have been obtained by exhaustive computer search. For polyominoes whose dual graph contains cycles, we considered all possible dual trees. Thus, we first generated all such dual trees for polyominoes of size up to 14. The third column of Table 1 shows their number, compared to the number of different free polyominoes, given in the second column. In both cases, elements which can be transformed into each other by translation, rotation and/or reflection are counted only once. While the number of different free polyominoes is currently known for shapes of size up to 28 (`http://oeis.org/A000105`), the number of different dual trees was known only for up to 10 elements, see `http://oeis.org/A056841`.

Based on the generated dual trees we checked each of them whether it can be folded into a unit cube. We did this in three different steps. First, only $90°$ folds have been allowed. Column 4 of Table 1 shows how many (dual trees of) polyominoes can be folded to a cube this way. It is interesting to observe that while for $n = 6$ only 11 polyominoes can be folded to a cube, for $n = 14$ it already works for over 98% of all shapes.

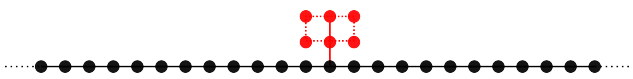


Figure 10: Infinite families that cannot fold into a unit cube, with at least a height 2 red subset (others are optional).

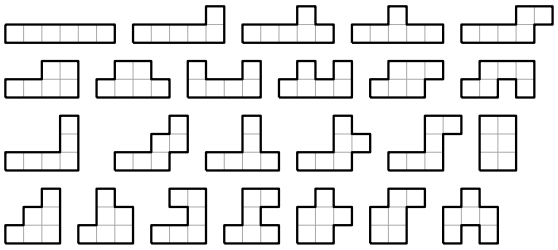


Figure 11: Polyominoes of size 6 that cannot be folded to a cube (for any dual tree).

In the second step we tried $\pm 90°$ and $\pm 180°$ folds for the remaining dual trees. Table 1 gives in column 5 how many additional cube foldings can be obtained this way. It is interesting to note, that never more than two $\pm 180°$ folds were needed, if the shape was foldable this way at all. For $n = 11$ and $n = 12$ there are each only one example which needs two $\pm 180°$ folds, and no such examples for $n \geq 13$ exist, i.e., in that case all foldable examples can be folded with just one $\pm 180°$ fold.

In the last step we allowed for the remaining dual trees also diagonal folds. Column 6 of Table 1 shows how many additional dual trees can be folded in this case, and the last column gives the number of remaining (non-foldable) dual trees. The most interesting result here is that for $n \geq 10$ all polyominoes, regardless which dual tree we select for them, can be folded this way. This partially affirms Theorem 4: here we do not allow half-grid folds, but covering a cube face with triangles from diagonal folds. Moreover, all such foldings need at most one such diagonal fold, with the exception of the 7×1 strip, which is the only example that needs two diagonal folds.

Figs. 11 and 12 show all polyominoes of size $n \geq 6$ for which dual trees exist, such that they cannot be

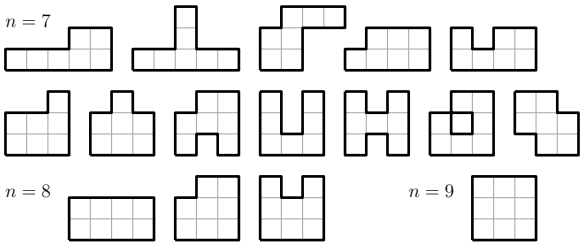


Figure 12: Polyominoes of size 7 to 9 that can be cut into a tree shape that cannot be folded to a cube.

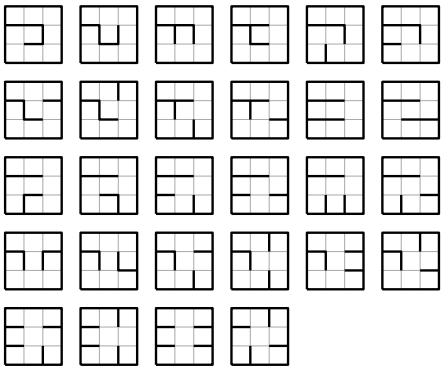


Figure 13: Puzzle: which cuttings of a 3×3 square fold to a cube? (For solution see the full version of the paper.)

folded to the unit cube using 90°, 180°, and diagonal folds. There are 24 such polyominoes with a total of 43 different dual trees for $n = 6$, 12 polyominoes with 59 dual trees for $n = 7$, 3 polyominoes with 36 dual trees for $n = 8$, and one polyomino with 10 dual trees for $n = 9$. Note, however, that for many of them there are cuts (i.e., dual trees) such that they can be folded to the cube. For example, the 3×3 square has 18 dual trees that can fold to a cube (Fig. 13).

5 Dynamic program for trees

Theorem 6 *Let S be a tree shape, C be a polycube with $O(1)$ cubes, no four squares meeting at an edge, and $\mathcal{F} = \{grid : \pm 90°\}$. Then it is possible in linear time to determine whether S can fold to C in folding model $\mathcal{F}$.*

Proof. (sketch) We choose an arbitrary root of S; for each square s of S define the subtree of s to be the tree shape consisting of all squares whose shortest path in S to the root passes through s. For a square s of S, define a *placement* of s to be an identification of s with a surface square of C together with the subset of the squares of C covered by squares in the subtree of s. We use a dynamic program that computes, for each square s of S, and each placement of s, whether there is a folding of the subtree of s that places s in the correct position and correctly covers the specified subset. Each square has $O(1)$ placements, and we can test whether a placement has a valid folding in constant time given the same information for the children of s. Therefore, the algorithm takes linear time. □

We have been unable to extend this result to folding models that allow 180° folds, nor to folds with interior faces, nor to polycubes for which four or more squares meet at an edge. The difficulty is that the dynamic program constructs a mapping from the polyomino to the polycube surface (topologically, an *immersion*) but what we actually want to construct is a three-dimensional embedding of the polyomino without self-intersections, and in general testing whether an immersion can be lifted to a three-dimensional embedding is NP-complete [3]. For 90° folds, a three-dimensional lifting always exists, as can be seen by induction on the number of squares in the tree shape: given a folding of all but one square of the tree shape, there can be nothing blocking the addition of the one remaining square to its neighbor in the tree shape. However, if a pentomino formed by a single row of five squares is given +180° folds at the two edges incident to its central square, the result cannot be embedded into three dimensional space: one of the two-square flaps will be blocked by the fold from the other flap.

It is tempting to attempt to extend our dynamic program to a fixed-parameter algorithm for non-trees (parameterized by feedback vertex number in the dual graph of the polyomino), by finding an approximate minimum feedback vertex set, trying all placements of the squares in this set, and using dynamic programming on the remaining tree components of the graph. However, the problem of parts of the fold blocking other parts of the fold becomes even more severe in this case, even for 90° folds. Additionally, we must avoid knots and twists in the three-dimensional embedding. These issues make it difficult to extend the dynamic program to the non-tree case.

6 Conclusion

Various open problems remain. We gave an example of a tree shape S that does fold into a polycube C for $\mathcal{F} = \{$grid: any°; interior faces$\}$, but not in weaker models, in particular, not without interior faces. C consists of 5 unit cubes; is it minimal? Moreover, we characterized tree shapes that fold into a unit cube in the $\mathcal{F} = \{$grid: any°$\}$ model—can we characterize polyominoes with holes (possibly of area zero) that fold into a unit cube? If a tree shape folds into a unit cube, can it be folded with rigid faces (continuous blooming)?

References

[1] Z. Abel, E. Demaine, M. Demaine, H. Matsui, G. Rote, and R. Uehara. Common developments of several different orthogonal boxes. In *Proc. 23rd Canad. Conf. Comput. Geom.*, 2011.

[2] G. Aloupis, P. K. Bose, S. Collette, E. D. Demaine, M. L. Demaine, K. Douïeb, V. Dujmović, J. Iacono, S. Langerman, and P. Morin. Common unfoldings of polyominoes and polycubes. In *Proc. 9th Int. Conf. Comput. Geom., Graphs & Appl.*, volume 7033 of *LNCS*, pages 44–54, 2011.

[3] D. Eppstein and E. Mumford. Self-overlapping curves revisited. In *Proc. 20th ACM-SIAM Symp. Discrete Algorithms*, pages 160–169, 2009.

Touring a Sequence of Line Segments in Polygonal Domain Fences

Amirhossein Mozafari* mozafari@alum.sharif.edu
Alireza Zarei* zarei@sharif.edu

Abstract

In this paper, we consider the problem of touring a sequence of line segments in presence of polygonal domain fences. In this problem there is a sequence $\mathcal{S} = (s = S_0, \ldots, S_k, S_{k+1} = t)$ in which s and t are respectively start and target points and $S_1, \ldots, S_k$ are line segments in the plane. Also, we are given a sequence $\mathcal{F} = (F_0, \ldots, F_k)$ of planar polygonal domains called fences such that $S_i \cup S_{i+1} \subset F_i$. The goal is to obtain a shortest path from s to t which visits in order each of the segments in $\mathcal{S}$ in such a way that the portion of the path from S_i to S_{i+1} lies in F_i. In 2003, Dror *et. al.* proposed a polynomial time algorithm for this problem when the fences are simple polygons. Here, we propose an efficient polynomial time algorithm for this problem when the fences are polygonal domains (simple polygons with some polygonal holes inside).

1 Introduction

Computing a shortest path between two points having some desired properties is one of the classic and well studied problems in computer science and computational geometry. In many applications, we need to visit a sequence of certain regions while traversing from s to t. We call this class of problems as visiting problems. In such visiting problems, a desired path may be restricted to a fence (or fences) which means that the path must completely lie inside a special region. Zoo-Keeper [7], Safari [11] and Watchman Route [4] problems are famous examples of such visiting problems. In Zoo-keeper and Safari problems we need to obtain a shortest tour visiting a set of disjoint convex polygons (called cages) inside a simple polygon P (the fence) each of which shared an edge with the boundary of P. The difference between these two problems is that in the first one, the desired path cannot enter into the cages while this restriction does not exist in the second one. In watchman route problem (fixed source version) we have a point inside a simple polygon P (the fence) and we seek for a shortest tour inside P containing s such that every point in P is visible from at least one point of the tour.

*Department of Mathematical Science, Sharif University of Technology

Dror *et. al.* [5] in 2003 introduced a general version of these visiting problems called *touring polygons problem (TPP)*. In this problem there is a sequence $\mathcal{P} = (s = P_0, \ldots, P_k, P_{k+1} = t)$ of polygons where s and t are respectively start and target points and a sequence $(F_0, \ldots, F_k)$ of simple polygonal fences where $P_i \cup P_{i+1} \subset F_i$. The goal of this problem is to obtain a shortest path from s to t which intersects the polygons of $\mathcal{P}$ in order and its subpath from P_i to P_{i+1} lies inside F_i. They proved that TPP is NP-hard for intersecting polygons and proposed a $O(nk^2 \log n)$ time algorithm for convex polygons. They also gave a $O(kn \log(n/k))$ time algorithm for the cases where the polygons are convex and pairwise disjoint and the fences are the whole plane. Here, n is the total number of vertices of all fences and polygons. In 2006, Arkin *et al.* [3] considered the touring polygons problem in L_1 metric for the cases where polygons are pairwise disjoint segments and the fences are the whole plane and proposed a $O(k^2)$ time algorithm for this version.

For one decade the complexity of TPP for disjoint non-convex polygons was unknown and during these years several approximation algorithms have been proposed for solving this version of the problem [6, 10]. Finally, Ahadi *et al.* [1] in 2013 proved that TPP is NP-hard for disjoint polygons in any L_p norm.

Despite many investigations and results on various kinds of visiting problems with simple polygon fences, there are less results on visiting problems whose fences are polygonal domains. In 2014, Ahadi *et al.* [2] gave a polynomial time algorithm for touring polygonal objects problem in which the fences are polygonal domains. This problem is similar to TPP but in this problem a desired path cannot enter into the polygons (this is like the Zoo-Keeper problem in which the tour cannot enter the cages).

In this paper, we present a polynomial time algorithm for a version of the TPP called *Touring Line Segments Problem (TLSP)* in which the polygons are line segments and the fences are polygonal domains. This is the first polynomial time algorithm which considers TPP when the fences are polygonal domains. Figure 1 shows an example of TLSP. We show that this problem can be solved in $O(n^3k)$ time where n is the number of vertices of all fences and segments and k is the number of segments. We use the well-known continuous Dijkstra paradigm [9, 8] to obtain a shortest path for our

problem.

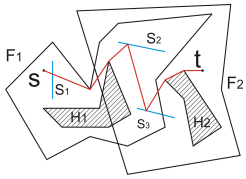


Figure 1: An example of TLSP where (s, S_1, S_2, S_3, t) is the sequence of segments and (PLN, F_1, PLN, F_2) is the sequence of fences and PLN represents the whole plane.

2 Preliminaries and Definitions

Let $\mathcal{T} = (\mathcal{S}, \mathcal{F})$ be an instance of TLSP with segments $\mathcal{S} = (s = S_0, S_1, \ldots, S_k, S_{k+1} = t)$ and fences $\mathcal{F} = (F_0, \ldots, F_k)$ in the plane such that $S_i \cup S_{i+1} \subset F_i$.

Observation 1. Any optimal path is a polygonal chain which bends only on some vertices of the fences, endpoints of segments, intersection point of two consecutive segments or on some reflection points on the interior of segments of $\mathcal{S}$.

According to the above observation, when an optimal path intersects the interior of a segment (not on the sgment endpoints) according to its order, the optimal path either passes or reflects on this segment. We denote by $\mathcal{T}_i(x)$ an instance of TLSP with $(S_0, \ldots, S_i, x)$ and $(F_0, \ldots, F_i)$ as its segments and fences, respectively. The set of optimal paths of $\mathcal{T}_i(x)$ is denoted by $P_i(x)$. Let $x \in F_i$ and $p \in P_i(x)$ and v be a vertex of F_j or an endpoint of S_j $(j \leq i)$. We call v an *origin* of x if v is the last fence vertex or segment endpoint on p when we traverse p from s to x. Thus, p must pass or reflect on the remaining segments from S_{j+1} to S_i to reach x. We say that v' is a *source* of x if it is obtained by the sequence of reflections of v on the supporting lines of segments that p reflects from v to x in order. For example, s is the *origin* of points x,x',x'' and x''' in Figure 2, and s,s',s'' and s''' are respectively the corresponding sources of the points x,x',x'' and x'''. According to this definition, the length of the portion of p from v to x is equal to $|v'x|$. According this definition, the number of distinct sources of all points in F_i can exponentially grows. Figure 2 shows an example of such situations. In this figure, if we consider $\mathcal{S} = (s)$, s is the only source of the points in the plane. After adding S_1 to $\mathcal{S}$, the set of sources of points in the plane becomes $\{s, s'\}$ and after adding S_3, the sources are $\{s, s', s'', s'''\}$. In this example, each segment doubles the set of sources of all points of the plane. We say that a point $x \in F_i$ is *straightly reachable from* $S_j (1 \leq j \leq i)$ if it has an optimal path with an origin in vertices of F_l or an endpoint of S_l where $l < j$.

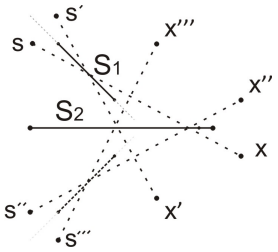


Figure 2: The number of sources can grow exponentially: s,s',s'' and s''' are respectively the corresponding sources of the points x,x',x'' and x'''.

To find an optimal path for $\mathcal{T}$, we use the continuous Dijkstra paradigm [9, 8] (In Appendix 1, we briefly describe the continuous Dijkstra paradigm using our terminology). Basically, we need some modifications on this paradigm to enforce shortest paths to visit the segments in order from s to t. But, as we said before, the number of sources of all points of the fences may grow exponentially and we cannot consider them as the set of initial sites in the continuous Dijkstra paradigm to obtain a shortest path map for each fence from which an optimal path is obtained. In next sections, we first prove some properties about optimal paths and based on these properties a modified version of the continuous Dijkstra paradigm is proposed to solve TLSP in polynomial time.

3 TLSP for Consecutively Disjoint Segments

In this section, we restrict ourselves to instances of TLSP in which the segments of $\mathcal{S}$ are consecutively disjoint. Our algorithm can be extended to solve TLSP for intersecting segments as described in Appendix 4 and we skip it because of the limited space here. Let $\mathcal{T} = (\mathcal{S}, \mathcal{F})$ be such an instance of the problem. If S_i, for $1 \leq i \leq k$, is a single point, we can break this problem into two sub-problems one from s to S_i and another from S_i to t. Therefore, we assume that s and t are the only points in $\mathcal{S}$. To obtain an optimal path for $\mathcal{T}$, we use the continuous Dijkstra paradigm to build a shortest path map for each F_i $(0 \leq i \leq k)$ namely SPM_i such that having the sequence $\mathcal{SPM} = (SPM_0, \ldots, SPM_k)$ we can obtain a solution for $\mathcal{T}$. For simplicity, we imagine $k+1$ planes such that the i^{th}-plane $(0 \leq i \leq k)$ contains only S_i, F_i and S_{i+1}, and construct SPM_i in this plane. To construct the shortest path maps we need $k+1$ wavefronts each of which propagates in one plane and sweeps its fence. To identify the initial wavelets of these wafefronts, we need some information about the configuration of $\mathcal{T}$. For this purpose, our algorithm consists of three phases: pre-processing $\mathcal{T}$, building $\mathcal{SPM}$

and computing an optimal path.

3.1 The pre-processing phase

Before we describe the first phase, we need some definitions. We inductively define the extensions of S_j in i^{th}-plane $(0 < j \leq i)$ as follows: For $i = j$, the extensions of S_j is simply the two half-lines along $\bar{S}_j$ starting from its endpoints and going away from S_j where $\bar{S}_j$ is the supporting line of S_j. For $i > j$, let r be the intersection of S_i and an extension e of S_j in the $(i-1)^{th}$-plane. For any one of these intersection points, the half-line from r along e and its reflection with respect to $\bar{S}_i$ are extensions of S_j in the i^{th}-plane. In fact, each extension of S_j in the $(i-1)^{th}$-plane which intersects S_i, generates two extensions of S_j in the i^{th}-plane (If S_i does not intersect any extension of S_j in the $(i-1)^{th}$-plane, we have no extension of S_j in the i^{th}-plane)(See Figure 3). According to this definition, S_i and extensions of S_j in the i^{th}-plane induces a subdivision which is called the j^{th}-subdivision of the i^{th}-plane.

Lets assign α and β marks arbitrary to the sides of the supporting line of each segment $S_j \in \{S_1, \ldots, S_k\}$. Based on this *initial marking assignment*, we mark all regions of the j^{th}-subdivision of the i^{th}-plane $(i \geq j)$ as follows: For $i = j$, we mark the half-plane lies on α-side of S_j as α-region and the other as β-region. For $i > j$, each region R of the j^{th}-subdivision of the i^{th}-plane contains exactly one sub-segment of S_i on its boundary. We assign to R the mark of the region of the j^{th}-subdivision of the $(i-1)^{th}$-plane which contains this sub-segment. Note that according to our definition, this sub-segment must entirely lie in one region of the j^{th}-subdivision of the $(i-1)^{th}$-plane (Figure 3).

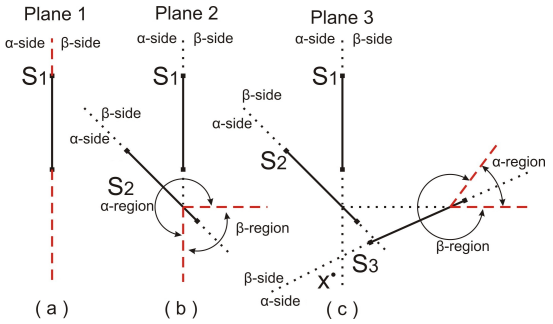


Figure 3: Extensions of S_1 in the 1^{th},2^{th} and 3^{th}-plane are shown by dashed half-lines.

Let x be a point in the i^{th}-plane. The j^{th}-component of x for $1 \leq j \leq i$ is defined to be the mark (α or β) of that region of the j^{th}-subdivision of the i^{th}-plane which contains x. According to Observation 1 and this definition, it is simple to verify that if x is straightly reachable from S_j $(1 \leq j \leq i)$ in some path $p \in P_i(x)$, then, for all $j \leq l < i$, the mark of the region of the j^{th}-subdivision of the l^{th}-plane in which p traverses from S_l to S_{l+1} is equal to the j^{th}-component of x. This fact gives us an intuition about the optimal paths to x that their origins are vertices of $F_l \cup S_l$ for some $l < j$. These paths will reflect on or pass through segments $S_j, S_{j+1}, \ldots, S_i$ and their traversal (passing through or reflecting on) from S_j to S_i is according to the j^{th}, $j+1^{th}$, $\ldots$,i^{th} component of x.

Let $\mathcal{M}$ be an initial marking assignment for sides of the supporting lines of segments $\{S_1, \ldots, S_k\}$. The characteristic sequence of a point $x \in i^{th}$-plane is defined as a vector $< x_1, \ldots, x_i >$ in which its j^{th}-term is computed as follows (this has been described formally in Algorithm 1 in Appendix 3. In this algorithm, procedure PChar(j,i,x) is used to find the j^{th}-term of the characteristic sequence of a point $x \in i^{th}$-plane). The last term x_i is equal to the mark of that side of $\bar{S}_i$ which contains x. For $j < i$, the j^{th}-term is recursively defined to be either the j^{th}-term of the characteristic sequence of x in the $(i-1)^{th}$-plane or the j^{th}-term of the characteristic sequence of the reflection of x on $\bar{S}_i$ in the $(i-1)^{th}$-plane. The former case happens when x lies in the α side of $\bar{S}_i$ and the later is used otherwise.

Note that the j^{th}-term of the characteristic sequence of a point in the i^{th}-plane $(i > j)$ is not necessarily equal to its j^{th}-component. For example, the 1^{th}-component of $x \in 3^{th}$-plane in Figure 3 is β while its first characteristic sequence term is α. However this difference is due to the initial marking assignment used in this example.

Assume that e is an extension of S_j in the $(i-1)^{th}$-plane $(j < i)$ which starts from point r on S_{i-1} and intersects S_i. We say that an initial marking assignment $\mathcal{M}$ has *passing property* if for all such extensions, r lies in β-side of S_i. The following lemma implies that there is always an initial marking assignment of $\mathcal{S}$ having passing property.

Lemma 1. The starting point of all extensions of all segments $S_j \in \{S_1, \ldots, S_{i-1}\}$ in the $(i-1)^{th}$-plane that intersect S_i lie on one side of $\bar{S}_i$.

Proof. See Appendix 2.□

Lemma 1 implies that there is always an initial marking assignment of sides of segments $\{S_1, \ldots, S_k\}$ having passing property and shows how to obtain such an assignment. But, before describing a method for obtaining such a marking assignment we describe how such a marking is used in obtaining optimal paths. As said before, identifying all j^{th}-component of a point $x \in i^{th}$-plane $(1 \leq j \leq i)$ are critical in obtaining an optimal path to x. The following lemma shows how we can obtain these data.

Lemma 2. Let $\mathcal{M}$ be an initial marking assignment of segments which satisfies the passing property, and $x \in F_i$ is straightly reachable from S_j $(j \leq i)$. Then, the j^{th}-term of the characteristic sequence of this point

for marking $\mathcal{M}$ is equal to its j^{th}-component.

Proof. See Appendix 2.□

Now, we return to the concept of the initial marking assignment and propose a method for obtaining a marking which has passing assignment. This method has been formally described as procedure MarkSides($\mathcal{S}$) in Algorithm 2 in Appendix 3.

Our marking algorithm iteratively marks both sides of segments from S_1 to S_k. According to the definition of the passing property, the starting points of all extensions of $S_1, \ldots, S_{i-1}$ that intersect S_i in the $(i-1)^{th}$-plane lie in the β-side of $\bar{S}_i$. Also, from the definition of the extensions in the $(i-1)^{th}$-plane, all these starting points lie on S_{i-1}. For S_1, we do not have any preceding segment and $s = S_0$ is a point which can be considered as a degenerate segment. Therefore, we mark the containing half-plane of s as β-side of $\bar{S}_1$ and the other as α-side. Then, each half-line extension of S_0 that starts from s and intersects S_1 has its starting point in the β-side of S_1. Therefore, marking S_1 this way supports the passing property.

Now assume that we have marked sides of segments $S_1, \ldots, S_{i-1}$ in such a way that supports the passing property. If S_{i-1} lies completely in one side of $\bar{S}_i$, the two extensions of S_{i-1} (from its endpoints) as well as extensions of preceding segments that intersect S_{i-1} have starting point on one side of $\bar{S}_i$. To support the passing property, we mark this side as β-side and the other as α-side. Otherwise (if S_{i-1} intersects $\bar{S}_i$), S_i must lie completely in one side of $\bar{S}_{i-1}$. This is due to our assumption that segments are consecutively disjoint. Then, none of the two extensions of S_{i-1} intersects S_i. Therefore, the extensions that intersect S_i in the i^{th}-plane are either extensions in $(i-2)^{th}$-plane that intersect S_{i-1} or the reflections of these extensions. Thinking inductively, we have already marked the sides of $\bar{S}_{i-1}$ which supports the passing property. This implies that all of the extensions of the $(i-2)^{th}$-plane that intersect S_{i-1} continue in α-side of $\bar{S}_{i-1}$ and their reflections continue in β-side. Then, if S_i lies in α-side of $\bar{S}_{i-1}$ it can be intersected by extensions of the $(i-2)^{th}$-plane and if it lies in β-side, it may be intersected by reflections of these extensions on S_{i-1}. For the first case, all extensions that intersect S_i have a starting point on S_{i-2}. This fact helps us to ignore S_{i-1} and mark sides of S_i considering S_{i-2} as its preceding segment. If we do this inductively, we will finally reach a base case from which the marks of sides of S_i are obtained. In the other case where S_i lies in β side of S_{i-1}, we first reflect S_i on $\bar{S}_{i-1}$ and obtain the marks for sides of this new segment namely S'_i. Then, the marks of the sides of S_i will be the marks of the corresponding sides of S'_i. It is important to note that the extensions of the $(i-1)^{th}$-plane diverse in both α and β sides of $\bar{S}_{i-1}$. This forces that when $\bar{S}_i$ intersects S_{i-1} it will be intersected by only extensions whose starting points from S_{i-1} lie on one side of $\bar{S}_i$.

There is still one flaw in this inductive algorithm. We assumed that consecutive segments do not intersect each other. But, when we reflect S_i on $\bar{S}_{i-1}$ and consider S_{i-2} as the preceding segment of S_i (ignoring S_{i-1}), S_i may intersect S_{i-2}. To solve this problem, we cut that part of S_{i-2} which lies on the α-side of S_{i-1}. Precisely, in each iteration after marking sides of a segment S_i we cut and remove from S_{i-1} that part that lies in α-side of S_i (line 16 in Algorithm 2 in Appendix 3). This does not affect our algorithm because non of the extensions of the $(i-2)^{th}$-plane that have their starting point on the α-side of $\bar{S}_{i-1}$ intersect S_{i-1}, and consequently, does not affect the passing property of S_i.

According to the above discussion, the pre-processing phase of our algorithm consists of two steps. First, marking the sides of $\mathcal{S}$ using procedure MarkSides and second, computing the characteristic sequences of all vertices of F_i and endpoints of S_{i+1} in the i^{th}-plane $(1 \leq i \leq k)$. These information enables us to efficiently construct the shortest path maps in the second phase of our algorithm.

3.2 Building shortest path maps

In order to build the $k+1$ shortest path maps $\mathcal{SPM} = (SPM_0, \ldots, SPM_k)$, we need to perform in parallel $k+1$ instances of our modified version of the continuous Dijkstra algorithm inside fences $F_0, \ldots, F_k$ in their corresponding planes. The initial wavefront of each F_i propagates from S_i and sweeps F_i to build SPM_i. For F_0, the initial wavefront is a complete circle with zero radius around s. For $i > 0$, the initial wavefront of F_i is determined according to the wavelets intersect S_i during the wavelet propagation of the wavefront of F_{i-1}. Precisely, when a wavelet ω in F_{i-1} intersects an endpoint of S_i, a new wavelet centred at this endpoint and zero radius starts to propagate in F_i. If ω intersects the interior of S_i, it can generate one or two wavelets in F_i according to the information we obtained in the pre-processing phase. In order to detect the wavelets which intersect S_i in F_{i-1} in our implementation of the continuous Dijkstra paradigm, we consider S_i as an obstacle (hole) in F_{i-1}. So, we can use the standard event handling of continuous Dijkstra paradigm [9, 8] to identify such events. Let ω be a wavelet in F_{i-1}. When it touches the interior of S_i, it can generate two wavelets in F_i. One, called *passing wavelet*, propagates in F_i along ω and the other, called *reflecting wavelet*, is the reflection of the passing wavelet with respect to $\bar{S}_i$. Let ω be a wavelet in F_{i-1} with center C_ω which intersects $S_i = ab$ at point I. Without lose of generality, assume that C_ω lies in the α-side of S_i. Two cases may happen: First, I is a point in the interior of ω and second, I is an endpoint of ω. In the first case, the passing wavelet of ω is a wavelet with center C_ω which propagates from I in

the α-side of S_i and is restricted to the angle $\widehat{aC_\omega b}$ in F_i (See Figure 4).

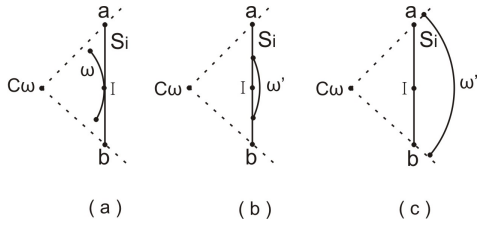


Figure 4: ω generates a passing wavelet ω' in F_i when an interior point of ω intersects S_i.

In the second case, the passing wavelet of ω is a wavelet with center C_ω which propagates from I in the α-side of S_i. This wavelet is restricted to the angle $\widehat{aC_\omega I}$ if a lies on that side of $C_\omega I$ which contains ω, and is restricted to $\widehat{IC_\omega b}$, otherwise (Figure 5).

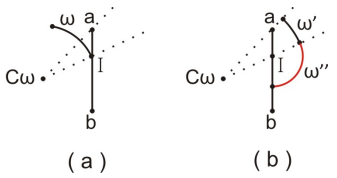


Figure 5: ω generates a passing wavelet ω' in F_i when an endpoint of ω intersects S_i. In this case, another wavelet (ω'') from this intersection point is propagated in F_i.

Note that the center of passing and reflecting wavelets may lie outside F_i. According to the above discussion, the initial wavelets of F_i are the wavelets propagate from the endpoints of S_i (these endpoints act like initial sites in F_i whose weights are assigned by the wavefront of F_{i-1}) and a set of passing or reflecting wavelets obtained according to the pre-processing information. Moreover, when a wavelet ω in F_{i-1} generates a passing or reflecting wavelet in F_i by intersecting S_i in its endpoint (the second case in the above discussion) we consider the intersection point (I in Figure 5) as an initial site in F_i whose weight is assigned by ω in F_{i-1}.

If we consider both passing and reflecting wavelets for all wavelets that intersect S_i in F_{i-1} for all $1 \leq i \leq k+1$, the corresponding cells of passing (resp. reflecting) wavelets in F_i represent the set of points $x \in F_i$ for which there is an optimal path in $P_i(x)$ that passes through (resp. reflects on) S_i. Furthermore, the site of each cell (center of the wavelet which sweeps this cell) in F_i will be a source of its containing points and for each $x \in F_i$ we can recursively construct an optimal path in the same way as what we do in obtaining an optimal path between two points in a polygonal domain using continuous Dijkstra paradigm [9]. But, as we said in Section 2, the number of sources and therefore the number of cells can grow exponentially this way. To overcome this problem, we filter the passing and reflecting wavelets generated by the wavelets in F_{i-1} according to the pre-processing information. For this purpose, we assign a sequence $SQ(\omega)$ to each wavelet ω in F_i $(0 \leq i \leq k)$ as follows: The sequence of all wavelets in F_i whose center is a vertex of F_i or an endpoint of S_i is the empty sequence. If a wavelet ω in F_{i-1} generates a wavelet ω' in F_i (by passing or reflecting) which propagates in α-side (resp. β-side) of S_i, we have $SQ(\omega') = (SQ(\omega), \alpha)$ (resp. $SQ(\omega') = (SQ(\omega), \beta)$).

Let $\mathcal{P}_i$ be the set of all vertices of F_i and endpoints of S_{i+1} $(1 \leq i \leq k)$ and ω be a wavelet in F_{i-1} with $SQ(\omega) = (w_1, \ldots, w_l)$ which intersects the interior of S_i. The wavelet ω generates a wavelet in F_i only when there exists a point $x \in \mathcal{P}_j$ $(j \geq i)$ with characteristic sequence $(x_1, \ldots, x_{i-1}, x_i, \ldots, x_j)$ where $(w_1, \ldots, w_l) = (x_{i-l}, \ldots, x_{i-1})$. Then, if $x_i = \alpha$, ω generates a wavelet (by passing or reflecting) in α-side of S_i and if $x_i = \beta$, ω generates a wavelet in β-side of S_i in F_i. This filtering mechanism prevents the exponential growth of the wavelets in our algorithm.

Lemma 3. The total size of the shortest path maps $\mathcal{SPM}$ obtained according to the above propagation rules is polynomial in terms of the number of vertices of all fences and k.

Proof. The complexity of $SPM_i \in \mathcal{SPM}$ is proportional to the number of wavelets propagates in F_i. Trivially, this is linear with respect to the number of vertices in F_0 for SPM_0. The wavelets of SPM_i for $i > 0$ are generated due to either a vertex in F_i or intersecting S_i by a wavelet in SPM_{i-1}. For the first case, each vertex of F_i generates exactly one wavelet. Wavelets of the second type have non-empty sequences. For each wavelet with non-empty sequence $(w_1, \ldots, w_l)$ in F_i, there must exist a vertex in $\mathcal{P}_j$ with characteristic sequence $(x_1, \ldots, x_{i-1}, x_i, \ldots, x_j)$ where $(w_1, \ldots, w_l) = (x_{i-l+1}, \ldots, x_i)$. Therefore, the number of distinct wavelet sequences of length l is proportional to the number of vertices in $\bigcup \mathcal{P}_j$. On the other hand, all wavelets with sequence of length l in F_i have emerged in a vertex of F_{i-l}. Therefore, $|\mathcal{P}_{i-l}|.|\bigcup_{j=k}^{i} \mathcal{P}_j|$ is an upper bound for the number of wavelets in F_i with sequence of length l. Summing this for all $1 \leq l \leq i$ results the lemma.□

Lemma 4. The weights assigned by the wavefront in SPM_i to each vertex v of $F_i \cup S_{i+1}$ is equal to the length of all optimal paths for $\mathcal{T}_i(v)$.

Proof. We prove this lemma by induction on i. According to the standard continuous Dijkstra paradigm, the theorem is true for vertices of $F_0 \cup S_1$. For $i > 0$, let x be a vertex of F_i or an endpoint of S_{i+1}. The vertex x

either has an origin v in the vertices of $F_j \cup S_{j+1}$ where $j < i$ or all of its origins belong to F_i. The first case means that x is straightly reachable from S_{j+1}. According to the induction hypothesis, the weight of v is the optimal length from s to v. Also, by Lemma 2 and the propagation rules, the wavelet with center v in F_j generates wavelets in fences $F_l (j < l \leq i)$ according to the characteristic sequence of x. This means that we traverse along an optimal path from v to x which implies the lemma for x. For the other case where in all optimal paths the origin of x belongs to F_i, let $v \in F_i$ be the origin of x in an optimal path. If v is straightly reachable from S_{i+1} then we have the optimal length to v according to the first case and our version of continuous Dijkstra paradigm computes the shortest path from v to x. Otherwise, we can repeat this process inductively until reaching an endpoint of S_{i+1} or a straightly reachable vertex of F_i from S_{i+1}. □

3.3 Obtaining an optimal path

As the final phase of our algorithm we find an optimal path from s to t. This is a straightforward usage of the data in $\mathcal{SPM}$. For this purpose, we use procedure $OPT(i, x)$ (Described formally in Algorithm 3 in Appendix 3) which returns an optimal path for $\mathcal{T}_i(x)$ as a list of points. Then, $OPT(k, t)$ will be an optimal path for $\mathcal{T}$. In this procedure, if $s = x$, the optimal path for $T_0(s)$ is trivially the single point s. If x is an endpoint of S_i, an optimal path for $\mathcal{T}_i(x)$ is just an optimal path for $\mathcal{T}_{i-1}(x)$ which is the optimal path computed by $OPT(i-1, x)$. Otherwise, there are two cases for the last segment of an optimal path to x in $\mathcal{T}_i(x)$: First, the site c of the cell C in SPM_i which contains x is a vertex of F_i or an endpoint of S_i. Second, this cell belongs to a passing or reflecting wavelet ω. For the first case the optimal path is $(OPT(i, c), x)$ which means that the last segment of the optimal path is cx. For the second case, let v be an origin of x. We obtain a sub-path from v to x by a list L of points according to $SQ(\omega)$. Then, $(OPT(i-|SQ(\omega)|, v), L)$ is an optimal path from s to x which means that the optimal path goes to v and then, from v to x it acts (passes or reflects) according to $SQ(\omega)$.

3.4 Complexity of the Algorithm

In the first phase of our algorithm we compute characteristic sequences of all vertices of fences and endpoints of segments in $\mathcal{T}$. We use procedure MarkSides to properly mark the sides of each segment in $\mathcal{S}$. The function MARK in this procedure takes $O(k)$ time. Therefore, procedure MarkSides can be run in $O(k^2)$ time. Each execution of procedure PChar takes $O(k)$ and each characteristic sequence have $O(k)$ terms which means that obtaining characteristic sequence of each point takes $O(k^2)$ time. Therefore, the time complexity of the first phase of our algorithm is $O(k^2 n)$ where n is the total number of all fences vertices and segments endpoints. In the second phase, as discussed in the proof of Lemma 3, the total number of wavelets in all planes is $O(n^2)$ and we should handle $O(n^2)$ events according to [8]. However, in each event handling procedure, we have to perform $O(k)$ comparison for all $O(n)$ points to decide whether this wavelet must generate a passing or reflecting wavelet in the next plane. Therefore, the complexity of the second phase is $O(n^3 k)$. The complexity of the third phase depends on the number of bends and intersection points on the obtained optimal path which is $O(n)$. We can store enough information in shortest path maps to restore such an optimal path in linear time in terms of the output path length (storing the site of each cell and the corresponding sequence of its wavelet). Summing all these costs implies that the complexity of our algorithm is $O(n^3 k)$.

References

[1] A. Ahadi, A. Mozafari, and A. Zarei. "Touring disjoint polygons problem is NP-hard." In *Combinatorial Optimization and Applications*, pp. 351-360. Springer International Publishing, 2013.

[2] A. Ahadi, A. Mozafari, and A. Zarei. "Touring a sequence of disjoint polygons: Complexity and extension." *Theoretical Computer Science* 556 (2014): 45-54.

[3] E. Arkin, A. Efrat, C. Erten, F. Hurtado, J. Mitchell, V. Polishchuk, and C. Wenk. "Shortest tour of a sequence of disjoint segments in L1." In *Proc. 16th Fall Workshop on Computational and Combinatorial Geometry*. 2006.

[4] W. Chin, and S. Ntafos. "Shortest watchman routes in simple polygons." *Discrete and Computational Geometry* 6, no. 1 (1991): 9-31.

[5] M. Dror, A. Efrat, A. Lubiw, and J. Mitchell. "Touring a sequence of polygons." In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pp. 473-482. ACM, 2003.

[6] F. Li, and R. Klette. "Rubberband algorithms for solving various 2D or 3D shortest path problems." In *Computing: Theory and Applications*, 2007. ICCTA'07. International Conference on, pp. 9-19. IEEE, 2007.

[7] J. Hershberger, and J. Snoeyink. "An Efficient Solution to the Zookeeper's Problem." In *CCCG*, pp. 104-109. 1994.

[8] J. Hershberger, and S. Suri. "An optimal algorithm for Euclidean shortest paths in the plane." *SIAM Journal on Computing* 28, no. 6 (1999): 2215-2256.

[9] J. Mitchell. "Shortest paths among obstacles in the plane." *International Journal of Computational Geometry and Applications* 6, no. 03 (1996): 309-332.

[10] X. Pan, F. Li, and R. Klette. "Approximate shortest path algorithms for sequences of pairwise disjoint simple polygons." In *CCCG*, pp. 175-178. 2010.

[11] X. Tan, and T. Hirata. "Shortest safari routes in simple polygons." In *Algorithms and Computation*, pp. 523-530. Springer Berlin Heidelberg, 1994.

Appendix 1: An Overview on the continuous Dijkstra paradigm

The Dijkstra paradigm is originally proposed to obtain a shortest path between two points namely p and q in a polygonal domain namely D [9, 8]. This is done by simulating the propagation of a wavefront starting to propagate from p and sweeping the entire polygonal domain. Precisely, If we define the length of a shortest path between p and x in D as the weight of x and denote it by $w(x)$, the wavefront at distance d is :

$$WF(d) := \{x \in D | w(x) = d\}$$

A wavefront consists of a set of wavelets each of which is a circular arc whose center is p or some vertex of D. The initial wavefront contains a single point p as its only wavelet (a circle of radius zero centred at p). When a wavefront propagates, its structure (the set of its wavelets) changes, i.e., some wavelets may disappear, some may break into two wavelets and new wavelets may appear. A wavelet is eliminated from the structure of a wavefront when its two neighbour wavelets collide on each other and a wavelet may break into two wavelets when it collides the interior of an edge of D. Also, if a wavefront collides a vertex v of D, a new wavelet with center v appears and starts to propagate in the region of D that v blocks the wavefront to sweep it.

When the wavefront propagates, the traces of these endpoints decompose the swept part of D into regions having this property that all points of a region have combinatorially equivalent shortest paths. Therefore, this wavefront propagation induces a subdivision called *shortest path map* on D (See Figure 6). The *site* of each *cell* of this subdivision is the center of the wavelet that has swept it. If q belongs to a cell with site r, then the last segment of a shortest path from p to q is rq and the length of the shortest path from p to q is $w(r) + |rq|$. We can replace q by r and use the shortest path map to obtain the last segment of a shortest path from p to r and repeat this procedure until a shortest path from p to q is obtained.

We can use the continuous Dijkstra paradigm to solve a more general shortest path problem in which instead of having one initial site (p in the above discussion) we have multiple initial weighted sites namely $\{p_1, \ldots, p_i\}$ with weights $w(p_j)$ $(1 \leq j \leq i)$. Then, for a query point q in D we seek a shortest path from q to an initial site p_j such that $w(p_j) + d(p_j, q)$ is minimum where $d(p_j, q)$ is the length of a shortest path from p_j to q. This can be done by delaying wavelet propagation of each site according to its weight. Hershberger and Suri in [8] gave an implementation of the continuous Dijkstra paradigm which handles such cases.

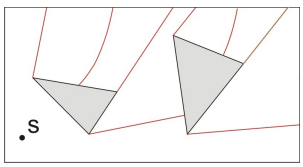


Figure 6: An example of a shortest path map

Appendix 2: Proofs

Lemma 1. The starting point of all extensions of all segments $S_j \in \{S_1, \ldots, S_{i-1}\}$ in the $(i-1)^{th}$-plane that intersect S_i lie on one side of $\bar{S}_i$.

Proof. Let e_1 be an extension of S_{l_1} and e_2 be an extension of S_{l_2} in the $(i-1)^{th}$-plane with starting points r_1 and r_2 respectively which intersect S_i $(1 \leq l_1, l_2 < i)$. We prove that both, r_1 and r_2 lie in one side of $\bar{S}_i$. According to the fact that the definition of extensions is independent to the fences, we assume that the fences are the whole plane. It is simple to see that if x lies on an extension e in the $(i-1)^{th}$-plane, there is always an optimal path $p \in P_{i-1}(x)$ for which e overlaps the last segment of p. For the sake of a contradiction, assume that r_1 and r_2 lie on different sides of S_i. Then, e_1 and e_2 must intersect each other on a point like x. Without loss of generality, we assume that e_1 intersects S_i after x (Figure 7). Then, there will be two optimal paths in $P_{i-1}(x)$: one reaches x from r_1 and the other from r_2. But, this implies that for each point x' after S_i on e_1 in the i^{th}-plane there exist an optimal path with two last segments r_2x and xx' which contradicts Observation 1. □

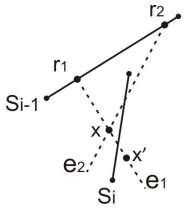


Figure 7: r_1 and r_2 must lie on one side of S_i.

Lemma 2. Let $\mathcal{M}$ be an initial marking assignment of segments which satisfies the passing property, and $x \in F_i$ is straightly reachable from S_j $(j \leq i)$. Then, the j^{th}-term of the characteristic sequence of this point for marking $\mathcal{M}$ is equal to its j^{th}-component.

Proof. Let $x \in F_i$ be straightly reachable from S_j in some optimal path $p \in P_i(x)$. This means that x is straightly reachable from all $S_l (j \leq l \leq i)$. Trivially, all fences $F_j, \ldots, F_{i-1}$ dos not affect the optimal path p. Therefore, we can consider this fences as the whole plane and ignore them at all.

We can prove the lemma by induction on $i - j$. For $i = j$, the lemma follows from the definition of the i^{th}-component and i^{th}-term of the characteristic sequence. In both cases, it is the mark of the half-plane of $\bar{S}_i$ that contains x.

Now, suppose that the lemma holds for all $0 \leq i - j < l$. To prove the lemma for $i - j = l$, we first assume that x lies in α-side of $\bar{S}_i$. According to the definition, j^{th}-component of x is equal to the mark of the containing region of x in j^{th}-subdivision of the i^{th}-plane. As shows in Figure 8, let R be this region. The mark of this region in this subdivision is equal to the mark of the containing region of segment ab in j^{th}-subdivision of the $(i-1)^{th}$-plane. While x lies in α-side of $\bar{S}_i$ and our initial marking assignment has passing property, all extensions of S_j in this half-plane are exactly the extensions of S_j in the $(i-1)^{th}$-plane that intersect S_i

and by considering x as a point in the $i-1^{th}$-plane, it is still straightly reachable from S_j. This means that x and ab lie in the same region in j^{th}-subdivision of the $(i-1)^{th}$-plane and by induction the j^{th}-term of the characteristic sequence of x in the $(i-1)$-plane and its j^{th}-component are equal. On the other hand, the j^{th}-term of x in the i^{th}-plane is considered to be equal to the j^{th}-term of its characteristic sequence in the $(i-1)$-plane. These two equalities imply that j^{th}-term of the characteristic sequence of $x \in F_i$ is equal to its j^{th}-component.

When x lies in β-side of S_i, because of having passing property, all regions of the j^{th}-subdivision of the i^{th}-plane in this side are reflections of the regions in α-side of $\bar{S}_i$ in this subdivision, and the marks of corresponding regions are the same. This means that the j^{th}-component of x in β-side of $\bar{S}_i$ is equal to the j^{th}-component of the reflection of x on $\bar{S}_i$. Similarly, our algorithm computes the j^{th}-term of the characteristic sequence of the reflection of x on $\bar{S}_i$ as the j^{th}-term of x which follows the lemma in this case as well. □

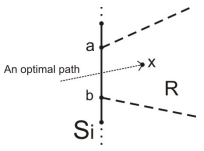


Figure 8: An optimal path to x from a point of ab.

Appendix 3: Algorithms

Algorithm 1 PChar(j,i,x)

if $j = i$ **then**
  if $x \in \alpha$-side of $\bar{S}_j$ **then**
    return α
  else
    return β
  end if
else
  if x lies in the α-side of $\bar{S}_i$ **then**
    return PChar(j,$i-1$,x)
  else
    Let x' be the reflection of x on $\bar{S}_i$
    return PChar(j,$i-1$,x')
  end if
end if

Algorithm 2 MarkSides($\mathcal{S}$)

function MARK(S,i)
  if S_i completely lies on one side of $\bar{S}$ **then**
    Mark the side of $\bar{S}$ which contains S_i as β-side and the other as α-side.
  else
    if S lies in the α-side of $\bar{S}_i$ **then**
      MARK(S,$i-1$)
    else
      Let S' be the reflection of S on $\bar{S}_i$
      MARK(S',$i-1$)
      Mark sides of $\bar{S}$ the same as marks of their corresponding sides of $\bar{S}'$.
    end if
  end if
end function
for $i \leftarrow 1$ to k **do**
  MARK(S_i,$i-1$)
  Cut S_{i-1} by removing from it the part that lies on the α-side of $\bar{S}_i$.
end for

Algorithm 3 OPT(i,x)

if $i = 0$ and $x = s$ **then**
  return s.
end if
if x is an endpoint of S_i **then**
  return OPT($i-1$,x).
end if
Let C with sequence $SQ(C) = (c_1, \ldots, c_l)$ be the cell with site c of SPM_i which contains x.
if the length of $SQ(C)$ is zero **then**
  return <OPT(i,c),x >.
end if
Let $j = l$ and L be a list with $L =< x >$.
while $j \neq 0$ **do**
  Let I be the intersection of cx and $S_{i-(l-j)}$.
  Append I to the beginning of L.
  if x does not lie on the c_j-side of $S_{i-(l-j)}$ **then**.
    Let c be the reflection of c on $\bar{S}_{i-(l-j)}$.
  end if
  $x = I$.
  $j = j - 1$.
end while
return <OPT($i-l$,c),L >.

Appendix 4: TLSP for intersecting line segments

In this section, we briefly describe how to extend our algorithm to solve TLSP when a segment $S_i \in \mathcal{S}$ may have intersection with S_{i+1}. Two types of intersections can be happened when S_i intersects S_{i+1}: *point intersection* and *interval intersection*. In point intersection, S_i and S_{i+1} have only one point in common but in interval intersection $S_i \cap S_{i+1}$

is a segment. If S_i and S_{i+1} overlap in interval I, we can simply remove S_i and S_{i+1} from $\mathcal{S}$ and replace them by segment I. So, we consider that there is no interval intersection between consecutive segments of $\mathcal{S}$.

Let $S_j = ab$ intersects S_{j+1} at point r. We define the extensions of S_j in the i^{th}-plane ($i \geq j$) as the union of extensions of ar and br in the i^{th}-plane (their interior is disjoint from S_{j+1}). So, If we mark the sides of S_j by α and β, each segment ar and br induces its own marking on the j^{th}-subdivision of the i^{th}-plane. Therefore, the mark of each region R of the j^{th}-subdivision of the i^{th}-plane is in $\{\alpha\alpha, \alpha\beta, \beta\alpha, \beta\beta\}$ which is composed of two letters. Assume that we have marked R as x_1x_2 where $x_1, x_2 \in \{\alpha, \beta\}$. If we replace S_j by ar, R completely lies in the x_1-region and if we replace S_j by rb, R completely lies in the x_2-region of the j^{th}-subdivision of the i^{th}-plane.

If we have characteristic sequences of all vertices and endpoints of segments, we can filter the wavelets in the second phase as follows: Let ω be a wavelet in F_{j-1} and $SQ(\omega)$ coincides $(x_{j-|SQ(\omega)|}, \ldots, x_{j-1})$. Assume that the j^{th}-term of the characteristic sequence of a point in $\mathcal{P}_j$ is $\alpha\beta$ (other cases are similar). This means that if ω intersects ar, it must generate a wavelet by passing or reflecting on ar in the α-side of S_j in F_j and if it intersects rb it must generate a wavelet by passing or reflecting on br in the β side of S_j in F_j. Note that we must also consider r as an initial site in F_j and its weight is computed by the wavefront of F_{j-1}. In both of the above cases the sequences of the generated wavelets in F_j are $(SQ(\omega), \alpha\beta)$.

Similar to Section 3, we use characteristic sequences and according to Lemma 2 if a point x is straightly reachable from S_{j-1}, the $(j-1)^{th}$-term of its characteristic sequence is equal to its $(j-1)^{th}$-component if we have a marking with passing property. When $S_j = ab$ intersects $S_{j+1} = cd$, two extensions in the j^{th}-plane may be intersected by S_{j+1} while their starting points lie on different sides of S_{j+1}. Figure 9 shows an example of such situations.

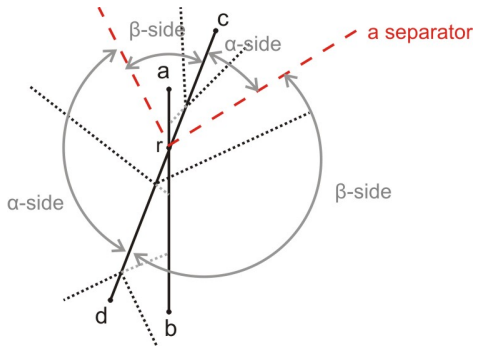


Figure 9: Extensions of S_j which intersect S_{j+1} are shown by gray dots and the extensions of S_{j+1} are shown by solid dots.

To handle such situations we extend the definition of α-side and β-side of S_{j+1} such that using this new definition, we can use procedure MarksSides and PChar to obtain characteristic sequences of vertices of fences and endpoints of segments.

Assume that function MARK marks the sides of cr and rd in such a way that the starting point of each extension in the j^{th}-plane which intersects these segments lies in β side of cr and rd. Lets consider a half-line from r which does not intersect any extension of S_l in the $(j+1)^{th}$-plane $(1 \leq l \leq j)$. We call this half-line and its reflection on $\bar{S}_{j+1}$ as *separators* (there is infinite separators of S_{j+1} but we need one in our algorithm) in the $(j+1)^{th}$-plane (separators are shown by dashed lines in Figure 9). These separators divide the plane into two regions namely R_1 and R_2. Lets R_1 be the one containing rc. We say that a point x or a segment S lies in α-side of S_{j+1} if it completely lies in α-side of rc and R_1 or α-side of rd and R_2. Otherwise, we say that it lies in β-side of S_{j+1}. By this definition, we see that all extensions in α-side of S_{j+1} coincide their corresponding extensions in the j^{th}-plane which is exactly the definition of the passing property. But, we need an extra clause in function MARK to handle the situation when a segment S intersects the separators in the $(j+1)^{th}$-plane: If a segment intersects a separator with starting point r in the $(j+1)^{th}$-plane, we mark the side of it which contains r as β-side and the other as α-side.

Therefore, in procedure MarkSides, when S_{j+1} intersects S_j we need to use function MARK for both cr and dr. Then, using a separator we can define its α and β-side. Now, because the marking obtained by MarkSides has the passing property, we can use procedure PChar to obtain the characteristic sequences of vertices of fences and endpoints of the segments.

Note that a separator for S_{j+1} from a point r in it can be easily obtained by considering r or its reflection recursively on α sides of $S_j, \ldots, S_1$ (Similar to procedure PChar) and in each step $l \leq j$ we maintain the bounds on the slope of a separator in which a separator does not intersect an extension of S_l.

According to the above discussion, if S_{j+1} intersects S_j, we need a linear time to obtain a separator for it. Also MarkSides performs function MARK at most $2k$ times. Also, the extra clause in MARK does not affect its complexity. Therefore, the running time of our algorithm remains $O(n^3k)$ for intersecting segments.

A Geometric Perspective on Sparse Filtrations*

Nicholas J. Cavanna† Mahmoodreza Jahanseir‡ Donald R. Sheehy§

Abstract

We present a geometric perspective on sparse filtrations used in topological data analysis. This new perspective leads to much simpler proofs, while also being more general, applying equally to Rips filtrations and Čech filtrations for any convex metric. We also give an algorithm for finding the simplices in such a filtration and prove that the vertex removal can be implemented as a sequence of elementary edge collapses.

A video illustrating this approach is available [7].

1 Introduction

Given a finite data set in a Euclidean space, it is natural to consider the balls around the data points as a way to fill in the space around the data and give an estimate of the missing data. The union of balls is often called the *offsets* of the point set. Persistent homology was originally invented as a way to study the changes in topology of the offsets of a point set as the radius increases from 0 to ∞. The input to persistent homology is usually a filtered simplicial complex, that is, an ordered collection of simplices (vertices, edges, triangles, etc.) such that each simplex appears only after its boundary simplices of one dimension lower. The Nerve Theorem and its persistent variant allow one to compute the persistent homology of the offsets by instead looking at a discrete object, a filtered simplicial complex called the nerve (see Fig. 1). The simplest version of this complex is called the Čech complex and it may be viewed as the set of all subsets of the input, ordered by the radius of their smallest enclosing ball. Naturally, the Čech complex gets very big very fast, even when restricting to subsets of constant size. A common alternative is the Rips complex but it suffers similar difficulties. Over the last few years, there have been several approaches to building sparser complexes that still give good approximations to the persistent homology [21, 17, 11, 3, 2].

Our main contributions are the following.

1. A much simpler explanation for the construction and proof of correctness of sparse filtrations. Our new geometric construction shows that the sparse complex is just a nerve in one dimension higher.
2. The approach easily generalizes to Rips, Čech and related complexes (the offsets for any convex metric). This is another advantage of the geometric view as the main result follows from convexity rather than explicit construction of simplicial map homotopy equivalences.
3. A simple geometric proof that the explicit removal of vertices from the sparse filtration can be done with simple edge contractions. This can be done without resorting to the full-fledged zigzag persistence algorithm [5, 4, 18, 19] or even the full simplicial map persistence algorithm [11, 1].

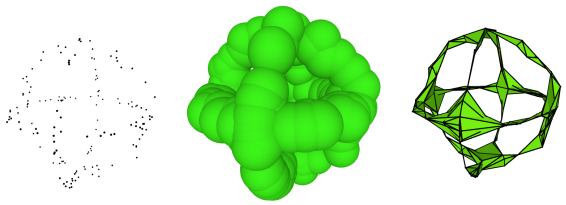


Figure 1: A point set sampled on a sphere, its offsets, and its (sparsified) nerve complex.

The most striking thing about this paper is perhaps more in what is absent than what is present. Despite giving a complete treatment of the construction, correctness, and approximation guarantees of sparse filtrations that applies to both Čech and Rips complexes, there is no elaborate construction of simplicial maps or proofs that they induce homotopy equivalences. In fact, we prove the results directly on the geometric objects, the covers, rather than the combinatorial objects, the complexes, and the result is much more direct. In a way, this reverses a common approach in computational geometry problems in which the geometry is as quickly as possible replaced with combinatorial structure; instead, we delay the transition from the offsets to a discrete representation until the very end of the analysis.

Related Work. Soon after the introduction of persistent homology by Edelsbrunner et al. [13], there was interest in building more elaborate complexes for larger and larger data sets. Following the full algebraic characterization of persistent homology by Zomorodian and

*Partially supported by the National Science Foundation under grant number CCF-1464379

†University of Connecticut nicholas.j.cavanna@uconn.edu
‡University of Connecticut reza@engr.uconn.edu
§University of Connecticut don.r.sheehy@gmail.com

Carlsson [23], a more general theory of zigzag persistence was developed [5, 4, 18, 19] using a more complicated algorithm. Zigzags gave a way to analyze spaces that did not grow monotonically; they could alternately grow and shrink such as by growing the scale and then removing points [22]. A variant of this techniques was first applied for specific scales by Chazal and Oudot in work on manifold reconstruction [9] and was implemented as a full zigzag by Morozov in his Dionysus library [12]. Later, Sheehy gave a zigzag for Rips filtrations that came with guaranteed approximation to the persistent homology of the unsparsified filtration [21]. Other later works gave various improvements and generalizations of sparse zigzags [20, 17, 11, 2].

2 Background

Distances and Metrics. Throughout, we will assume the input is a finite point set P in $\mathbb{R}^d$ endowed with some convex metric $\mathbf{d}$. A closed ball with center c and radius r will be written as $\text{ball}(c,r) = \{x \in \mathbb{R}^d | \mathbf{d}(x,c) \leq r\}$. For illustrative purposes, we will often draw balls as Euclidean (ℓ_2) balls.

For a non-negative $\alpha \in \mathbb{R}$, the *α-offsets* of P are defined as

$$P^\alpha := \bigcup_{p \in P} \text{ball}(p, \alpha).$$

The sequence of offsets as α ranges from 0 to ∞ is called the *offsets filtration* $\{P^\alpha\}$.

The *doubling dimension* of a metric space is $\log_2 \gamma$, where γ is the maximum over all balls B, of the minimum number of balls of half the radius of B required to cover B. Metric spaces with a small constant doubling dimension are called *doubling metrics*. Such metrics allow for packing arguments similar to those used in Euclidean geometry.

Simplicial Complexes. A *simplicial complex* K is a family of subsets of a vertex set that is closed under taking subsets. The sets $\sigma \in K$ are called *simplices* and $|\sigma| - 1$ is called the *dimension* of σ. A nested family of simplicial complexes is called a *simplicial filtration*. Often the family of complexes will be parameterized by a nonnegative real number as in $\{K^\alpha\}_{\alpha \geq 0}$. Here, the filtration property guarantees that $\alpha \leq \beta$ implies that $K^\alpha \subseteq K^\beta$. In this case, the value of α for which a simplex first appears is called its birth time, and so, if there is a largest complex K^α in the filtration, the whole filtration can be represented by K^α and the birth time of each simplex. For this reason, simplicial filtrations are often called *filtered simplicial complex*.

Persistent Homology. Homology is an algebraic tool for characterizing the connectivity of a space. It captures information about the connected components, holes, and voids. For this paper, we will only consider homology with field coefficients and the computations will all be on simplicial complexes. In this setting, computing homology is done by reducing a matrix D called the boundary matrix of the simplicial complex. The boundary matrix has one row and column for each simplex. If the matrix reduction respects the order of a filtration, i.e. columns are only combined with columns to their left, then the reduced matrix also represents the so-called *persistent homology* of the filtration. Persistent homology describes the changes in the homology as the filtration parameter changes and this information is often expressed in a *barcode* (See Fig. 2). Barcodes give topological signatures of a shape [14].

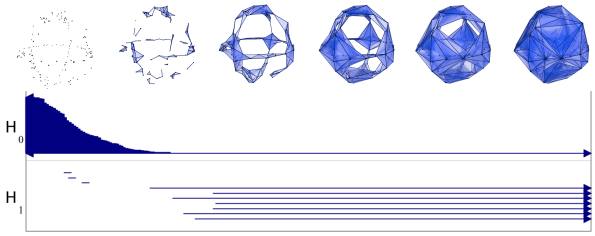


Figure 2: A filtration and its barcode.

Each bar of a barcode is an interval encoding the lifespan of a topological feature in the filtration. We say that a barcode B_1 is a (multiplicative) c-approximation to another barcode B_2 if there is a partial matching between B_1 and B_2 such that every bar $[b,d]$ with $d/b > c$ is matched and every matched pair of bars $[b,d],[b',d']$ satisfies $\max\{b/b', b'/b, d/d', d'/d\} \leq c$. A standard result on the stability of barcodes [8] implies that if two filtrations $\{F^\alpha\}$ and $\{G^\alpha\}$ are c-interleaved in the sense that $F^{\alpha/c} \subseteq G^\alpha \subseteq F^{c\alpha}$, then the barcode of $\{F^\alpha\}$ is a c-approximation to $\{G^\alpha\}$.

Nerve Complexes and Filtrations. Let $\mathcal{U} = \{U_1, \ldots, U_n\}$ be a collection of closed, convex sets. Let $\bigcup \mathcal{U}$ denote the union of the sets in $\mathcal{U}$, i.e. $\bigcup \mathcal{U} := \bigcup_{i=1}^n U_i$. We say that the set $\mathcal{U}$ is a *cover* of the space $\bigcup \mathcal{U}$. The *nerve* of $\mathcal{U}$, denoted $\text{Nrv}(\mathcal{U})$ is the abstract simplicial complex defined as

$$\text{Nrv}(\mathcal{U}) := \left\{ I \subseteq [n] \mid \bigcap_{i \in I} U_i \neq \emptyset \right\}.$$

This construction is illustrated in Fig.3. The Nerve Theorem [16, Cor. 4G.3] implies that $\text{Nrv}(\mathcal{U})$ is homotopy equivalent to $\bigcup \mathcal{U}$.

Similarly, one can construct a nerve filtration from a cover of a filtration by filtrations. Let $\mathcal{U} = \{\{U_1^\alpha\}, \ldots \{U_n^\alpha\}\}$ be a collection of filtrations parameterized by real numbers such that for each $i \in [n]$ and each $\alpha \geq 0$, the set U_i^α is closed and convex. Let $\mathcal{U}^\alpha$

Figure 3: The nerve has an edge for each pairwise intersection, a triangle for each 3-way intersection (right), etc.

to denote the set $\{U_1^\alpha, \ldots, U_n^\alpha\}$. As before, the Nerve Theorem implies that $\bigcup \mathcal{U}^\alpha$ is homotopy equivalent to $\mathrm{Nrv}(\mathcal{U}^\alpha)$. The Persistent Nerve Lemma [9] implies that the filtrations $\{\bigcup \mathcal{U}^\alpha\}_{\alpha \geq 0}$ and $\{\mathrm{Nrv}(\mathcal{U}^\alpha)\}_{\alpha \geq 0}$ have identical persistent homology.

Čech and Rips Filtrations. A common filtered nerve is the *Čech filtration*. It is defined as $\{\mathcal{C}_\alpha(P)\}$, where

$$\mathcal{C}_\alpha(P) := \mathrm{Nrv}\{\mathrm{ball}(p_i, \alpha) \mid i \in [n]\}.$$

Notice that this is just the nerve of the cover of the α-offsets by the α-radius balls. Thus, the Persistent Nerve Lemma implies that $\{P^\alpha\}$ and $\{\mathcal{C}_\alpha(P)\}$ have identical persistence barcodes.

A similar filtration that is defined for any metric is called the *(Vietoris-)Rips filtration* and is defined as $\{\mathcal{R}_\alpha(P)\}$, where

$$\mathcal{R}_\alpha(P) := \{J \subseteq [n] \mid \max_{i,j \in J} \mathbf{d}(p_i, p_j) \leq 2\alpha\}.$$

Note that if $\mathbf{d}$ is the max-norm, ℓ_∞, then $\mathcal{R}_\alpha(P) = \mathcal{C}_\alpha(P)$. Moreover, because every finite metric can be isometrically embedded into ℓ_∞, every Rips filtration is isomorphic to a nerve filtration.

Greedy Permutations. Let P be a set of points in some metric space with distance $\mathbf{d}$. A *greedy permutation* of P goes by many names, including landmark sets, farthest point sampling, and discrete center sets. We say that $P = \{p_1, \ldots, p_n\}$ is ordered according to a greedy permutation if each p_i is the farthest point from the first $i-1$ points. We let p_1 be any point. Formally, let $P_i = \{p_1, \ldots, p_i\}$ be the ith *prefix*. Then, the ordering is greedy if and only if for all $i \in \{2, \ldots, n\}$,

$$\mathbf{d}(p_i, P_{i-1}) = \max_{p \in P} \mathbf{d}(p, P_{i-1}).$$

For each point p_i, the value $\lambda_i := \mathbf{d}(p_i, P_{i-1})$ is known as the *insertion radius*. By convention, we set $\lambda_1 = \infty$. It is well-known (and easy to check) that P_i is a λ_i-net in the sense that it satisfies the conditions: for all distinct $p, q \in P_i$, $\mathbf{d}(p, q) \geq \lambda_i$ (**packing**) and $P \subseteq P_i^{\lambda_i}$ (**covering**).

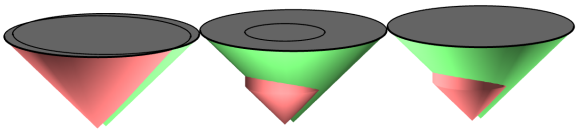


Figure 4: Left: two growing balls trace out cones in one dimension higher. Center: One of the cones has a maximum radius. Right: Limiting the height of one cone guarantees that the top is covered.

3 Perturbed Distances

A convenient first step in making a sparse version of the Čech filtration is to "perturb" the distance. Given a greedy permutation, we perturb the distance function so that as the radius increases, only a sparse subset of points continues to contribute to the offsets. This can most easily be viewed as changing the radius of the balls slightly so that some balls will be completely covered by their neighbors and thus will not contribute to the union. Fix a constant $\varepsilon < 1$ that will control the sparsity. As we will show in Lemma 1, at scale α, there is an $\varepsilon\alpha$-net of P whose perturbed offsets cover the perturbed offsets of P. Assuming the points $P = \{p_1, \ldots, p_n\}$ are ordered by a greedy permutation with insertion radii $\lambda_1, \ldots, \lambda_n$, we define the radius of p_i at scale α as

$$r_i(\alpha) := \begin{cases} \alpha & \text{if } \alpha \leq \lambda_i(1+\varepsilon)/\varepsilon \\ \lambda_i(1+\varepsilon)/\varepsilon & \text{otherwise.} \end{cases}$$

The *perturbed α-offsets* are defined as

$$\widetilde{P}^\alpha := \bigcup_{i \in [n]} \mathrm{ball}(p_i, r_i(\alpha)).$$

To realize the sparsification as described, we want to remove balls associated with some of the points as the scale increases. This is realized by defining the α-ball for a point $p_i \in P$ to be

$$b_i(\alpha) := \begin{cases} \mathrm{ball}(p_i, r_i(\alpha)) & \text{if } \alpha \leq \lambda_i(1+\varepsilon)^2/\varepsilon \\ \emptyset & \text{otherwise.} \end{cases}$$

The usefulness of this perturbation is captured by the following covering lemma, which is depicted in the tops of the cones in Fig. 4.

Lemma 1 (Covering Lemma) *Let $P = \{p_1, \ldots, p_n\}$ be a set of points ordered by a greedy permutation with insertion radii $\lambda_1, \ldots, \lambda_n$. For any $\alpha, \beta \geq 0$, and any $p_j \in P$, there exists a point $p_i \in P$ such that*

1. *if $\beta \geq \alpha$ then $b_j(\alpha) \subseteq b_i(\beta)$, and*
2. *if $\beta \geq (1+\varepsilon)\alpha$, then $\mathrm{ball}(p_j, \alpha) \subseteq b_i(\beta)$.*

Proof. Fix any $p_j \in P$. We may assume that $\beta \geq \lambda_j(1+\varepsilon)^2/\varepsilon$, for otherwise, choosing $p_i = p_j$ suffices to satisfy both clauses, the first because $b_j(\alpha) \subseteq b_j(\beta)$ and the second because $\text{ball}(p_j, \alpha) = b_j(\alpha) \subseteq b_j(\beta)$. This assumption is equivalent to the assumption that $b_j(\beta) = \emptyset$.

By the covering property of the greedy permutation, there is a point $p_i \in P$ such that $\mathbf{d}(p_i, p_j) \leq \varepsilon\beta/(1+\varepsilon)$ and $\lambda_i \geq \varepsilon\beta/(1+\varepsilon)$. It follows that $r_i(\beta) = \beta$ and $b_i(\beta) = \text{ball}(p_i, \beta)$. Recall that $\lambda_1 = \infty$ by convention, so $b_1(\beta) \neq \emptyset$, and for large values of β, choosing $p_i = p_1$ suffices.

To prove the first clause, fix any point $x \in b_j(\alpha)$. By the triangle inequality,

$$\begin{aligned}\mathbf{d}(x, p_i) &\leq \mathbf{d}(x, p_j) + \mathbf{d}(p_i, p_j) \leq r_j(\alpha) + \varepsilon\beta/(1+\varepsilon) \\ &\leq \lambda_j(1+\varepsilon)/\varepsilon + \varepsilon\beta/(1+\varepsilon) \leq \beta = r_i(\beta).\end{aligned}$$

So, $x \in b_i(\beta)$ and thus, $b_j(\alpha) \subseteq b_i(\beta)$ as desired.

To prove the second clause of the lemma, fix any $x \in \text{ball}(p_j, \alpha)$. By the triangle inequality,

$$\begin{aligned}\mathbf{d}(x, p_i) &\leq \mathbf{d}(x, p_j) + \mathbf{d}(p_i, p_j) \leq \alpha + \varepsilon\beta/(1+\varepsilon) \\ &\leq \beta/(1+\varepsilon) + \varepsilon\beta/(1+\varepsilon) = r_i(\beta).\end{aligned}$$

So, as before, $x \in b_i(\beta)$ and thus, $\text{ball}(p_j, \alpha) \subseteq b_i(\beta)$ as desired. □

Corollary 2 *Let $P = \{p_1, \ldots, p_n\}$ be a set of points ordered by a greedy permutation with insertion radii $\lambda_1, \ldots, \lambda_n$. For all $\alpha \geq 0$, $\widetilde{P}^\alpha = \bigcup_i b_i(\alpha)$ and $\widetilde{P}^\alpha \subseteq P^\alpha \subseteq \widetilde{P}^{(1+\varepsilon)\alpha}$.*

A proof may be found in the full paper [6]. Corollary 2 implies the following proposition using standard results on the stability of persistence barcodes [8].

Proposition 3 *The persistence barcode of the perturbed offsets $\{\widetilde{P}^\alpha\}_{\alpha\geq 0}$ is a $(1+\varepsilon)$-approximation to the persistence barcode of the offsets $\{P^\alpha\}_{\alpha\geq 0}$.*

4 Sparse Filtrations

The *sparse Čech complex* is defined as $Q^\alpha := \text{Nrv}\{b_i(\alpha) \mid i \in [n]\}$. Notice that because $b_i(\alpha) = \emptyset$ unless λ_i is sufficiently large compared to α, there are fewer vertices as the scale increases. This is the desired sparsification. Unfortunately, it means that the set of complexes $\{Q^\alpha\}$ is not a filtration, but this is easily remedied by the following definition. The *sparse Čech filtration* is defined as $\{S^\alpha\}$, where

$$S^\alpha := \bigcup_{\delta\leq\alpha} Q^\delta = \bigcup_{\delta\leq\alpha} \text{Nrv}\{b_i(\delta) \mid i \in [n]\}.$$

This definition makes it clear that the sparse complex is a union of nerves, but it not obvious that it has the same persistent homology as the filtration defined by the perturbed offsets $\widetilde{P}^\alpha := \bigcup_i b_i(\alpha)$. For such a statement, it would be much more convenient if $\{S^\alpha\}$ was itself a nerve filtration rather than a union of nerves, in which case the Persistent Nerve Lemma could be applied directly. In fact, this can be done by adding an extra dimension corresponding to the filtration parameter extending the balls $b_i(\alpha)$ into the perturbed cone shapes

$$U_i^\alpha := \bigcup_{\delta\leq\alpha} (b_i(\delta) \times \{\delta\}).$$

These sets, depicted in Figs. 4 and 5, allow the following equivalent definition of the complexes in the sparse Čech filtration.

$$S^\alpha := \text{Nrv}\{U_i^\alpha \mid i \in [n]\}.$$

Theorem 4 *The persistence barcode of the sparse nerve filtration $\{S^\alpha\}_{\alpha\geq 0}$ is a $(1+\varepsilon)$-approximation to the persistence barcode of the offsets $\{P^\alpha\}_{\alpha\geq 0}$.*

Proof. For all i, the set U_i^α is convex because r_i is concave (see the full paper [6] for a proof). It follows that the sets U_i^α satisfy the conditions of the Persistent Nerve Lemma. So, $\{S^\alpha\}$ has the same persistence barcode as the filtration $\{B^\alpha\}$, where $B^\alpha := \bigcup_i U_i^\alpha$.

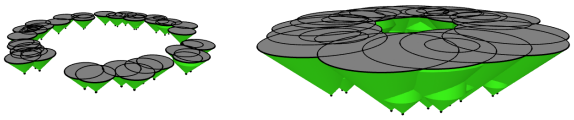


Figure 5: The collection of cones B^α at two different scales. The top of B^α is the union of (perturbed) balls.

The Covering Lemma implies that the linear projection of B^α to $\widetilde{P}^\alpha$ that maps (x, δ) to x is a homotopy equivalence as each fiber is simply connected. Moreover, the projection clearly commutes with the inclusions $B^\alpha \hookrightarrow B^\beta$ and $\widetilde{P}^\alpha \hookrightarrow \widetilde{P}^\beta$, from which, it follows that $\text{Pers}\{\widetilde{P}^\alpha\} = \text{Pers}\{B^\alpha\} = \text{Pers}\{S^\alpha\}$. So, the claim now follows from Proposition 3. □

5 Algorithms

In this section, we present an algorithm to construct the sparse filtration. In previous work, it was shown how to use metric data structures [15] to compute the sparse Rips filtration in $O(n \log n)$ time [21] when the doubling dimension is constant. The same approach also works for the sparse nerve filtrations described here. However, in our implementation, we found it to be efficient to construct the edge set in $O(n^2)$ time and then find the remaining simplices in linear time.

In order to find the edges in the sparse filtration, we consider every two points and determine whether their

corresponding balls have a common intersection. If the balls intersect, it returns the birth time of the corresponding edge and ∞ otherwise. We store every edge as a directed edge, which is used to find other k-simplices. Edges are directed from smaller to larger insertion radius. This will allow us to charge the simplices we find to their vertex of minimum insertion radius.

Let $E(v)$ be the vertices adjacent to a vertex v with larger insertion radius. To find a k-simplex for $k > 1$ containing a vertex v, we consider all subsets $\{u_1, \ldots, u_k\}$ of k vertices in $E(v)$. If $\{v, u_1, \ldots, u_k\}$ forms a $(k+1)$-clique, we check the clique to see whether it creates a k-simplex and compute its birth time. The birth time of a k-simplex σ in a nerve filtration is the minimum α such that $U_j^\alpha \neq \emptyset$. If no such α exists, then we define the birth time to be ∞. We assume the user provides a method, SimplexBirthTime, to compute birth times for their metric that runs in time polynomial in k. This function takes a $(k+1)$-clique as input. If at some scale α, the corresponding balls have a common intersection, it returns the minimum such α, otherwise, it returns ∞ indicating the $(k+1)$-clique is not a k-simplex in the sparse filtration.

For the case of Rips filtrations (i.e. ℓ_∞), SimplexBirthTime(σ) needs to compute the maximum birth time of the edges and compare it to $\min_{p_i \in \sigma} \lambda_i(1+\varepsilon)^2/\varepsilon$ (the first time t after which some $p_i \in \sigma$ has $b_i(t) = \emptyset$). For ℓ_2, the corresponding computation is a variation of the minimum enclosing ball problem.

Algorithm 1 finds the k-simplices and birth times in a sparse filtration. Here, $G = (V, E)$ is a directed graph, and the output S is the set of pairs (σ, t), where σ is a k-simplex with birth time t.

Algorithm 1 Find all k-simplices and birth times

```
1:  procedure FindSimplices(G = (V, E), k)
2:      S ← ∅
3:      for all v ∈ V do
4:          for all {u_1, ..., u_k} ⊆ E(v) do
5:              if {v, u_1, ..., u_k} is a (k+1)-clique then
6:                  σ ← {v, u_1, ..., u_k}
7:                  t ← SimplexBirthTime(σ)
8:                  if t < ∞ then
9:                      S ← S ∪ (σ, t)
10:     return S
```

Theorem 5 *Given the edges of a sparse filtration, Algorithm 1 finds the k-simplices of $\{S^\alpha\}$ in $((1+\varepsilon)^2/\varepsilon)^{O(k\rho)}n$ time, where ρ is the doubling dimension of the input metric.*

Proof. In Algorithm 1, for every vertex v in the directed graph G, there are $\binom{|E(v)|}{k}$ subsets with size k. Therefore, if we find an upper bound Δ on the number of adjacent vertices for all $v \in V$, the total running time of the algorithm will be $O(\Delta^k n)$.

In the directed graph G, a vertex p_j is adjacent to vertex p_i if the insertion radius of p_i is less than insertion radius of p_j and their corresponding balls intersect at some scale α, i.e. $b_i(\alpha) \cap b_j(\alpha) \neq \emptyset$. We know that $\lambda_i \leq \lambda_j$, and also they intersect before p_i disappears, so

$$b_i(\lambda_i(1+\varepsilon)^2/\varepsilon) \cap b_j(\lambda_i(1+\varepsilon)^2/\varepsilon) \neq \emptyset.$$

The distance between p_i and p_j is bounded as follows.

$$\begin{aligned}\mathbf{d}(p_i, p_j) &\leq r_i(\lambda_i(1+\varepsilon)^2/\varepsilon) + r_j(\lambda_i(1+\varepsilon)^2/\varepsilon) \\ &\leq \lambda_i(1+\varepsilon)/\varepsilon + \lambda_i(1+\varepsilon)^2/\varepsilon < 2\lambda_i(1+\varepsilon)^2/\varepsilon\end{aligned}$$

Thus, all adjacent vertices to p_i lie in a ball with center p_i and radius $2\lambda_i(1+\varepsilon)^2/\varepsilon$. Moreover, the insertion radii of the neighbors are all at least λ_i, so by a standard packing argument for doubling metrics, $|E(p_i)| = ((1+\varepsilon)^2/\varepsilon)^{O(\rho)}$. Consequently, the running time of this algorithm will be $((1+\varepsilon)^2/\varepsilon)^{O(k\rho)}n$. □

6 Removing Vertices

Because the sparse filtration is a true filtration, no vertices are removed. When the cone is truncated, no new simplices will be added using that vertex, but it is still technically part of the filtration. The linear-size guarantee is a bound on the total number of simplices in the complex. Thus, by using methods such as zigzag persistence or simplicial map persistence to fully remove these vertices when they are no longer needed cannot improve the asymptotic performance. Still, it may be practical to remove them (see [2]). A full theoretical or experimental analysis of the cost tradeoff of using a heavier algorithm to do vertex removal is beyond the scope of this paper.

In this section, we show that the geometric construction leads to a natural choice of elementary simplicial maps (edge collapses) which all satisfy the so-called link condition. In the persistence by simplicial maps work of Dey et al. [11] and Boissonat et al. [1], a key step in updating the data structures to contract an edge is to first add simplices so that the so-called Link Condition is satisfied. The *link* of a simplex σ in a complex K is defined as $\mathrm{Lk}\,\sigma = \{\tau \setminus \sigma \mid \tau \in K \text{ and } \sigma \subseteq \tau\}$. That is, the link σ is formed by removing the vertices of σ from each of its cofaces. An edge $\{u, v\} \in K$ satisfies the *Link Condition* if and only if

$$\mathrm{Lk}\,\{u, v\} = \mathrm{Lk}\,\{u\} \cap \mathrm{Lk}\,\{v\}.$$

Dey et al. [10] proved that edge contractions induce homotopy equivalences when the link condition is satisfied. Thus, it gives a minimal local condition to guarantee that the contraction preserves the topology. More recently, it was shown that such a contraction does not change the persistent homology [11].

Proposition 6 *If $(P, \mathbf{d})$ is a finite subset of a convex metric space and $\{S^\alpha\}$ is its corresponding sparse filtration, then the last vertex p_n has a neighbor p_i such that the edge $\{p_n, p_i\} \in S^\alpha$ satisfies the link condition, where $\alpha = \lambda_n(1+\varepsilon)^2/\varepsilon$ and λ_n is the insertion radius of p_n.*

Proof. It follows directly from the definition of a link that $\text{Lk}\,\{u,v\} \subseteq \text{Lk}\,\{u\} \cap \text{Lk}\,\{v\}$ for all edges $\{u,v\}$. By the Covering Lemma (Lemma 1), we know that there exists a $p_i \in P$ such that $b_n(\alpha) \subseteq b_i(\alpha)$. Thus, it suffices to check that $\text{Lk}\,\{i\} \cap \text{Lk}\,\{n\} \subseteq \text{Lk}\,\{i,n\}$. Because the vertices are ordered according to a greedy permutation, $\lambda_n \geq \lambda_j$ for all $p_j \in P$. It follows that a simplex $J \in S^\alpha$ if and only if $\bigcap_{i\in J} b_j(\alpha) \neq \emptyset$.

Let J be any simplex in $\text{Lk}\,\{i\} \cap \text{Lk}\,\{n\}$. So, $i, n \notin J$ and $\bigcap_{j\in J\cup\{n\}} b_j(\alpha) \neq \emptyset$. Because $b_n(\alpha) \cap b_i(\alpha) = b_n(\alpha)$, it follows that $\bigcap_{j\in J\cup\{i,n\}} b_j(\alpha) \neq \emptyset$. Thus, we have $J \in \text{Lk}\,\{i,n\}$ as desired. □

7 Conclusion

In this paper, we gave a new geometric perspective on sparse filtrations for topological data analysis that leads to a simple proof of correctness for all convex metrics. By considering a nerve construction one dimension higher, the proofs are primarily geometric and do not require explicit construction of simplicial maps. This geometric view clarifies the non-zigzag construction, while also showing that removing vertices can be accomplished with simple edge contractions.

References

[1] J.-D. Boissonnat, T. K. Dey, and C. Maria. The compressed annotation matrix: An efficient data structure for computing persistent cohomology. In *Algorithms – ESA 2013*, pages 695–706, 2013.

[2] M. B. Botnan and G. Spreemann. Approximating persistent homology in Euclidean space through collapses. *Applicable Algebra in Engineering, Communication and Computing*, pages 1–29, 2015.

[3] M. Buchet, F. Chazal, S. Y. Oudot, and D. R. Sheehy. Efficient and robust persistent homology for measures. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 168–180, 2015.

[4] G. Carlsson and V. de Silva. Zigzag persistence. *Foundations of Computational Mathematics*, 10(4):367–405, 2010.

[5] G. Carlsson, V. de Silva, and D. Morozov. Zigzag persistent homology and real-valued functions. In *Proceedings of the 25th ACM Symposium on Computational Geometry*, pages 247–256, 2009.

[6] N. J. Cavanna, M. Jahanseir, and D. R. Sheehy. A geometric perspective on sparse filtrations. 2015, arXiv:1506.03797.

[7] N. J. Cavanna, M. Jahanseir, and D. R. Sheehy. Visualizing sparse filtrations. In *Proceedings of the 31st Symposium on Computational Geometry (Multimedia Session)*, 2015.

[8] F. Chazal, D. Cohen-Steiner, M. Glisse, L. J. Guibas, and S. Y. Oudot. Proximity of persistence modules and their diagrams. In *Proceedings of the 25th ACM Symposium on Computational Geometry*, pages 237–246, 2009.

[9] F. Chazal and S. Y. Oudot. Towards persistence-based reconstruction in Euclidean spaces. In *Proceedings of the 24th ACM Symposium on Computational Geometry*, pages 232–241, 2008.

[10] T. K. Dey, H. Edelsbrunner, S. Guha, and D. V. Nekhayev. Topology preserving edge contraction. *Publications de l'Institut Mathematique (Beograd)*, 60:23–45, 1999.

[11] T. K. Dey, F. Fan, and Y. Wang. Computing topological persistence for simplicial maps. In *Proceedings of the 30th Annual Symposium on Computational Geometry*, pages 345–354, 2014.

[12] Dionysus. By Dmitriy Morozov (`http://www.mrzv.org/software/dionysus/`).

[13] H. Edelsbrunner, D. Letscher, and A. Zomorodian. Topological persistence and simplification. *Discrete & Computational Geometry*, 4(28):511–533, 2002.

[14] R. Ghrist. Barcodes: The persistent topology of data. *Bull. Amer. Math. Soc.*, 45(1):61–75, 2008.

[15] 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.

[16] A. Hatcher. *Algebraic Topology*. Cambridge University Press, 2001.

[17] M. Kerber and R. Sharathkumar. Approximate Čech complexes in low and high dimensions. In *24th International Symposium on Algorithms and Computation*, volume LNCS 8283, pages 666–676, 2013.

[18] C. Maria and S. Y. Oudot. Zigzag Persistence via Reflections and Transpositions. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, January 2015.

[19] N. Milosavljevic, D. Morozov, and P. Skraba. Zigzag persistent homology in matrix multiplication time. In *Proceedings of the 27th ACM Symposium on Computational Geometry*, 2011.

[20] S. Y. Oudot and D. R. Sheehy. Zigzag zoology: Rips zigzags for homology inference. *Foundations of Computational Mathematics*, pages 1–36, 2014.

[21] D. R. Sheehy. Linear-size approximations to the Vietoris-Rips filtration. *Discrete & Computational Geometry*, 49(4):778–796, 2013.

[22] A. Tausz and G. Carlsson. Applications of zigzag persistence to topological data analysis. arxiv:1108.3545, Aug 2011.

[23] A. Zomorodian and G. Carlsson. Computing persistent homology. *Discrete & Computational Geometry*, 33(2):249–274, 2005.

Online Packing of Equilateral Triangles

Shahin Kamali* Alejandro López-Ortiz* Zahed Rahmati*

Abstract

We investigate the online triangle packing problem in which a sequence of equilateral triangles with different sizes appear in an online, sequential manner. The goal is to place these triangles into a minimum number of squares of unit size. We provide upper and lower bounds for the competitive ratio of online algorithms. In particular, we introduce an algorithm which achieves a competitive ratio of at most 2.474. On the other hand, we show that no online algorithm can have a competitive ratio better than 1.509.

1 Introduction

The classic 1-dimensional bin packing problem asks for assignment of a set of items of different sizes into a minimum number of bins of unit capacity. For convenience, it is often assumed that bins have capacity 1 and items' sizes are in the range $(0, 1]$. In the online version, the items are revealed in a sequential manner, and an algorithm has to place an item into a bin without any information about forthcoming items. Online algorithms are often compared according to their competitive ratio, which is the maximum ratio between the cost of an online algorithm and that of an optimal offline algorithm, denoted by Opt, for serving the same sequence. For bin packing, we are particularly interested in *asymptotic* competitive ratio which only considers sequences for which the cost of Opt is arbitrary large.

Online bin packing has many applications in practice, from server consolidation to cutting stock. In the latter application, the goal is to cut patterns of given sizes from stocks of unit size. Clearly, this application can be extended into two dimensions. In the 2-dimensional bin packing problem, bins are typically squares of unit size while items are similar objects of different sizes. Two studied variations are box packing and square packing in which items are boxes (rectangles) and squares, respectively, of different sizes.

In this paper, we consider the *equilateral triangle packing problem*, which is stated as follows. The problem can be thought as a two-dimensional version of the classic bin packing problem.

Let $\sigma = \langle x_1, x_2, \ldots, x_n \rangle$ be an online sequence of equilateral triangles, where $x_i \in (0, 1.035]$ indicates the side length of the i^{th} equilateral triangle, $1 \leq i \leq n$. The online equilateral triangle packing problem is to place these equilateral triangles into a minimum number of squares of unit size so that no two triangles overlap. Upon receiving an equilateral triangle, an online algorithm makes an irrevocable decision for placing the triangle into a square. For that, the algorithm does not have any information about the (sizes of) forthcoming triangles. Triangles are allowed to be rotated.

The assumption $x_i \in (0, 1.035]$ comes from the fact that no equilateral triangle of length larger than $1/\cos\frac{\pi}{12} \approx 1.035$ fits in a square of unit size. In the rest of the paper by "triangle of size x" we mean "an equilateral triangle whose side length is x". We interchangeably use terms "bin" for "square of unit size", and "items" for incoming "equilateral triangles of different sizes".

Related work. The 1-dimensional bin packing problem has been extensively studied in the past few decades (see [2, 3] for excellent surveys). The most practical online algorithms are First Fit and Best Fit, which are greedy in the sense that they avoid opening a new bin if possible. The competitive ratio of both algorithms is 1.7 [11]. The Harmonic family of algorithms is based on classifying items by their sizes [12]. A member of this family is Harmonic++ of Seiden [13] with a competitive ratio of 1.588 which is the best among online bin packing algorithms.

For online square packing, the first set of results included an upper bound of 2.6875 and a lower bound of 4/3 [4]. The upper bound was later improved a few times ([5, 6, 9]). The best existing upper bound is given by an algorithm of competitive ratio 2.1187 [10]. In [14], a lower bound of 1.62176 was proved for the competitive ratio of any online algorithm. This lower bound was later improved to 1.6406 [6].

The best existing online algorithm for the box packing problem has a competitive ratio of 2.5545 [8] while there is a lower bound of 1.907 for the competitive ratio of any online box packing algorithm [1].

To our knowledge, there is no previous work addressing online packing of triangles. While we currently have no particular application in mind, we believe that triangle packing is of inherent and compelling interest.

*School of Computer Science, University of Waterloo
{s3kamali, alopez-o, zrahmati}@uwaterloo.ca

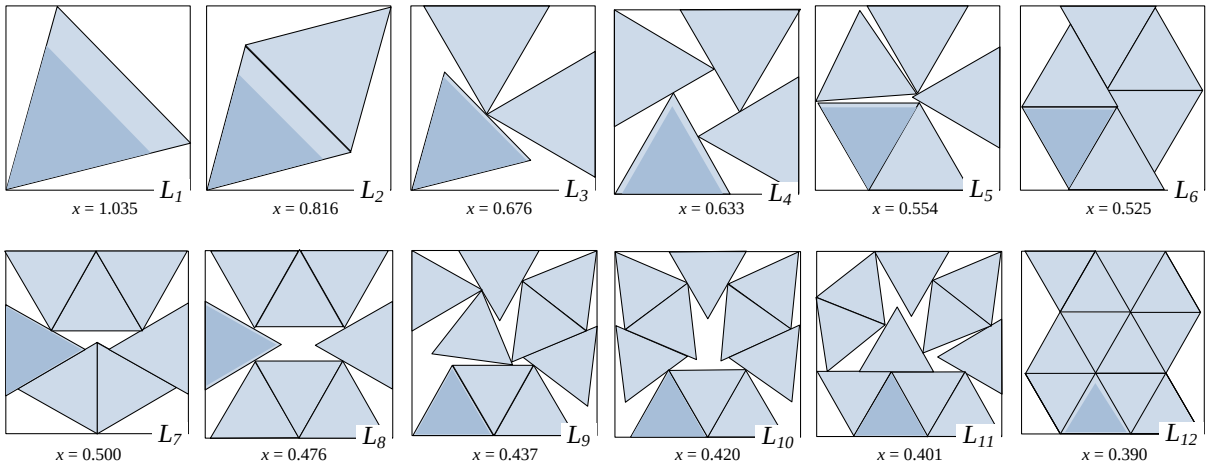


Figure 1: The triangle spots of a classification of large triangles of size x. The dark triangles indicate the lower bound for the size of the triangles.

Our contributions. We provide an online algorithm for packing equilateral triangles, which achieves a competitive ratio of 2.474. Our algorithm is *bounded-space* in the sense that it only keeps a constant number of squares opened at any given time. We prove a lower bound of 1.509 for *any* online equilateral triangle packing algorithm.

Our algorithm, in Section 2, classifies the triangles based on their sizes into different classes, and places triangles of the same class into the same bin. It requires a careful classification, which is more detailed compared to prior packing algorithms of boxes and squares.

For lower bound, in Section 3, we consider a sequence formed by triangles of three different sizes. To achieve a competitive ratio, we provide a linear program which captures the requirements for different subsequences of the original sequence.

2 Algorithm

In this section, we introduce our algorithm for packing of equilateral triangles, which has a competitive ratio of 2.474.

Similar to Harmonic family of algorithms, we classify triangles by their sizes. We refer to a triangle as being *large* if its size is larger than $1/3$, *medium* if its size is in the range $(1/20, 1/3]$ and *small* if it is at most $1/20$. Triangles that belong to any of these three groups are classified further into smaller classes, and members of each class are packed separately from others. In what follows, we describe the classification and packing for each group separately.

2.1 Large Triangles

The large triangles are classified into 12 groups, denoted by L_c, $c = 1, \ldots, 12$, based on their sizes. A large triangle of size x belongs to class L_c if at most c items of size x fit into a bin. Figure 1 shows the (best known) way of placing c ($1 \leq c \leq 12$) equal triangles of maximum size into a square [7]. If $0.816 < x \leq 1.035$, at most one item of size x can fit in a unit square and the triangle belongs to class L_1. If $0.676 < x \leq 0.816$, at most two items of size x fit in the same bin and it belongs to class L_2. Similarly, we can obtain the boundaries $(l_c, r_c]$ for any class L_c; see Table 1.

Triangles of each class are treated separately from other classes. For each class L_c ($1 \leq c \leq 12$) with boundaries $(l_c, r_c]$, the algorithm has at most one *active bin* of type c. When a bin of type L_c is opened, it is declared as the active bin of the class and c triangle spots of size r_c are reserved in that (this is feasible by definition of classes). Upon arrival of an item of type L_c, it is placed in one of the c spots of the active bin. If all these spots are occupied by previous items, a new bin of type L_c is opened. This ensures that all bins of type L_c, except potentially the current active bin, include c items of size greater than l_c.

2.2 Medium Triangles

The size of a medium triangle is in the range $(1/20, 1/3]$. Similar to that of Section 2.1, medium triangles are classified and items of each class are treated separately. Here, the classification is performed in a more regulated manner. We define 34 groups with ranges $(1/20, 1/19.5], (1/19.5, 1/19], \ldots, (1/3.5, 1/3]$ as boundaries of the classes. In particular, we say a triangle of

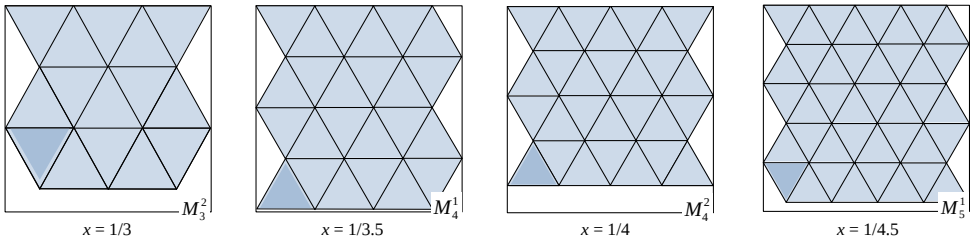


Figure 2: The triangle spots of a classification of medium triangles of size x. The dark triangles indicate the lower bound for the size of the triangles.

size x belongs to class M_k^1 if $x \in (1/k, 1/(k-0.5)]$, and belongs to class M_k^2 if $x \in (1/(k+0.5), 1/k]$, where $3 \leq k \leq 20$; classes M_3^1 and M_{20}^2 are not defined.

The item placement is similar to large triangles. As before, there is at most one active bin for each class L_c with boundaries $(l_c, r_c]$. When an active bin of type M_c is opened, X_c spots of size r_c are reserved in it. Upon arrival of an item of type M_c, it is placed in one of the spots in the active bin (and a new bin is opened if required). The following lemma gives the value of X_c.

Lemma 1 *It is possible to place $2k^2 - 2k$ triangles of class M_k^1 in the same bin. Similarly, $2k^2 - k$ triangles of class M_k^2 can be placed in the same bin ($3 \leq k \leq 20$).*

Proof. Consider a horizontal triangle strip constructed by r connected equilateral triangles of size x, for an integer $r \geq 5$. The vertical width of the triangle strip is the height of an equilateral triangle, *i.e.*, $\sqrt{3}x/2$. Assume the horizontal width of the triangle strip is 1. Therefore, $x = 2/(r+1)$.

For a triangle of size $x = 1/k$, which is associated with class M_k^2, we obtain $r = 2k - 1$. Note that, without overlappings, we can place at most k copies of the corresponding strip in a bin, where $k \geq 3$; see M_3^2 and M_4^2 in Figure 2. Therefore, there exist $2k^2 - k$ triangles of class M_k^2 in the same bin.

In a similar way, for a triangle with $x = 1/(k-0.5)$ of class M_k^1, we obtain $r = 2k - 2$. Since we can place at most k ($k \geq 4$) copies of the corresponding strip in a bin, there exist $2k^2 - 2k$ triangles of class M_k^1 in the same bin; see M_4^1 and M_5^1 in Figure 2. □

2.3 Small Triangles

A small triangle has size at most to 1/20. We maintain at most one active bin for placing small items. When a bin is opened for these items, we reserve four *triangle spots* of size 0.633, *i.e.*, the four triangles of class L_4 in Figure 1. These triangle spots are used as bins for placing small items.

Epstein and van Stee [5] provide an approach to assign small squares into unit squares. The algorithm is to find an appropriate sub-bin for an item, which can be found by partitioning a (sub-)bin into four identical sub-bins (see [5], for more details). A similar approach to that of Epstein and van Stee can be used for assigning equilateral triangles into unit equilateral triangle bins. The analysis of the algorithm for *the case when both the items and bins are equilateral triangles* is the same to *the case when both the items and bins are squares*. Therefore, we can use Claim 3 of [5] for our case. That is,

Claim 1 Given an online sequence of equilateral triangles of sizes no more than $1/M$, for some integer M, one can pack items into equilateral triangle bins of unit size so that the total occupied space in each bin is at least $\frac{M^2-1}{(M+1)^2}$.

Lemma 2 *The occupied area of each bin opened for small items, except possibly a constant number of them, is more than* 0.585.

Proof. Note that the side length ($\leq 1/20$) of a small triangle is within a factor of at most 1/12.66 of the side length (0.633) of the triangle spots. Assuming $M = 12$, by Claim 1, a fraction of at least $(12^2-1)/13^2 = 0.846$ of all triangle spots, except potentially a constant number of them, is occupied by small triangles.

Since the area of each bin covered by four triangle spots is $4 \times 0.173 = 0.692$ (see L_4 of Figure 1), the occupied area of each bin opened for small items is more than $0.692 \times 0.846 = 0.585$. □

2.4 Analysis

For analyzing our algorithm, we use a weighting argument. Corresponding to each triangle of size x, we define a weight $w(x)$ as follows.

Recall that a large triangle of a class L_c ($1 \leq c \leq 12$) is placed in a square which has c spots for items of this class. All bins opened for these triangles, except possibly the last active bin, include c items of this class. We define the weight of items of class c to be $1/c$. This

Class	Side length x	Occupied Area	Weight	Density
L_1	$(0.816, 1.035]$	$> 1(0.288) = 0.288$	1	< 3.472
L_2	$(0.676, 0.816]$	$> 2(0.197) = 0.395$	1/2	< 2.538
L_3	$(0.633, 0.676]$	$> 3(0.173) = 0.520$	1/3	< 1.926
L_4	$(0.554, 0.633]$	$> 4(0.132) = 0.531$	1/4	< 1.893
L_5	$(0.525, 0.554]$	$> 5(0.119) = 0.596$	1/5	< 1.680
L_6	$(0.500, 0.525]$	$> 6(0.108) = 0.649$	1/6	< 1.543
L_7	$(0.476, 0.500]$	$> 7(0.098) = 0.686$	1/7	< 1.457
L_8	$(0.437, 0.476]$	$> 8(0.082) = 0.661$	1/8	< 1.524
L_9	$(0.420, 0.437]$	$> 9(0.076) = 0.682$	1/9	< 1.461
L_{10}	$(0.401, 0.420]$	$> 10(0.069) = 0.696$	1/10	< 1.449
L_{11}	$(0.390, 0.401]$	$> 11(0.065) = 0.724$	1/11	< 1.398
L_{12}	$(1/3, 0.390]$	$> 12(0.048) = 0.577$	1/12	< 1.732
M_3^2	$(1/3.5, 1/3]$	$> 15(0.035) = 0.530$	1/15	< 1.886
M_4^1	$(1/4, 1/3.5]$	$> 24(0.027) = 0.649$	1/24	< 1.540
M_4^2	$(1/4.5, 1/4]$	$> 28(0.021) = 0.598$	1/28	< 1.672
M_5^1	$(1/5, 1/4.5]$	$> 40(0.017) = 0.692$	1/40	< 1.445
...	...	...	...	...
M_{19}^2	$(1/19.5, 1/19]$	$> 703(0.00113) = 0.800$	1/703	< 1.250
M_{20}^1	$(1/20, 1/19.5]$	$> 760(0.00108) = 0.822$	1/760	< 1.216
Small	$(0, 1/20]$	> 0.585	$\frac{\sqrt{3}/4x^2}{0.585}$	< 1.708

Table 1: A summary of classes: range of the side length x, minimum occupied area, weights, and densities.

way, the total weight of items in bins opened for large triangles, except possibly 12 of them, is exactly 1.

For medium triangles of class M_k^1, we define the weight to be $1/(2k^2 - 2k)$, where $4 \leq k \leq 20$. By Lemma 1, the algorithm places $2k^2 - 2k$ triangles of this class in each open bin (except possibly the last active bin), which implies the weight of all bins opened for this class is exactly 1. Similarly, the weight of triangles of class M_k^2 is defined as $1/(2k^2 - k)$, where $3 \leq k \leq 19$. This implies a weight of 1 for all bins (except possibly the last one) opened for this class. To summarize, the total weight of triangles in all bins opened for medium items, except possibly 34 of them (one for each class), is 1.

For a small triangle Δ of size x, we define its weight as $area(\Delta)/0.585$, where $area(\Delta) = \sqrt{3}x^2/4$ is the area of Δ. By Lemma 2, the occupied area of all bins opened for small items (except a constant number of them) is at least 0.585 which implies their total weight is at least $0.585/0585 = 1$.

Table 1 gives a summary of the weights of items in different classes. From the above argument, we conclude the following:

Lemma 3 *The total weight of triangles in each bin opened by the algorithm, except possibly a constant number of then, is at least 1.*

Next, we provide an upper bound for the total weight of items in a bin of the optimal offline algorithm (Opt).

Lemma 4 *The total weight of items in a bin of* Opt *is less than 2.474.*

Proof. Define the *density* of an item as the ratio between its weight and its area. An upper bound for the density of items of each class is reported in Table 1. For large and medium triangles, these values are simply the ratio between the weight and minimum area of items in each bin. For a small triangle Δ, the density is less than $1/0.585 = 1.708$, which is the ratio between the weight $((\sqrt{3}x^2/4)/0.585)$ and the area of Δ.

Note that the density of all triangles except those of types L_1 and L_2 cannot be more than 1.926 (by Table 1). In the following, we consider different cases, and show that no bin B of Opt can have a total weight more than 2.474.

First, assume there are no triangles of type L_1 or L_2 in B. The density of all triangles is less than 1.926, which implies the total weight of items in B is less than 1×1.926 (which is < 2.474).

Assuming there is no item of type L_1, there exist two cases: `(I)` If there are two items of type L_2, the total weight of these two items is $2 \times 1/2 = 1$. Since the remaining area is at most $1 - (2 \times 0.197) < 0.606$, the total weight of the items that fit in the remaining area would be at most $0.606 \times 1.926 < 1.168$. Therefore, the total weight of all items in B is less than $1+1.168$ (which is < 2.474). `(II)` If there is only one item of type L_2, it would have a weight of $1/2$ and the remaining area in B is at most $1 - 0.197 = 0.803$. The total weight of other items that fit in the remaining area of B would be at most $0.803 \times 1.926 < 1.547$. Therefore, the total weight of items in the bin will be less than $1/2 + 1.547$ (which is < 2.474).

Note that no two items of type L_1 fit in B. Assume there is one item of type L_1. The remaining area is at most $1 - 0.288 = 0.712$. In such case, we can only place at most one item of type L_2 in B. If there is no item of type L_2, the total weight of items that fit in the remaining area of B is at most $0.712 \times 1.926 < 1.372$. Thus the total weight of items in B would be less than $1 + 1.372$ (which is < 2.474).

The only remaining case is when B contains a triangle Δ_1 of type L_1 and a triangle Δ_2 of type L_2 (see Figure 3). Assume the remaining area is tightly covered by triangles of smaller sizes. It easy to check that when B contains Δ_1 and Δ_2, there is no way to place triangles of smaller sizes of class L_3 in B. The total weights of Δ_1 and Δ_2 is $1 + 1/2 = 1.5$. The remaining area in B has size at most $1 - 0.288 - 0.197 = 0.515$. Since the density of items (except those of types $L_1, \ldots, L_3$) is at most 1.893 (by Table 1), the total weight of the items would be at most $0.515 \times 1.893 < 0.974$. Therefore, the total weight of items in B is less than $1.5+0.974$ (which is equal to 2.474). □

Now, we give the main result of this section.

Theorem 5 *There exists an online algorithm for packing equilateral triangles into squares with a competitive ratio less than 2.474.*

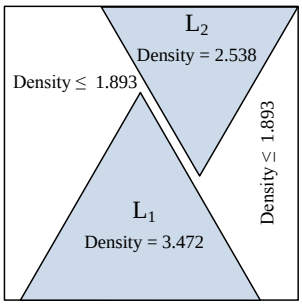


Figure 3: The maximum weight for a bin of OPT is achived when it contains an item of class L_1 and an item of class L_2. Other items that fit in the bin will have denisity smaller than 1.893.

Proof. For an input σ, let $A(\sigma)$ and $\text{OPT}(\sigma)$ denote the cost of the presented algorithm and OPT, respectively. Let $w(\sigma)$ denote the total weight of items of σ. Lemmas 3 and 4 imply that $A(\sigma) \leq w(\sigma) + c$, where c is a constant independent of the length of σ, and $\text{OPT}(\sigma) \geq w(\sigma)/2.474$. From these inequalities we conclude $A(\sigma) \leq 2.474\,\text{OPT}(\sigma) + c$, which proves an upper bound 2.474 for the competitive ratio of the algorithm. □

3 General Lower Bound

In this section, we show that no online algorithm for triangle packing can achieve a competitive ratio better than $80/53 \approx 1.509$. In our proof, we build sequences containing only triangles of sizes $x = 0.554$, $y = 0.676 + \epsilon$, and $z = 0.816 + \epsilon$, where ϵ is a sufficiently small constant. Note that triangles of size x are the largest triangles of the class L_5, and triangles of sizes y and z belong to the classes L_2 and L_1, respectively.

Let $\sigma = \sigma_1\sigma_2\sigma_3$ be a sequence of triangles, where σ_1, σ_2, and σ_3 are n replicas of the triangles of sizes x, y, and z, respectively. Assume n is sufficiently large. We compare the cost of any online algorithm A with that of OPT after serving sequences σ_1, $\sigma_1\sigma_2$, and $\sigma_1\sigma_2\sigma_3$.

Lemma 6 $\text{OPT}(\sigma_1) = n/5 + 1$, $\text{OPT}(\sigma_1\sigma_2) \leq n/2 + 1$, *and* $\text{OPT}(\sigma_1\sigma_2\sigma_3) \leq n + 1$.

Proof. For σ_1, OPT places five triangles in one bin; see Figure 4a. Thus each bin, except potentially the last one, contains five triangles, which gives a total of $n/5 + 1$ bins for placing σ_1. For $\sigma_1\sigma_2$, OPT can place two triangles of size x with two triangles of size y in the same bin (see Figure 4b), so all bins (except potentially the last one), contain four triangles, which gives a total of $2n/4+1$ bins. For $\sigma_1\sigma_2\sigma_3$, OPT can place one triangle of each size x, y, z in the same bin; see Figure 4c. Thus all bins include three triangles, which results a total of at most $3n/3 + 1$ bins for placing $\sigma_1\sigma_2\sigma_3$. □

Now, we obtain the main result of this section.

Theorem 7 *The competitive ratio of any online algorithm* A *for triangle packing is at least* $\frac{80}{53} \approx 1.509$.

Proof. Let a_{ij} be the number of bins which include i replicas of size x and j replicas of size y, where $1 \leq i \leq 5$ and $0 \leq j \leq 2$.

Denote by $\text{A}(\sigma_1)$ the number of bins opened by the algorithm A for σ_1. Note that if a square contains four or five triangles of size x, then it cannot contain a triangle of size y. Similarly, if a square contains three triangles of size x, then it cannot contain more than one triangle of size y. In summary,

$$\begin{aligned}\text{A}(\sigma_1) = \; & a_{10} + a_{11} + a_{12} + a_{20} + a_{21} + a_{22} + a_{30} + a_{31} \\ & + a_{40} + a_{50}.\end{aligned}$$

Assume A has a competitive ratio of at most r. By Lemma 6, for some constant c_1,

$$\text{A}(\sigma_1) \leq r \times n/5 + c_1. \tag{1}$$

By counting the number of triangles of size x, we obtain

$$\begin{aligned}a_{10} + a_{11} + a_{12} + 2a_{20} + 2a_{21} + 2a_{22} + 3a_{30} \\ + 3a_{31} + 4a_{40} + 5a_{50} = n.\end{aligned} \tag{2}$$

Let b_1 be the number of bins that include only one replica of size y, and also let b_2 be the number of bins that include only two replicas of size y and no replica of size x. Consider a packing of A after serving $\sigma_1\sigma_2$, meaning that A has placed n triangles of size x and n triangles of size y. In a similar way to that of $\text{A}(\sigma_1)$, we can obtain $\text{A}(\sigma_1\sigma_2)$, the number of bins opened by A for $\sigma_1\sigma_2$:

$$\begin{aligned}\text{A}(\sigma_1\sigma_2) = \; & a_{10} + a_{11} + a_{12} + a_{20} + a_{21} + a_{22} + a_{30} + a_{31} \\ & + a_{40} + a_{50} + b_1 + b_2,\end{aligned}$$

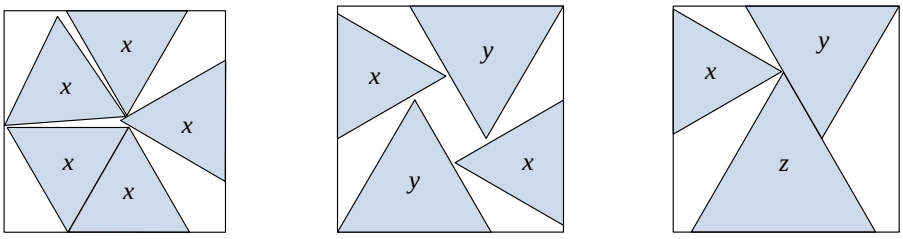
which can be bounded as follows (by Lemma 6, for some constant c_2):

$$\text{A}(\sigma_1\sigma_2) \leq r \times n/2 + c_2. \tag{3}$$

Counting the number of triangles of size y gives

$$a_{11} + 2a_{12} + a_{21} + 2a_{22} + a_{31} + b_1 + 2b_2 = n. \tag{4}$$

Now, consider a packing of A after placing triangles of size z. The algorithm A can place triangles of size z in bins which either `(I)` include at most two triangles of size x and no triangle of size y (*i.e.*, $a_{10} + a_{20}$ bins), or `(II)` include one triangle of size x and one triangle of size y (*i.e.*, a_{11} bins), or `(III)` include only one triangle



(a) A bin of $\textsc{Opt}(\sigma_1)$ (b) A bin of $\textsc{Opt}(\sigma_1\sigma_2)$ (c) A bin of $\textsc{Opt}(\sigma_1\sigma_2\sigma_3)$

Figure 4: Opt uses different packings for different prefix sequences of $\sigma_1\sigma_2\sigma_3$.

of size y (*i.e.*, b_1 bins). Except these bins, A has to open one bin for each triangle of size y, which implies $n - a_{10} - a_{20} - a_{11} - b_1$ new bins. In summary,

$$\begin{aligned}\mathrm{A}(\sigma_1\sigma_2\sigma_3) = \; & a_{12} + a_{21} + a_{22} + a_{30} + a_{31} + a_{40} + a_{50} \\ & + b_2 + n\end{aligned}$$

By Lemma 6, for some constant c_3, the number of bins opened by A for $\sigma_1\sigma_2\sigma_3$ bounds as follows:

$$\mathrm{A}(\sigma_1\sigma_2\sigma_3) \leq r \times n + c_3. \tag{5}$$

Equations 1-5 should hold for a competitive ratio of r. If we scale these equations by $1/n$, the constants c_1, c_2, and c_3 can be ignored. Let $\alpha_{ij} = a_{ij}/n$, $\beta_1 = b_1/n$, and $\beta_2 = b_2/n$. The following linear program summarizes the above discussion.

minimize r subject to

$$\alpha_{10} + \alpha_{11} + \alpha_{12} + \alpha_{20} + \alpha_{21} + \alpha_{22} + \alpha_{30} + \alpha_{31} + \alpha_{40} + \alpha_{50} \leq r/5;$$

$$\alpha_{10} + \alpha_{11} + \alpha_{12} + \alpha_{20} + \alpha_{21} + \alpha_{22} + \alpha_{30} + \alpha_{31} + \alpha_{40} + \alpha_{50} + \beta_1 + \beta_2 \leq r/2;$$

$$\alpha_{12} + \alpha_{21} + \alpha_{22} + \alpha_{30} + \alpha_{31} + \alpha_{40} + \alpha_{50} + \beta_2 + 1 \leq r;$$

$$\alpha_{10} + \alpha_{11} + \alpha_{12} + 2\alpha_{20} + 2\alpha_{21} + 2\alpha_{22} + 3\alpha_{30} + 3\alpha_{31} + 4\alpha_{40} + 5\alpha_{50} = 1;$$

$$\alpha_{11} + 2\alpha_{12} + \alpha_{21} + 2\alpha_{22} + \alpha_{31} + \beta_1 + 2\beta_2 = 1;$$

$$\alpha_{ij}, \beta_1, \beta_2 \geq 0.$$

This linear program obtains the optimal value $r = 80/53$. Therefore, we conclude the actual competitive ratio is at least $80/53$. □

References

[1] D. Blitz, A. van Vliet, and G. J. Woeginger. Lower bounds on the asymptotic worst-case ratio of online bin packing algorithms. Unpublished manuscript, 1996.

[2] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In D. Hochbaum, editor, *Approximation algorithms for NP-hard Problems*. PWS Publishing Co., 1997.

[3] E. G. Coffman Jr., J. Csirik, G. Galambos, S. Martello, and D. Vigo. Bin packing approximation algorithms: survey and classification. In P. M. Pardalos, D.-Z. Du, and R. L. Graham, editors, *Handbook of Combinatorial Optimization*, pages 455–531. Springer, 2013.

[4] D. Coppersmith and P. Raghavan. Multidimensional online bin packing: Algorithms and worst case analysis. *Oper. Res. Lett.*, 8:17–20, 1989.

[5] L. Epstein and R. van Stee. Optimal online bounded space multidimensional packing. In *Proc. 15th Symp. on Discrete Algorithms (SODA)*, pages 214–223, 2004.

[6] L. Epstein and R. van Stee. Online square and cube packing. *Acta Inform.*, 41(9):595–606, 2005.

[7] E. Friedman. Triangles in squares. URL: http://www.stetson.edu/ efriedma/triinsqu/.

[8] X. Han, F. Y. L. Chin, H.-F. Ting, G. Zhang, and Y. Zhang. A new upper bound 2.5545 on 2d online bin packing. *ACM Trans. Algorithms*, 7(4):1–18, Sept. 2011.

[9] X. Han, D. Ye, and Y. Zhou. Improved online hypercube packing. In *Proc. 4th International Workshop in Approximation and Online Algorithms (WAOA)*, pages 226–239, 2006.

[10] X. Han, D. Ye, and Y. Zhou. A note on online hypercube packing. *Cent. Eur. J. Oper. Res.*, 18(2):221–239, 2010.

[11] D. S. Johnson, A. J. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. Comput.*, 3:256–278, 1974.

[12] C.-C. Lee and D.-T. Lee. A simple online bin packing algorithm. *J. ACM*, 32:562–572, 1985.

[13] S. S. Seiden. On the online bin packing problem. *J. ACM*, 49:640–671, 2002.

[14] S. S. Seiden and R. van Stee. New bounds for multidimensional packing. *Algorithmica*, 36(3), 2003.

Relaxed Disk Packing*

Herbert Edelsbrunner† Mabel Iglesias-Ham‡ Vitaliy Kurlin§

Abstract

Motivated by biological questions, we study configurations of equal-sized disks in the Euclidean plane that neither pack nor cover. Measuring the quality by the probability that a random point lies in exactly one disk, we show that the regular hexagonal grid gives the maximum among lattice configurations.

Keywords. Packing and covering, disks, lattices, Voronoi domains, Delaunay triangulations.

1 Introduction

High-resolution microscopic observations of the DNA organization inside the nucleus of a human cell support the *Spherical Mega-base-pairs Chromatin Domain model* [3, 10]. It proposes that inside the chromosome territories in eukaryotic cells, DNA is compartmentalized in sequences of highly interacting segments of about the same length [4]. Each segment consists of roughly a million base pairs and resembles a round ball. The balls are tightly arranged within a restricted space, tighter than a packing since they are not rigid, and less tight than a covering to allow for external access to the DNA needed for gene expression.

Motivated by these biological findings, [8] considered configurations in which the overlap between the balls is limited and the quality is measured by the density, which we define as the expected number of balls containing a random point. We introduce a new measure that favors configurations between packing and covering without explicit constraints on the allowed overlap. Specifically, we measure a configuration by *the probability a random point is contained in exactly one ball*. Since empty space and overlap between disks are both discouraged, the optimum lies necessarily between packing and covering. The interested readers can find references for traditional packing and covering in two and higher dimensions in [2, 5]. In this paper, we restrict attention to equal-sized disks in the plane whose centers form a lattice, leaving three and higher dimensions as well as non-lattice configurations as open problems. Our main result is the following non-surprising fact.

Theorem 1 (Main) *Among all lattice configurations in $\mathbb{R}^2$, the regular hexagonal grid in which each disk overlaps the six neighboring circles in 30° arcs maximizes the probability that a random point lies exactly in one disk.*

For obvious reasons, we call this the 12-*hour clock configuration*. While preparing the final version of this paper, we have learnt that László Fejes Tóth in his book on *Regular Figures* introduced the same measure and conjectured that the 12-hour clock configuration is optimal among all configurations in the plane. He mentions that J. Balázs proved the conjecture for lattices [6, page 195]. Without finding any written record, we are unsure whether we rediscover Balázs' proof or we give a different proof for the same result. In any case, we present the proof in four sections: preparing the background in Section 2, showing an equilibrium condition in Section 3, developing the main argument in Section 4, and giving the technical details in Appendix A.

2 Background

In this section, we introduce notation for lattices, Voronoi domains, and Delaunay triangulations.

Lattices. Depending on the context, we interpret an element of $\mathbb{R}^2$ as a point or a vector in the plane. Vectors $a, b \in \mathbb{R}^2$ are *linearly independent* if $\alpha a + \beta b = 0$ implies $\alpha = \beta = 0$. A *lattice* is defined by two linearly independent vectors, $a, b \in \mathbb{R}^2$, and consists of all integer combinations of these vectors:

$$L(a, b) = \{ia + jb \mid i, j \in \mathbb{Z}\}. \quad (1)$$

Its *fundamental domain* is the parallelogram of points $\alpha a + \beta b$ with real numbers $0 \leq \alpha, \beta \leq 1$. Writing $\|a\|$ for the length of the vector and γ for the angle between a and b, the area of the fundamental domain is $\det L(a, b) = \|a\|\|b\| \sin(\gamma)$. The same lattice is generated by different pairs of vectors, and we will see shortly that at least one of these pairs defines a non-obtuse triangle. We will be more specific about this condition shortly, as it is instrumental in our proof of the optimality of the regular hexagonal grid.

Voronoi domain. Given a lattice L, the *Voronoi domain* of a point $p \in L$ is the set of points for which p is the closest:

$$V(p) = \{x \in \mathbb{R}^2 \mid \|x - p\| \leq \|x - q\|, \forall q \in L\}. \quad (2)$$

*All three authors acknowledge support from ESF under the ACAT Research Network Programme. The first author acknowledges partial support from the TOPOSYS project FP7-ICT-318493-STREP.

†IST Austria (Institute of Science and Technology Austria), Klosterneuburg, Austria, edels@ist.ac.at.

‡IST Austria (Institute of Science and Technology Austria), Klosterneuburg, Austria, mabel.iglesias-ham@ist.ac.at.

§Microsoft Research Cambridge and Mathematical Sciences, Durham University, United Kingdom, vitaliy.kurlin@gmail.com.

It is a convex polygon that contains p in its interior. Any two Voronoi domains have disjoint interiors but may intersect in a shared edge or a shared vertex. The lattice looks the same from every one of its points, which implies that all Voronoi domains are translates of each other: $V(p) = p + V(0)$. Similarly, central reflection through the origin preserves the lattice, which implies that $V(0)$ is centrally symmetric.

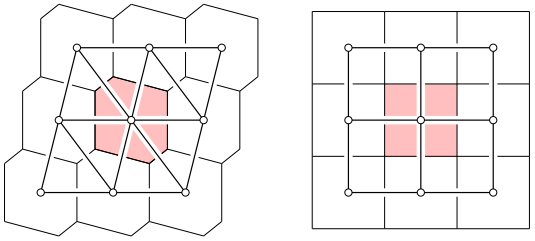


Figure 1: A primitive Voronoi diagram on the *left*, and a non-primitive Voronoi diagram on the *right*. The Delaunay triangulations are superimposed.

The *Voronoi diagram* of L is the collection of Voronoi domains of its points. It is *primitive* if the maximum number of Voronoi domains with non-empty common intersection is 3. In this case, the Voronoi domain is a centrally symmetric hexagon; see Figure 1. In the non-primitive case, there are generators that enclose a right angle, and the Voronoi domains are rectangles. To the first order of approximation, the area of any sufficiently simple and sufficiently large subset of $\mathbb{R}^2$ is the number of lattice points it contains times the area of $V(0)$. Similarly, it is the number of lattice points times the area of the fundamental domain. It follows that the area of $V(0)$ is equal to $\det L$.

Packing and covering. For $\varrho > 0$, we write $B(p, \varrho)$ for the closed disk with center p and radius ϱ. The *packing radius* is the largest radius, r_L, and the *covering radius* is the smallest radius, R_L, such that $B(0, r_L) \subseteq V(0) \subseteq B(0, R_L)$. The *density* of the configuration of disks with radius ϱ centered at the points of L is the area of a disk divided by the area of the Voronoi domain:

$$\delta_L(\varrho) = \tfrac{\varrho^2 \pi}{\det L}. \tag{3}$$

It is also the expected number of disks containing a random point in $\mathbb{R}^2$. The *packing density* is $\delta_L(r_L)$, which is necessarily smaller than 1. It is maximized by the regular hexagonal grid, H, for which we have $\delta_H(r_H) = 0.906\ldots$. The *covering density* is $\delta_L(R_L)$, which is necessarily larger than 1. It is minimized by the regular hexagonal grid for which we have $\delta_H(R_H) = 1.209\ldots$. More generally, it is known that H maximizes the density among all configuration of congruent disks whose interiors are pairwise disjoint [11], and it minimizes the density among all configurations that cover the entire plane [9]. Elegant proofs of both optimality results can be found in Fejes Tóth [5].

Delaunay triangulations. Drawing a straight edge between points p and q in L iff $V(p)$ and $V(q)$ intersect along a shared edge, we get the *Delaunay triangulation* of L. In the primitive case, the edges decompose the plane into triangles. Among the six triangles sharing 0 as a vertex, three are translates of each other and, going around 0, they alternate with their central reflections. It follows that all six triangles are congruent and, in particular, they have equally large circumcircles that all pass through 0. Since their centers are vertices of the Voronoi domain of 0, we have the following result.

Lemma 2 (Inscribed Voronoi Domain) *The vertices of $V(0)$ all lie on the circle bounding $B(0, R_L)$.*

The discussion above proves the Inscribed Voronoi Domain Lemma in the primitive case. It is also true in the simpler, non-primitive case in which $V(0)$ is a rectangle. Returning to the primitive case, we note that the two angles opposite to a shared edge in the Delaunay triangulation add up to less than $180°$. In a lattice, these two angles are the same and therefore both acute. The two types of triangles in the Delaunay triangulation of a lattice can be joined across a shared edge in three different ways. We can therefore make the same argument three times and conclude that all angles are less than $90°$. A slightly weaker bound holds in the non-primitive case.

Lemma 3 (Non-obtuse Generators) *Every lattice L in $\mathbb{R}^2$ has vectors $a, b \in \mathbb{R}^2$ with $L = L(a, b)$ such that*

(i) in the primitive case $0, a, b$ are the vertices of an acute triangle,

(ii) in the non-primitive case $0, a, b$ are the vertices of a non-obtuse triangle with a right angle at 0.

Assuming a, b satisfy the Non-obtuse Generators Lemma, the triangle $0ab$ has edges of length $\|a\|$, $\|b\|$, and $\|c\| = \|a - b\|$. In the non-primitive case (ii), $\|a\|$ and $\|b\|$ are the lengths of the sides of the rectangle $V(0)$, and $\|c\|$ is the length of a diagonal. We have $\|c\|^2 = \|a\|^2 + \|b\|^2$, and therefore $\|a\| \leq \|b\| \leq \|c\|$, possibly after swapping a and b. In the primitive case (i), we can choose a, b, and $c = a - b$ such that $\|a\| \leq \|b\| \leq \|c\|$. In this case, $V(0)$ is a centrally symmetric hexagon with distances $\|a\|, \|b\|, \|c\|$ between antipodal edge pairs.

3 Equilibrium Configurations

Given a lattice in $\mathbb{R}^2$, we are interested in the radius of the disks for which the probability that a random point lies inside exactly one disk is maximized. Further maximizing this probability over all lattices, we get the main result of this paper.

Partial disks. Fix a lattice L in $\mathbb{R}^2$. For a radius $\varrho > 0$, consider the set of points that belong to the disk centered at the origin but not to any other disk centered at a point of L:

$$D(0,\varrho) = B(0,\varrho) \setminus \bigcup_{0 \neq p \in L} B(p,\varrho). \tag{4}$$

As illustrated in Figure 2, for radii strictly between the packing radius and the covering radius, this set is partially closed and partially open. We distinguish between the *convex boundary* that belongs to the circle bounding $B(0,\varrho)$, and the *concave boundary* that belongs to other circles:

$$\partial_x D(0,\varrho) = \partial B(0,\varrho) \cap D(0,\varrho), \tag{5}$$
$$\partial_v D(0,\varrho) = \partial D(0,\varrho) \setminus \partial_x D(0,\varrho). \tag{6}$$

We note that $\partial_x D(0,\varrho) = \partial B(0,\varrho) \cap V(0)$. By the Inscribed Voronoi Domain Lemma, the vertices of $V(0)$ are all at the same distance from 0. This implies that for $r_L < \varrho < R_L$, the convex boundary consists of 2, 4, or 6 circular arcs that alternate with the same number of circular arcs in the concave boundary.

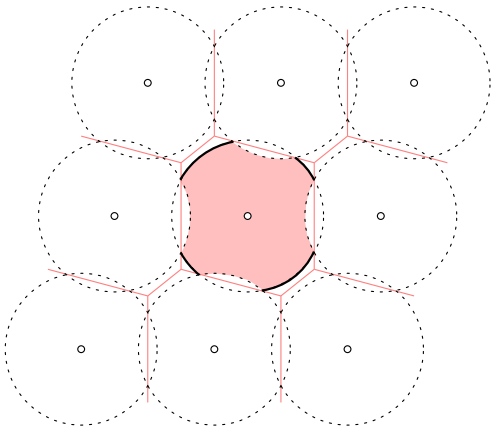


Figure 2: The shaded partial disk $D(0,\varrho)$ with its boundary divided into a solid convex portion and a dotted concave portion.

Angles. Recall that the concave boundary consists of at most three pairs of arcs, and let $\varphi_i(\varrho)$ be the angle of each of the two arcs in the i-th pair, for $i = 1, 2, 3$. The total angle of the concave boundary is

$$\Phi_L(\varrho) = \sum_{i=1}^{3} 2\varphi_i(\varrho), \tag{7}$$

and the total angle of the convex boundary is $2\pi - \Phi_L(\varrho)$. We have $\Phi_L(r_L) = 0$ and $\Phi_L(R_L) = 2\pi$, and between these two limits, the function is continuous and monotonically increasing.

Lemma 4 (Monotonicity) *Let L be a lattice in $\mathbb{R}^2$. Then $\Phi_L\colon [r_L, R_L] \to [0, 2\pi]$ is continuous, with $\Phi_L(\varrho_1) < \Phi_L(\varrho_2)$ whenever $\varrho_1 < \varrho_2$.*

Proof. The continuity of the function follows from the fact that $\partial B(0,\varrho)$ intersects $\partial V(0)$ in at most a finite number of points.

To prove monotonicity, we recall that $2\pi - \Phi_L(\varrho)$ is the total angle of $\partial_x D(0,\varrho) = \partial B(0,\varrho) \cap V(0)$. The Voronoi domain is a convex polygon with $0 \in V(0)$. Drawing circles with radii $\varrho_1 < \varrho_2$ centered at 0, we let $0 \le \theta < 2\pi$ and write $p_1(\theta)$ and $p_2(\theta)$ for the points on the circles in direction θ. Either both points belong to $V(0)$, both points do not belong to $V(0)$, or $p_1(\theta) \in V(0)$ but $p_2(\theta) \notin V(0)$. The fourth combination is not possible, which implies $2\pi - \Phi_L(\varrho_1) \ge 2\pi - \Phi_L(\varrho_2)$ or, equivalently, $\Phi_L(\varrho_1) \le \Phi_L(\varrho_2)$. To prove the strict inequality, we just need to observe that there is an arc of non-zero length in $\partial_x D(0,\varrho_1)$ such that the corresponding arc in $\partial B(0,\varrho_2)$ lies outside $V(0)$ and therefore does not belong to $\partial_x D(0,\varrho_2)$. □

Area. The probability that a random point belongs to exactly one disk is the area of $D(0,\varrho)$ over the area of $V(0)$. The latter is a constant independent of the radius. We will prove shortly that the former is a unimodal function in ϱ with a single maximum at the radius $\varrho = \varrho_L$ that balances the lengths of the two kinds of boundaries; see Figure 3. We call ϱ_L the *equilibrium radius* of L. Write $A_L\colon [r_L, R_L] \to \mathbb{R}$ for the function that maps ϱ to the area of $D(0,\varrho)$.

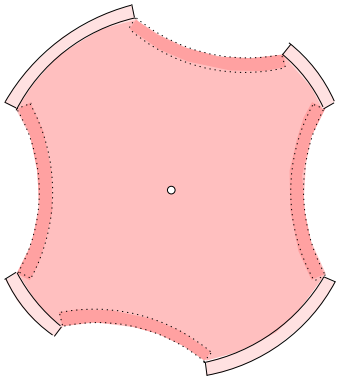


Figure 3: Increasing the radius grows $D(0,\varrho)$ along the convex boundary and shrinks it along the concave boundary.

Lemma 5 (Equilibrium Radius) *Let L be a lattice in $\mathbb{R}^2$. The function $A_L\colon [r_L, R_L] \to \mathbb{R}$ is strictly concave, with a unique maximum at the equilibrium radius ϱ_L that satisfies $\Phi_L(\varrho_L) = \pi$.*

Proof. Recall that $\Phi_L(\varrho)$ is the total angle of the concave boundary of $D(0,\varrho)$, and $2\pi - \Phi_L(\varrho)$ is the total angle of the convex boundary. When we increase the radius, the partial disk grows along the convex boundary and shrinks along the concave boundary. Indeed, the derivative is the difference between the two lengths:

$$\tfrac{\partial A_L}{\partial \varrho}(\varrho) = \varrho[2\pi - \Phi_L(\varrho) - \Phi_L(\varrho] \tag{8}$$
$$= 2\varrho[\pi - \Phi_L(\varrho)]. \tag{9}$$

The derivative vanishes when $\Phi_L(\varrho) = \pi$, is positive when $\Phi_L(\varrho) < \pi$ and negative when $\Phi_L(\varrho) > \pi$, as claimed. □

4 Optimality of the Regular Hexagonal Grid

In this section, we present the proof of our main result. After writing the probability that a random point lies in exactly one disk as a function of the radius, we distinguish between three cases, showing that the maximum is attained at the regular hexagonal grid.

Probability. Given a lattice L in $\mathbb{R}^2$, we write $r_L < \varrho_L < R_L$ for the packing, equilibrium, and covering radii. Recall that the probability in question is

$$P_L(\varrho_L) = \tfrac{A_L(\varrho_L)}{\|a\|\|b\|\sin\gamma}, \tag{10}$$

in which $L = L(a,b)$ and γ is the angle between a and b. Recall furthermore that the convex boundary consists of at most six arcs, two each with angle φ_1, φ_2, φ_3, in which we set the angle to zero if the arc degenerates to a point or is empty.

Lemma 6 (Equilibrium Area) *Let $L = L(a,b)$ be a lattice in $\mathbb{R}^2$ with angle γ between a and b. Then*

$$P_L(\varrho_L) = \tfrac{2\varrho_L^2}{\|a\|\|b\|\sin\gamma} \cdot \sum_{i=1}^{3} \sin\varphi_i. \tag{11}$$

Proof. Recall that $\|a\|\|b\|\sin\gamma$ is the area of $V(0)$. Let A_{in} be the area of $B(0,\varrho_L) \cap V(0)$, let A_{out} be the area of $B(0,\varrho_L) \setminus V(0)$, and note that $A_{\text{in}} - A_{\text{out}}$ is the area of $D(0,\varrho_L)$. Since $A_{\text{in}} + A_{\text{out}} = \varrho_L^2\pi$, we have $A_{\text{in}} - A_{\text{out}} = \varrho_L^2\pi - 2A_{\text{out}}$. The portion of $B(0,\varrho_L)$ outside the Voronoi domain consists of up to three symmetric pairs of disk segments, with total area

$$A_{\text{out}} = 2\sum_{i=1}^{3} \tfrac{\varrho_L^2}{2}(\varphi_i - \sin\varphi_i) \tag{12}$$

$$= \varrho_L^2 \left(\tfrac{\pi}{2} - \sum_{i=1}^{3} \sin\varphi_i \right), \tag{13}$$

in which the second line is obtained using $\sum_{i=1}^{3} \varphi_i = \frac{\pi}{2}$ from the Equilibrium Radius Lemma. The probability is $A_{\text{in}} - A_{\text{out}}$ divided by the area of the Voronoi domain:

$$P_L(\varrho_L) = \tfrac{\varrho_L^2\pi - 2A_{\text{out}}}{\|a\|\|b\|\sin\gamma}. \tag{14}$$

Together with (13) this implies the claimed relation. □

Case analysis. We focus on the primitive case in which the Voronoi domain is a hexagon, considering the non-primitive case a limit situation in which two of the edges shrink to zero length. Let $a, b \in \mathbb{R}^2$ be generators of the lattice satisfying the condition in the Non-obtuse Generators Lemma, set $c = a - b$, and assume $\|a\| \le \|b\| \le \|c\|$. Recall that these three lengths are the distances between parallel edges of the hexagon. Further notice that we have $r_L = \frac{\|a\|}{2}$ for the packing radius and $R_L > \frac{\|c\|}{2}$ for the covering radius. As before, we write φ_i for the angles of the arcs of $\partial_x D(0,\varrho)$, and we index such that $\varphi_1 \ge \varphi_2 \ge \varphi_3$.

CASE 1: $\frac{\|a\|}{2} < \varrho_L \le \frac{\|b\|}{2}$. Then $\varphi_1 > 0$ and $\varphi_2 = \varphi_3 = 0$.

CASE 2: $\frac{\|b\|}{2} < \varrho_L \le \frac{\|c\|}{2}$. Then $\varphi_1 \ge \varphi_2 > 0$ and $\varphi_3 = 0$.

CASE 3: $\frac{\|c\|}{2} < \varrho_L < R_L$. Then $\varphi_1 \ge \varphi_2 \ge \varphi_3 > 0$.

For example the configuration depicted in Figure 2 falls into Case 2. Using the expression for the probability in the Equilibrium Area Lemma, we determine the maximum for each of the three cases. Here we state the results, referring to Appendix A for the proofs. By *the probability* we mean of course the probability that a random point belongs to exactly one disk.

Lemma 7 (Two Arcs) *In Case 1, the maximum probability is attained for $\|b\| = \sqrt{2}\|a\|$, $\gamma = \arccos\frac{1}{2\sqrt{2}}$, and $\varrho_L = \|a\|/\sqrt{2}$, which gives $P_L(\varrho_L) = 0.755\ldots$.*

Lemma 8 (Four Arcs) *In Case 2, the maximum probability is attained for $\|b\| = \|a\|$, $\gamma = \arccos(\sqrt{2}-1)$, and $\varrho_L = \|a\|\sqrt{1 - 1/\sqrt{2}}$, which gives $P_L(\varrho_L) = 0.910\ldots$.*

Lemma 9 (Six Arcs) *In Case 3, the maximum probability is attained for $\|b\| = \|a\| = \|c\|$, $\gamma = \frac{\pi}{3}$ and $\varrho_L = \|a\|/(2\cos\frac{\pi}{12})$, which gives $P_L(\varrho_L) = 0.928\ldots$.*

Note that the lattice in the Six Arcs Lemma is the regular hexagonal grid. Comparing the three maximum probabilities, we see that the regular hexagonal grid gives the global optimum; see Figures 4. For this lattice, we get ϱ_H such that

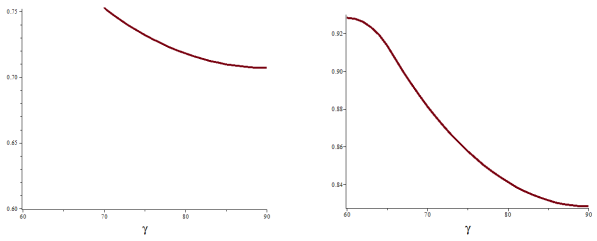


Figure 4: *Left:* for $\sqrt{2}\|a\| = \|b\| \le \|c\|$, the angle γ is between $\arccos\frac{1}{2\sqrt{2}}$ and $\frac{\pi}{2}$. *Right:* for $\|a\| = \|b\| \le \|c\|$, the angle γ is between $\frac{\pi}{3}$ and $\frac{\pi}{2}$. In both cases, the probability increases as the angle decreases, attaining its maximum at the minimum angle.

each disk overlaps with six others and in each case covers $30°$ of the bounding circle: the 12-hour clock configuration in the plane. This implies the Main Theorem stated in Section 1. We further illustrate the result by showing the graph of the function that maps $\|a\|/\|b\|$ and the angle γ to the probability at the equilibrium radius; see Figure 5.

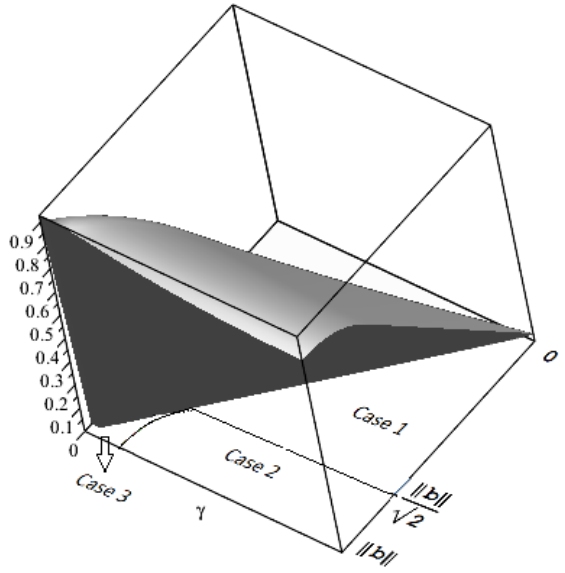


Figure 5: The graph of the function that maps $0 \leq \|a\|/\|b\| \leq 1$ and $\arccos \frac{\|a\|}{2\|b\|} \leq \gamma \leq 90^\circ$ to the probability that a random point lies in exactly one disk. The thus defined domain resembles a triangle and decomposes into three regions corresponding to Cases 1, 2, 3. The regular hexagonal grid is located at the lower left corner of the domain.

5 Discussion

The main result of this paper is a proof that the 12-hour clock configuration of disks in the plane maximizes the probability that a random point lies in exactly one of the disks. Other criteria favoring configurations between packing and covering can be formulated, see [8], and it would be interesting to decide which one fits the biological data about DNA organization within the nucleus best. There are also concrete mathematical questions related to the work in this paper:

- Is the 12-hour clock configuration optimal among all configurations of congruent disks in the plane?
- What is the optimal lattice configuration of balls in $\mathbb{R}^3$?

To appreciate the difficulty of the second question, we note that the FCC lattice gives the densest packing [7], while the BCC lattice gives the sparsest covering [1]. Does one of them also maximize the probability that a random point lies inside exactly one ball?

Acknowledgements

The authors thank Michael Kerber for help in proof-checking the maple file that supports the computations in this paper.

References

[1] R.P. BAMBAH. On lattice coverings by spheres. *Proc. Natl. Inst. Sci. India* **20** (1954), 25–52.

[2] J.H. CONWAY AND N.J.A. SLOANE. *Sphere Packings, Lattices and Groups.* Third edition, Springer-Verlag, New York, New York, 1999.

[3] T. CREMER, G. KRETH, H. KOESTER, R.H.A. FINK, R. HEINTZMANN, M. CREMER, I. SOLOVEI, D. ZINK AND C. CREMER. Chromosome territories, interchromatin domain compartment, and nuclear matrix: An integrated view of the functional nuclear architecture. *Critical Reviews in Eukaryotic Gene Expression* **10** (2000) 179–212.

[4] J. DIXON, S. SELVARAJ, F. YUE, A. KIM, Y. LI, Y. SHEN, M. HU, J. LIU AND B. REN. Topological domains in mammalian genomes identified by analysis of chromatin interactions. *Nature* **485** (2012) 376–80.

[5] L. FEJES TÓTH. *Lagerungen in der Ebene, auf der Kugel und im Raum.* Grundlehren der mathematischen Wissenschaften **65**, Springer, Berlin, Germany, 1953.

[6] L. FEJES TÓTH. *Regular Figures.* Macmillan, New York, New York, 1964.

[7] C.F. GAUSS. Untersuchungen über die Eigenschaften der positiven ternären quadratischen Formen von Ludwig August Seeber. *Göttingische Gelehrte Anzeigen* (1831), reprinted in *Werke II*, Königliche Gesellschaften der Wissenschaften (1863), 188–196.

[8] M. IGLESIAS-HAM, M. KERBER AND C. UHLER. Sphere packing with limited overlap. Online Proceedings of Canad. Conf. Comput. Geom., 2014.

[9] R. KERSHNER. The number of circles covering a set. *Amer. J. Math.* **61** (1939), 665–671.

[10] G. KRETH, P. EDELMANN AND C. CREMER. Towards a dynamical approach for the simulation of large scale, cancer correlated chromatin structures. *Supplement of Italian Journal of Anatomy and Embryology* **106** (2001) 21–30.

[11] A. THUE. Über die dichteste Zusammenstellung von kongruenten Kreisen in einer Ebene. *Norske Vid. Selsk. Skr.* **1** (1910), 1–9.

[12] MAPLE 17. Maplesoft, a division of Waterloo Maple Inc., Waterloo, Ontario.

A Proofs

In this appendix, we give detailed proofs of the three Arc Lemmas. As described in Section 4, the three lemmas add up to a proof of the Main Theorem stated in Section 1 of this paper. We begin with a few relations that will be useful in all three proofs. Given a triangle with edges of lengths $\|a\|, \|b\|, \|c\|$ and angle γ opposite the edge c, the *law of cosines* implies

$$\|c\|^2 = \|a\|^2 + \|b\|^2 - 2\|a\|\|b\| \cos\gamma. \tag{15}$$

Assuming $\|a\| \leq \|b\| \leq \|c\|$, the angle γ is at least as large as each of the other two angles. From (15) together with $\|b\| \leq \|c\|$, we get $\cos\gamma \leq \frac{\|a\|}{2\|b\|}$. As justified by the Non-obtuse Generators Lemma, we may assume the triangle is non-obtuse, which implies

$$\arccos \tfrac{\|a\|}{2\|b\|} \leq \gamma \leq \tfrac{\pi}{2}. \tag{16}$$

The range of possible angles is largest for $\|a\| = \|b\|$ where we get $60^\circ \leq \gamma \leq 90^\circ$. Furthermore, we write the angles φ_i of the arcs in the convex boundary of the partial disk in terms of the edge lengths and the radius:

$$\cos\tfrac{\varphi_1}{2} = \tfrac{\|a\|}{2\varrho}, \tag{17}$$

$$\cos\tfrac{\varphi_2}{2} = \tfrac{\|b\|}{2\varrho}, \tag{18}$$

$$\cos\tfrac{\varphi_3}{2} = \tfrac{\|c\|}{2\varrho}. \tag{19}$$

The first relation holds provided $\frac{\|a\|}{2} \leq \varrho < R_L$, and similar for the second and third relations. Finally, we note that scaling does not affect the density of a configuration. We can therefore set $\|b\| = 1$, which we will do to simplify computations.

Proof of the Two Arcs Lemma. Case 1 is defined by $\frac{\|a\|}{2} < \varrho_L \leq \frac{\|b\|}{2}$, which implies $\varphi_1 > 0$ and $\varphi_2 = \varphi_3 = 0$. Since $\sin\varphi_2 = \sin\varphi_3 = 0$, the probability at the equilibrium radius simplifies to

$$P_L(\varrho_L) = \tfrac{2\varrho_L^2}{\|a\|\|b\|\sin\gamma} \cdot \sin\varphi_1 \tag{20}$$

$$= \tfrac{\|a\|}{\|b\|\sin\gamma}, \tag{21}$$

where we get the second line by combining $\varphi_1 = \frac{\pi}{2}$ with (17) to imply $\varrho_L = \|a\|/\sqrt{2}$. For the remainder of this proof, we normalize by setting $\|b\| = 1$. To maximize the probability, we choose $\|a\|$ as large as possible and γ as small as possible. From $\|a\|/\sqrt{2} = \varrho_L \leq \frac{1}{2}$, we get $\|a\| \leq 1/\sqrt{2}$, and from $\|b\| \leq \|c\|$ we get $\gamma \geq \arccos\frac{1}{2\sqrt{2}}$. The two parameters can be optimized simultaneously, which gives

$$P_L(\varrho_L) = \frac{1}{\sqrt{2}\sin\left(\arccos\frac{1}{2\sqrt{2}}\right)} = 0.755\ldots. \tag{22}$$

Proof of the Four Arcs Lemma. Case 2 is defined by $\frac{\|b\|}{2} < \varrho_L \leq \frac{\|c\|}{2}$, which implies $\varphi_1 \geq \varphi_2 > 0$ and $\varphi_3 = 0$. The probability at the equilibrium radius is therefore

$$P_L(\varrho_L) = \tfrac{2\varrho_L^2}{\|a\|\|b\|\sin\gamma} \cdot (\sin\varphi_1 + \sin\varphi_2). \tag{23}$$

To get a handle on the maximum of this function, we first write the the sum of $\sin\varphi_1$ and $\sin\varphi_2$ and second the equilibrium radius in terms of other parameters. Using $\cos 2\alpha = \cos^2\alpha - \sin^2\alpha$ and $\cos^2\alpha + \sin^2\alpha = 1$, we get $\cos\varphi_2 = 2\cos^2\frac{\varphi_2}{2} - 1$, and since $\varphi_1 + \varphi_2 = \frac{\pi}{2}$, we have $\sin\varphi_1 = \cos\varphi_2$. Recalling (18), we get $\sin\varphi_1 = \|b\|^2/(2\varrho_L^2) - 1$, and recalling (17), we get $\sin\varphi_2 = \|a\|^2/(2\varrho_L^2) - 1$. Adding the two relations gives

$$\sin\varphi_1 + \sin\varphi_2 = \tfrac{\|a\|^2+\|b\|^2}{2\varrho_L^2} - 2. \tag{24}$$

To find a substitution for the equilibrium radius, we begin with (18), use $\frac{\varphi_2}{2} = \frac{\pi}{4} - \frac{\varphi_1}{2}$, and finally apply $\cos(\alpha+\beta) = \cos\alpha\cos\beta - \sin\alpha\sin\beta$:

$$\tfrac{\|b\|}{2\varrho_L} = \cos\left(\tfrac{\pi}{4} - \tfrac{\varphi_1}{2}\right) \tag{25}$$

$$= \tfrac{1}{\sqrt{2}}\left(\cos\tfrac{\varphi_1}{2} + \sin\tfrac{\varphi_1}{2}\right). \tag{26}$$

Next, we substitute the two trigonometric functions using (17) and $\sin^2\alpha = 1 - \cos^2\alpha$. Simplifying the resulting relation and squaring it, we get

$$\varrho_L^2 = \tfrac{1}{2}\left(\|a\|^2 + \|b\|^2 - \sqrt{2}\|a\|\|b\|\right). \tag{27}$$

Plugging (24) and (27) into the equation for the probability and normalizing by setting $\|b\| = 1$, we get

$$P_L(\varrho_L) = \tfrac{2\sqrt{2}\|a\| - \|a\|^2 - 1}{\|a\|\sin\gamma} \tag{28}$$

$$= \frac{2\sqrt{2}\|a\| - \|a\|^2 - 1}{\|a\|\sqrt{1 - \left(\sqrt{2} - \frac{\|a\|^2+1}{2\|a\|}\right)^2}} \tag{29}$$

where we maximize to get the second line by choosing γ as small as possible. Specifically, γ is implicitly restricted by $\frac{\|b\|}{2} < \varrho_L \leq \frac{\|c\|}{2}$, so we can use $4\varrho_L^2 \leq \|c\|^2$ together with (15) and (27) to get

$$\cos\gamma \leq \sqrt{2} - \tfrac{\|a\|^2+\|b\|^2}{2\|a\|\|b\|}. \tag{30}$$

Checking with the Maple software [12], we find that the right-hand-side of (29) increases in $[0, 1]$ attaining its maximum at $\|a\| = 1$. We therefore get $\gamma = \arccos\left(\sqrt{2} - 1\right)$ from (30), $\varrho_L^2 = 1 - 1/\sqrt{2}$ from (27), and

$$P_L(\varrho_L) = \sqrt{2\sqrt{2} - 2} = 0.910\ldots \tag{31}$$

from (29).

Proof of the Six Arcs Lemma. Case 3 is defined by $\frac{\|c\|}{2} < \varrho_L < R_L$, which implies $\varphi_1 \geq \varphi_2 \geq \varphi_3 > 0$. Starting with the expression for the probability given in the Equilibrium Area Lemma, we first express the $\sin\varphi_i$ in terms of the other parameters:

$$\sin\varphi_1 = \frac{\|a\|\sqrt{4\varrho_L^2 - \|a\|^2}}{2\varrho_L^2}, \tag{32}$$

$$\sin\varphi_2 = \frac{\|b\|\sqrt{4\varrho_L^2 - \|b\|^2}}{2\varrho_L^2}, \tag{33}$$

$$\sin\varphi_3 = 1 - \frac{\left[\|a\|\sqrt{4\varrho_L^2 - \|b\|^2} + \|b\|\sqrt{4\varrho_L^2 - \|a\|^2}\right]^2}{8\varrho_L^4}. \tag{34}$$

To get (32), we use $\sin 2\alpha = 2\sin\alpha\cos\alpha$ with $\alpha = \frac{\varphi_1}{2}$, together with (17). To get (33), we use the same trigonometric identity with $\alpha = \frac{\varphi_2}{2}$, together with (18). To get (34), we use the equilibrium condition together with $\sin\varphi_3 = \sin(\frac{\pi}{2} - \varphi_1 - \varphi_2) = \cos(\varphi_1 + \varphi_2) = 1 - 2\sin^2\frac{\varphi_1+\varphi_2}{2}$, and finally substitute $\sin(\alpha+\beta) = \sin\alpha\cos\beta + \sin\beta\cos\alpha$, with $\alpha = \frac{\varphi_1}{2}$ and $\beta = \frac{\varphi_2}{2}$.

To do the same for $\sin\gamma$, we take the cosine of both sides of the equilibrium condition, which is $\frac{\varphi_3}{2} = \frac{\pi}{4} - \frac{\varphi_1}{2} - \frac{\varphi_2}{2}$. Writing $c_i = \cos\frac{\varphi_i}{2}$ and $s_i = \sin\frac{\varphi_i}{2}$, for $i = 1, 2, 3$, and applying standard trigonometric identities, we get

$$c_3 = \tfrac{1}{\sqrt{2}}[c_1c_2 - s_1s_2 + s_1c_2 + c_1s_2]. \tag{35}$$

$$c_3^2 = \tfrac{1}{2} + (2c_1c_2^2 - c_1)\sqrt{1 - c_1^2} + (2c_1^2c_2 - c_2)\sqrt{1 - c_2^2}. \tag{36}$$

Using (17), (18), (19) and substituting $\|c\|^2$ using (15), we get the following relation after a few rearrangements:

$$\cos\gamma = \frac{\|a\|^2 + \|b\|^2 - 2}{2\|a\|\|b\|} - \frac{\|b\|^2 - 2\varrho_L^2}{4\varrho_L^2\|b\|}\sqrt{4\varrho_L^2 - \|a\|^2} - \frac{\|a\|^2 - 2\varrho_L^2}{4\varrho_L^2\|a\|}\sqrt{4\varrho_L^2 - \|b\|^2}. \tag{37}$$

Using $\cos^2\gamma = 1 - \sin^2\gamma$, we can substitute $\sin\gamma$ in the formula for $P_L(\varrho_L)$. We thus arrived at a relation that gives the probability in terms of $\|a\|$, $\|b\|$, and ϱ_L only. While being lengthy, this relation is readily obtained by plugging (32), (33), (34), and (37) into (11). We therefore take the liberty to omit the formula here and refer the interested readers to the website of the second author of this paper[1].

It remains to determine the parameters that maximize the probability. To simplify this task, we normalize by setting $\|b\| = 1$. The probability is thus a function of two variables, $\|a\|$ and ϱ_L. Using the Maple software, we compute the two partial derivatives, $\partial P_L/\partial\|a\|$ and $\partial P_L/\partial\varrho_L$. Setting both to zero, we get $\|a\| = 1$ matched up with three radius values:

$$\varrho_L = \tfrac{1}{2}\left(\sqrt{6} - \sqrt{2}\right) = 0.517\ldots, \tag{38}$$

$$\varrho_L = \tfrac{1}{2}\sqrt{C^{\frac{1}{3}} - 1 + C^{-\frac{1}{3}}} = 0.582\ldots, \tag{39}$$

$$\varrho_L = \tfrac{1}{2}\left(\sqrt{6} + \sqrt{2}\right) = 1.931\ldots, \tag{40}$$

with $C = 3 + 2\sqrt{2}$. Setting $\|a\| = \|b\| = 1$, the covering radius depends only on the angle γ, which ranges from $\frac{\pi}{3}$ to $\frac{\pi}{2}$. It is largest for $\gamma = \frac{\pi}{2}$, where $R_L = \sqrt{2}/2 = 0.707\ldots$. The radius we get at the third root is larger than that and can therefore be excluded.

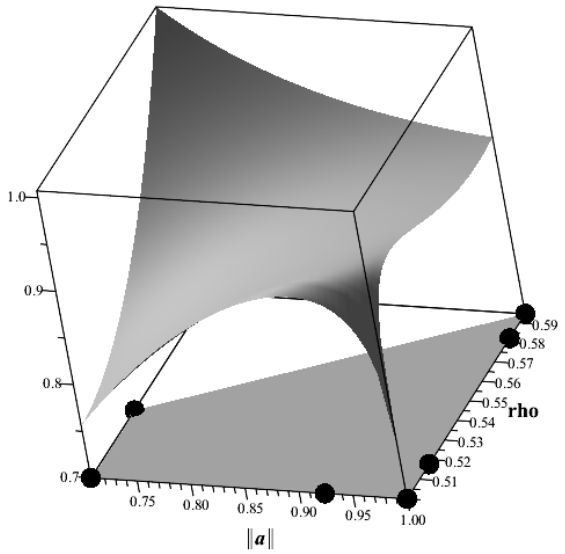


Figure 6: The quadrangle in the plane defined by the length of a and the radius is shaded. Along its boundary, we encounter three local minima (two corners and a point along the right edge) and three local maxima (a corner and a point each along the lower edge and the right edge).

The maximum probability is attained at one of the two remaining roots or along the boundary of the domain region that corresponds to Case 3. As illustrated in Figure 6, we simplify the computation by taking a quadrangle that contains this region. The quadrangle is defined by

$$\tfrac{\sqrt{2}}{2} \leq \|a\| \leq 1 \tag{41}$$

$$\tfrac{1}{2} \leq \varrho \quad \leq \tfrac{\|c\|}{2\sin\gamma_{max}}, \tag{42}$$

in which the first interval follows from the bounds that define Case 3, and the second interval is obtained by limiting the radius by the covering radius of the configuration in which a and b enclose its maximum angle. The maximum angle for Case 3, γ_{max}, corresponds to the minimal angle for Case 2 derived from (30). The upper boundary on ϱ is a convex curve and so we replace it with the straight line connecting its extremes.

[1] A Maple file with the main steps in the formulas related with this paper is available at `http://mabelih9.wix.com/mabelhome#!publications/cee5`.

We evaluate the probability at the four vertices and, using the Maple software, at the roots of the derivatives along the four edges. As shown in Figure 6, we find three local minima alternating with three local maxima along the boundary of the quadrangle. The maximum and minimum in the interior of the right edge coincide with the roots in (38) and (39). Among the three local maxima, the probability is largest at (38), which is characterized by $\|a\| = \|b\| = 1$ and $\varrho_L = \frac{1}{2}\left(\sqrt{6} - \sqrt{2}\right) = 1/(2\cos\frac{\pi}{12})$. This gives $P_L(\varrho_L) = 0.928\ldots$, as claimed in the Six Arcs Lemma. Indeed, plugging the values into the equilibrium condition of Case 3, we get $\|c\| = 1$, which shows that the probability is maximized by the regular hexagonal grid.

Approximating the Minimum Closest Pair Distance and Nearest Neighbor Distances of Linearly Moving Points

Timothy M. Chan* Zahed Rahmati*

Abstract

Given a set of n moving points in $\mathbb{R}^d$, where each point moves along a linear trajectory at arbitrary but constant velocity, we present an $\tilde{O}(n^{5/3})$-time algorithm[1] to compute a $(1+\epsilon)$-factor approximation to the *minimum closest pair distance* over time, for any constant $\epsilon > 0$ and any constant dimension d. This addresses an open problem posed by Gupta, Janardan, and Smid [12].

More generally, we consider a data structure version of the problem: for any linearly moving query point q, we want a $(1+\epsilon)$-factor approximation to the *minimum nearest neighbor distance* to q over time. We present a data structure that requires $\tilde{O}(n^{5/3})$ space and $\tilde{O}(n^{2/3})$ query time, $\tilde{O}(n^5)$ space and polylogarithmic query time, or $\tilde{O}(n)$ space and $\tilde{O}(n^{4/5})$ query time, for any constant $\epsilon > 0$ and any constant dimension d.

1 Introduction

In the last two decades, there has been a lot of research on problems involving objects in motion in different computer science communities (*e.g.*, robotics and computer graphics). In computational geometry, maintaining attributes (*e.g.*, closest pair) of moving objects has been studied extensively, and efficient kinetic data structures are built for this purpose (see [17] and references therein). In this paper, we pursue a different track: instead of maintaining an attribute over time, we are interested in finding a time value for which the attribute is minimized or maximized.

Let P be a set of moving points in $\mathbb{R}^d$, and denote by $p(t)$ the position (trajectory) of $p \in P$ at time t. Let $d(p(t), q(t))$ denote the Euclidean distance between $p(t)$ and $q(t)$. The following gives the formal statements of the two kinetic problems we address in this paper, generalizing two well-known standard problems for stationary points, *closest pair* and *nearest neighbor search*:

- *Kinetic minimum closest pair distance*: find a pair (p, q) of points in P and a time instant t, such that $d(p(t), q(t))$ is minimized.
- *Kinetic minimum nearest neighbor distance*: build a data structure so that given a moving query point q, we can find a point $p \in P$ and a time instant t such that $d(p(t), q(t))$ is minimized.

Related work. The *collision detection problem*, *i.e.*, detecting whether points ever collide [10], has attracted a lot of interest in the context. This problem can trivially be solved in quadratic time by brute force. For a set P of n linearly moving points in $\mathbb{R}^2$, Gupta, Janardan, and Smid [12] provided an algorithm, which detects a collision in P in $O(n^{5/3}\log^{6/5} n)$ time.

Gupta *et al.* also considered the minimum diameter of linearly moving points in $\mathbb{R}^2$, where the velocities of the moving points are constant. They provided an $O(n\log^3 n)$-time algorithm to compute the minimum diameter over time; the running time was improved to $O(n\log n)$ using randomization [7, 9]. Agarwal *et al.* [2] used the notion of ϵ-kernel to maintain an approximation of the diameter over time. For an arbitrarily small constant $\delta > 0$, their kinetic data structure in $\mathbb{R}^2$ uses $O(1/\epsilon^2)$ space, $O(n+1/\epsilon^{3s+3/2})$ preprocessing time, and processes $O(1/\epsilon^{4+\delta})$ events, each in $O(\log(1/\epsilon))$ time, where s is the maximum degree of the polynomials of the trajectories; this approach works for higher dimensions.

For a set of n stationary points in $\mathbb{R}^d$, the closest pair can be computed in $O(n\log n)$ time [5]. Gupta *et al.* [12] considered the kinetic minimum closest pair distance problem. Their solution is for the $\mathbb{R}^2$ case, and works only for a limited type of motion, where the points move with the same constant velocity along one of the two orthogonal directions. For this special case their algorithm runs in $O(n\log n)$ time. Their work raises the following open problem: Is there an efficient algorithm for the kinetic minimum closest pair distance problem in the more general case where points move with constant but possibly different velocities and different moving directions?

For a set of stationary points in $\mathbb{R}^d$, there are data structures for approximate nearest neighbor search with linear space and logarithmic query time [4]. We are not aware of any prior work on the kinetic minimum nearest neighbor distance problem. Linearly moving points in $\mathbb{R}^d$ can be mapped to lines in $\mathbb{R}^{d+1}$ by viewing time as an extra dimension. There have been previous papers

*Cheriton School of Computer Science, University of Waterloo, {tmchan, zrahmati}@uwaterloo.ca

[1] The notation $\tilde{O}$ is used to hide polylogarithmic factors. That is, $\tilde{O}(f(n)) = O(f(n)\log^c n)$, where c is a constant.

on approximate nearest neighbor search in the setting when the data objects are lines and the query objects are points [13], or when the data objects are points and the query objects are lines [16] (in particular, the latter paper contains some results similar to ours for any constant dimension d). However, in our problem, both data and query objects are mapped to lines; moreover, our distance function does not correspond to Euclidean distances between lines in $\mathbb{R}^{d+1}$.

An approach to solve the kinetic minimum closest pair distance and nearest neighbor distance problems would be to track the closest pair and nearest neighbor over time using known kinetic data structures [3, 18, 19]. The chief drawback of this approach is that the closest pair can change $\Omega(n^2)$ times in the worst case, and the nearest neighbor to a query point can change $\Omega(n)$ times (even if approximation is allowed). The challenge is to solve the kinetic minimum closest pair distance problem in $o(n^2)$ time, and obtain a query time $o(n)$ for the kinetic minimum nearest neighbor distance problem. To this end, we will allow approximate solutions.

Our contributions. We focus on the setting where each point in P (and each query point) has an arbitrary, constant velocity, and moves along an arbitrary direction.

We present an algorithm to compute a $(1+\epsilon)$-factor approximation to the minimum closest pair distance in $\tilde{O}(n^{5/3})$ time for any constant $\epsilon > 0$. More generally, we present a data structure for the kinetic minimum nearest neighbor distance problem with approximation factor $1+\epsilon$ with $\tilde{O}(m)$ preprocessing time and space, and $\tilde{O}(n/m^{1/5})$ query time for any m between n and n^5. For example, setting m appropriately, we obtain a data structure with $\tilde{O}(n^{5/3})$ space and $\tilde{O}(n^{2/3})$ query time, $\tilde{O}(n^5)$ space and $\tilde{O}(1)$ query time, or $\tilde{O}(n)$ space and $\tilde{O}(n^{4/5})$ query time. The results hold in any constant dimension d. Our solution uses techniques from range searching (including multi-level data structures and parametric search).

Perhaps the most notable feature of our results is that the exponents do not grow as a function of the dimension. (In contrast, for the exact kinetic minimum closest pair problem, it is possible to obtain subquadratic-time algorithms by range searching techniques, but with much worse exponents that converge to 2 as d increases.)

2 Kinetic Minimum Nearest Neighbor Distance

Let $p(t) = \mathbf{p}' + t\mathbf{p}''$ denote the linear trajectory of a point $p \in P$, where $\mathbf{p}' \in \mathbb{R}^d$ is the initial position vector of p, and $\mathbf{p}'' \in \mathbb{R}^d$ is the velocity vector of p. For any moving query point q with a linear trajectory $q(t) = \mathbf{q}' + t\mathbf{q}''$, we want to approximate the minimum nearest neighbor distance to q over time.

We first consider the following decision problem:

Decision Problem 1 *Given a point q and a real parameter r, determine whether there exists $p \in P$ with*

$$\min_{t\in\mathbb{R}} d(p(t), q(t)) \leq r. \tag{1}$$

Afterwards we use the parametric search technique to find the minimum nearest neighbor distance of q in P.

Approximating Decision Problem 1. Let $\mathbf{w}$ be a vector in $\mathbb{R}^d$. The Euclidean norm $\|\mathbf{w}\|$ of $\mathbf{w}$ can be approximated as follows [6]. Assume $\theta = \arccos(1/(1+\epsilon))$, for a small $\epsilon > 0$. The d-dimensional space around the origin can be covered by a set of $b = O(1/\theta^{d-1}) = O(1/\epsilon^{(d-1)/2})$ cones of opening angle θ [20]. *i.e.*, there exists a set $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_b\}$ of unit vectors in $\mathbb{R}^d$ that satisfies the following property: for any $\mathbf{w} \in \mathbb{R}^d$ there is a unit vector $\mathbf{v}_i \in V$ such that $\angle(\mathbf{v}_i, \mathbf{w}) \leq \theta$. Note that $\angle(\mathbf{v}_i, \mathbf{w}) = \arccos(\mathbf{v}_i \cdot \mathbf{w}/\|\mathbf{w}\|)$, where $\mathbf{v}_i \cdot \mathbf{w}$ denotes the inner product of the unit vector $\mathbf{v}_i$ and $\mathbf{w}$. Therefore,

$$\|\mathbf{w}\|/(1+\epsilon) \leq \max_{i\in B} \mathbf{v}_i \cdot \mathbf{w} \leq \|\mathbf{w}\|, \tag{2}$$

where $B = \{1, \ldots, b\}$.

From (2), we can use the following as an approximation of $d(p(t), q(t))$:

$$\max_{i\in B} \mathbf{v}_i \cdot (\mathbf{p}' - \mathbf{q}' + t(\mathbf{p}'' - \mathbf{q}'')).$$

Let $p'_i = \mathbf{v}_i \cdot \mathbf{p}'$, $p''_i = \mathbf{v}_i \cdot \mathbf{p}''$, $q'_i = \mathbf{v}_i \cdot \mathbf{q}'$, and $q''_i = \mathbf{v}_i \cdot \mathbf{q}''$. From the above discussion, a solution to Decision Problem 1 can be approximated by deciding the following.

Decision Problem 2 *Given a point q and a real parameter r, test whether there exists $p \in P$ with*

$$\min_{t\in\mathbb{R}} \max_{i\in B} (p'_i - q'_i) + t(p''_i - q''_i) \leq r. \tag{3}$$

Solving Decision Problem 2. Consider the inequality in (3). Minimizing the maximum of $\gamma_i(t) = (p'_i - q'_i) + t(p''_i - q''_i)$, over $i \in B$, is equivalent to finding the lowest point on the upper envelope of the linear functions $\gamma_i(t)$ in the $t\gamma$-plane; see Figure 1(a). Thus (3) is equivalent to checking whether the lowest point of the upper envelope is on or below the line $\gamma = r$.

Let $t_i = (r - p'_i + q'_i)/(p''_i - q''_i)$ denote the time that $\gamma_i(t)$ intersects with $\gamma = r$, *i.e.*, the root for $\gamma_i(t) = r$. Let $m_i = p''_i - q''_i$ denote the slope of the linear function $\gamma_i(t)$.

Deciding the following is equivalent to deciding whether the lowest point on the upper envelope of $\gamma_i(t)$ is on or below the line $\gamma = r$.

- The maximum root of the linear functions $\gamma_i(t) = r$ with *negative* slope is less than or equal to the

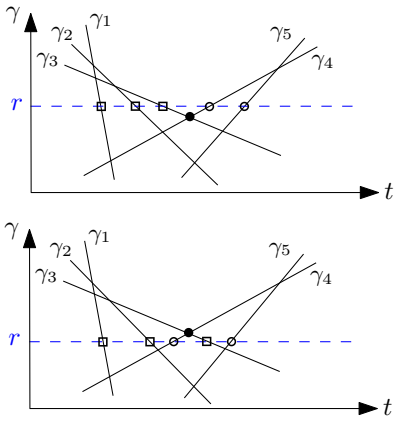


Figure 1: The intersections of γ_i with $\gamma = r$ are shown by empty circles (○) if the slope m_i of γ_i is positive, or empty squares (□) if $m_i < 0$. (a) The lowest point (bullet point ●) on the upper envelope is below $\gamma = r$. (b) The lowest point on the upper envelope is above $\gamma = r$, $m_3 < 0$, $m_4 > 0$, and $t_3 > t_4$.

minimum root of the linear functions with *positive* slope. In other words,

$$\max_{i:m_i<0} t_i \le \min_{j:m_j>0} t_j. \tag{4}$$

Note that `if` the lowest point of the upper envelope is *above* the line $\gamma = r$, then there exists a pair (i, j) of indices such that the clause $(m_i > 0) \vee (m_j < 0) \vee (t_i < t_j)$ is false (see Figure 1(b)); `otherwise`, the conjunction of all clauses (for all $i, j \in B$) is true. Therefore, we can obtain the following.

Lemma 1 *The inequality of (3) is satisfied iff the following conjunction is true:*

$$\bigwedge_{i,j\in B} ((p_i'' > q_i'') \vee (p_j'' < q_j'') \vee (t_i < t_j)),$$

where $t_i = (r - p_i' + q_i')/(p_i'' - q_i'')$.

Each condition in the clauses in Lemma 1 may be represented as a half-space in the following manner.

Consider the inequality $t_i < t_j$, *i.e.*,

$$\frac{r - p_i' + q_i'}{p_i'' - q_i''} < \frac{r - p_j' + q_j'}{p_j'' - q_j''}.$$

Assuming $(p_i'' < q_i'') \wedge (p_j'' > q_j'')$, $t_i < t_j$ is equivalent to

$$(r - p_i' + q_i')(p_j'' - q_j'') - (r - p_j' + q_j')(p_i'' - q_i'') > 0,$$

which can be expanded as

$$rp_j'' - rq_j'' - p_i'p_j'' + p_i'q_j'' + q_i'p_j'' - q_i'q_j'' - rp_i'' + rq_i'' + p_j'p_i'' - p_j'q_i'' - p_i''q_j' + q_i''q_j' > 0.$$

By factoring some terms in the above inequality, we obtain

$$p_i'(q_j'') + p_i''(-r - q_j') + p_j'(-q_i'') + p_j''(r + q_i') + p_i''p_j' - p_i'p_j'' + rq_i'' - rq_j'' - q_i'q_j'' + q_i''q_j' > 0,$$

which can be expressed in the form

$$A_1X_1 + A_2X_2 + A_3X_3 + A_4X_4 + X_5 > A_5,$$

where $X_1 = p_i'$, $X_2 = p_i''$, $X_3 = p_j'$, $X_4 = p_j''$, $X_5 = p_i''p_j' - p_i'p_j''$, $A_1 = q_j''$, $A_2 = -r - q_j'$, $A_3 = -q_i''$, $A_4 = r + q_i'$, and $A_5 = -rq_i'' + rq_j'' + q_i'q_j'' - q_i''q_j'$.

Lemma 2 *For each pair (i, j) of indices, the clause $(p_i'' > q_i'') \vee (p_j'' < q_j'') \vee (t_i < t_j)$ in Lemma 1 can be represented as*

$$(X_2 > -A_3) \vee (X_4 < A_1) \vee \tag{5}$$
$$(A_1X_1 + A_2X_2 + A_3X_3 + A_4X_4 + X_5 > A_5). \tag{6}$$

From Lemmas 1 and 2, we have reduced Decision Problem 2 to a searching problem $\mathcal{S}$, which is the conjunction of $O(b^2)$ simplex range searching problems $\mathcal{S}_l$, $l = 1, \ldots, O(b^2)$. Each $\mathcal{S}_l$ is a 5-dimensional simplex range searching problem on a set of points, each with coordinates $(X_1, X_2, X_3, X_4, X_5)$ that is associated with a point $p \in P$. The polyhedral range (5–6) for $\mathcal{S}_l$, which can be decomposed into a constant number of simplicial ranges, is given at query time, where $A_1, \ldots, A_5$ can be computed from the query point q and the parameter r.

Data structure for the searching problem $\mathcal{S}$. *Multi-level data structures* can be used to solve complex range searching problems [1] involving a conjunction of multiple constraints. In our application, we build a multi-level data structure $\mathcal{D}$ to solve the searching problem $\mathcal{S}$ consisting of $O(b^2)$ levels. To build a data structure for a set at level l, we form a collection of canonical subsets for the 5-dimensional simplex range searching problem $\mathcal{S}_l$, and build a data structure for each canonical subset at level $l + 1$. The answer to a query is expressed as a union of canonical subsets. For a query for a set at level l, we pick out the canonical subsets corresponding to all points in the set satisfying the l-th clause by 5-dimensional simplex range searching in $\mathcal{S}_l$, and then answer the query for each such canonical subset at level $l + 1$.

A multi-level data structure increases the complexity by a polylogarithmic factor (see Theorem 10 of [1] or the papers [8, 14]). In particular, if $S(n)$ and $Q(n)$ denote the space and query time of 5-dimensional simplex range searching, our multi-level data structure $\mathcal{D}$ requires $O(S(n)\log^{O(b^2)} n)$ space and $O(Q(n)\log^{O(b^2)} n)$ query time.

Assume $n \le m \le n^5$. A 5-dimensional simplex range searching problem can be solved in $\tilde{O}(\frac{n}{m^{1/5}})$ query time

with $\tilde{O}(m)$ preprocessing time and space [8, 14]. We conclude:

Lemma 3 *Let $n \leq m \leq n^5$. A data structure $\mathcal{D}$ for Decision Problem 2 can be built that uses $O(m \log^{O(b^2)} n)$ preprocessing time and space and can answer queries in $O(\frac{n}{m^{1/5}} \log^{O(b^2)} n)$ time.*

Solving the optimization problem. Consider Decision Problem 2, and denote by r^* the smallest r satisfying (3). We use Megiddo's parametric search technique [15] to find r^*. This technique uses an efficient *parallel* algorithm for the decision problem to provide an efficient *serial* algorithm for the optimization problem (computing r^*); the running time typically increases by logarithmic factors. Suppose that the decision problem can be solved in T time sequentially, or in τ parallel steps using π processors. Then the running time to solve the optimization problem would be $O(\tau \cdot \pi + T \cdot \tau \cdot \log \pi)$.

In our case, $T = \pi = O(\frac{n}{m^{1/5}} \log^{O(b^2)} n)$ (by Lemma 3) and $\tau = O(\log^{O(b^2)} n)$, where $b^2 = O(1/\epsilon^{d-1})$. Therefore, we obtain the main result of this section:

Theorem 4 *Let $n \leq m \leq n^5$. For a set P of n linearly moving points in $\mathbb{R}^d$ for any constant d, there exists a data structure with $O(m \log^{O(1/\epsilon^{d-1})} n)$ preprocessing time and space that can compute a $(1+\epsilon)$-factor approximation of the minimum nearest neighbor distance to any linearly moving query point over time in $O(\frac{n}{m^{1/5}} \log^{O(1/\epsilon^{d-1})} n)$ time.*

Remark 1 Our approach can be modified to compute the minimum distance over all time values inside any query interval $[t_0, t_f]$. The conjunction in Lemma 1 becomes $\bigwedge_{i,j \in B}((p_i'' > q_i'') \vee (p_j'' < q_j'') \vee (t_i < t_j) \vee (t_i > t_f) \vee (t_j < t_0))$. The condition $t_i > t_f$ is equivalent to $r - p_i' + q_i > t_f(p_i'' - q_i'')$, which can be expressed in the form $B_1 Y_1 + Y_2 < B_2$, where $Y_1 = p_i''$, $Y_2 = p_i'$, $B_1 = t_f$, and $B_2 = r + q_i + t_f q_i''$. This corresponds to a 2-dimensional halfplane range searching problem. The condition $t_j < t_0$ can be handled similarly. We can expand the entire expression into a disjunction of $5^{O(b^2)}$ subexpressions, where each subexpression is a conjunction of $O(b^2)$ conditions and can then be handled by a multi-level data structure similar to $\mathcal{D}$.

Remark 2 Our approach can be used to compute the *exact* minimum nearest neighbor distance in the L_∞ metric to any moving query point. Let $\mathbf{v}_j$ and $\mathbf{v}_{d+j}$ be the unit vectors of the negative x_j-axis and positive x_j-axis, respectively, in the d dimensional Cartesian coordinate system, where $1 \leq j \leq d$. We define $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_b\}$ with $b = 2d$, and solve the problem as before.

3 Kinetic Minimum Closest Pair Distance

To approximate the kinetic minimum closest pair distance, we can simply preprocess P into the data structure of Theorem 4, and for each point $p \in P$, approximate the minimum nearest neighbor distance to p. The total time is $O((m + \frac{n^2}{m^{1/5}}) \log^{O(1/\epsilon^{d-1})} n)$ time. Setting $m = 5/3$ gives the main result:

Theorem 5 *For a set of n linearly moving points in $\mathbb{R}^d$ for any constant d, there exists an algorithm to compute a $(1+\epsilon)$-factor approximation of the minimum closest pair distance over time in $O(n^{5/3} \log^{O(1/\epsilon^{d-1})} n)$ time.*

Remark 3 By Remark 2, we can compute the *exact* minimum closest pair distance in the L_∞ metric, of a set of n linearly moving points in $\mathbb{R}^d$, in $O(n^{5/3} \log^{O(d^2)} n)$ time.

4 Discussion

For a set P of linearly moving points in $\mathbb{R}^d$, we have given efficient algorithms and data structures to approximate the minimum value of two fundamental attributes: the closest pair distance and distances to nearest neighbors. We mention some interesting related open problems along the same direction:

- The Euclidean minimum spanning tree (EMST) on a set P of n moving points in $\mathbb{R}^2$ can be maintained by handling nearly cubic events [19], each in polylogarithmic time. Can we compute the minimu weight of the EMST on P, for linearly moving points, in subcubic time?
- For a set of n moving unit disks, there exist kinetic data structures [11] that can efficiently answer queries in the form "Are disks D_1 and D_2 in the same connected component?". This kinetic data structure handles nearly quadratic events, each in polylogarithmic time. Can we find the first time when all the disks are in the same connected component in subquadratic time?

References

[1] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. *Contemporary Mathematics*, 223:1–56, 1999.

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

[3] P. K. Agarwal, H. Kaplan, and M. Sharir. Kinetic and dynamic data structures for closest pair and all nearest neighbors. *ACM Transactions on Algorithms*, 5:4:1–37, 2008.

[4] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[5] J. L. Bentley and M. I. Shamos. Divide-and-conquer in multidimensional space. In *Proceedings of the 8th Annual ACM Symposium on Theory of Computing (STOC)*, pages 220–230, ACM, 1976.

[6] T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. *International Journal of Computational Geometry & Applications*, 12(1-2):67–85, 2002.

[7] 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, SIAM, 2004.

[8] T. M. Chan. Optimal partition trees. *Discrete & Computational Geometry*, 47(4):661–690, 2012.

[9] K. L. Clarkson. Algorithms for the minimum diameter of moving points and for the discrete 1-center problem, 1997. http://kenclarkson.org/moving_diam/p.pdf.

[10] K. Fujimura. *Motion Planning in Dynamic Environments*. Springer-Verlag, Secaucus, NJ, USA, 1992.

[11] L. Guibas, J. Hershberger, S. Suri, and L. Zhang. Kinetic connectivity for unit disks. *Discrete & Computational Geometry*, 25(4):591–610, 2001.

[12] P. Gupta, R. Janardan, and M. Smid. Fast algorithms for collision and proximity problems involving moving geometric objects. *Computational Geometry*, 6(6):371–391, 1996.

[13] S. Mahabadi. Approximate nearest line search in high dimensions. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 337–354, SIAM, 2015.

[14] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete & Computational Geometry*, 10(1):157–182, 1993.

[15] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *Journal of the ACM*, 30(4):852–865, 1983.

[16] W. Mulzer, H. L. Nguyen, P. Seiferth, and Y. Stein. Approximate k-flat nearest neighbor search. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC)*, 2015.

[17] Z. Rahmati. *Simple, Faster Kinetic Data Structures*. PhD thesis, University of Victoria, 2014.

[18] Z. Rahmati, M. A. Abam, V. King, and S. Whitesides. Kinetic k-semi-Yao graph and its applications. *Computational Geometry* (to appear).

[19] Z. Rahmati, M. A. Abam, V. King, S. Whitesides, and A. Zarei. A simple, faster method for kinetic proximity problems. *Computational Geometry*, 48(4):342–359, 2015.

[20] A. C.-C. Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing*, 11(4):721–736, 1982.

Time-Windowed Closest Pair*

Timothy M. Chan† Simon Pratt†

Abstract

Given a set of points in any constant dimension, each of which is associated with a time during which that point is active, we design a data structure with $O(n \log n)$ space that can find the closest pair of active points within a query interval of time in $O(\log \log n)$ time using a quadtree-based approach in the word-RAM model.

1 Introduction

Let P be a set of n points in $\mathbb{R}^d$. Additionally, let each point $p \in P$ be associated with a time t_p at which that point is active. In the *time-windowed closest pair problem*, we want to preprocess P into a data structure that can efficiently determine the closest pair of points that are active during an interval of time $[t_1, t_2]$ called a *time window*.

Time-windowed geometric problems are motivated by *Geographic Information System* (GIS) data which sometimes consists not only of longitude, latitude, and altitude coordinates but also time. Bannister *et al.* [3] examined time-windowed versions of convex hull, approximate spherical range searching, and approximate nearest neighbor queries. Their solution for this last problem is to store the points in a balanced binary search tree indexed by time, and store the Z-order (also known as Morton or shuffle order) [4] of subsets of the points. However, the time-windowed closest pair problem appears more challenging, because it involves minimizing *quadratically* many pairs.

We can consider the time of each point as its coordinate in the $(d+1)^{\text{th}}$ dimension. If we do so, the problem becomes finding the closest pair of points within a strip in $\mathbb{R}^{d+1}$. Sharathkumar and Gupta [10] wrote a technical report in which they solve the problem of finding the closest pair in 2 dimensions within a query range, a special case of which is when the range is a strip. Their solution can be used to solve the time-windowed closest pair problem in 1 dimension in $O\big(n \log^2 n\big)$ space and $O(\log n)$ query time.

Our approach answers queries in $O(\log \log n)$ time using $O(n \log n)$ space and $O(n \log n \log \log n)$ preprocessing time in the w-bit word-RAM model for any constant dimension d. (The algorithm finds the exact closest pair.) Here, we assume the time value of each point is an integer from 0 to $n-1$ and that each point has a distinct time value. If time is given as a w-bit integer instead, we can pre-sort the time values and replace them by their ranks; this adds the cost of a predecessor search to the query time (which is at most $O\Big(\min\{\log w, \log_w n, \sqrt{\log n / \log \log n}\}\Big)$ [9]).

The main idea behind our approach is to compute a centroid cell B of a quadtree, consider $O(n)$ pairs in which one point is inside the centroid cell and the other is outside, then recurse on $P \cap B$ and $P \setminus B$.

The idea of the centroid cell of a quadtree is due to Arya *et al.* [2]. In that paper they describe a data structure related to quadtrees called a *Balanced Box Decomposition* (BBD) tree, which they use to answer approximate nearest neighbor queries efficiently.

2 Preliminaries

We work in the w-bit word-RAM model, which models the computer as a sequence of w-bit memory locations each indexed by a w-bit integer. In this model, we assume that $w \geq \log n$ and standard operations on words take constant time.

Let P be a set of n points in $\mathbb{R}^d$, and B is a hypercube (which we call a *cell*) containing those points. To build the *quadtree* for P, we divide B into 2^d congruent child hypercubes. For each of these child hypercubes which contain more than a single point, we recursively build a quadtree for that box.

The following lemma, similar to Lemma 4 from Arya *et al.* [2], bounds the number of points in a hypercube by a constant with respect to the hypercube's side length and the distance between the closest pair within that hypercube.

Lemma 1 (Packing) *If a point set has closest-pair distance at least r and lies in a d-dimensional hypercube with side length at most $b \cdot r$, then there are less than $c_0(b+1)^d$ points, where c_0 is some constant that depends only on d. We call c_0 the* packing constant*.*

The following lemma is due to Arya *et al.* [2] (Lemma 1).

Lemma 2 (Centroid) *Given a point set P containing n points, there exists a quadtree cell B, which we call a*

*Research supported by The Natural Sciences and Engineering Research Council of Canada and the Ontario Graduate Scholarship.

†{tmchan,s2pratt}@cs.uwaterloo.ca

centroid cell, *such that* $|P \cap B| \leq \alpha n$ *and* $|P \setminus B| \leq \alpha n$ *for some constant* $\alpha < 1$ *that depends only on* d.

Recursive application of this lemma gives a data structure on a set of points P, called a *balanced quadtree* [5], defined as a binary tree where the root stores B, the left subtree is the balanced quadtree for $P \cap B$, and the right subtree is the balanced quadtree for $P \setminus B$ where B is a centroid cell of P.

We have the following lemma due to Chan [5] (Observation 3.2, Lemma 3.3), that says if we draw a constant number of grids over our points, each shifted by some amount, then we can guarantee that any pair of points must be in the same cell in at least one such grid. Since quadtree cells are related to grid cells, this also implies that the closest pair will be in the same quadtree cell if we build a constant number of quadtrees.

Lemma 3 (Shifting) *Suppose* d *is even. Let* $v^{(j)} = (\lfloor j2^w/(d+1) \rfloor, \ldots, \lfloor j2^w/(d+1) \rfloor) \in \mathbb{R}^d$. *For any points* p *and* q *and* $r = 2^\ell$ *such that* $||p-q||_\infty \leq r$, *there exists* $j \in \{0, 1, \ldots, d\}$ *such that* $p + v^{(j)}$ *and* $q + v^{(j)}$ *belong to the same* $c_1 r$*-grid cell, where* c_1 *is the smallest power of 2 bigger than or equal to* $2d+2$. *We call* c_1 *the* shifting constant.

While the preceding lemma requires that d be even, for odd values of d we can use $d+1$.

3 Decision Problem

Before we solve the time-windowed closest pair problem, it helps to consider the decision problem version, in which we are additionally given a fixed distance r and we want to preprocess P into a data structure which can efficiently determine, for any query time window, if there exists a pair of active points pq such that the distance between p and q is at most r. We call such a pair a *satisfying pair*.

The main idea of our approach is to use a constant number of shifted grids, which by Lemma 3 ensures that any two points will appear in the same cell together in at least one such shifted grid. For each point, we consider a constant number of its time-order predecessors and successors within the same cell, which by Lemma 1 we know must include a satisfying pair if one exists. From there, we reduce the problem to a standard 2-dimensional dominance range searching problem.

3.1 Computing Candidate Satisfying Pairs

We begin by bucketing the points of P into grid cells with side length $c_1 r'$, where c_1 is the shifting constant from Lemma 3 and r' is the smallest power of 2 bigger than or equal to r. Each grid cell is assigned a label ℓ and for each point p we create a tuple (ℓ, t_p, p). We can determine the label of the grid cell containing each point by hashing in $O(n)$ total expected time.

For each cell, we build a time-ordered array of the points within that cell. This is done by running radix sort on the tuples created in the previous step, sorting first by grid cell label, and then by time. The radix sort takes $O(n)$ time.

For each such point p, we consider its $c_0(c_1+1)^d$ predecessors and the same number of successors in the time-ordered array, where c_0 is the packing constant from Lemma 1. Let q be such a predecessor or successor. If the distance between p and q is at most r, then t_p and t_q form a *candidate satisfying pair*.

We do the preceding steps $d+1$ times, where each time the cells are shifted by $v^{(j)}$ for $j \in \{0, 1, \ldots, d\}$ as defined in Lemma 3. We union together the results to build the full set of candidate satisfying pairs.

Lemma 4 *There are* $O(n)$ *candidate satisfying pairs.*

Proof. Over the $d+1$ shifts, the total is upper-bounded by $(d+1) \cdot c_0(c_1+1)^d n = O(n)$. □

Lemma 5 *If a time window contains a satisfying pair, then the time window must contain a candidate satisfying pair.*

Proof. Let pq be a satisfying pair for the window which is closest in terms of time order. From Lemma 3, there exists $j \in \{0, 1, \ldots, d\}$ such that $p + v^{(j)}$ (which we will call p') and $q + v^{(j)}$ (which we will call q') are in the same grid cell of side length $c_1 r$. Since p' and q' are active, all points between them in time order must also be active. No two points strictly between p' and q' can have distance smaller than r, for otherwise we would have a satisfying pair that is closer than pq in terms of time order. By Lemma 1 there are less than $c_0(c_1+1)^d$ points strictly between p' and q'. Therefore $p'q'$ must be among the candidate satisfying pairs. □

3.2 Reduction to 2D Dominance Range Emptiness

Now that the number of pairs we need to consider is reduced to $O(n)$, we would like to store these pairs in a data structure to support efficient querying. Specifically, given a query window $[t_1, t_2]$ we wish to determine if there exists a candidate satisfying pair pq such that $t_1 \leq \min\{t_p, t_q\}$ and $t_2 \geq \max\{t_p, t_q\}$.

Consider each candidate satisfying pair as a point in 2 dimensions with coordinates $(-\min\{t_p, t_q\}, \max\{t_p, t_q\})$. Our query problem is equivalent to determining whether the quadrant $(-\infty, -t_1] \times (-\infty, t_2]$ contains any of these points. This is exactly the *2D dominance range emptiness* problem. This problem can be solved by computing

the *minima* of the 2D point set [8] and testing whether the query point is above or below the *staircase* formed by the minima. Computing the minima of $O(n)$ points takes $O(n)$ time by a standard sweep-line algorithm, assuming that the x-coordinates have been pre-sorted. Since the x-coordinates are in $\{0, \ldots, n-1\}$, pre-sorting takes $O(n)$ time.

We can use an array to store the y-value of the staircase at every x-value; this requires $O(n)$ words of space.

To save space, we can build a succinct rank/select data structure [7] with all the same time bounds but using just $2n + o(n)$ bits of space. We do so by considering the staircase as a monotone chain (after negating the x-coordinates) through the $n \times n$ grid from the origin to $(n-1, n-1)$. This grid is effectively a plot with start time on the x-axis, and end time on the y-axis. We can encode a monotone chain as a sequence of $2n$ bits. Starting at the origin, whenever the chain moves upwards from end time i to $i+1$, we store a 0 bit. Similarly, whenever the chain moves rightwards, we store a 1 bit. The answer to the query is yes if and only if $t_2 \geq rank(select(t_1))$, where $select(i)$ denotes the position of the i^{th} 1 in the sequence and $rank(j)$ denotes the number of 1s in the first j positions of the sequence. The rank and select operations take constant time.

We have thus proven the following result:

Theorem 6 *The decision problem version of the time-windowed closest pair problem in any fixed dimension can be solved in $O(1)$ time using $O(n)$ bits of space and $O(n)$ expected preprocessing time in the word-RAM model.*

4 Closest Pair

To solve the original time-windowed closest pair problem, the main new idea is to replace shifted grids with shifted balanced quadtrees. For each point outside of the centroid cell, we consider a constant number of its time-order predecessors and successors within the centroid cell. We then recurse separately on the points inside and outside of the centroid cell. This divide-and-conquer approach gives us $O(n \log n)$ candidate pairs. From there, we reduce the problem to a 2-dimensional dominance range minimum problem.

4.1 Computing Candidate Pairs

We describe our algorithm to generate candidate pairs recursively. We first compute the centroid cell B of the given point set P. Define the set of neighbors $N(p)$ of a point p as its $c_0(2c_1+1)^d$ time-order predecessors and successors within the centroid cell B, where c_0 is the packing constant from Lemma 1 and c_1 is the shifting constant from Lemma 3. For each point p outside of the centroid, we consider each pair pq for $q \in N(p)$ as a *candidate pair*. We then recurse on $P \cap B$ and $P \setminus B$.

We run the preceding algorithm $d+1$ times, where each time the quadtrees are shifted by $v^{(j)}$ for $j \in \{0, 1, \ldots, d\}$ as defined in Lemma 3. We union together the results to build the full set of candidate pairs.

Lemma 7 *There are $O(n \log n)$ candidate pairs.*

Proof. For each fixed shift, the number of candidate pairs is given by the recurrence $P(n) \leq P(n_1) + P(n_2) + c_0(2c_1+1)^d n$, where n_1 and n_2 are the number of points inside and outside of the centroid respectively.

Since $n_1 + n_2 = n$ and $n_1, n_2 \leq \alpha n$, the recurrence solves to $P(n) = O(n \log n)$. □

Lemma 8 *The closest pair for any time window must be among the candidate pairs.*

Proof. Let pq be the closest pair in the window, with distance r. From Lemma 3, there exists $j \in \{0, 1, \ldots, d\}$ such that $p + v^{(j)}$ (which we will call p') and $q + v^{(j)}$ (which we will call q') are in the same quadtree cell of side length $c_1 r'$ where r' is the smallest power of 2 greater than r.

There are 3 cases. Either p', q' are both inside or outside of the centroid cell B, or one is inside and the other is outside of B. The first 2 cases can be handled by induction. Now we are in case 3, so suppose q' is inside the centroid. (The case where p' is inside the centroid is symmetric.)

From Lemma 1, there are no more than $c_0(2c_1+1)^d$ active points in the centroid cell B, since B has side length at most $2c_1 r$. Since p and q are active during the time window, all points between them in time order must also be active. Therefore, there are less than $c_0(2c_1+1)^d$ points between p and q in time order, so $q \in N(p)$. □

4.2 Reduction to 2D Dominance Range Minimum

Now that the number of pairs we need to consider is reduced to $O(n \log n)$, we would like to store these pairs in a data structure to support efficient querying. Specifically, given a query window $[t_1, t_2]$ we wish to find a candidate pair pq such that $t_1 \leq \min\{t_p, t_q\}$ and $t_2 \geq \max\{t_p, t_q\}$ while minimizing the distance $d(p, q)$.

Consider each candidate pair as a weighted point in 2 dimensions with coordinates $(-\min\{t_p, t_q\}, \max\{t_p, t_q\})$ and weight $d(p, q)$. Our query problem is equivalent to finding a point in the quadrant $(-\infty, -t_1] \times (-\infty, t_2]$ with the minimum weight. This is exactly the *2D dominance range minimum* problem, which we can solve by using standard techniques. Namely, we first lift the 2D weighted points to 3D where the weights become

z-coordinates. We compute the *staircase polyhedron* of the 3D point set, defined as the region of all points that are not dominated by any input point. Then a query can be answered by finding the highest point on the staircase polyhedron at a given x- and y-coordinate. Computing the staircase polyhedron is related to the standard problem of computing the *minima* of the 3D point set [8, 1] and can be done by a standard sweep-plane algorithm. For a set of N points in 3D, the sweep-plane algorithm takes $O(N \log\log N)$ time using van Emde Boas trees, assuming that the x- and y-coordinates have been pre-sorted (the z-coordinates need not be pre-sorted). Since the x-coordinates are in $\{0, \ldots, n-1\}$, pre-sorting can be done in $O(N+n)$ time.

Finding the highest point of the staircase polyhedron (a monotone polyhedron in 3D) at a query x- and y-coordinate reduces to point location in a 2D subdivision of $O(N)$ size, after projecting the faces onto the xy-plane. We can use Chan's *planar orthogonal point location* structure [6] as a black box to answer queries in $O(\log\log N)$ time using $O(N)$ space and $O(N)$ preprocessing time.

Setting $N = O(n \log n)$ gives our main result:

Theorem 9 *Time-windowed closest pair queries in any fixed dimension can be answered in $O(\log\log n)$ time using $O(n \log n)$ words of space and $O(n \log n \log\log n)$ preprocessing time in the word-RAM model.*

4.3 A Lower Bound on the Number of Candidate Pairs

As a final remark, we point out that any approach which stores all candidate pairs must use $\Omega(n \log n)$ space by proving the following observation.

Observation 1 *There exists a set of n points, where each point is associated with a time value, such that there are $\Omega(n \log n)$ distinct closest pairs over all possible time windows.*

Proof. Our construction works in one dimension. Suppose n is a power of 2. The base case $n = 2$ is trivial. To construct a set S of n points on a line, we first recursively construct a set S_1 of $n/2$ points, and duplicate S_1 to create S_2. We increase the labels of points in S_2 by $n/2$ and we shift the points along the line by δ for a sufficiently small $\delta > 0$ (less than half of the closest pair distance in S_1). Since the time labels of S_1 and S_2 are disjoint, any closest pair between points in S remains a closest pair for some time window. Symmetrically, we have the same closest pairs between points in S_2. In addition, for each time value $i \in \{1, \ldots, n/2\}$, the pair of points with time values i and $i + n/2$ is a closest pair for the time window $[i, i+n/2]$, because the pair has the smallest possible distance δ, and any other pair with distance δ has time values of the form j and $j+n/2$, which can't both lie inside $[i, i+n/2]$. This gives $n/2$ additional closest pairs. Therefore, the number of distinct closest pairs is given by the recurrence $C(n) \geq 2C(n/2) + n/2$, which solves to $C(n) = \Omega(n \log n)$.

(Note: the construction can alternatively be described without recursion using *bit-reversal permutations*.) □

References

[1] P. Afshani. Fast computation of output-sensitive maxima in a word RAM. In *Proceedings of the 25th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1414–1423. SIAM, 2014.

[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] M. J. Bannister, W. E. Devanny, M. T. Goodrich, J. A. Simons, and L. Trott. Windows into geometric events: Data structures for time-windowed querying of temporal point sets. In *Proceedings of the 26th Canadian Conference on Computational Geometry*, 2014.

[4] M. Bern, D. Eppstein, and S.-H. Teng. Parallel construction of quadtrees and quality triangulations. *International Journal of Computational Geometry & Applications*, 9(06):517–532, 1999.

[5] T. M. Chan. Approximate nearest neighbor queries revisited. *Discrete & Computational Geometry*, 20(3):359–373, 1998.

[6] T. M. Chan. Persistent predecessor search and orthogonal point location on the word RAM. *ACM Transactions on Algorithms (TALG)*, 9(3):22, 2013.

[7] J. I. Munro. Tables. In *Foundations of Software Technology and Theoretical Computer Science*, pages 37–42. Springer, 1996.

[8] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag New York, Inc., New York, NY, USA, 1985.

[9] Pătraşcu, Mihai. Predecessor search. In *Encyclopedia of Algorithms*. 2008.

[10] R. Sharathkumar and P. Gupta. Range-aggregate proximity queries. Technical report, IIIT/TR/2007/80, IIIT Hyderabad, 2007.

An Output-Sensitive Algorithm for Computing Weighted α-Complexes*

Donald R. Sheehy†

Abstract

An α-complex is a subcomplex of the Delaunay triangulation of a point set $P \subset \mathbb{R}^d$ that is topologically equivalent to the union of balls of radius α centered at the points of P. In this paper, we give an output-sensitive algorithm to compute α-complexes of n-point sets in constant dimensions, whose running time is $O(f \log n \log \frac{\alpha}{s})$, where s is the smallest pairwise distance and f is the number of simplices in the $c\alpha$-complex for a constant c. The algorithm is based on a refinement of a recent algorithm for computing the full Delaunay triangulation of P. We also extend the algorithm to work with weighted points provided the weights are appropriately bounded. The new analysis, which may be of independent interest, bounds the number of intersections of k-faces of a Voronoi diagram with $(d-k)$-faces of the Voronoi diagram of a carefully constructed superset.

1 Introduction

The starting point for many algorithmic problems in computational geometry is the discrete representation of continuous objects. The α-complex gives a topologically faithful representation of a union of balls as a subcomplex of the Delaunay triangulation of the centers [6]. Weighted α-complexes model the case where the radii of the balls are permitted to vary.

As with the Delaunay triangulation, the α-complex has a dramatic difference in the number of simplices in best- and worst-case examples. However, it can be that even though the Delaunay triangulation may be large, say $\Theta(n^{\lceil d/2 \rceil})$ simplices, the α-complex may still be quite small. Thus, our goal is to compute the α-complex without computing the full Delaunay triangulation. Our approach will be to modify a recent output-sensitive algorithm for computing Delaunay triangulations [11] as well as providing a new perspective to the analysis that gives nearly tight bounds on the number of bistellar flips in a restricted case of kinetic Delaunay triangulations, a result of independent interest.

Contributions Our main contributions are the following.

1. We introduce a generalization of the aspect ratio of a Voronoi cell that applies also to the cells of dimension less than d and relate the aspect ratio to the number of flips needed in removing a subset of vertices from a Delaunay triangulation in the output-sensitive Delaunay triangulation algorithm of Miller and Sheehy [11]. This gives a tighter analysis and also leads to the following algorithmic results.

2. We give a generalization of the Miller-Sheehy algorithm to handle weighted points, assuming the weight of any point is less than half the distance to its nearest neighbor.

3. We give a variation of the algorithm that returns the (weighted) α-complex of the point set without computing the full Delaunay triangulation.

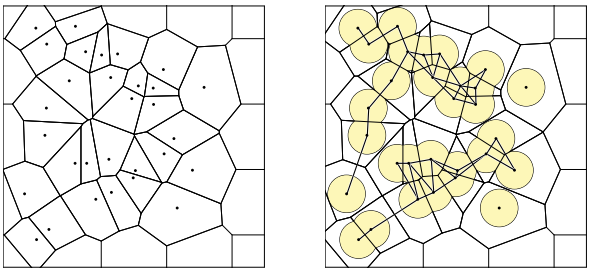


Figure 1: The Voronoi diagram and α-complex of a point set in the plane.

Related Work The classic reference for α-complexes is the survey by Edelsbrunner [6]. Several interesting variations of α-complexes have been proposed including conformal α-complexes [8] which use an alternative to weighting to approximate variations in radii and alpha-beta witness complexes [1] which relax the condition that the output be embedded in $\mathbb{R}^d$.

We put a restriction on the class of weight functions that are permitted. A generalization to arbitrary weights would mean a new output-sensitive algorithm for convex hulls. The restriction is precisely that used by Cheng et al. [4] on sliver exudation, a method that adds weights to Delaunay triangulations to eliminate certain badly shaped simplices. That work is closely related to

*Partially supported by the National Science Foundation under grant number CCF-1464379

†University of Connecticut don.r.sheehy@gmail.com

further use of weights in surface reconstruction to account for curvature [3]. Other curve and surface reconstruction algorithms explicitly use α-complexes [2, 13].

2 Background

Voronoi and Power Diagrams The *Euclidean norm* of a point $x \in \mathbb{R}^d$ is denoted $\|x\|$ and the *Euclidean distance* between points $x, y \in \mathbb{R}^2$ is $\|x - y\|$. The distance from a point to a set is defined as $\mathbf{d}(x, P) := \min_{p \in P} \|x - p\|$. A *weighted point set* is a finite set $P \subset \mathbb{R}^d$ and a weight function $w : P \to \mathbb{R}_{\geq 0}$. An unweighted set may be viewed as a weighted set with all weights 0. The *power* of a weighted point p is the function $\pi_p(x) := \|p - x\|^2 - w(p)^2$. The *power distance* between a point x and a weighted set P is defined as $\pi_P(x) := \min_{p \in P} \pi_p(x)$. The *power diagram* Vor_P of a weighted point set P is the set of nonempty polyhedra called *Voronoi cells* indexed by subsets $S \subseteq P$ as follows.

$$\mathrm{Vor}_P(S) := \{x \in \mathbb{R}^d \mid \forall s \in S : \pi_P(x) = \pi_s(x)\}$$

One can easily check that for unweighted point sets, this yields the standard Euclidean Voronoi diagram.

The dual diagram, Del_P, is the set of convex closures of the sets S such that $\mathrm{Vor}_P(S)$ is nonempty. Duals of power diagrams go by several names including *weighted Delaunay triangulations*, *regular triangulations*, and *coherent triangulations*. These names assume that the points are in sufficiently general position that the duals are triangulations. In general the weighted Delaunay triangulation is the orthogonal projection in $\mathbb{R}^d$ of the lower convex hull of the points $P^+ := \{(p, \|p\|^2 - w(p)^2) \in \mathbb{R}^{d+1} \mid p \in P\}$.

We say that a point set P is *mildly weighted* if for all $p \in P$, we have $w(p) < \frac{1}{2} \min_{q \in P \setminus \{p\}} \|p - q\|$. In particular, this implies that if the points are viewed as balls with radii equal to the weights, then the balls are disjoint. Unweighted points are mildly weighted.

For mildly weighted points P, we define the *weighted feature size* as

$$\mathbf{f}_{P,w}(x) := \sqrt{\min_{(u,v) \in \binom{P}{2}} \max_{p \in \{u,v\}} \pi_p(x)}.$$

This is the square root of the second smallest power distance from x to a point of P. If the points were not mildly weighted, the feature size could be imaginary at some points. If the points are unweighted, then we will simply write $\mathbf{f}_P$ for the feature size, and, in this case, the square root of the power distance is just the Euclidean distance. The function $\mathbf{f}_P$ is sometimes called the Ruppert local feature size and is ubiquitous in the analysis of Delaunay and Voronoi refinement mesh generation [12].

Weighted α-Complexes An *orthoball* of a set of weighted points S is a ball B with center c and radius r such that $\pi_p(c) = r^2$ for all $p \in S$. The minimum radius for an orthoball of S is called the *orthoradius*. For unweighted points, the orthoball is called the *circumball* and the orthoradius is called the *circumradius*.

The α-offsets of a weighted point set are defined as $P^\alpha := \{x \in \mathbb{R}^d \mid \pi_P(x) \leq \alpha^2\}$. A Voronoi cell of a subset $\sigma \subseteq P$ *restricted* to the offsets is defined as $\mathrm{Vor}_P^\alpha(\sigma) := \mathrm{Vor}_P(\sigma) \cap P^\alpha$ and the corresponding Voronoi diagram is $\mathrm{Vor}_P^\alpha := \{\mathrm{Vor}_P^\alpha(\sigma) \mid \sigma \subseteq P\}$. The *$\alpha$-complex* is the subcomplex of the Delaunay triangulation restricted to the α-offsets as follows.

$$\mathrm{Del}_P^\alpha := \{\sigma \in \mathrm{Del}_P \mid \mathrm{Vor}_P^\alpha(\sigma) \neq \emptyset\}.$$

Equivalently the α-complex may be defined as the *nerve* of set of clipped Voronoi cells $\{\mathrm{ball}(p, \alpha) \cap \mathrm{Vor}_P(p) \mid p \in P\}$, i.e. an abstract simplicial complex with a simplex for every subset of P whose corresponding clipped Voronoi cells have a common intersection. The Nerve Theorem, a standard result in algebraic topology guarantees that Del_P^α is homotopy equivalent to P^α. This topological guarantee was extended by Edelsbrunner and Shah [7] and forms the foundation of many of the topological guarantees in surface reconstruction [5].

Aspect Ratios of Voronoi Cells We will assume here and throughout that all Voronoi cells are bounded and convex. There are two different ways this will be enforced. First, we will consider a global bounding ball Ω that contains all the points and restrict our attention to the intersection of the full Voronoi cells with Ω. Second, when considering α-complexes, we will intersect the Voronoi cell of a point p with the ball of radius α centered at P. Having bounded cells allows the following definition (illustrated in Figure 2).

Definition 1 *If P is a set of mildly weighted points and $F \in Vor_P$, the* aspect ratio *of F is defined as*

$$\mathrm{aspect}_P(F) := \frac{\max_{x \in F} \mathbf{f}_{P,w}(x)}{\min_{y \in F} \mathbf{f}_{P,w}(y)}.$$

More generally, we let aspect_P denote the geometric mean of the aspect ratios of all Voronoi cells (of all dimensions) in Vor_P, i.e.

$$\mathrm{aspect}_P := \left(\prod_{F \in \mathrm{Vor}_P} \mathrm{aspect}(F) \right)^{1/f},$$

where $f = |\mathrm{Vor}_P|$. A more useful way to write this definition is the following.

$$f \log(\mathrm{aspect}_P) = \sum_{F \in \mathrm{Vor}_P} \log(\mathrm{aspect}_P(F)). \qquad (1)$$

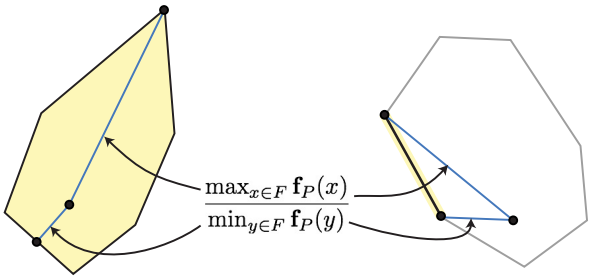


Figure 2: The aspect ratio of two cells of a Voronoi diagram. Left: a 2-dimensional cell. Right: a 1-dimensional cell. In both cases, the distances are measured from the point of P (in the interior of the 2-dimensional cell) to the nearest and farthest points in the cell F.

Well-Spaced Points A set of points M inside a bounding domain Ω is called *τ-well-spaced* if for all $q \in M$, $\text{aspect}_M(\text{Vor}_M(q)) \leq \tau$. As Voronoi cells of well-spaced points are nearly balls, simple packing arguments imply that there is a constant c_1 such that $\text{Vor}_M(q)$ has at most c_1 faces for all $q \in M$. Given a set of n points P and a bounding ball Ω, there exists a τ-well-spaced superset M of P as long as $\tau > 2$. Asymptotically minimal well-spaced supersets are *graded* in the sense that there is a constant K such that for all $v \in M$, we have $\mathbf{f}_P(v) \leq K\mathbf{f}_M(v)$. The grading condition implies that there is a constant c_2 such that for all $r > 0$, at most c_2 points of M have Voronoi cells intersecting $\text{annulus}(q, r, 2r)$ for any $q \in P$ [14], where $\text{annulus}(q, r, 2r)$ denotes $\text{ball}(q, 2r) \setminus \text{ball}(q, r)$. The constants K, c_1 and c_2 only depend on d and τ. Moreover, such a superset can be found in $O(n \log n + |M|)$ time [10]. Finally, we will use another important fact about graded, well-spaced point sets, namely that there is a constant γ such that $r \leq \gamma \mathbf{f}_M(x)$ for all x in any empty ball of radius r (see [9, Lemma 6.1]).

We will say a point set P is *annulus-free* if there is no point p and radius R such that $\text{ball}(p, r)$ contains more than one point of P and $\text{annulus}(p, r, 10r)$ contains no points of P. The constant 10 here is arbitrary. The size of a τ-well-spaced superset M from an annulus-free set P is known to be $O(n)$, so the running time to compute M is $O(n \log n)$ [14]. For α-complexes, point sets that are not annulus-free are rather uninteresting: if $r > \alpha$ then the ball points in the ball form a separate component; if $r \ll \alpha$ then the points are much closer than the scale and so replacing them with a single point results in (Hausdorff-)close offsets.

A Kinetic View of Refinement Given a set $S \subset \mathbb{R}^d$ of $d+2$ points in general position, there are precisely two different triangulations of S. A bistellar flip is a local change in a triangulation that swaps between the two triangulations of such a subset of $d+2$ points. Given a point set P in a bounding ball B and a constant τ, there exists a τ-well-spaced superset $M \supseteq P$. Starting from the Delaunay triangulation of M, one may obtain the Delaunay triangulation of P, by incremental bistellar flips that ultimately remove the points of $M \setminus P$ except those on the convex hull. This is done by changing the weights linearly and tracking the incremental changes that occur in the weighted Delaunay triangulation. As the change in weights may be viewed as a change in heights for a kinetic convex hull problem, the combinatorial changes can all be computed by replacing the coordinates in the usual Delaunay in-sphere predicate with the linear functions describing the motion. These changes are stored in a heap and are processed one at a time.

3 The Algorithm

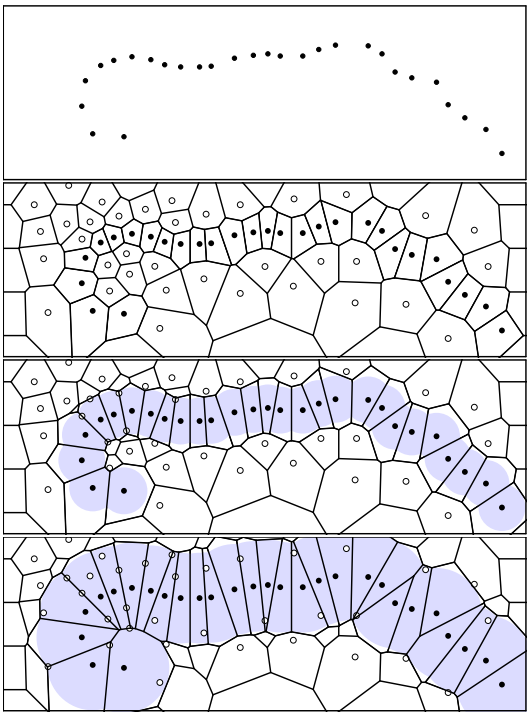


Figure 3: The algorithm is illustrated from top to bottom in terms of the Voronoi diagram. Starting from the input points (black), Steiner points are added (white). Weight is then added to the input points causing local changed to the Voronoi diagram until the weighted Voronoi cells of the input points contain the α-offsets.

In this section, we describe the algorithm for computing the α-complex of a set of mildly weighted points and prove its correctness. The algorithm starts by building a linear-size Delaunay triangulation of a well-spaced superset M of the input points P. The extra points

are called *Steiner points*. Then it adjusts the weights to match the input weights, leaving the weights of the Steiner points as zero. The projective view of weighted Delaunay triangulation assigns a height to a point p equal to $\|p\|^2 - w(p)^2$. Now, this height is treated as a $(d+1)$st coordinate and the weighted Delaunay triangulation is the orthogonal projection of the lower convex hull back into $\mathbb{R}^d$. In the algorithm, we treat the weights as a function of time, so, the weight $w(p)$ is specified in the input but the algorithm uses

$$w(p,t) = \begin{cases} \sqrt{w(p)^2 + t} & \text{if } p \in P \\ w(p) & \text{otherwise.} \end{cases}$$

The weighted point set at time t is denoted M_t. The height of a point p at time t is

$$h(p,t) := \|p\|^2 - w(p,t)^2.$$

As t increases, the input points get pulled downward. Generically, each combinatorial change in the weighted Delaunay triangulation is a single bistellar flip. As the input points move downward, the Steiner points are flipped out of the triangulation. Note that the definition of $w(p,t)$ guarantees that the height $h(p,t)$ is either constant or a linear function of t. Weighted points $S = \{p_1, \ldots, p_{d+2}\}$ in $\mathbb{R}^d$ have a common orthoball exactly when they lie on a common hyperplane after lifting. Thus, we can check this condition by computing

$$\text{flip}_S(t) = \det \begin{bmatrix} p_{1,1} & \cdots & p_{d+2,1} \\ \vdots & \cdots & \vdots \\ p_{1,d} & \cdots & p_{d+2,d} \\ h(p_1,t) & \cdots & h(p_{d+2},t) \\ 1 & \cdots & 1 \end{bmatrix}.$$

Note that $\text{flip}_S(t)$ is a linear function of t and so we can compute the *flip time* t_S satisfying $\text{flip}_S(t_S) = 0$. More generally, when the coordinates of the points (and not just the heights) are polynomials in t, $\text{flip}_S(t)$ is some polynomial, and computing the roots of $\text{flip}_S(t)$ gives the changes in the Delaunay triangulation as the points move. This more general setting is the quintessential example in the field of kinetic data structures, a generalization of the line-sweep paradigm.

In our case, we are only modifying the height and so the algebraic computations are much simpler. The main data structure is a heap called the *flip heap* that stores the possible flips ordered by time. We identify each flip with a facet in Del_{M_t} The steps of the construction given in Algorithm 1.

The following lemma guarantees that stopping the kinetic part of the algorithm at time $t = \alpha^2$, will still allow us to construct the α-complex.

Lemma 2 *If a simplex $\sigma \in \text{Del}_P$ is contained in a d-simplex $\sigma' \in \text{Del}_P$, of orthoradius at most $\sqrt{t}$ then the flip time when σ first appears in the* AlphaComplex *algorithm is at most t.*

Algorithm 1 Compute the α complex for a mildly weighted point set.

1: **procedure** AlphaComplex(P, α)
2: Compute a graded, τ-well-spaced superset M of P in a bounding ball B containing P.
3: For each facet F of Del_M, compute the flip time t_F and insert the key-value pair (t_F, F) into the flip heap. Skip the insertion if $t_F > \alpha^2$.
4: **while** The flip heap is nonempty **do**
5: Pop a facet F off the heap
6: Attempt to flip F, and push any new facets to the heap if their flip time is at most α^2.
7: Output all simplices containing only points of P that have an orthoradius at most α.

Proof. First, observe that for sufficiently large α, every simplex of Del_P will appear at some time. Let t_0 be the time when σ first appears and let c be the orthocenter of the corresponding flip. Let p be any vertex of σ, so $t_0 = \pi_p(c)$. This means that c is the orthocenter of σ', the smallest d-dimensional simplex (by orthoradius) in Del_P containing σ. Suppose for contradiction that $t_0 > t$. At time t, some vertex q of M has a Voronoi cell containing c such that $q \neq p$. So, $\pi_q(c,t) < \pi_p(c,t)$ and so it follows from the definition of the power distance that $0 \leq \pi_q(c) \leq \pi_p(c) - t$. Because $t_0 = \pi_p(c)$, the preceding inequality implies that $t_0 \leq t$, a contradiction. Therefore, we conclude that the flip time t_0 when σ first appears is at most t as claimed. □

Lemma 2 now implies the following theorem as it guarantees that the algorithm finds all simplices of the α-complex despite stopping at time α^2.

Theorem 3 *Given a set P of mildly weighted points and a parameter α, the* AlphaComplex *algorithm above returns the weighted α-complex of P.*

4 Analysis

The starting point for the analysis of the running time of our output-sensitive algorithm for α-complexes is the following lemma of Miller & Sheehy (proven as a first step in Lemma 6 of [11]) describing the flips in the algorithm; it says that the flips are in one-to-one correspondence with intersections between Vor_M and Vor_P. We extend it to the weighted case.

Lemma 4 *Let $P \subset \mathbb{R}^d$ be a finite point set and let $M \supseteq P$ be any superset of P. A set S of $d+2$ points of M will be involved in a flip in the kinetic refinement reversal if and only if $\text{Vor}_M(S \setminus P) \cap \text{Vor}_P(S \cap P) \neq \emptyset$.*

The preceding lemma gives a static way to count the kinetic changes in the algorithm; it suffices to count intersections between the starting and ending Voronoi diagrams. In previous work, a coarse bound on the number of intersections was given by exploiting the fact that the point set M in the algorithm is well-spaced. That analysis give a bound of $O(\log \Delta)$ flips per simplex. We will now give a more refined analysis that bounds these flips instead in terms of a more local parameter. Incidentally, this will also improve the running-time guarantee for certain known hard instances for Delaunay triangulation, such some that are known to produce $\Theta(n^2)$ simplices in $\mathbb{R}^3$.

Lemma 4 implies a natural way to partition the set of flips by assigning the set of points S in the flip to the simplex $S \cap P$ of Del_P. This is clearly possible, because according to the lemma $\mathrm{Vor}_P(S \cap P)$ must be nonempty for the flip to occur and so $S \cap P \in \mathrm{Del}_P$. To count the total flips, it will suffice to bound the number of flips assigned to each simplex of Del_P.

Lemma 5 *Let $P \subset \mathbb{R}^d$ be a set of mildly weighted points. Let M be a τ-well-spaced superset of P. Let $F \in \mathrm{Vor}_P$ be any face. There is a constant c that depends only on d and τ such that*

$$|\{G \in \mathrm{Vor}_M \mid F \cap G \neq \emptyset\}| \leq c \log(\mathrm{aspect}_P(F)).$$

Proof. Let $S \subset P$ be such that $F = \mathrm{Vor}_P(S)$ and let $q \in S$ be any point. Recall from Section 2, for graded supersets M, there is a constant c_2 such that for all $r > 0$, at most c_2 points of M have Voronoi cells intersecting $\mathrm{annulus}(q, r, 2r)$. Moreover, there is a constant c_1 such that each such Voronoi cell has at most c_1 faces. Let $x = \mathrm{argmax}_{x \in F} \mathbf{f}_{P,w}(x)$ and $y = \mathrm{argmin}_{y \in F} \mathbf{f}_{P,w}(y)$. Let $A_i = \mathrm{annulus}(q, 2^i r, 2^{i+1} r)$ for all integers i, where $r = \|q - y\|$. By Lemma 9, $\|x - q\| \leq r \ \mathrm{aspect}_P(F)$. It follows that $F \subset \bigcup_{i=0}^{\lceil \log(\mathrm{aspect}_P(F)) \rceil - 1} A_i$. As there are at most $c_1 c_2$ intersections in any A_i and thus at most $c_1 c_2 \lceil \log(\mathrm{aspect}_P(F)) \rceil$ faces of Vor_M intersect F. □

We first give an upper bound on the total number of flips in terms of the aspect ratio of Vor_P. This bound applies independent of the value of α and thus gives a potentially tighter bound on the number of flips in computing the full Delaunay triangulation of P.

Theorem 6 *For a mildly weighted point set P and any constant $\alpha \geq 0$, the total number of flips in the* AlphaComplex *algorithm is $O(f \log(\mathrm{aspect}_P))$, where $f = |\mathrm{Vor}_P|$.*

Proof. By Lemma 4 and (1), it will suffice to prove that each k-face F of Vor_P intersects at most $O(\log \mathrm{aspect}(F))$ $(d-k)$-faces of Vor_M, which is precisely the conclusion of Lemma 5. □

The following lemma guarantees that the flips that occur in the algorithm, all happen "close" to the input points. That is, none of the intersections causing a flip are farther than a constant times α from the points of P. This is the key to proving an output-sensitive running time for α-complexes as it says that the output simplices are discovered (approximately) in order of their orthoradius.

Lemma 7 *Let P be a set of mildly weighted points, let $p \in P$, let $\alpha \in \mathbb{R}$, and let x be the center of a flip in the* AlphaComplex *algorithm that occurs at time t. There is a constant c_3 that depends only on τ and d such that if $\mathbf{f}_P(p) \leq 2\sqrt{2}\alpha$ and $t \leq \alpha^2$, then $\|x - p\| \leq c_3 \alpha$.*

Proof. Let $b := w(p,t) = \sqrt{w(p)^2 + t} \leq \sqrt{2}\alpha$, where the last inequality follows from the mildness assumption and the hypothesis that $t \leq \alpha^2$. Let r be the radius of the orthoball of the flip centered at x, so $r = \sqrt{\|x - p\|^2 - b^2}$. Let y be the point on the line segment $\overline{xp}$ such that $\|x - y\| = r$. Now, for τ-well-spaced points M, it is known that there is a constant γ such that any ball of radius r that contains no points of M has the property that $r \leq \gamma \mathbf{f}_P(z)$ for all z in the ball. So, in our case, this implies that $r \leq \gamma \mathbf{f}_P(y)$ because $y \in \mathrm{ball}(x, r)$ and no points M lie in this ball.

We consider two cases. First, if $r < b$, then $\|x-p\|^2 = r^2 + b^2$ implies that $\|x - p\| \leq 2\alpha$. So in that case, it suffices to choose $c_3 \geq 2$. Second, if $r \geq b$, then similarly,

$$\begin{aligned}
\|x - p\| &\leq \sqrt{2} r \\
&\leq \sqrt{2}\gamma \mathbf{f}_M(y) \\
&\leq \sqrt{2}\gamma \mathbf{f}_P(y) \\
&\leq \sqrt{2}\gamma (\mathbf{f}_P(p) + \|p - y\|) \\
&\leq \sqrt{2}\gamma (2\sqrt{2}\alpha + b) \\
&\leq 6\gamma\alpha.
\end{aligned}$$

So, $c_3 = 6\gamma$ is the desired constant. □

The preceding lemma provides the main new tool for analyzing the running time of AlphaComplex and also indicates why the output-sensitive term is not precisely the output size but rather the size of a constant factor larger α-complex.

Theorem 8 *For a mildly weighted, annulus-free point set $P \subset \mathbb{R}^d$, the total running time of* AlphaComplex *(P, α) is $O(f \log(n) \log(\alpha/s))$ where $s := \min_{p \in P} \mathbf{f}_P(p)$, $f = |\mathrm{Del}_P^{c_3 \alpha}|$, and c_3 is the constant from Lemma 7.*

Proof. The preprocessing phase to compute M and Del_M takes $O(n \log n)$ time [12]. Adjusting Del_M to form Del_{M_t} requires only a constant number of flips per vertex. This fact is used in the work of Cheng et al. on

sliver exudation [4], but also follows from the arguments of Lemma 7.

The main loop processes flips and adds them to a heap. Lemma 7 implies that all of the flips are contained in $P^{c_3\alpha}$. Let f be the number of faces of $\mathrm{Vor}_P^{c_3\alpha}$. The aspect ratio of each such face is at most $\log(c_3\alpha/s)$. So, Theorem 6 implies that the total number of flips is $O(f\log(\alpha/s))$. Not every flip on the heap is performed, but at most a constant number of potential flips are added to the flip heap for every actual flip, so the total number of heap operations is also $O(f\log(\alpha/s))$. Thus, the total running time is $O(f\log(n)\log(\alpha/s))$. □

5 Conclusion

In this paper, we generalized the output-sensitive algorithm of Miller and Sheehy for Delaunay triangulations to also give guarantees for mildly weighted points and for α-complexes. Along the way, we generalized the notion of the aspect ratio of a Voronoi cell to give a meaningful definition for weighted Voronoi cells of dimension less than d. This new definition gives a sharper analysis for the Miller and Sheehy algorithm and also simplifies the analysis of the modified algorithm.

One future direction is to look at more recent generalizations of α-complexes introduced for topological inference from point cloud data such as the alpha-beta witness complexes of Attali et al. [1]. It may also be useful to apply this approach to Voronoi-based manifold reconstruction as many use the Delaunay triangulation restricted to the manifold which is a subcomplex of the Delaunay triangulation restricted to a union of balls (see for example [2, 13]). We are also interested in relaxing the mild weighting assumption and replacing it with a Lipschitz condition on the weights. In that case, weights could be larger as long as they don't vary too quickly.

References

[1] D. Attali, H. Edelsbrunner, J. Harer, and Y. Mileyko. Alpha-beta witness complexes. In *WADS*, 2007.

[2] F. Bernardini and C. L. Bajaj. Sampling and reconstructing manifolds using alpha-shapes. In *In Proc. 9th Canad. Conf. Comput. Geom*, pages 193–198, 1997.

[3] J.-D. Boissonnat and A. Ghosh. Manifold reconstruction using tangential Delaunay complexes. *Discrete & Computational Geometry*, 51(1):221–267, 2014.

[4] S.-W. Cheng, T. K. Dey, H. Edelsbrunner, M. A. Facello, and S.-H. Teng. Sliver exudation. *JACM: Journal of the ACM*, 47, 2000.

[5] T. K. Dey. *Curve and Surface Reconstruction : Algorithms with Mathematical Analysis*. Cambridge University Press, 2007.

[6] H. Edelsbrunner. The union of balls and its dual shape. *Discrete & Computational Geometry*, 13:415–440, 1995.

[7] H. Edelsbrunner and N. R. Shah. Triangulating topological spaces. In *Proceedings of the 10th ACM Symposium on Computational Geometry*, pages 285–292, 1994.

[8] J. Giesen, F. Cazals, M. Pauly, and A. Zomorodian. The conformal alpha shape filtration. *The Visual Computer*, 22(8):531–540, 2006.

[9] B. Hudson, G. Miller, and T. Phillips. Sparse Voronoi Refinement. In *Proceedings of the 15th International Meshing Roundtable*, pages 339–356, Birmingham, Alabama, 2006. Long version available as Carnegie Mellon University Technical Report CMU-CS-06-132.

[10] G. L. Miller, T. Phillips, and D. R. Sheehy. Beating the spread: Time-optimal point meshing. In *Proceedings of the 26th ACM Symposium on Computational Geometry*, pages 321–330, 2011.

[11] G. L. Miller and D. R. Sheehy. A new approach to output-sensitive construction of Voronoi diagrams and Delaunay triangulations. *Discrete & Computational Geometry*, 52(3):476–491, 2014.

[12] G. L. Miller, D. R. Sheehy, and A. Velingker. A fast algorithm for well-spaced points and approximate Delaunay graphs. In *Proceedings of the 29th annual Symposium on Computational Geometry*, pages 289–298, 2013.

[13] S. H. Park, S. S. Lee, and J. H. Kim. A surface reconstruction algorithm using weighted alpha shapes. In L. Wang and Y. Jin, editors, *Fuzzy Systems and Knowledge Discovery*, volume 3613 of *Lecture Notes in Computer Science*, pages 1141–1150. Springer, 2005.

[14] D. R. Sheehy. New Bounds on the Size of Optimal Meshes. *Comput. Graph. Forum*, 31(5):1627–1635, 2012.

A The Weighted Aspect Ratio

In the following lemma, we show that if we replace the power distance with the Euclidean distance, the aspect ratio cannot go up.

Lemma 9 *Let P be a mildly weighted point set. Let $S \subset P$ and let $F = \mathrm{Vor}_P(S)$, where $|S| \geq 2$. Let $x = \mathrm{argmax}_{x\in F}\,\mathbf{f}_{P,w}(x)$ and $y = \mathrm{argmin}_{y\in F}\mathbf{f}_{P,w}(y)$. For all $q \in S$, $\|q-x\| \leq \|q-y\|\mathrm{aspect}_P(F)$.*

Proof. First, observe that by the choice of x and y, we know that $\mathbf{f}_{P,w}(x)^2 = \pi_q(x)$ and $\mathbf{f}_{P,w}(y)^2 = \pi_q(y)$. The desired inequality now follows from the definitions of π_q and $\mathrm{aspect}_P(F)$ as follows.

$$\begin{aligned}\|q-x\|^2 &= \pi_q(x) + w(q)^2\\ &= \pi_q(y)\mathrm{aspect}_P(F)^2 + w(q)^2\\ &\leq \mathrm{aspect}_P(F)^2(\pi_q(y) + w(q)^2)\\ &= \|q-y\|^2\mathrm{aspect}_P(F)^2.\end{aligned}$$

□

Dynamic data structures for approximate Hausdorff distance in the word RAM

Timothy M. Chan* Dimitrios Skrepetos†

Abstract

We give a fully dynamic data structure for maintaining an approximation of the Hausdorff distance between two point sets in a constant dimension d, a standard problem in computational geometry. Our solution has an approximation factor of $1+\varepsilon$ for any constant $\varepsilon > 0$ and expected update time $O(\frac{\log U}{\log\log n})$, where U is the universe size, and n is the number of the points. The result of the paper greatly improves over the previous exact method, which required $O(n^{5/6}\text{polylog}\, n)$ time and worked only in a semi-online setting. The model of computation is the word RAM model.

1 Introduction

The problem of computing the Hausdorff distance between a red point set R and a blue point set B is to find $\max_{b\in B}\min_{r\in R} d(b,r)$ where $d(\cdot,\cdot)$ is a distance function. The dynamic version of the Hausdorff distance problem is to solve the problem under a series of insertions and deletions of points both from the red point set and from the blue point set. The approximate dynamic version of the Hausdorff distance is to solve the dynamic version of the problem allowing a factor of approximation.

Chan [4] provided a solution for the semi-online maintenance of Hausdorff distance in 2-d that had an update time of $O(n^{5/6}\text{polylog}\, n)$ in the worst case. Here *semi-online* means that when we insert an element we are given its deletion time in advance. Chan also gave a fully dynamic solution for a decision version of the problem in 2-d that had an amortized update time of $O(n^{1/2}\text{polylog}\, n)$.

In this paper we show that the update time can be greatly improved if we allow approximation. We give a dynamic data structure that maintains the Hausdorff distance within an approximation factor of $1+\varepsilon$ in *sublogarithmic* update time—or more concretely, $O(\frac{\log U}{\log\log n})$ time.

Note that our update time of $O(\frac{\log U}{\log\log n})$ greatly improves over the previous bound of $O(n^{5/6}\text{polylog}\, n)$ for the semi-online maintenance of the exact Hausdorff distance, as it is purely polylogarithmic and fully dynamic. Furthermore, our solution works for any constant dimension d, while the previous method works only in 2-d.

We originally started this research for a closely related problem, the dynamic bichromatic closest pair problem, studied by Eppstein [3], which involves finding $\min_{b\in B}\min_{r\in R} d(b,r)$. However, this seems easier to approximate than the Hausdorff distance, as one could obtain a constant-factor approximation with update time of $O(\log\log U)$ by extending the technique of Chan [1].

1.1 Model of computation

We assume the word RAM model of computation; that is, the point coordinates are drawn from the set $[U] = \{0,1,\ldots,U-1\}$, the words are composed of $\Theta(\log U)$ bits, and each standard word operation (such as shifting, arithmetic and logical operations) takes constant time. The distance function is assumed to be the Euclidean distance function. The approximation factor ε and the dimension d are considered to be constant in both the running time and space of our algorithms.

2 Approximate dynamic Hausdorff distance

In section 2.1 we give a dynamic data structure that solves the approximate decision Hausdorff distance problem. That is, given a distance r and an approximation factor ε fixed at preprocessing time, we want to answer decision queries of the form: "Is the Hausdorff distance greater than r?". The data structure should return "yes" if the Hausdorff distance is greater than $r(1+\varepsilon)$ and "no" if it is less than r. In section 2.2 a stronger version of the decision problem is considered: instead of a fixed r, the data structure can answer decision queries for all $r \in \{2^i \mid 0 \le i \le \log U\}$ at the same time. Finally, in section 2.3 we solve the original version of the approximate Hausdorff distance problem; that is, we compute a $(1+\varepsilon)$-approximation of the Hausdorff distance.

*Cheriton School of Computer Science, University of Waterloo, tmchan@uwaterloo.ca

†Cheriton School of Computer Science, University of Waterloo, dskrepet@uwaterloo.ca

2.1 Decision queries for a fixed distance r

The first step is to compute a grid G_s of the d-dimensional space composed of cells that are hypercubes with $s = r\varepsilon$ side length. For a cell c, another cell c' is a neighbor of c if the minimum distance between the centers of c and c' is less than or equal to r. We imagine that all the points that fall inside a cell are *rounded* to the center of the cell. Let $neighbors_c$ denote the set of the neighbors of the cell c that is produced by this definition. It is clear that the number of neighbors of a cell is dependent only on ε and d, so it will be treated as a constant in the rest of the paper. Because of rounding the approximation factor is $1 + O(\varepsilon)$.

Each cell c of the grid has a unique ID; thus, for each point $p = (x_1, x_2, \ldots, x_d) \in c$ the ID of the cell is $id(c) = (\lfloor \frac{x_1}{s} \rfloor, \lfloor \frac{x_2}{s} \rfloor, \ldots, \lfloor \frac{x_d}{s} \rfloor)$. Each cell c also has three counters, one for the number of blue points inside the cell, $blueCount_c$, one for the number of red points inside the cell, $redCount_c$, and one for the number of red points in any cell c' that belongs to the set $neighbors_c$, $redNeighborsCount_c$. Furthermore, each cell c maintains a boolean flag, $flag_c$, with the following semantic: $flag_c$ is set if and only if $blueCount_c > 0$ and there is no neighbor cell c' such that $redCount_{c'} > 0$, which is the same with checking whether $redNeighborsCount_c > 0$. If there is at least one such flag set, then the Hausdorff distance is greater than r (after rounding). Otherwise, it is at most r. We keep a global variable *globalCount* that stores the number of cells with flags that have been set. With this variable, we can answer decision queries in constant time.

In order to keep the total space linear in the number of points in the red and blue point sets, we use a hash table to store only the $O(n)$ cells that are nonempty (where n is the number of points of both the point sets), since in the worst case each point occupies a different cell. We use the ID of the cell as key. The hash table supports insertions, deletions, and searching in expected constant time.

To insert a point p, we compute the ID of the cell c that contains the point (that is, $id(c) = (\lfloor \frac{x_1}{s} \rfloor, \lfloor \frac{x_2}{s} \rfloor, \ldots, \lfloor \frac{x_d}{s} \rfloor)$). If the cell does not exist in the hash table, a new entry for the cell is created and inserted to the hash table with $redCount_c$ and $blueCount_c$ equal to zero, $redNeighborsCount_c$ equal to the accumulation of all the $redCount_{c'}$ values for every cell c' in the set $neighbors_c$, and $flag_c$ equal to zero. Assume that the inserted point is red. Then, the $redCount_c$ value is incremented. Afterwards, for each cell c' in the set $neighbors_c$ we increment the value $redNeighborsCount_{c'}$, and if $blueCount_{c'} > 0$ and $redNeighborsCount_{c'}$ was zero prior to the insertion, then the $flag_{c'}$ is reset, and the *globalCount* is decremented. Now, assume that the inserted point is blue. Then the $blueCount_c$ value is incremented. Afterwards, if the $blueCount_c$ value was zero prior to the insertion and $redNeighborsCount_c > 0$, the $flag_c$ is set, and *globalCount* is incremented. The pseudocode of the algorithm for inserting points is given in Algorithm 1.

```
1  Find the cell c that contains the point p
2  if c does not exist in the hash table then
3  |  create it and insert to the hash table
4  |  set blueCount_c, redCount_c,
   |  redNeighborsCount_c, flag_c to 0
5  |  for all c' ∈ neighbors_c do
6  |  |  redNeighborsCount_c += redCount_c'
7  if p is red then
8  |  increment redCount_c
9  |  for all c' ∈ neighbors_c do
10 |  |  increment redNeighborsCount_c'
11 |  |  if blueCount_c' == 1 then
12 |  |  |  set flag_c'
13 |  |  |  increment globalCount
14 else
15 |  increment blueCount_c
16 |  if blueCount_c == 1 then
17 |  |  if redNeighborsCount_c > 0 then
18 |  |  |  set flag_c increment globalCount
```

Algorithm 1: Insert-Point

To delete a point p, the ID of the cell c that contains the point is computed $(id(c) = (\lfloor \frac{x_1}{s} \rfloor, \lfloor \frac{x_2}{s} \rfloor, \ldots, \lfloor \frac{x_d}{s} \rfloor))$, and $redCount_c$ or $blueCount_c$ is accordingly decremented. If the point is red, for every c' in the set $neighbors_c$ we decrement the $redNeighborsCount_{c'}$ value, and if this value becomes zero, the $flag_{c'}$ is reset and the *globalCount* is decremented. If the point is blue and the cell c does not have any other blue points after the deletion, the $flag_c$ is reset and the *globalCount* is decremented. Finally, if the cell does contain any points (either red or blue), it is deleted from the hash table. The pseudocode of the algorithm for deleting points is given in Algorithm 2.

In order to answer a distance query, we check the value *globalCount*, and we return "yes" if it is zero and "no" otherwise.

The aforementioned data structure can handle insertions and deletions of points in constant time. The queries can be answered in constant time as well. Consequently, this data structure with a linear space and preprocessing time can handle point insertions and deletions and can answer queries for a fixed distance r and a fixed ε in constant time.

2.2 Decision queries for all distances $r \in \{2^i \mid 0 \leq i \leq \log U\}$

The naive way to perform the query for all distances $r \in 1, 2, 4, ..., U$ would be to create $O(\log U)$ instances

```
1  Find the cell c that contains the point p
2  if p is red then
3  |  decrement redCount_c
4  |  for all c' ∈ neighbors_c do
5  |  |  decrement redNeighborsCount_c'
6  |  |  if redNeighborsCount_c' == 0 then
7  |  |  |  reset flag_c'
8  |  |  |  decrement globalCount
9  else
10 |  decrement blueCount_c
11 |  if blueCount_c == 0 then
12 |  |  reset flag_c
13 |  |  decrement globalCount
14 if (redCount_c == blueCount_c == 0) then
15 |  delete cell c from the hash table
```

Algorithm 2: Delete-Point

of the above data structure, but this would increase the required space to $O(n \log U)$ words and the required update time to $O(\log U)$. In order to reduce these bounds, we use standard word RAM techniques, more specifically word packing tricks.

2.2.1 Interpreting the grids as a quadtree

In order to handle multiple distance values of r at the same time, we imagine the space decomposition that is implicitly imposed by the grids of the multiple side lengths as a d-dimensional complete quadtree with all the leaves at the same depth (although such a tree is not explicitly maintained) for the hypercube of side length U and with height $O(\log U)$. Such a quadtree creates hypercubes of side length r with $r \in \{2^i \mid 0 \leq i \leq \log U\}$, but we need hypercubes of side length $r\varepsilon$ as explained in section 2.1. Therefore, in order to adjust the standard quadtree decomposition of the space, we have to create $\frac{1}{\varepsilon^d}$ children to each of the leaves of the tree. Notice that all the nodes of the tree at depth i correspond to the grid G_s for $s = \frac{U}{2^i}\varepsilon$, and each node of the tree at depth i corresponds to a cell of that grid.

The $flag_c$ value of a cell c of a grid G_s has the same meaning as in section 2.1, but it is not explicitly maintained. The single register *globalCount* is replaced by the array *globalCountArray* that has $O(\log U)$ size, and each value at index i corresponds to the number of the cells c with nonzero $flag_c$ for the grid G_s with $s = \frac{U}{2^i}\varepsilon$. The goal of the update process is to find the array *changeArray* that stores the changes that need to be done to *globalCountArray*.

Two flags are related to each node of the tree, $emptyRed_c$ and $emptyBlue_c$. The first flag is set to 1 if there are no red points in the cell c of the grid G_s with $s = \frac{U}{2^i}\varepsilon$ that corresponds to that node, and 0 otherwise. The second flag is similarly defined for the blue points.

2.2.2 Applying word RAM techniques

The key idea in achieving sublogarithmic time is to perform the operations for multiple grids in constant time by taking advantage of the tools that word RAM provides us with; that is, we have $O(\log U)$ grids, but we need to compress them. Starting at the root of the quadtree, we store all the $emptyRed_c$ flags of the nodes of the quadtree that are in the subtree s of the root of depth b, where b is a value to be determined later, in a RAM word ($emptyRed_s$), and we store all the $emptyBlue_c$ flags of that subtree in another word ($emptyBlue_s$). The same is done for each subtree of depth b of every node at depth $ib+1$ for $0 \leq i \leq \frac{\log U}{b} - 1$. We note that only the words of the subtrees that are not entirely composed of zeros need to be stored, and we store them in a hash table using the ID of the cell that corresponds to the root of the subtree and the depth of the subtree as key. The $redCount_c$ and the $blueCount_c$ of the nonempty cells c of the most detailed grid G_s with $s = \varepsilon$ are stored in a separate word in another hash table using their ids as keys.

2.2.3 Maintaining the flags

When a point is inserted (deleted), we find the leaf cell c that corresponds to the grid G_s with $s = \varepsilon$, and we increment (decrement) $redCount_c$ or $blueCount_c$ accordingly. We create or delete subtrees from the hash table when needed. If the number of the points of the same color with the inserted point is nonzero (zero for deleted point), there is nothing that needs to be done. Otherwise, we start processing the subtrees that lie in the path from c to the root of the quadtree in a bottom-up manner as follows.

If the inserted (deleted) point is red, for each subtree s on the path we update the $emptyRed_s$ word. To find the part of the *changeArray* that corresponds to s, we need to count the changes of the $flag_{c'}$ values of the cells $c' \in neighbors_c$ for each cell c in the path from the leaf of s in which the insertion (deletion) took place to the root of s, as only these cells are affected from the insertion (deletion). First we fetch the $emptyBlue_{s'}$ words of the subtrees s' that contain neighbors $c'_{root} \in neighbors_{c_{root}}$ of the root cell c_{root} of s (it is easy to see that these words also contain the neighbors for all cells c described above), and then we fetch the $emptyRed_{s''}$ words of subtrees s'' that contain the neighbors $c''_{root} \in neighbors_{c'_{root}}$ for all the root cells c'_{root} of the subtrees s' (similarly, these words also contain the neighbors for all cells in the subtree s'). With these words we can find the cells c' of each subtree s' whose $flag_{c'}$ needs to be set (reset) since we have access to all the neighbors of these cells. That can be done by checking for each cell c' if

there is at least one neighbor cell other than the cells c of p with nonzero number of red points. If there is at least one such cell, and we have an insertion of a red point, then the $flag_{c'}$ does not change. Otherwise, it is reset. On the contrary, if there is at least one such cell, and we have a deletion of a red point, then the $flag_{c'}$ does not change. Otherwise, it is set.

If the inserted (deleted) point is blue, for each subtree s on the path we update the *emptyBlue* word. To find the part of the *changeArray* that corresponds to s, we need to count the changes of the $flag_c$ values for each cell c in the path from the leaf of s in which the insertion (deletion) took place to the root of s, as only these cells are affected from the insertion (deletion). We fetch the constant number of *emptyRed* words of the subtrees of the same depth that contain neighbors $c'_{root} \in neighbors_{c_{root}}$ of the root cell c_{root} of s (these words contain also the neighbors for all cells c in the subtree s). With these words, if we have an insertion, we can determine if $flag_c$ for each cell c needs to be set by checking if there is at least one neighbor cell c' with nonzero number of red points. On the contrary, if we have a deletion, we only need to reset $flag_c$.

2.2.4 Choosing the value of b

We notice that the number of subtrees that need to be fetched while processing a subtree s is constant because a cell in a grid has only a constant number of neighbors. Supposing that the word operations take constant time, the update requires $O(\frac{\log U}{b})$ RAM words to be accessed and processed because there are that many subtrees on the path from the root to the leaf in which the update was done. The value of b needs to be maximized in order to minimize the update time, but there is an upper bound on b that is imposed by the need to fit all the *empty* flags of a subtree of depth b in a single word. More concretely, there are $O(2^{db})$ nodes in a tree of branching factor b and depth d, and each node needs only a bit for its flag. Therefore, we choose b to be $\delta \log\log n$ for a sufficiently small constant $\delta > 0$, so that $O(2^{db}) = o(\log n)$. Since the space usage is $O(n)$ words of nonempty subtrees with the same root depth and there are $O(\frac{\log U}{\log\log n})$ different depths of subtrees' roots, our data structure requires $O(n\frac{\log U}{\log\log n})$ words.

The word operations described in the previous section require only constant time, as it is easy to create a look-up table of $o(n)$ words with preprocessing time of $o(n)$. This is because the number of bits in a word in the above word operations is $O(2^{db}) = o(\log n)$, so the number of possible words is sublinear.

2.2.5 Maintaining the *globalCountArray* in $O(\frac{\log U}{\log\log n})$ worst-case time

Since each update adds or subtracts at each index of *globalCountArray* a is bounded by $O(1)$, which is dependent on the number of set $flag_{c'}$ for $c' \in neighbors_c$, the changes at every position of that array are of constant size per update, so we can encode $O(\log\log n)$ consecutive indexes of *changeArray* in a single word; thus we maintain an array called *tempCountArray* of size $O(\log U/\log\log n)$ words. In a period of $O(\log U/\log\log n)$ updates, we store the changes in the *tempCountArray* array in $O(\frac{\log U}{\log\log n})$ time per update. After a period of $O(\log U/\log\log n)$ updates, we update the *globalCountArray* in $O(\log U)$ steps. Therefore, the overall update of the *globalCountArray* can be done in $O(\frac{\log U}{\log\log n} + \log\log n) = O(\frac{\log U}{\log\log n})$ amortized time. A decision query to an index of *globalCountArray* is accomplished by adding the value at that position with the corresponding value from the *tempCountArray* in constant worst-case time.

The aforementioned data structure can be deamortized by following ideas by Dietz [2] for the partial sums problem. In a sequence of $O(\frac{\log U}{\log\log n})$ updates, we do all the steps that were described in the previous paragraph, but we also update *globalCountArray*[i mod $\frac{\log U}{\log\log n}$] in the i^{th} update, by adding to it the value in *tempCountArray*[i] in $O(\log\log U)$ time per update. It is easy to see that the words in the *tempCountArray* do not exceed the word size. Therefore, we obtain $O(\frac{\log U}{\log\log n})$ update time of *globalCountArray* in worst case.

We conclude that, with the above data structure we can answer the decision query for $r \in \{2^i \mid 0 \leq i \leq \log U\}$ using $O(n\frac{\log U}{\log\log n})$ words, $O(n\frac{\log U}{\log\log n} + n) = O(n\frac{\log U}{\log\log n})$ preprocessing time, $O(\frac{\log U}{\log\log n})$ update time, and constant query time.

2.3 Original version of the problem

With the current scheme we can maintain fully dynamically the Hausdorff distance with an approximation factor of 2. We now show how to make the data structure work with approximation factor of $1 + \varepsilon$. Let k be a constant parameter to be determined later. First create k instances of the data structure described in section 2.2 for distances $r_i = 2^{\frac{ki+j}{k}}$ with each data structure having a subscript $j = 0, \ldots, k-1$. The only twist that needs to be done to the j^{th} data structure is to rescale the coordinates of points by a factor of $\frac{1}{2^{j/k}}$.

To handle an update, we update the information to each one of the k data structures, and since k is constant (because it is dependent only on ε), and an update to one data structure requires $O(\frac{\log U}{\log\log n})$ time, one update takes $O(\frac{\log U}{\log\log n})$ time.

To perform a distance query to compute an approximation of the Hausdorff distance, we collect from each one of the k data structure the biggest distance that has an answer "yes" to a decision query. Afterwards, we return the the biggest of these distances. Both the decision and the distance query can take place in constant time.

It is clear that with the aforementioned scheme and with $k = \lfloor \frac{1}{\varepsilon} \rfloor$ we can obtain an approximation factor of $2^{\frac{1}{k}}(1+O(\varepsilon)) = 2^{O(\varepsilon)}(1+O(\varepsilon)) = (1+O(\varepsilon))(1+O(\varepsilon)) = 1 + O(\varepsilon)$, which can be made $1 + \varepsilon$ if we readjust ε. Therefore, the original version of the problem can be solved in $O(\frac{\log U}{\log \log n})$ expected update time.

3 Conclusion

Finding solutions with still better update time (e.g. $O(\log \log U)$) or with linear space remains interesting open problems.

References

[1] T. M. Chan. Closest-point problems simplified on the RAM. *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete algorithms*, 472–473, 2002.

[2] P. F. Dietz. Optimal algorithms for List Indexing and Subset Rank. *Proceedings of the First Annual Workshop on Algorithms and Data Structures*, 39–46, 1989.

[3] David Eppstein. Dynamic Euclidean minimum spanning trees and extrema of binary functions. *Discrete & Computational Geometry*, 13(1):111–122, 1995.

[4] T. M. Chan. Semi-Online Maintenance of Geometric Optima and Measures. *SIAM Journal on Computing*, 32(3):700–716, 2003.

Local Doubling Dimension of Point Sets

Aruni Choudhary * Michael Kerber†

Abstract

We introduce the notion of t-restricted doubling dimension of a point set in Euclidean space as the local intrinsic dimension up to scale t. In many applications information is only relevant for a fixed range of scales. We present an algorithm to construct a hierarchical net-tree up to scale t which we denote as the *net-forest*. We present a method based on *Locality Sensitive Hashing* to compute all near neighbors of points within a certain distance. Our construction of the net-forest is probabilistic, and we guarantee that with high probability, the net-forest is supplemented with the correct neighboring information. We apply our net forest construction to various applications including approximate Rips and Čech complexes and pair decompositions.

1 Introduction

Motivation Often, one wants to perform tasks on data which lives in high dimensional spaces. Typically, algorithms for manipulating such high dimensional data take exponential time with respect to the ambient dimension. This is frequently quoted as the "curse of dimensionality". In many cases, however, practical input instances lie on low dimensional manifolds and a natural question arises as to how do we exploit this structural property to invent computationally feasible algorithms.

A well-established approach is to define a special notion of dimension on a point set to capture its intrinsic dimension. The *doubling dimension* of a point set P is the smallest integer Δ such that every ball centered at any $p \in P$ of radius R is covered by at most 2^{Δ} non-empty balls of radius $R/2$ for any R. For instance, if P is a sample of an affine subspace of dimension d, it holds that $\Delta = \Theta(d)$, and often, $\Delta \ll d$ holds for more general samples of d-manifolds. A common goal is therefore to replace the exponential dependency on d by Δ in the complexity of geometric algorithms.

The concept of *(hierarchical) net-trees* can be visualized as a generalization of quadtrees and allows for the translation of quadtree-based algorithms (which are exponential in d) to instances with small Δ. Technically, a net-tree provides a hierarchy of *nets* which summarize the point set in terms of a clustering scheme at different scales. For n points with doubling dimension Δ, a net-tree can be constructed in expected $2^{O(\Delta)}O(n \log n)$ time, matching the time for constructing a quadtree except for replacing d with Δ [10]. As an application of particular importance, net-trees permit the efficient construction of *well-separated pair decomposition* (WSPD) which have various applications in geometric approximation, such as constructing spanners, finding approximate nearest-neighbors, approximating the diameter and the closest-pair distance.

In some applications, it is natural to investigate only an interval of scales. Applications which use net-trees can easily be adapted to ignore low scales by pruning off the lower part of the tree up to the appropriate scale. We instead concentrate on the more challenging question of upper bounding the range of scales. In such cases, the doubling dimension does not capture the intrinsic complexity of the problem at hand, since it may be defined by a ball that is beyond the range of considered scales. Moreover, the net-tree construction of [10] proceeds in a top-down fashion, considering the high scales of the point set first. Therefore, it suffers from potentially bad large-scale properties of the point set, even when these properties are irrelevant for the given application.

Contributions In this paper, we introduce the concept of t-restricted doubling dimension Δ_t, which is the smallest integer such that any ball centered at any $p \in P$ of radius $R \leq t$ is covered by at most 2^{Δ_t} non-empty balls of radius $R/2$. In this paper, we restrict our attention to the case of point sets in Euclidean space. We present an algorithm to construct a *net-forest*, which contains the relevant data of a net-tree up to scale t. The runtime of the construction depends on Δ_{Ct}, where C is a constant independent of n and d and is defined later on. We hence eliminate the dependence on the doubling dimension Δ. The major geometric primitive of our algorithm is to find all points which are at a distance at most $\Theta(t)$ from a given $p \in P$. We propose an approach based on *Locality Sensitive Hashing* (LSH) from [6]. The LSH based construction of the net-forest yields an expected runtime of $O\left(dn^{1+\rho} \log n(\log n + (14/\rho)^{\Delta_{7t/\rho}})\right)$ where $\rho \in (0,1)$ is a parameter which can be chosen to be as small as desired. Comparing this bound with the full net-tree con-

*Max Planck Institute for Informatics Saarbrücken, Germany, aruni.choudhary@mpi-inf.mpg.de

†Max Planck Institute for Informatics Saarbrücken, Germany, mkerber@mpi-inf.mpg.de

struction, our approach makes sense if $n^\rho \log n \ll 2^{O(\Delta)}$ and $\Delta_{O(t)} \ll \Delta$.

As a consequence of our result, we can construct the part of the WSPD where all pairs are in distance at most $\Theta(t)$, adapting the construction scheme of [10, Sec.5]. That means that any application of WSPD that restricts its attention to low scales can profit from our approach. Our approach also extends to the related concept of *semi-separated pair decomposition* [1].

As a further application, we show how to approximate Rips and Čech complexes using net-forests. These simplicial complexes are standard tools for capturing topological properties of a point cloud. Such a complex depends on a scale parameter; in particular, in the context of persistent homology [7], filtrations are considered, which encode complexes at various scales. In [13], an approximate Rips filtration of size at most $n(\frac{2}{\varepsilon})^{O(k\cdot\Delta)}$ has been constructed using net-trees. "Approximate" means that the exact and approximate filtrations are interleaved in the sense of [4] and therefore yield similar *persistence diagrams*, which summarize the topological properties of the point set. On the other hand, it is common to limit the construction of filtrations to an upper threshold value t. In such a scenario, our results yield a filtration of size $n(\frac{2}{\varepsilon})^{O(k\cdot\Delta_{O(t)})}$, thus replacing the exponential dependence on the doubling dimension by the $O(t)$-restricted doubling dimension. We also show how to approximate Čech complexes up to scale t, with the same size bound as for Rips complexes, building upon the framework of Choudhary et al. [5].

Organization of the paper Section 2 gives a brief overview of doubling spaces and net-trees. We introduce the concept of the restricted version of the doubling dimension in Section 3. In Section 4 we present an algorithm to construct the net-forest up to a desired scale. Our algorithm uses the concept of LSH which we detail in Section 5. In Section 6 we present applications of the net-forest. We summarize our results and conclude in Section 7.

2 Background

We fix P to be a finite point set consisting of n points throughout. As mentioned before, we restrict our attention to the Euclidean case $P \subset \mathbb{R}^d$, although some of the presented concepts could be extended to arbitrary metric spaces with some additional effort. In particular, the distance between any two points can be computed in $O(d)$ time in the Euclidean case.

Doubling dimension A *discrete ball* centered at a point $p \in P$ with radius r is the set of points $Q \subseteq P$ which satisfy $\|p - q\| \leq r$ for all $q \in Q$. The *doubling constant* [2, 14] is the smallest integer λ such that for all $p \in P$ and all $r > 0$, the discrete ball centered at p of radius r is covered by at most λ discrete balls of radius $r/2$. The *doubling dimension* Δ of P is $\lceil \log_2 \lambda \rceil$. For example, a point set that is sampled from a k-dimensional subspace has a doubling dimension of $\Theta(k)$, independent of the ambient dimension d. In contrast, the d boundary points of the standard $(d-1)$-simplex form a doubling space of dimension $\lceil \log_2 d \rceil$. Even worse, we can construct a subset of doubling dimension $\Theta(d)$ by placing $2^{\Theta(d)}$ points inside the unit ball in $\mathbb{R}^d$ such that any two points have a distance of at least $3/2$ (the existence of such a point set follows by a simple volume argument). It is NP-hard to calculate the doubling dimension of a metric [9], but it can be approximated within a constant factor [10, Sec.9].

Nets and Net-trees A subset $Q \subseteq P$ is an (α, β)-*net*, denoted by $\mathcal{N}_{\alpha,\beta}$, if all points in P are in distance at most α from some point in Q and the distance between any two points in Q is at least β. Usually, α and β are coupled, that is, $\beta = \Theta(\alpha)$, in which case we talk about a *net at scale* α.

We can represent a nested sequence of nets for increasing scales α using a rooted tree structure, called the *net-tree* [10]. It has n leaves, each representing a point of P, and each internal node has at least two children. Every tree-node v represents the subsets of points given by the sub-tree rooted at v; we denote this set by P_v. Every v has a representative, $\text{rep}_v \in P_v$ that equals the representative of one of its children if v is not a leaf. Moreover, v is associated with an integer $\ell(v)$ called the *level* of v which satisfies $\ell(v) < \ell(\text{parent}(v))$, where $\text{parent}(v)$ is the parent of v in the tree. Finally, each node satisfies the following properties

- *Covering* : $P_v \subseteq \mathbb{B}(\text{rep}_v, \frac{2\tau}{\tau-1} \cdot \tau^{\ell(v)})$
- *Packing* : $P_v \supseteq P \bigcap \mathbb{B}(\text{rep}_v, \frac{\tau-5}{2\tau(\tau-1)} \cdot \tau^{\ell(\text{parent}(v))})$

where $\mathbb{B}(p, r)$ denotes the ball centered at p with radius r and $\tau = 11$.

The covering and packing properties ensure that each node v has at most $\lambda^{O(1)}$ children where λ is the doubling constant for P. Moreover, for any α, a net at scale α can be accessed from the net-tree immediately; see [10, Prop.2.2] for details. A net-tree can be constructed deterministically in time $2^{O(\Delta)}O(dn \log(n \cdot \Phi))$ where Φ represents the *spread* of P, using the greedy clustering scheme of Gonzalez [8] as a precursor to the tree construction. The dependence on the spread can be eliminated by constructing the tree in $2^{O(\Delta)}dn \log n$ time in expectation (the additional factor of d compared to [10] accounts for the fact that we work with the Euclidean metric, and therefore take into account the cost of computing distances in our computational model). The net-tree construction is oblivious to knowing the value of

Δ. One can extract a net at scale ℓ [10, Pro.2.2] by collecting the set of nodes from T satisfying the condition $\mathcal{N}(\ell) = \{\text{rep}_v | \ell(v) < \ell \leq \ell(\text{parent}(v)\}$. The net-tree can be *augmented* to maintain, for each node u, a list of close-by nodes with similar diameter. Specifically, for each node u the data structure maintains the set

$$\text{Rel}(u) := \{v \in T \mid \ell(v) \leq \ell(u) < \ell(\text{parent}(v)) \text{ and } \|\text{rep}_u - \text{rep}_v\| \leq 14\tau^{\ell(u)}\}.$$

Rel(.) is computed during the construction without additional cost.

3 t-restricted doubling dimension

Definition 1 *The* t-restricted doubling constant *of* P *is the smallest positive integer* λ_t *such that all the points in any discrete ball centered at* $p \in P$ *of radius* r *with* $r \leq t$ *are covered by at most* λ_t *non empty balls of radius* $r/2$. *The corresponding* t-restricted doubling dimension Δ_t *is* $\lceil \log \lambda_t \rceil$.

By definition, $\Delta_t \leq \Delta$ for any P. More precisely, Δ_t is zero for t smaller than the closest-pair distance of P, and equals Δ when t is the diameter of P. While Δ for samples from an affine subspace of dimension k is bounded by $\Theta(k)$, this is not generally true for samples of k-manifolds where Δ increases due to curvature. To sketch an extreme example, consider an *almost space-filling* curve γ in $\mathbb{R}^d$ which has distance at most ε to any point of the unit ball, where ε is chosen small enough. We let P be a sufficiently dense sample of γ. While $\Delta_t = 1$ for small values of t, we claim that $\Delta_t = \Theta(d)$ for $t = 1$; indeed, any sparser covering of the unit ball with balls of radius $1/2$ would leave some portion of the ball uncovered, and by construction, γ goes through that uncovered region, so that some point in P is missed.

The "badness" of the previous example stems from the difference between Euclidean and geodesic distance of points lying on a lower-dimensional manifold. A common technique for approximating the geodesic distance is through the *shortest-path metric*: Let $G = (P, E)$ denote the graph whose edges are defined by the pairs of points of Euclidean distance at most t; we call such a graph a t-intersection graph. The distance of two points p and q is then defined as the length of the shortest path from p to q (we assume for simplicity that G is connected). The concept of doubling dimensions extends to any metric space and we let Δ' denote the doubling dimension of P equipped with the shortest path metric. While Δ_t and Δ' appear to be related, Δ' can be much larger than Δ_t: an example is shown in Figure 1. Moreover, using the shortest-path metric raises the question of how to compute shortest path distances efficiently, if the cost of metric queries is taken into account.

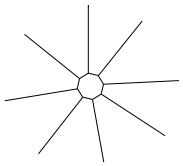


Figure 1: Consider a regular k-gon in the plane with all vertices on the unit circle. To each vertex, attach an "arm" of length 4 in the direction outwards the origin. We call the endpoint of such an arm a *tip*. Let M denote the obtained shape (as depicted). Let P denote a sufficiently dense point sample of M. Let G be the t-intersection graph on P; we choose t appropriately so that the shortest path metric on G approximates the distances on M very closely. Fixing an arbitrary vertex of the k-gon, the furthest tip has a distance of at most $4 + \pi < 8$ on M, so there is a ball of radius less than 8 containing all of P. However, any pair of tips has a distance of more than 8, so no ball of radius less than 4 can contain two tips. It follows that any covering of P with balls of radius less than 4 requires at least one ball per tip. Therefore, the doubling dimension Δ' of this metric is at least $\lceil \log_2 k \rceil$. On the other hand, for the Euclidean metric on a plane, we can easily see that the doubling dimension $\Delta = O(1)$, and therefore, Δ_t is a constant as well.

4 Net-forests

We next define an appropriate data structure for point sets of small t-restricted doubling dimension, where t is a parameter of the construction. Informally, a *net-forest* is the subset of a net-tree obtained by truncating all nodes above scale t. More precisely, it is defined as a collection of net-trees with roots $v_1, \ldots, v_k$ such that the representatives $\text{rep}_{v_1}, \ldots, \text{rep}_{v_k}$ form a (t, t)-net and the point sets $P_{v_1}, \ldots, P_{v_k}$ are disjoint and their union covers P. We define $\text{Rel}(u)$ for a node in the forest the same way as for net-trees: it is the set of net-forest nodes that are close to u and have similar diameter. As for net-trees, we call a net-forest *augmented* if each node u is equipped with $\text{Rel}(u)$.

Construction Our algorithm for constructing a net-forest is a simple adaptation of the net-tree algorithm: we construct a (t, t)-net of P by clustering the point set and assign each point in P to its closest net-point. Each root in the net-forest represents one of the clusters. We also compute $\text{Rel}(u)$ for each root by finding the close-by clusters to u. Having this information, we can simply run the net-tree algorithm from [10] individually on each cluster to construct the net-forest. For augmenting it, we use the top-down traversal strategy as described in [10, Sec.3.4], inferring the neighbors of a node from the neighbors of its parent –since we have set up Rel(·)

for the roots of the forest, this strategy is guaranteed to detect neighboring nodes even if they belong to different trees of the forest.

Both the initial net construction and the Rel(·)-construction require the following primitive for a point set Q, which we call a *near-neighbor query*: *Given a point $q \in Q$ and a radius r, return a list of points in Q containing exactly the points at distance r or smaller from q).* In the remainder of the section, we give more details on how to compute the net and the associated clusters, and how to find the neighbors for each such cluster, assuming that we have a primitive which can perform near-neighbor queries. In Section 5, we show the implementation of such a primitive.

Net construction We construct the net using a greedy scheme: For any input point, store a pointer $\mathrm{N}(p)$ pointing to the net point assigned to point p. Initially, $\mathrm{N}(p) \leftarrow \varnothing$ for all p. As long as there is a point p with $\mathrm{N}(p) = \varnothing$, we set $\mathrm{N}(p) \leftarrow p$ and query the near-neighbor primitive to get a list of points with distance at most t from p. For any point q in the list we update $\mathrm{N}(q) \leftarrow p$ if either $\mathrm{N}(q) = \varnothing$ or $\|p - q\| < \|N(q) - q\|$. Then we pick the next point p with $\mathrm{N}(p) = \varnothing$.

At the end, the set of points p with $\mathrm{N}(p) = p$ represents the net at scale t and the points q satisfying $\mathrm{N}(q) = p$ constitute p's cluster. The net thus constructed is a (t, t)-net. Moreover, we set $\ell(v) = \lfloor \log_\tau \left(\frac{\tau-1}{2\tau} t\right) \rfloor$ for each root node v.

Computing the Rel(.) set for the roots After computing the net-points and their respective clusters, we need to augment the net-points with neighboring information. Recall that $\mathrm{Rel}(u)$ contains nodes in distance at most $14\tau^{\ell(u)}$ from rep_u. Since we have a (t, t)-net, the level of any root node u satisfies $14\tau^{\ell(u)} \leq 7t$. Hence we need to find neighbors of net-points within $7t$, the minimum distance between any two net-points being more than t. By the doubling property, any root net-node can have at most $\lambda_{7t}^{\log_2 \frac{7t}{t/2}}$ such neighbors which simplifies to $14^{\Delta_{7t}}$. We use the near-neighbor primitive to compute such neighbors.

5 Near-neighbors primitive

We describe the primitive used in the previous section which performs near-neighbor queries. Our approach follows the notion of *locality-sensitive hashing* (LSH) introduced by [11] for the Hamming metric and extended to Euclidean spaces in [6]. LSH is a popular approach to find approximate near-neighbors in high dimensions because of its near linear complexity in n and d.

Locality Sensitive Hashing LSH applies several hash functions on a point set such that close points are more likely to map to the same hash-buckets than points which are sufficiently far apart. A typical application of LSH is the (r, c)-*nearest neighbor* problem: If there exists a point within distance r of the query point q, report some point within distance cr of q, $c > 1$.

However, for our construction we wish to solve the following problem: report *all* points within distance r of the query point. We need the LSH oracle for two steps in our construction: constructing the net at scale t and computing the Rel(·) for the root-nodes. We show that both these steps requires a runtime sub-quadratic in n by a slight modification of the method presented in [6]. We repeat some of their definitions for clarity:

Definition 2 *A family of hash functions $\mathcal{H} = \{h : S \to U\}$ is called* (r_1, r_2, p_1, p_2)-sensitive *if for all $a, b \in S$, the following holds for $p_1 \geq p_2$ and $r_1 \leq r_2$:*

- *if* $\|a - b\| \leq r_1, P_{r_1} = P[h(a) = h(b)] \geq p_1$
- *if* $\|a - b\| \geq r_2, P_{r_2} = P[h(a) = h(b)] \leq p_2$

We amplify the gap between P_{r_1} and P_{r_2} by concatenating k such hash functions, creating the family of hash functions $\mathcal{G} = \{g : S \to U^k\}$ such that $g(x) = (h_1(x), h_2(x), ..., h_k(x))$. For $g(x)$, we have the modified properties:

- if $\|a - b\| \leq r_1, P[g(a) = g(b)] \geq p_1^k$
- if $\|a - b\| \geq r_2, P[g(a) = g(b)] \leq p_2^k$

We describe our near-neighbor primitive next: The input is a point set Q with n points and a distance $r > 0$. As a pre-processing step, we choose l hash functions $g_1, \cdots, g_l$ uniformly at random from $\mathcal{G}$ [6, Sec.3]. and hash each $p \in Q$ to the buckets $g_i(p) \forall i \in [1, l]$. Given a query point $q \in Q$ (we *only* consider near neighbor queries for points within Q), we check for any point p in bucket $g_i(q)$ whether the distance to q is at most r. We output such points as the near-neighbors of q.

We need to specify the parameters of LSH in the above description. Most importantly, we have to ensure that, with high probability, the output contains all points in distance r from q. Moreover, we want the buckets to be of small size so that the primitive does not have to filter out too many false positives.

The performance of the LSH scheme depends upon a parameter $\rho \in (0, 1)$ which appears as an exponent of n in the runtime. We choose the parameters p_1, p_2, r_1 and r_2 of the hashing scheme such that $\rho = \frac{\log p_1}{\log p_2} \approx \frac{r_1}{r_2}$ [6, Sec.4]. In the following parts of the section, we let $\rho = \frac{r_1}{r_2}$.

Lemma 1 *Let $r_1 := r$ and $r_2 := r/\rho$, $k := \lceil -\log_{p_2} n \rceil$ and $l := \lceil 2n^\rho \ln \frac{n}{\sqrt{\delta}} \rceil$ with an arbitrarily small constant δ. The near-neighbor primitive has the following properties:*

(i) With probability at least $1 - \delta$, all points in distance at most r are reported for all query points.

(ii) For any query point q, the aggregate expected size of all buckets $g_1(q), ..., g_l(q)$ is at most $l(\widetilde{C}+1)$, where $\widetilde{C}$ is the number of points in Q with distance at most r_2 to q.

(iii) The pre-processing runtime is $O(dnkl)$ and the expected query runtime for a point is $O(dl(k+\widetilde{C}))$, where $\widetilde{C}$ is defined as in (ii).

We defer the proof to Appendix A.

Net-forest construction using LSH We analyze the complexity of our net-forest construction from Section 4 with our near-neighbor primitive in Lemma 5 under Appendix B. The primitive is used in the construction of the (t,t)-net, where we find the near-neighbors in distance at most t for a subset of points that form the net in the end. That means, we initialize the primitive with $r \leftarrow t$ and $Q \leftarrow P$.

The second appearance of the near-neighbor primitive is in the construction of the Rel(.) sets for the roots of the net forest. Recall that the roots are represented by the net-points constructed before; let M denote the set of net-points and $|M|m$.We simply have to find all pairs of points of distance at most $7t$ among the net-points; and to do so we call the near neighbor primitive with $r \leftarrow 7t$ and $Q \leftarrow M$ for all $q \in M$ (see Lemma 6).

Theorem 2 *The expected time for constructing the net-forest using LSH is*

$$O\left(dn^{1+\rho}\log n\left(\log n + \left(\frac{14}{\rho}\right)^{\Delta_{7t/\rho}}\right)\right).$$

We defer the details of the proof to Appendix B.

We see how the choice of ρ affects the complexity bound: For ρ very close to zero, we get a almost linear complexity in n, to the price that we have to consider larger balls in our algorithm and thus increase the restricted doubling dimension.

6 Applications

Well-Separated Pair Decomposition A pair of net-tree nodes (u,v) is *ε-well-separated* if it satisfies $\max\{\mathrm{diam}(P_u), \mathrm{diam}(P_v)\} \leq \varepsilon\|\mathrm{rep}_u - \mathrm{rep}_v\|$, where $\mathrm{diam}(P_u)$ denotes the *diameter* of the points stored in u. Informally speaking, all pairs of points (p,q) with $p \in P_u$, $q \in P_v$ have a similar distance to each other if (u,v) is well-separated. An *ε-well-separated pair decomposition* (ε-WSPD) [3] is a collection of ε-well-separated pairs such that for any pair $(p,q) \in P \times P$, there exists a well-separated pair (u,v) such that $p \in P_u$ and $q \in P_v$; we say that such a pair (p,q) is *covered* by (u,v).

An ε-WSPD of size $n\varepsilon^{-O(\Delta)}$ can be computed in time $d\left(2^{O(\Delta)}n\log n + n(1/\varepsilon)^{O(\Delta)}\right)$ [10]. A WSPD considers pairs over all scales of distance, because it has to cover any pair of points. Our structure only requires that all pairs of points in distance at most t are covered. We call the resulting structure a *t-restricted ε-WSPD*.

We construct the t-restricted ε-WSPD as follows: We start by constructing the corresponding augmented net forest; let $u_1, \ldots, u_m$ be its roots. Since we know the $\mathrm{Rel}(\cdot)$ set for any root, we can identify pairs (u_i, u_j) such that u_i is in $\mathrm{Rel}(u_j)$ and vice versa (this also includes pairs where $u_i = u_j$). For any such pair, we call `genWSPD`(u_i, u_j) from [10, Sec.5], which simply traverses the sub-trees until it finds well-separated pairs. We output the union of all pairs generated in this way.

Theorem 3 *For $0 < \varepsilon < 1$ and $t > 0$, our algorithm computes a t-restricted ε-WSPD of size $n\varepsilon^{-O(\Delta_{7t})}$ in expected time $NF + dn\varepsilon^{-O(\Delta_{7t})}$, where NF is the complexity for computing the net-forest from Theorem 2.*

We defer the proof to Appendix C.

Semi-Separated Pair Decomposition(SSPDs) SSPDs [1] are a related concept to the WSSDs with some advantages. We briefly describe them and present our *t-restricted* version of SSPDs in Appendix D.

Approximating Simplicial Complexes For a point set P, let $\mathrm{rad}(Q)$ denote the radius of the minimum enclosing ball of $Q \subseteq P$. The Čech complex on P at scale r is defined as: $\check{C}ech(r) := \{Q \subseteq P \mid \mathrm{rad}(Q) \leq r\}$. The Rips complex is defined as: $\mathrm{Rips}(r) := \{Q \subseteq P \mid \mathrm{diam}(Q) \leq 2r\}$. Rips and Čech complexes are standard constructions for topological analysis of point clouds; more precisely, one constructs a sequence of Rips or Čech complexes for growing r (called a *filtration*) and tracks the evolution of topological features in the process. This gives rise to the *persistence diagram* [7], a multi-scale summary of the topological properties of the point cloud. However, a major computational drawback of this approach is that both Rips and Čech complexes become prohibitively large when P has high ambient dimension. A common remedy is to bound the maximal dimension k of the simplices constructed in the filtration (bounding the filtration size to $O(n^{k+1})$) and/or bound the maximal scale r which is considered.

Several recent works have come up with *approximate Rips (Čech) filtrations* of significantly smaller size. In this case, "approximate" means that the persistence diagram of the exact and approximate filtrations are ε-close in *interleaving distance* [4]. Sheehy [13] showed how to compute an $(\frac{1}{1-2\varepsilon})$-approximate Rips filtration of size $n(\frac{2}{\varepsilon})^{O(k\cdot\Delta)}$ in expected time $d2^{O(\Delta)}n\log n + n(\frac{2}{\varepsilon})^{O(\Delta)}$, where k denotes the maximal dimension of the constructed approximation. His algorithm works by creating simplicial complexes on a hierarchically sparsified point set, obtained through the net-tree. In [5], Choudhary et

al. gave an alternative algorithm for $(1+\varepsilon)$-approximate Čech filtrations with the same guarantees and running time, building upon the framework from [12].

For the case that the highest scale in the construction is bounded for t, our results imply that Δ can be replaced with Δ_{7t} in the size bound above.

Theorem 4 *An approximate filtration of size at most* $n(\frac{2}{\varepsilon})^{O(k\cdot\Delta_{7t})}$ *can be computed for both* Rips *and* Čech *filtrations for the range* $[0,t]$.

Proof. For Rips-complexes, this follows directly by applying the algorithm of [13, Sec.10] on a net forest of scale t. Since $w = \{u \cup \mathrm{Rel}(u)\}$ captures all edges of length at most $5t > t$ between a point $p \in P_u$ and $q \in P_w$, the algorithm is able to compute the $E(p)$ sets [13, Sec.10] successfully, thereby yielding a filtration of size at most $n(\frac{2}{\varepsilon})^{O(k\cdot\Delta_{7t})}$.

To prove the same for Čech complexes, we need to apply our net-forests to a generalization of WSPDs (called well-separated simplicial decomposition) first; we postpone the details to Appendix E. □

Approximating the t-doubling dimension One can approximate Δ_t for any point set P up to a constant factor by constructing a net-forest T of scale t over P. Let x denote the maximum out-degree of any node in T. Then $\log x$ is a constant-factor approximation of Δ_t. This follows directly from the arguments of [10, Sec.9].

7 Conclusion and future work

In this paper we presented an algorithm to construct a hierarchical net-forest up to a certain scale and applied it to the construction of WSPDs, SSPDs, and approximate Rips and Čech complexes. Finding more applications tailored for the t-restricted doubling dimension is an appealing research direction we would like to look into.

Acknowledgement This research is supported by the Max Planck Center for Visual Computing and Communication.

References

[1] Mohammad A. Abam and Sariel Har-Peled. New constructions of SSPDs and their applications. *Comput. Geom. Theory Appl.*, 45(5-6):200–214, July 2012.

[2] Patrice Assouad. Plongements Lipschitziens dans $\mathbb{R}^n$. *Bulletin de la Societe Mathematique de France*, 111:429–448, 1983.

[3] Paul B. Callahan and S. Rao Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42(1):67–90, January 1995.

[4] Frédéric Chazal, David Cohen-Steiner, Marc Glisse, Leonidas J. Guibas, and Steve Y. Oudot. Proximity of persistence modules and their diagrams. In *Proceedings of the Twenty-fifth Annual Symposium on Computational Geometry*, pages 237–246, 2009.

[5] Aruni Choudhary, Michael Kerber, and R. Sharathkumar. Approximate Čech complexes in low and high dimensions. `http://people.mpi-inf.mpg.de/~achoudha/Files/Papers/AppCech.pdf`.

[6] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, pages 253–262, 2004.

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

[8] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

[9] Lee-Ad Gottlieb and Robert Krauthgamer. Proximity algorithms for nearly doubling spaces. *SIAM J. Discrete Math.*, 27(4):1759–1769, 2013.

[10] Sariel Har-Peled and Manor Mendel. Fast construction of nets in low dimensional metrics, and their applications. In *SIAM J. Comput*, pages 150–158, 2005.

[11] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, 1998.

[12] Michael Kerber and R. Sharathkumar. Approximate Čech complexes in low and high dimensions. In *International Symposium on Algorithms and Computation*, pages 666–676, 2013.

[13] Donald R. Sheehy. Linear-size approximations to the Vietoris-rips filtration. *Discrete & Computational Geometry*, 49(4):778–796, 2013.

[14] Kunal Talwar. Bypassing the embedding: Algorithms for low dimensional metrics. In *Proceedings of the Thirty-sixth Annual ACM Symposium on Theory of Computing*, STOC '04, pages 281–290, 2004.

A Proof of Lemma 1

Proof. First we bound the expected aggregate size of the buckets. A bucket contains "close" points which are in distance at most r_2 from q and "far" points which are further away. However, since the probability of a far point falling in the same bucket as q is at most p_2^k, the expected size of a single bucket is at most $\widetilde{C}+np_2^k \leq \widetilde{C}+1$ by our choice of k. Since there are l buckets, (ii) is satisfied.

For (i), fix two points $q_1, q_2 \in Q$ such that $\|q_1 - q_2\| \leq r_1$. We have to ensure that $g_j(q_1) = g_j(q_2)$ for some $j \in \{1, \ldots, l\}$; this implies that q_1 will be reported for query point q_2, and vice versa for at least one of the l buckets. The probability for $g_j(q_1) = g_j(q_2)$ for a fixed j is at least p_1^k, which is $p_1^{-\log_{p_2} n} = n^{-\rho}$. Hence the probability that $g_j(q_1) \neq g_j(q_2)$ holds for all $j \in \{1, \ldots, l\}$ is at most $(1 - n^{-\rho})^l$ because we choose the hash functions uniformly at random. There are at most n^2 point pairs within distance at most r_1. By the union bound, the probability that at least one such pair maps into different buckets is at most $n^2(1 - n^{-\rho})^l$. Now we can bound

$$n^2(1 - n^{-\rho})^l = n^2(1 - n^{-\rho})^{2n^\rho \ln \frac{n}{\sqrt{\delta}}}$$
$$= n^2(1 - \frac{1}{n^\rho})^{n^\rho \ln \frac{n^2}{\delta}} \leq n^2 e^{-\ln \frac{n^2}{\delta}} = \delta,$$

where we used the fact that $(1 - 1/x)^x \leq 1/e$ for all $x \geq 1$. It follows that the probability that all pairs of points in distance at most r_1 fall in at least one common bucket is at least $1 - \delta$. This implies (i).

It remains to show (iii): in the pre-processing step, we have to compute $k \cdot l$ hash functions for n points. Computing the hash value for a point p, $h_i(p)$ takes $O(d)$ time [6, Sec.3.2]. For a query, we have to identify the buckets to consider in $O(dkl)$ time and then iterate through the (expected) $l(\widetilde{C} + 1)$ candidates (using (ii)), spending $O(d)$ for each. □

B Proof of Theorem 2

To prove Theorem 2, we first need to prove two supporting Lemmas, which we do next.

Lemma 5 *The expected time to construct the (t, t)-net using LSH is*

$$O\Big(dn^{1+\rho} \log n \Big(\log n + \Big(\frac{2}{\rho}\Big)^{\Delta_{t/\rho}}\Big)\Big).$$

Proof. We consider the time spent on all near-neighbor queries: Let the resulting net consist of $m \leq n$ points. This implies that the algorithm proceeds in m rounds and queries the near-neighbors of m points. Let $\widetilde{C}_i$ denote the number of points in distance t/ρ from the i-th query point. By Lemma 1, the total complexity for all near-neighbor queries is:

$$O\left(ndkl + \sum_{i=1}^{m} dl(k + \widetilde{C}_i)\right) = O(ndkl + dl\sum_{i=1}^{m} \widetilde{C}_i)$$

We only need to bound the sum of the $\widetilde{C}_i$. For that, we fix some $q \in P$ and count in how many sets $\widetilde{C}_i$ may it appear. Let p_i denote the net-point chosen in the i-th iteration. We call such a net-point *close* to q if the distance to q is at most t/ρ. By definition, the net-points close to q lie in a ball of radius t/ρ centered at q. Since any pair of net-points has a distance of more than t, any ball of radius $t/2$ can contain at most one close net point. Following the definition of the t-restricted doubling dimension, the number of such net-points can be at most $\lambda_{t/\rho}^{\log_2 \frac{t/\rho}{t/2}}$ which simplifies to $\left(\frac{2}{\rho}\right)^{\Delta_{t/\rho}}$. It follows that

$$\sum_{i=1}^{m} \widetilde{C}_i \leq n\left(\frac{2}{\rho}\right)^{\Delta_{t/\rho}}.$$

Hence, we get the claimed running time, observing that $k = O(\log n)$ and $l = O(n^\rho \log n)$ by Lemma 1. All additional operations in the net construction besides the calls of the primitive are dominated by that complexity. □

Lemma 6 *Computing the* Rel(.) *sets using LSH takes expected time*

$$O\left(dn^{1+\rho} \log n \left(\log n + \left(\frac{14}{\rho}\right)^{\Delta_{7t/\rho}}\right)\right).$$

Proof. The proof is analogous to the proof of Lemma 5: Let $\widetilde{C}_i$ (for $i = 1, \ldots, m$) denote the number of net-points in distance at most $\frac{7t}{\rho}$ to the i-th net point. The same packing argument as in the previous Lemma shows that any $\widetilde{C}_i$ can be at most $(\frac{14}{\rho})^{\Delta_{7t/\rho}}$, so that their sum is bounded by $m(\frac{14}{\rho})^{\Delta_{7t/\rho}}$. Analogous to the proof of Lemma 5, we can thus bound the runtime to be as required, noting that $m \leq n$. □

B.1 Proof of Lemma 2

Proof. Using Lemma 5 and Lemma 6, constructing the net and its Rel(·) sets are within the complexity bound. Constructing a single net-tree for a node containing n_i points takes time at most $2^{14\Delta_{2t}} n_i \log n_i$ (the factor of 14 in the exponent can be seen by a careful analysis of [10, Sec.3.4]). Constructing individual net-trees for the clusters takes time: $\sum_{i=1}^{m} 2^{14\Delta_{2t}} dn_i \log n_i$ Since $\sum_{i=1}^{m} n_i = n$, the above runtime simplifies to be at most $2^{14\Delta_{2t}} dn \log n$. Augmenting the net-forest takes time $dn2^{14\Delta_{7t}}$ [10, Sec.3.4]. The runtimes for the latter steps are dominated by the Rel(·) and net construction for sufficiently large values of n. □

C Proof of Theorem 3

Proof. For correctness, any pair of nodes generated is ε-well-separated by definition. For the relaxed covering property, consider a pair (p, q) of points in distance at most t. There are roots u_1, u_2 in the net-forest with $p \in P_{u_1}$ and $q \in P_{u_2}$. Since the diameter of u_1 and u_2 is at most $2t$, the distance of rep_{u_1} and rep_{u_2} is at most $5t < 7t$. Therefore, $u_2 \in \text{Rel}(u_1)$ (and vice versa), and there will be a pair generated that covers (p, q).

For the size bound, we can use the same charging argument as in [10, Sec.5]. We can additionally ensure by our construction that in all doubling arguments, the radius of the balls in question is at most $7t$ and therefore replace the doubling dimension by Δ_{7t} in the bound. The running time follows because the number of recursive calls of `genWSPD` is proportional to the output size, and we spend $O(d)$ time per recursion step. □

D Semi-Separated Pair Decomposition

A pair of net-tree nodes (u, v) is called *ε-semi-separated* if $\min(diam_u, diam_v) \leq \varepsilon \|\text{rep}_u - \text{rep}_v\|$. An *$\varepsilon$-semi-separated pair decomposition* (ε-SSPD) is a collection of ε-semi-separated pairs such that for any pair of points $(p, q) \in P \times P$ there exists a semi-separated pair (u, v) such that $p \in P_u$ and $q \in P_v$; we say that such a pair (p, q) is *covered* by (u, v). This is a weaker notion than the WSPD, because it only requires that the *smaller* diameter of the participating sets be small compared to the distance between them. The advantage of using a SSPD over a WSPD is reduced weight: The weight of a WSPD W is defined as $w = \sum_{(x,y)\in W}(|x| + |y|)$. The weight of any WSPD can be $\Omega(n^2)$ in the worst case [1]. In contrast, it is possible to construct a SSPD with near linear weight, where the weight in question is an analogous definition to the one used for WSPDs. An ε-SSPD of expected weight $O(\varepsilon^{-O(\Delta)} n \log n)$ can be calculated in $O(\varepsilon^{-O(\Delta)} n \log n)$ expected time [1].

We adapt the algorithm of [1, Sec.4] to compute a t-restricted SSPD, which requires that only pairs of points in distance at most t be covered. The construction is as follows: first we construct the net-forest at scale t. Let $u_1, \cdots, u_m$ denote the root nodes of the net-forest. We invoke the algorithm of [1, Sec.4] on each of the sets $\{u_i \cup \text{Rel}(u_i)\}$. We output the union of the pairs generated by each invocation. The SSPD generated this way also ensures coverage of points which are at most $5t$ apart and hence, covers all pairs of points at most t apart. Moreover, the pairs involved in the SSPD have the additional property that their diameter is at most $16t$.

Theorem 7 *For $0 < \varepsilon < 1$, our algorithm computes a t-restricted ε-SSPD of P of expected weight $(\frac{2}{\varepsilon})^{O(\Delta_{16t})} O(n \log n)$ in $(\frac{2}{\varepsilon})^{O(\Delta_{16t})} O(n \log n)$ expected time, after computing the net forest at scale t.*

Proof. To prove correctness, consider points p and q such that $\|p - q\| \leq 5t$. Let u_i and u_j be the root nodes of the net-forest covering p and q respectively. By a simple application of the triangle inequality, $\|\text{rep}_{u_i} - \text{rep}_{u_j}\| \leq 7t$. Hence, $u_i \in \text{Rel}(u_j)$ and vice versa. This ensures that the algorithm creates a semi-separated pair which covers the pair (p, q). For the diameter, let us say that we work with the set $P_m = \{u \cup \text{Rel}(u)\}$ and wish to upper bound $\text{diam}(P_m)$. For any $u_i, u_j \in \text{Rel}(u)$ covering points a and b respectively, it follows from the triangle inequality that $\|a - b\| \leq 16t$. Hence the claim about the diameter is true.

Next we bound the weight of the SSPD. Consider the individual invocations of the algorithm of [1] after constructing the net-forest. Let N_i denote the number of points in $u_i \cup \text{Rel}(u_i)$. Then the total weight of the SSPD is $\sum_{i=1}^{m} O(\varepsilon^{-O(\Delta_{16t})} N_i \log N_i) \leq O(\varepsilon^{-O(\Delta_{16t})} \log n) \sum_{i=1}^{m} N_i$. To bound the sum $S = \sum_{i=1}^{m} N_i$, consider any point p covered by u_i. This point participates in at most $2^{O(\Delta_{7t})}$ invocations of the algorithm, since that is precisely the maximum possible size of the Rel(.) set. This implies that the contribution of point p to S is at most $2^{O(\Delta_{7t})}$. Hence the sum S is at most $2^{O(\Delta_{7t})} n$, and bound follows. A similar argument bounds the runtime as well. □

E Approximate Čech Complexes

Well Separated Simplicial Decomposition The concept of well-separated simplicial decomposition (WSSDs) of point sets, introduced by Kerber and Sharathkumar [12] and extended to doubling spaces by Choudhary et al [5], generalizes the concept of WSPD to larger tuples. A $(k+1)$-tuple $(v_0, v_1, \ldots, v_k)$ is called *ε-well separated* if each v_i is a node of the net-tree and for any ball $\mathbb{B}$ which contains at least one point of each v_i, it holds that

$$v_0 \cup v_1 \cup .... \cup v_k \subseteq (1 + \varepsilon)\mathbb{B}$$

where $(1+\varepsilon)\mathbb{B}$ denotes a ball with same center as $\mathbb{B}$ and radius multiplied by $(1 + \varepsilon)$. An (ε, k)-WSSD is a set of ε-well-separated tuples of length $(k + 1)$ such that any k-simplex is covered by some tuple. In [5], an (ε, k)-WSSD of size $n(2/\varepsilon)^{O(\Delta \cdot k)}$ is constructed in expected time $d\left(2^{O(\Delta)} n \log n + n(2/\varepsilon)^{O(\Delta \cdot k)}\right)$.

Similar as before, we define a *t-restricted (ε, k)-WSSD* to be a collection of ε-well-separated tuples such that each k-simplex that fits into a ball of radius t is covered by a tuple. The statement is equivalent to the condition that the radius of the smallest minimum enclosing ball containing points from each node of the tuple is at most t.

Construction of the t-restricted WSSD We describe the algorithm to construct the t-restricted (ε, k)-WSSD and prove its correctness and runtime. In this appendix, we will heavily rely on the notations, algorithms, and results presented in [5]. The algorithm proceeds iteratively; for $k = 1$, we construct a $(2t)$-restricted $\varepsilon/2$-WSPD using the algorithm from Section 6. To construct Γ_{k+1} from Γ_k, we iterate over the tuples $\gamma \in \Gamma_k$. We use the scheme of [5, Sec.3], computing an approximate meb of γ and then exploring ancestors of v_0 and their descendants at appropriate levels. The only complication arises when the algorithm requests for an ancestor higher than root of the tree of v_0. In such a case, our algorithm uses the root as the ancestor. In the following lemma, we will show that with this approach, we still cover all simplices with meb radius of at most t.

Lemma 8 *The algorithm computes a t-restricted (ε, k)-WSSD.*

Proof. We show by induction that with modified ancestor search, we still cover all simplices with meb radius at most t. For $k = 1$, the correctness of the algorithm follows from Theorem 3 in Section 6, Let Γ_{k-1} cover all $(k-1)$-simplices γ which satisfy $\mathrm{rad}(\gamma) \le t$. Consider any k-simplex $\sigma = (m_0, \ldots, m_k)$ with $\mathrm{rad}(\sigma) \le t$. From [5, Lem.9], there exists a point (say m_k) such that $m_k \in 2\mathrm{meb}(\sigma')$ where $\sigma' := \sigma \setminus \{m_k\}$ and $2\mathrm{meb}(\sigma')$ represents a ball with twice the radius and the same center as $\mathrm{meb}(\sigma')$. Since σ' is a $(k-1)$-simplex and $\mathrm{rad}(\sigma') \le \mathrm{rad}(\sigma) \le t$, it is covered by some k-tuple $\gamma = (v_0, \ldots, v_{k-1}) \in \Gamma_{k-1}$. To prove correctness, we show that when our algorithm reaches tuple γ, it produces a $(k+1)$-tuple (γ, x) such that $m_k \in P_x$ which implies that the simplex σ is covered by the $(k+1)$-tuple (γ, x).

When handling γ, the algorithm searches for an ancestor of v_0 at an appropriate scale. If this ancestor is found within the tree of v_0 in the net-forest, the arguments from [5, Lem.12] carry over to ensure that a suitable x is found. So let us assume that the algorithm chooses the root of the tree of the net-forest that v_0 lies in. Call that root node a_0. The algorithm considers all nodes in $\mathrm{Rel}(a_0)$ and creates new tuples with their descendants. Moreover, the net-forest contains a leaf representing the point m_k; let a' denote the root of its tree. It suffices to show that $a' \in \mathrm{Rel}(a_0)$. Since $\mathrm{rad}(\sigma) \le t$, the distance of m_0 and m_k is at most $2t$. Moreover, the distance of m_0 to rep_{a_0} is at most t, because the representatives of the roots form a (t,t)-net. The same holds for m_k and a'. Using triangle inequality, the distance of rep_{a_0} and $\mathrm{rep}_{a'}$ is at most $4t$. This implies that $a' \in \mathrm{Rel}(a_0)$. □

Lemma 9 *The size of the computed t-restricted (ε, k)-WSSD Γ_k is $n(\frac{2}{\varepsilon})^{O(\Delta_{7t} \cdot k)}$.*

Proof. The proof of [5, Lem.13] carries over directly – indeed, we can replace all occurrences of Δ by Δ_{7t}. This comes from the fact that a node u has at most $14^{\Delta_{7t}}$ nodes in $\mathrm{Rel}(u)$, and for any node in $\mathrm{Rel}(u)$ we reach descendants of a level of at most $O(\log(2/\varepsilon))$ smaller then u (see the proof of [5, Lem.13] for details). Since every node in the net-forest has at most $2^{O(\Delta_t)}$ children, we create at most

$$14^{\Delta_{7t}} \left(\frac{2}{\varepsilon}\right)^{O(\Delta_t)} = \left(\frac{2}{\varepsilon}\right)^{O(\Delta_{7t})}$$

tuples in Γ_k from a tuple in Γ_{k-1}. With that, the bound can be proved by induction. □

Lemma 10 *Computing a t-restricted (ε)-WSSD takes time $nd(2/\varepsilon)^{O(\Delta_{7t} \cdot k)}$ after computing the net forest at scale t.*

Proof. The proof is analogous to [5, Lem.14], plugging in the running time for t-restricted ε-WSPD from Theorem 11 and the size bound from Lemma 9. □

Computing the approximate Čech filtration We use the scheme of [5, Sec.4] to construct the $(1+\varepsilon)$-approximate filtration on the t-restricted WSSD. The original construction works without modification. Using the notation from [5, Sec.4]., for any WST $\sigma = (v_0, v_1, \ldots, v_k)$ with $\ell(v_i) \le h$, we add $\sigma' = (\mathrm{vcell}(v_0, h), \mathrm{vcell}(v_1, h), \ldots, \mathrm{vcell}(v_k, h))$ to $\mathcal{A}_\alpha$ if $\mathrm{rad}(\sigma') \le \theta_\Delta$. The only potential problem with the t-restricted case is that such a vcell() might be a node higher than a root of the net-forest. This cannot happen, however, since h is chosen such that

$$\frac{2\tau}{\tau - 1}\tau^h \le \frac{\varepsilon}{7}\alpha.$$

Since $\alpha \le t$ and $\varepsilon \le 1$, we have that

$$h < \lfloor \log_\tau \frac{\tau - 1}{2\tau} t \rfloor = \ell(u)$$

for any root u in the net-forest.

Theorem 11 *A* t-restricted (ε, k)-WSSD *of size $n(\frac{2}{\varepsilon})^{O(\Delta_{7t} \cdot k)}$ can be computed in time*

$$NF + nd\left(\frac{2}{\varepsilon}\right)^{O(\Delta_{7t} \cdot k)},$$

where NF is the complexity for computing the net-forest from Theorem 2. Within the same time bound, we can construct a sequence of approximation complexes $(\mathcal{A}_\alpha)_{\alpha \in [0,t]}$ of size $n(\frac{2}{\varepsilon})^{O(\Delta_{7t} \cdot k)}$ whose persistence module is an $(1+\varepsilon)$-approximation (in the sense that the two modules are interleaved *[4]) of the* truncated Čech filtration $(\mathcal{C}_\alpha)_{\alpha \in [0,t]}$.

The claim follows directly from Lemmas 8, 9 and 10 and the preceding construction.

A Streaming Algorithm for 2-Center with Outliers in High Dimensions

Behnam Hatami* Hamid Zarrabi-Zadeh†

Abstract

We study the 2-center problem with outliers in high-dimensional data streams. Given a stream of points in arbitrary d dimensions, the goal is to find two congruent balls of minimum radius covering all but z points. We provide a $(1.8+\varepsilon)$-approximation streaming algorithm for the problem, improving upon the previous $(4+\varepsilon)$-approximation algorithm available for the problem. The space complexity and update time of our algorithm is $\text{poly}(d, z, \frac{1}{\varepsilon})$, independent of the size of the stream.

1 Introduction

The k-center problem—covering a set of points using k congruent balls of minimum radius—is a fundamental problem, arising in many applications such as data mining, machine learning, statistics, and image processing. In real-world applications, where input data is often noisy, it is very important to consider outliers, as even a small number of outliers can greatly affect the quality of the solution. The k-center problem is particularly very sensitive to outliers, and even a constant number of outliers can increase the radius of the k-center unboundedly. Therefore, it is natural to consider the following generalization of the the k-center problem: given a set P of n points in arbitrary d dimensions and a bound z on the number of outliers, find k congruent balls of minimum radius to cover at least $n-z$ points of P. See Figure 1 for an example.

In this paper, we focus on the *data stream* model of computation where only a single pass over input is allowed, and we have only a limited amount of working space available. This model is in particular useful for processing large data sets, as it does not require the entire data set to be stored in memory.

The Euclidean k-center problem has been extensively studied in the literature. If k is part of the input, the problem in known to be NP-hard in two and more dimensions [10], and is even hard to approximate to within a factor better than 1.82, unless P = NP [9]. Factor-2 approximation algorithms are available for the problem in any dimension [11, 9]. For small k and d, better solutions are available. The 1-center problem in fixed dimensions is known to be LP-type and can be solved in $O(n)$ time [7]. For 2-center in the plane, the current best algorithm runs in $O(n \log^2 n \log^2 \log n)$ time [4].

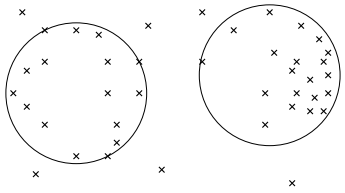


Figure 1: An example of 2-center with 6 outliers.

For k-center with outliers, Charikar *et al.* [6] gave the first algorithm with an approximation factor of 3, which works in any dimension. Better results are known for small k in the plane. The 1-center problem with z outliers in the plane can be solved in $O(n \log n + z^3 n^{\varepsilon})$ time, for any $\varepsilon > 0$, using Matoušek's framework [14]. Agarwal [1] gave a randomized $O(nz^7 \log^3 z)$-time algorithm for 2-center with z outliers in the plane.

In the streaming model, McCutchen *et al.* [15] and Guha [12] presented algorithms to maintain $(2+\varepsilon)$-approximation to the k-center problem in $O(\frac{kd}{\varepsilon} \log \frac{1}{\varepsilon})$ space. For $k=1$, a factor-$((1+\sqrt{3})/2)$ approximation was presented by Agarwal and Sharathkumar [2] in high dimensions, using $O(d)$ space. The approximation factor was later improved to 1.22 by Chan and Pathak [5]. For $k=2$, Kim and Ahn [13] have recently obtained a $(1.8+\varepsilon)$-approximation using $O(\frac{d}{\varepsilon})$ space and update time.

For k-center with z outliers in the streaming model, McCutchen *et al.* [15] gave a $(4+\varepsilon)$-approximation algorithm using $O(\frac{zk}{\varepsilon})$ space. When dimension is fixed, a $(1+\varepsilon)$-approximation to 1-center with outliers can be maintained in $O(z/\varepsilon^{((d-1)/2)})$ space using the notion of robust ε-kernels [3, 16]. For 1-center with outliers in high dimensions, Zarrabi-Zadeh and Mukhopadhyay [17] gave a $(\sqrt{2}\alpha)$-approximation, where α is the approximation factor of the underlying algorithm for maintaining 1-center. Combined with the 1.22-approximation algorithm of Chan and Pathak [5], it yields an approximation factor of $(\sqrt{2} \times 1.22) < 1.73$ using $O(d^3 z)$ space and $\text{poly}(d, z)$ update time.

Our result In this paper, we study the 2-center problem with outliers in high dimensions. We present a

*Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. bhatami@ce.sharif.edu

†Department of Computer Engineering, Sharif University of Technology, Tehran, Iran. zarrabi@sharif.edu

Problem	Approximation Factor	
	Without Outliers	With Outliers
1-Center	1.22 [5]	1.73 [17]
2-Center	$1.8+\varepsilon$ [13]	$1.8+\varepsilon$ [Here]
k-Center	$2+\varepsilon$ [12, 15]	$4+\varepsilon$ [15]

Table 1: Summary of the streaming algorithms for k-center with and without outliers in high dimensions.

streaming algorithm for the problem that achieves an approximation factor of $1.8+\varepsilon$, for any $\varepsilon > 1$, using $\text{poly}(d, z, \frac{1}{\varepsilon})$ space and update time. This improves the current streaming algorithm available for the problem which has an approximation factor of $4+\varepsilon$. The approximation factor of our algorithm matches that of the best streaming algorithm for the 2-center problem without outliers. This is somewhat surprising, considering that the current best approximation factors for streaming k-center with and without outliers differ by a multiplicative factor of $\sqrt{2}$ for $k=1$, and by a factor of 2 for general k. See Table 1 for a comparison.

To obtain our result, we have used a combination of several ideas including parallelization, far/close ball separation, centerpoint theorem, and keeping a lower/upper bound on the radius and distance of the optimal balls. We have also utilized ideas used in [13] for the 2-center problem with no outliers. However, our problem is much harder here, as we not only need to find balls of minimum radius, but we also need to decide which subset of points to cluster. This is in particular more challenging in the streaming model, where we only have a single pass over the input, and we must decide on the fly which point is an outlier, and which one can be safely ignored as a non-outlier point, to comply with the working space restriction enforced by the model.

2 Preliminaries

Let $B(c,r)$ denote a ball of radius r centered at c. We use $r(B)$ to denote the radius of a ball B. For two points p and q, the distance between p and q is denoted by $\|pq\|$. Given two balls $B(c,r)$ and $B'(c',r')$, we define $\delta(B,B') = \max\{0, \|cc'\| - r - r'\}$ to be the *distance* between B and B'. Two balls B_1 and B_2 are said to be *α-separated*, if $\delta(B_1,B_2) \geqslant \alpha \cdot \max\{r(B_1), r(B_2)\}$.

Given an n-point set P in d-dimensions, a point $c \in \mathbb{R}^d$ is called a *centerpoint* of P, if any halfspace containing c contains at least $\lceil n/(d+1) \rceil$ points of P. It is well-known that any finite set of points in d-dimensional space has a centerpoint [8]. The following observation is a corollary of this fact.

Observation 1 *Given a set P of $k(d+1)$ points in d-dimensional space, the centerpoint of P has the property that any convex object not covering the centerpoint, leaves at least k points of P uncovered.*

Given a point set P, the *k-furthest point from $p \in P$* is a point whose distance to p is the k-th largest among all points in P. We assume the standard word-RAM model of computation. Each coordinate value takes a unit of space. Thus, a d-dimensional point takes $O(d)$ space, and basic operations on the points take $O(d)$ time.

3 A Simple Algorithm for 1-Center with Outliers

To warm up, we present a simple 2-approximation streaming algorithm for the 1-center problem with outliers. It utilized a parallelization technique that will be used extensively in the rest of the paper. The pseudocode is provided in Algorithm 1. The algorithm receives as input a stream of points, P, and the number of outliers, z. It assumes that the first point p_1 of the stream is non-outlier. We will show later how to remove this assumption. The algorithm returns a ball B covering all but at most z points of P.

Algorithm 1 1-CENTER(P, z)

```
1: c ← the first point in P
2: B ← B(c, 0)
3: Q ← ∅
4: for each p in P do
5:     if p ∉ B then
6:         insert p into Q
7:         if |Q| = z + 1 then
8:             q ← closest point to c in Q
9:             remove q from Q
10:            B ← B(c, ‖cq‖)
11: return B
```

Theorem 1 *Algorithm 1 computes a 2-approximation to the 1-center problem with z outliers, assuming that the first point of the stream is not outlier.*

Proof. Let $B^*(c^*, r^*)$ be the optimal solution, and c be a non-outlier point in the optimal solution. Since c is covered by B^*, for all points $p \in B^*$, we have $\|cp\| \leqslant \|cc^*\| + \|c^*p\| \leqslant 2r^*$. Among the $z+1$ points furthest from c, there is at least one point q which is not outlier, and therefore, is contained in B^* (see Figure 2). Thus, $\|cq\| \leqslant 2r^*$, and hence, the ball $B(c, \|cq\|)$ returned by Algorithm 1 is a 2-approximation. □

Algorithm 1 assumes that the first point of the stream is not outlier. To remove this assumption, we run $z+1$ instances of Algorithm 1 in parallel, each of which is given as input one of the first $z+1$ points of the stream, followed by the rest of the points. Clearly, there exists a point among the first $z+1$ points of P which is not an

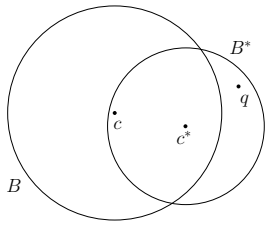


Figure 2: Proof of Theorem 1

outlier in the optimal solution. Therefore, the smallest ball among the $z+1$ balls computed in parallel is always within factor 2 of the optimal solution. Since the space complexity of Algorithm 1 for one instance is $O(zd)$, and its update time is $O(zd \log z)$, we get the following result.

Theorem 2 *Given a stream of points in d dimensions, we can maintain a 2-approximation to the 1-center with z outliers in $O(z^2 d)$ space and $O(z^2 d \log z)$ update time.*

4 The 2-Center Problem with Outliers

In this section, we provide a $(1.8 + \varepsilon)$-approximation algorithm for the 2-center problem with outliers. In all algorithms presented in this section, we assume that the first point of the stream, p_1, is non-outlier. This assumption can be easily removed by considering $z + 1$ parallel instances of the algorithm, similar to what we did in Section 3.

Let B_1^* and B_2^* be the balls in an optimal solution to the 2-center problem with z outliers on a point set P. We denote by r^* the optimal radius, and by δ^* the distance between B_1^* and B_2^*. To prove our main result, we distinguish between two cases. In Section 4.1, we address the case where $\delta^* > \alpha r^*$, for some constant α to be fixed later. (It will turn out that $\alpha = 16$ is a proper choice.) We then present in Section 4.2 our algorithm for the case of $\delta^* \leqslant \alpha r^*$.

4.1 The Case $\delta^* > \alpha r^*$

In this section, we present a 1.8-approximation algorithm for the case where optimal balls are separated by a distance greater than αr^*. We start by two useful observations.

Observation 2 *Let B_1 and B_2 be two congruent balls of radius r, with distance $\delta > \alpha r$. For any two points $p \in B_1$ and $q \in B_2$, we have $1 \leqslant \frac{\|pq\|}{\delta} < \frac{\alpha+4}{\alpha}$.*

Proof. The distance between p and q is at most $\delta + 4r$. Hence, $\frac{\|pq\|}{\delta} \leqslant 1 + \frac{4r}{\delta} < 1 + \frac{4}{\alpha}$. □

Observation 3 *Let B_1 and B_2 be two disjoint balls of distance δ, and let B be an arbitrary ball of radius less than $\frac{\delta}{2}$. Then B intersects at most one of B_1 and B_2.*

We next prove some properties regarding the optimal balls, B_1^* and B_2^*.

Lemma 3 *Let B_1^* and B_2^* be α-separated, with $\alpha > 4$. If p is a point in B_1^*, and S is a $(z+1)$-subset of P furthest from p, then $S \cap B_2^*$ is non-empty.*

Proof. Suppose by way of contradiction that $S \cap B_2^*$ is empty. Since $|S| = z + 1$, there is at least one point in S which is not outlier, and hence, it is in B_1^*. Let q be a point in $S \cap B_1^*$ furthest from p. Consider the ball $B(p, \|pq\|)$. For any point $s \in P \setminus S$, we have $\|ps\| \leqslant \|pq\|$, because $s \notin S$ and $q \in S$. Therefore, B covers $P \setminus S$. Since $p, q \in B_1^*$, $\|pq\|$ is at most $2r^*$. Thus, by Observation 3, $B_2^* \cap B = \emptyset$. Therefore, $B_2^* \cap P = \emptyset$, and hence, B_2^* is empty, which contradicts the optimality of the solution. □

Lemma 4 *Let p be a point in B_1^*, and q be the $(z+1)$-furthest point from p. Then, $\delta^* > \frac{\alpha}{\alpha+4}\|pq\|$.*

Proof. By Lemma 3, there exists a point $q' \in B_2^*$ such that $\|pq'\| \geqslant \|pq\|$. Thus, by Observation 2, $\frac{\|pq\|}{\delta^*} \leqslant \frac{\|pq'\|}{\delta^*} < \frac{\alpha+4}{\alpha}$. □

Lemma 5 *If $p \in B_1^*(c_1, r^*)$ and $q \in B_2^*(c_2, r^*)$, then $B_1^* \subset B(p, 2r^*)$ and $B_2^* \subset B(q, 2r^*)$, and hence, at most z points of P lie outside $B(p, 2r^*) \cup B(q, 2r^*)$.*

Proof. For an arbitrary point $p' \in B_1^*$, $\|pp'\| \leqslant \|pc_1\| + \|p'c_1\| \leqslant 2r^*$, and as a result, $B_1^* \subset B(p, 2r^*)$. Similarly, we have $B_2^* \subset B(q, 2r^*)$. Considering that at most z points of P are outlier, the proof is complete. □

Lemma 6 *Let S be a subset of P of size at least $(d+1)(z+1)$, enclosed by a ball B of radius less than $\delta^*/2$. Then the centerpoint c_p of S lies inside either B_1^* or B_2^*.*

Proof. Not all points in S can be outlier, because $(d+1)(z+1) > z$. Thus, by Observation 3, B intersects exactly one of B_1^* and B_2^*. Assume, w.l.o.g., that B intersect B_1^*. Now, by Observation 1, if c_p is not in B_1^*, then $z + 1$ points of S remain uncovered by B_1^*, contradicting the fact that there at most z outliers. □

The Algorithm We now describe our algorithm for handling the case $\delta^* > \alpha r^*$. At any time, our algorithm maintains a partition of P into three disjoint subsets B_1, B_2, and Buffer. The first point p_1 is assumed, w.l.o.g, to be in B_1^*. (We have already assumed that p_1 is not outlier.) The algorithm tries to partition points in such a way that at the end, B_1 contains the whole B_1^*, and B_2 contains the whole B_2^*, with possibly some outliers being contained in B_1 and B_2. The algorithm sets $c_1 = p_1$ as the fixed center of B_1, and picks c_2 among the points processed so far as a candidate for being the center of

Algorithm 2 2-Center-Separated(P)

```
1: c_1 ← p_1, r ← 0, δ ← 0
2: for each p ∈ P do
3:     if not (AddToB_1(p) or AddToB_2(p)) then
4:         add p to Buffer
5:         while |Buffer| > z do
6:             if |B_2| ⩾ (d+1)(z+1) then
7:                 B_1 ← B_1 ∪ B_2, B_2 ← ∅
8:             else if c_2 is set then
9:                 B_1 ← B_1 ∪ {c_2}
10:            T ← Buffer ∪ B_2 \ {c_2}
11:            B_2 ← ∅
12:            c_2 ← (z+1)-furthest point from c_1 in T
13:            r ← (2/α)‖c_1c_2‖
14:            for p ∈ T do
15:                AddToB_2(p)
16:            Buffer ← T \ B_2
```

B_2. Moreover, the algorithm maintains two values δ and r, where at any time, δ is a lower bound of δ^*, and r is an upper bound of $2r^*$ (under a certain condition).

Our algorithm is presented in Algorithm 2. For each input point $p \in P$, the algorithm first tries to add p to either B_1 or B_2, using functions AddToB$_1$ and AddToB$_2$, respectively. If none of them fits, the point is added to Buffer. The function AddToB$_1$ adds a point p to B_1 only if it is within distance δ of the center c_1. Similarly, AddToB$_2$ adds a point p to B_2 only if it is within r-radius of c_2. The two functions also update the values of δ and r whenever necessary, to maintain the invariants to be defined in Lemma 7.

Algorithm 3 AddToB$_1$(p)

```
1: if at least z + 1 points have been processed then
2:     q ← (z+1)-furthest point from c_1
3: else
4:     q ← c_1
5: δ ← (α/(α+4))‖c_1q‖
6: if p ∈ B(c_1, δ) then
7:     B_1 = B_1 ∪ {p}
8:     return true
9: return false
```

Whenever the buffer overflows (in line 5 of Algorithm 2), the algorithm takes one of the following actions depending on the size of B_2. If $|B_2| \geqslant (d+1)(z+1)$, then the points of B_2 are moved to B_1, and B_2 is reset. Otherwise, the old c_2 (if already set) is moved to B_1, and another point from $T = B_2 \cup \text{Buffer} \setminus \{c_2\}$ is picked as c_2. The while loop iterates at most $O(dz)$ times, because after the first iteration, we are sure that T has at most $(d+1)(z+1)+z$ points, from which one point (i.e., c_2) is removed at each subsequent iteration.

For the sake of analysis, we maintain a "central point", denoted by c_p, which is defined as follows: if $|B_2| < (d+1)(z+1)$, then $c_p = c_2$, otherwise, c_p is the centerpoint of the first $(d+1)(z+1)$ points currently in B_2.

Algorithm 4 AddToB$_2$(p)

```
1: if c_2 is set and p ∈ B(c_2, r) then
2:     B_2 ← B_2 ∪ {p}
3:     if |B_2| = (d+1)(z+1) then
4:         r ← (2 + 2/α) × r
5:         for p in Buffer do
6:             if p ∈ B(c_2, r) then
7:                 B_2 ← B_2 ∪ {p}
8:                 remove p from Buffer
9:     return true
10: return false
```

Lemma 7 *The following invariants are maintained during the execution of the algorithm:*

(a) $\delta < \delta^*$

(b) $r \leqslant \delta/2$

(c) $B_1 \cap B_2^* = \emptyset$

(d) if $c_p \in B_2^*$, *then*

1. $2r^* \leqslant r$
2. $B_2 \cap B_1^* = \emptyset$
3. *all points in Buffer are outlier*

Proof. Invariant (a): At the beginning, $\delta = 0$, which clearly satisfies the invariant. After $z+1$ points of the stream is processed, function AddToB$_1$ starts updating δ to $\frac{\alpha}{\alpha+4}\|c_1q\|$, where q is the $(z+1)$-furthest point from c_1 in the current stream. Now, since $c_1 \in B_1^*$, Lemma 4 implies that $\delta < \delta^*$.

Invariant (b): When c_2 is set by Algorithm 2, it is the $(z+1)$-furthest point from c_1 in a set $T \subseteq P$, and r is set to $\frac{2}{\alpha}\|c_1c_2\|$. Let q be the $(z+1)$-furthest point from c_1 in the stream at that moment. Then $\|c_1c_2\| \leqslant \|c_1q\|$. Assuming $\alpha \geqslant 16$, we have $\frac{2}{\alpha}\|c_1c_2\| \leqslant \frac{1}{6}\frac{\alpha\|c_1q\|}{(\alpha+4)} \leqslant \delta/6$, and hence,

$$r \leqslant (2+\frac{2}{\alpha})\frac{2}{\alpha}\|c_1c_2\| \leqslant 3 \times \frac{2}{\alpha}\|c_1c_2\| \leqslant \delta/2,$$

which means that the invariant holds, even after increasing r by function AddToB$_2$.

Invariant (c): The proof is provided in Appendix C.

Invariant (d1): By Observation 2, if $c_1 \in B_1^*$ and $c_p \in B_2^*$, then $1 \leqslant \frac{\|c_1c_p\|}{\delta^*} \leqslant \frac{\|c_1c_p\|}{\alpha r^*}$, and as a result, $2r^* \leqslant \frac{2}{\alpha}\|c_1c_p\|$. If $|B_2| < (d+1)(z+1)$, then $c_p = c_2$, and by Algorithm 2, $r = \frac{2}{\alpha}\|c_1c_2\|$, and therefore, $2r^* \leqslant r$. If $|B_2| \geqslant (d+1)(z+1)$, then similar to invariant (b),

$$2r^* \leqslant \frac{2}{\alpha}\|c_1c_p\| \leqslant (1+\frac{2}{\alpha})\frac{2}{\alpha}\|c_1c_2\| \leqslant (2+\frac{2}{\alpha})\frac{2}{\alpha}\|c_1c_2\| = r.$$

Invariant (d2): By invariant (d1), if $c_p \in B_2^*$ then $2r^* \leqslant r \leqslant \delta/2$ and $c_p \in B_2$. Now, by invariant (a) and Observation 3, B_2 intersect only B_2^*, and hence, $B_2 \cap B_1^* = \emptyset$.

Invariant (d3): By invariants (c) and (d1), $2r^* \leqslant r \leqslant \delta/2 < \delta^*/2$. Therefore, by Lemma 5, all points outside $B_1 \cup B_2$ are outlier. □

Theorem 8 *If $\delta^* > \alpha r^*$, a 1.8-approximation to the 2-center problem with z outliers can be maintained in $O(d^3z^2)$ space and* poly(d, z) *update/query time.*

Proof. Our algorithm for answering queries is provided in Appendix A. It uses the current partition B_1, B_2, and Buffer, to compute an optimal solution to 2-center with z outliers. In the streaming model, we cannot afford keeping all the points of B_1 and B_2. Therefore, we maintain the sets B_1 and B_2 in a data structure that supports adding points, and gives a β-approximation to 1-center with k outliers, for $k = 0, \ldots, z$. Moreover, we maintain a set $B_u = B_1 \cup B_2$ in a similar data structure. Note that these data structures do not need to maintain all the points. They only need to have a buffer of size $(d+1)(z+1)$ to keep the most recently added points.

To maintain B_1, B_2, and B_u, we use the streaming algorithm of [17, 5], which provides an approximation factor of $1.22 \times \sqrt{2} < 1.8$. The algorithm uses $O(d^3z)$ space and has poly(d, z) update time. Since we need to run $z+1$ instances of Algorithm 2 in parallel, the space and update time are multiplied by a factor of z. □

4.2 The Case $\delta^* \leqslant \alpha r^*$

Our idea in this section is to carefully adopt the algorithm of Kim and Ahn [13], originally designed for maintaining an approximate 2-center. To avoid duplication, we just sketch the main steps of their algorithm, and explain our modifications to it. Kim and Ahn's algorithm, which we refer to as the KA algorithm, has 9 different states, shown in Figure 3. Depending on the points arrived so far, the algorithm is in one of the states. In each state, the algorithm keeps at most two balls as a candidate solution. A transition between the states occurs whenever a point not covered by any of the two balls arrive.

The algorithm starts at node 1, and proceed through the transition graph as points arrive. In some states, there is more than one state to follow, and the algorithm has no prior information which one is the correct choice. However, there are only three different paths to follow in the transition graph. Hence, we can easily run three instances of the algorithm in parallel, each of which follows one of the paths deterministically, to make sure that at any time, at least one of the instances is in a correct state.

Our modification is on the transition part. Points that are covered by the current solution can be safely ignored, as they do not cause any change in the current solution, and hence, they cause no transition. Only those points that lie outside the current solution are candidates for being outliers. Since the number of outliers in each state is unknown, we try all possible choices. The observation here is that the transition graph is a DAG of depth four. If n_i $(1 \leqslant i \leqslant 4)$ represents the number of outliers in depth i, then it suffices to consider all tuples $(n_1, \ldots, n_4)$ such that $\sum_{i=1}^{4} n_i = z$, It is easy to verify that there are $O(z^3)$ such tuples.

The pseudocode of our algorithm is presented in Algorithm 5. For each possible choices of n_1 to n_4, and each of the three paths in the transition graph, numbered from 1 to 3, the algorithm keeps a candidate solution (B_1, B_2) to the 2-center of non-outlier points, a parameter j representing the current level in the transition graph, and four counters to keep track of number of outliers seen so far at each level.

The algorithm starts with $B_1 = B(p_1, r)$ and $B_2 = \emptyset$, which corresponds to Case 1 of the KA algorithm. For each new point p, we first check if it is contained in the current solution. If so, then we are done. Otherwise, if the number of outliers seen in the current level has not yet reached n_j, we consider p as an outlier and pro-

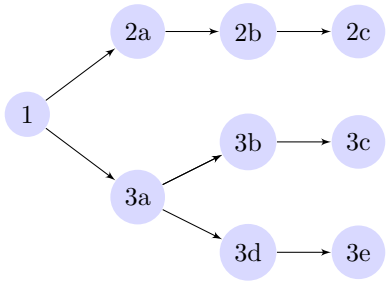


Figure 3: State diagram of the KA algorithm. Labels are taken from [13].

Algorithm 5 2-Center-Close(P, z, r)

```
1: solutions ← {}
2: for each (n_1, n_2, n_3, n_4) such that ∑ n_i = z do
3:     for each π ∈ {1, 2, 3} do
4:         counter_i ← 0, for i = 1, ..., 4
5:         B_1 ← B(p_1, r), B_2 ← ∅
6:         j ← 1    ▷ j represents current level
7:         for each p ∈ P do
8:             if p ∉ B_1 ∪ B_2 then
9:                 counter_j ← counter_j + 1
10:                if counter_j > n_j then
11:                    j ← j + 1
12:                (B_1, B_2) ← KA.INSERT(p, π)
13:        if j ⩽ 4 then
14:            add max{r(B_1), r(B_2)} to solutions
15: return min{solutions}
```

ceed. Otherwise, we go to the next level, and update the current candidate solution, (B_1, B_2), using the KA algorithm. We give the transition path π along with the point p to the KA algorithm to help it deterministically decide which state to choose as the next one.

After all points in P are processed, if we are in one of the four states in the current path, then the obtained solution is added to the feasible solutions. Otherwise, the solution is not feasible, and is abandoned as in the KA algorithm. Finally, we return the best solution among all computed feasible solutions. Kim and Ahn [13] proved that in all feasible solutions computed this way, the larger ball among B_1 and B_2 has radius at most $3/2r$, provided $\delta^* \leqslant \alpha r^*$. (Their proof is stated for $\alpha = 2$, but can be extended to any $\alpha \geqslant 2$.) Assuming that we have a good estimate r satisfying $1.2r^* \leqslant r < (1.2 + 2\varepsilon/3)r^*$, we get the following.

Theorem 9 *For $1.2r^* \leqslant r < (1.2 + \frac{2}{3}\varepsilon)r^*$ and $\delta^* \leqslant \alpha r^*$, Algorithm 5 computes a $(1.8+\varepsilon)$-approximation to the 2-center with z outliers in $O(dz^3)$ space and $O(dz^3)$ update time, assuming that the first point of the stream is not outlier.*

As shown in Appendix B, a desired estimate for r can be obtained by running $O(1/\varepsilon)$ instances of Algorithm 5 in parallel. Adding another level of parallelization to remove the assumption of p_1 being a non-outlier, we get the following.

Theorem 10 *If $\delta^* \leqslant \alpha r^*$, a $(1.8+\varepsilon)$-approximation to the 2-center problem with z outliers can be maintained in $O(\frac{dz^4}{\varepsilon})$ space and $O(\frac{dz^4}{\varepsilon})$ update/query time.*

Theorems 8 and 10 together yield the following main result of the paper.

Theorem 11 *Given a stream of points in d dimensions, we can maintain a $(1.8+\varepsilon)$-approximation to the 2-center problem with z outliers using $O(dz^2(d^2+z^2/\varepsilon))$ space and $\mathrm{poly}(d, z, \frac{1}{\varepsilon})$ update/query time.*

5 Conclusions

In this paper, we presented a $(1.8 + \varepsilon)$-approximation streaming algorithm for 2-center problem with outliers in Euclidean space. It improves the previous $(4 + \varepsilon)$-approximation algorithm available for the problem due to McCutchen and Khuller [15]. Finding better approximation factor or space complexity is an interesting problem that remains open. It is also interesting to see if the ideas in this paper can be extended to the k-center problem with outliers in the data stream model, even for small values of $k \geqslant 3$.

Acknowledgement The authors would like to thank Kiana Ehsani and Sahand Mozaffari for their thoughtful discussions, and for their very helpful comments.

References

[1] P. K. Agarwal and J. M. Phillips. An efficient algorithm for 2d Euclidean 2-center with outliers. In *Proc. 16th Annu. European Sympos. Algorithms*, pages 64–75. 2008.

[2] P. K. Agarwal and R. Sharathkumar. Streaming algorithms for extent problems in high dimensions. In *Proc. 21st ACM-SIAM Sympos. Discrete Algorithms*, pages 1481–1489, 2010.

[3] P. K. Agarwal and H. Yu. A space-optimal data-stream algorithm for coresets in the plane. In *Proc. 23rd Annu. ACM Sympos. Comput. Geom.*, pages 1–10, 2007.

[4] T. M. Chan. More planar two-center algorithms. *Comput. Geom. Theory Appl.*, 13(3):189–198, 1999.

[5] T. M. Chan and V. Pathak. Streaming and dynamic algorithms for minimum enclosing balls in high dimensions. *Comput. Geom. Theory Appl.*, 47(2):240–247, 2014.

[6] M. Charikar, S. Khuller, D. M. Mount, and G. Narasimhan. Algorithms for facility location problems with outliers. In *Proc. 12th ACM-SIAM Sympos. Discrete Algorithms*, pages 642–651, 2001.

[7] B. Chazelle and J. Matoušek. On linear-time deterministic algorithms for optimization problems in fixed dimension. *J. Algorithms*, 21(3):579–597, 1996.

[8] L. Danzer, B. Gruenbaum, and V. Klee. Helly's theorem and its relatives. In *Proc. Symposia in Pure Mathematics 7*, pages 101–180, 1963.

[9] T. Feder and D. Greene. Optimal algorithms for approximate clustering. In *Proc. 20th Annu. ACM Sympos. Theory Comput.*, pages 434–444, 1988.

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

[11] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoret. Comput. Sci.*, 38:293–306, 1985.

[12] S. Guha. Tight results for clustering and summarizing data streams. In *Proc. 12th Internat. Conf. Database Theory*, pages 268–275, 2009.

[13] S.-S. Kim and H.-K. Ahn. An improved data stream algorithm for clustering. In *Proc. 11th Latin American Theoret. Inform. Sympos.*, pages 273–284. 2014.

[14] J. Matoušek. On geometric optimization with few violated constraints. *Discrete Comput. Geom.*, 14(1):365–384, 1995.

[15] R. M. McCutchen and S. Khuller. Streaming algorithms for k-center clustering with outliers and with anonymity. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques*, pages 165–178. 2008.

[16] H. Zarrabi-Zadeh. An almost space-optimal streaming algorithm for coresets in fixed dimensions. *Algorithmica*, 60(1):46–59, 2011.

[17] H. Zarrabi-Zadeh and A. Mukhopadhyay. Streaming 1-center with outliers in high dimensions. In *Proc. 21st Canad. Conf. Computat. Geom.*, pages 83–86, 2009.

A Answering Queries

In the following, we show how the information maintained by Algorithm 2 can be used to answer queries of this kind: find two α-separated congruent balls of minimum radius to cover all but at most z points of the stream processed so far.

Our query algorithm is presented in Algorithm 6. The idea behind the algorithm is as follows. By our initial assumption about p_1 and by invariants (c) and (d), if $c_p \in B_2^*$, then we know that B_1 completely contains B_1^*, and B_2 completely contains B_2^*. However, it might be the case that our assumption about c_p was incorrect, and therefore, B_1 (resp., B_2) may not completely contain B_1^* (resp., B_2^*). To overcome this issue, we try all possible candidates for c_2 (which in turn, determines c_p), and compute, for each resulting partition of P into B_1, B_2, and Buffer, the best solution for 2-center with z outliers using the MinCover function presented in Algorithm 7.

Algorithm 6 Query

```
solutions ← {}
candidates ← B_2 ∪ Buffer
if |B_2| ⩾ (d+1)(z+1) then
    candidates ← {c_2} ∪ Buffer
    B_1 ← B_1 ∪ B_2
δ_0 = δ
for c ∈ candidates do
    r ← (2/α)‖c_1 c‖
    B'_1 ← B_1, δ ← max{δ_0, r}
    B'_2 ← ∅, Buffer' ← ∅
    for p ∈ candidates do
        if not (AddToB'_1(p) or AddToB'_2(p)) then
            add p to Buffer'
    add MinCover(B'_1, B'_2, Buffer') to solutions
return min{solutions}
```

Algorithm 7 MinCover$(B_1, B_2, \text{Buffer})$

```
solutions ← {}
for k ← 0, ..., (z − |Buffer|) do
    r_1 ← 1-Center(B_1, k)
    r_2 ← 1-Center(B_2, z − |Buffer| − k)
    add max{r_1, r_2} to solutions
return min{solutions}
```

Let C denote the set of candidates for c_2. By invariant (c), we know that $B_1 \cap B_2^* = \emptyset$. Therefore, there exists a point in $(B_2 \cup \text{Buffer}) \cap B_2^*$, and hence, C is $(B_2 \cup \text{Buffer}) \cap B_2^*$ in general. However, when $|B_2| \geqslant (d+1)(z+1)$, we will show in the following lemma that $(\{c_2\} \cup \text{Buffer}) \cap B_2^* \neq \emptyset$. Therefore, if $|B_2| \geqslant (d+1)(z+1)$, we only need to consider $\{c_2\} \cup \text{Buffer}$ as candidates for C.

Lemma 12 *At any time, if $|B_2| \geqslant (d+1)(z+1)$ and $c_p \notin B_2^*$, then $B_2 \cap B_2^* = \emptyset$.*

Proof. By invariants (a) and (b), we know that $r \leqslant \delta/2 < \delta^*/2$. By Lemma 6, $c_p \in B_1^* \cap B_2^*$. Since $c_p \notin B_2^*$, we have $c_p \in B_1^*$. On the other hand, by Observation 3, B_2 intersect at most one of B_1^* and B_2^*. Therefore, $B_2 \cap B_2^* = \emptyset$. □

Our query algorithm works as follows. For each candidate point $c \in C$, Algorithm 6 constructs $B'_1(c_1, \max\{\delta, \frac{2}{\alpha}\|c_1 c\|\})$ and $B'_2(c, \frac{2}{\alpha}\|c_1 c\|)$. If the candidate c equals the current c_2, then we have $B_1 = B'_1$. Since $\frac{2}{\alpha}\|c_1 c_2\| \leqslant r \leqslant \delta/2$ by invariant (b), and $B'_2 \subset B_2$, we do not need to construct any new set. For $c \neq c_2$, we know that $B_1 \subset B'_1$, and hence, we only need to see which points in Buffer $\cap B_2$ are inside B'_1. When $|B_2| \geqslant (d+1)(z+1)$ and $c \neq c_2$, then it means that $c_p \notin B_2^*$. Therefore, by Lemma 12, B_2 can be added to B'_1 without violating invariant (c). So in this case, we just need to see which points of Buffer must be added to B'_1. Algorithm 6 uses functions AddToB$'_1$ and AddToB$'_2$ for adding a point to B'_1 and B'_2 respectively. These functions are the same as AddToB$_1$ and AddToB$_2$, with the only exception that they add points to B'_i instead of B_i, for $i = 1, 2$.

Since Algorithm 6 considers all valid candidates for c, at least for one $c^* \in C$, we have $c^* \in B_2^*$. We denote the corresponding B'_1 and B'_2 by B''_1 and B''_2. Since by Observation 2, $1 \leqslant \frac{\|c_1 c^*\|}{\alpha^*} < \frac{\|c_1 c^*\|}{\alpha r^*}$, we have $2r^* \leqslant \frac{2}{\alpha}\|c_1 c^*\|$, and hence by Lemma 5, $B_1^* \subset B''_1$ and $B_2^* \subset B''_2$. On the other hand, since the distance of the new points added to B''_1 is less than $\|c_1 p\| \leqslant \max\{\delta, \frac{2}{\alpha}\|c_1 c^*\|\}$, we have by invariant (c) that $B''_1 \cap B_2^* = \emptyset$. As a result, B''_1 (resp., B''_2) completely covers B_1^* (resp., B_2^*), and the points in Buffer are all outliers. The only unknown part is that Algorithm 6 does not know how many outliers are in B''_1 and B''_2. Therefore, Algorithm 7 tries all possible cases and choose the one with the minimum radius.

B Estimating r

In this section, we show how to obtain a value r, such that $1.2r^* \leqslant r < (1.2 + 2\varepsilon/3)r^*$. The following lemma provides the main ingredient.

Lemma 13 *Given a point set P in $\mathbb{R}^d$, an optimal solution to the 1-center problem with z outliers on P gives a $(2 + \frac{\alpha}{2})$-approximation for the 2-center with z outliers on P, provided that $\delta^* \leqslant \alpha r^*$.*

Proof. Let r_1^* and r^* be the optimal radii for the 1-center and 2-center problems with z outliers on P, respectively. It is clear that $r^* \leqslant r_1^*$, because any feasible solution B^* for 1-center with z outliers yields a feasible solution (B^*, B^*) for 2-center with z outliers. Now,

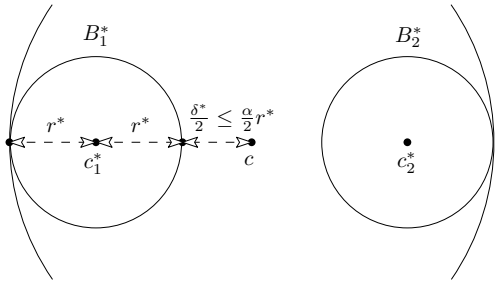


Figure 4: Proof of Lemma 13

suppose that $B_1^*(c_1^*, r^*)$ and $B_2^*(c_2^*, r^*)$ are the balls in an optimal solution for the 2-center problem with z outliers. Let c be the midpoint of the segment connecting c_1^* to c_2^* (see Figure 4). Clearly, $B\left(c, \frac{\delta}{2} + 2r^*\right)$ covers both B_1^* and B_2^*. Therefore, it is a feasible solution for the 1-center problem with z outliers. Hence, $r_1^* \leqslant \left(2 + \frac{\alpha}{2}\right) r^*$. □

The following is a direct corollary of Lemma 13 and Theorem 1.

Corollary 14 *If $\delta^* \leqslant \alpha r^*$, Algorithm 1 computes a $(4+\alpha)$-approximation to r^*.*

We use Algorithm 1 to find an estimate for r. Let r_i be the radius calculated by Algorithm 1 after receiving the i-th point, p_i. Clearly, the sequence of r_i's is increasing. Let k be an integer such that $2^{k-1} \leqslant r_i \leqslant 2^k$, and set $\ell_i = 2^k$. (If $r_i = 0$, we set $\ell_i = 0$.) Obviously, $\ell_i \leqslant 2r_i$, and hence, by Corollary 14, ℓ_i is a $(8+2\alpha)$-approximation to r^*.

We divide the interval $(0, 1.2\ell_i]$ into $m = \lceil 1.2(3\alpha+12)/\varepsilon \rceil$ equal segments, each of length $t_i = 1.2\ell_i/m$. Clearly, $t_i \leqslant (2\varepsilon/3)r^*$. Therefore, in the set $R_i = \{j \times t_i \,|\, j = 1, \ldots, m\}$, there is at least one value r for which the inequality $1.2r^* \leqslant r \leqslant (1.2 + \frac{2\varepsilon}{3})r^*$ holds.

We run m instances of Algorithm 5 for each value $r \in R_i$ in parallel. Whenever a new point p_i is added, if $\ell_i = \ell_{i-1}$, then $R_i = R_{i-1}$, and the new point is inserted to all parallel instances. If $\ell_i > \ell_{i-1}$, then the set R_i has two types of values. Those values in R_i which are less than $1.2\ell_i$ are also present in L_{i-1}, because t_i/t_{i-1} is a positive power of 2. For these values, we continue executing the corresponding instance. If a value $r \in R_i$ is not present in R_{i-1}, then we have $r \geqslant 1.2\ell_{i-1} \geqslant \ell_{i-1}$. Since those points not lying in the candidate solution are saved in the buffer of Algorithm 1 (which has size at most z), all non-outlier points of this algorithm lie in the candidate balls of Algorithm 5 which has center p_1 and radius at most l_{i-1}. These outliers have been stored in a buffer. Since Algorithm 5 maintains two balls with radius at least r, one of which (say B_1) is centered at p_1, then all non-outlier points of Algorithm 1 are in B_1, and hence, they do not make any transition in the states of Algorithm 5. Therefore, for any new value r, it suffices to execute Algorithm 5 with only the outlier points in the buffer of Algorithm 1.

C Proof of Invariant (c)

Here, we provide a proof for Invariant (c). The following technical claim will be used in our proof.

Claim 1 *If c_2 is set, then $B(c_p, \frac{2}{\alpha}\|c_1c_p\|) \subseteq B_2(c_2, r)$.*

Proof. If $|B_2| < (d+1)(z+1)$, then $c_p = c_2$ and $r = \frac{2}{\alpha}\|c_1c_2\|$, and hence, $B_2 = B(c_p, \frac{2}{\alpha}\|c_1c_p\|)$. When the size of B_2 reaches $(d+1)(z+1)$, the central point c_p moves to the centerpoint of B_2, and r is increased by a factor of $(2 + \frac{2}{\alpha})$. Because the centerpoint of B_2 lies in B_2, then $c_p \in B(c_2, \frac{2}{\alpha}\|c_1c_2\|)$. Thus, if $|B_2| \geqslant (d+1)(z+1)$ then $\|c_2c_p\| \leqslant \frac{2}{\alpha}\|c_1c_2\|$, and therefore,

$$\frac{2}{\alpha}\|c_1c_p\| \leqslant \frac{2(\|c_1c_2\| + \|c_2c_p\|)}{\alpha} \leqslant \frac{2}{\alpha}\|c_1c_2\|(1 + \frac{2}{\alpha}).$$

Hence, $B(c_p, \frac{2}{\alpha}\|c_1c_p\|) \subseteq B_2(c_2, \frac{2}{\alpha}\|c_1c_2\|(2 + \frac{2}{\alpha}))$. □

Now, we prove Invariant (c), which states $B_1 \cap B_2^* = \emptyset$.

Proof. A point p can be added to B_1 in two cases. The first case is in function AddToB$_1$, where the point is added to B_1 only if it is within distance δ of the center c_1, which by invariant (a), guaranties $\|pc_1\| < \delta^*$. Therefore, $p \notin B_2^*$ in this case.

The second case is in Algorithm 2, when the buffer overflows and B_2 is non-empty. The algorithm takes one of the following actions depending on the size of B_2. If $|B_2| < (d+1)(z+1)$, then $c_2 = c_p$. Algorithm 2 adds c_2 to B_1. Suppose by way of contradiction that $c_p \in B_2^*$. Then, by invariants (b) and (d1), $2r^* \leqslant r \leqslant \delta/2 < \delta^*/2$. Therefore, by Lemma 5, there must be at most z points outside $B_1 \cup B_2$, which contradicts the overflow of the buffer. If $|B_2| \geqslant (d+1)(z+1)$, then c_p is the centerpoint of the first $(d+1)(z+1)$ points currently in B_2. In this case, we add all points of B_2 to B_1. By invariant (b), $r \leqslant \delta/2 < \delta^*/2$. Therefore, By Lemma 6, $c_p \in B_1^*$ or $c_p \in B_2^*$. Suppose by way of contradiction that $c_p \in B_2^*$. In this case, by invariant (d1) and Claim 1, B_2 covers $B(c_p, \frac{2}{\alpha}\|c_1c_p\|)$ and $2r^* \leqslant r$. Therefore, Similar to the previous part, it contradicts the overflow of the buffer. □

A Streaming Algorithm for the Convex Hull

Raimi A. Rufai*† Dana S. Richards‡

Abstract

Consider a base station in a wireless sensor network that receives incoming input points and must maintain a running convex hull within a memory constraint. We give a new streaming algorithm that processes each point in time $\mathcal{O}(\log k)$ where k is the memory constraint, while maintaining an optimal area error of $\mathcal{O}\left(1/k^2\right)$.

1 Introduction

A streaming algorithm is an on-line approximation algorithm constrained to work within a memory budget. When more memory than the allowed budget is demanded, we must make decisions on what is worth keeping and what must be discarded. A streaming algorithm has three parts: an initialization procedure, a processing algorithm for each successive input, and a facility for answering queries using the restricted memory.

2 Related Work

Preparata gave an exact online algorithm [5] but with no memory constraints. The streaming algorithm proposed by Hershberger and Suri [1, 3, 2] maintains extreme points in k uniformly spaced directions and another k extreme points in adaptively sampled directions. Their algorithm has a distance error of $\mathcal{O}\left(1/k^2\right)$; no area measure was reported.

Lopez and Reizner [4] proposed an algorithm for approximating an n-gon by a k-gon, $k < n$. Their algorithm builds an inscribed k-gon by repeatedly removing an ear of minimum area until only k vertices remain. (An *ear* of a convex polygon is any triangle formed by three consecutive vertices.) However their algorithm, unlike ours, is not on-line, as all the vertices of the n-gon are known ahead of time.

3 Streaming Algorithm

Let $C = (p_1, p_2, ..., p_n)$ be a sequence of vertices of a convex polygon in counter-clockwise order. Each contiguous triple (p, q, r) in C defines a measure $\Delta_q = \text{GOODNESS}(p, q, r)$, which is associated with the vertex q. We will think of Δ_q as measuring the *goodness* of q. Note that Δ_q is a local measure and depends only on q and its two immediate neighbors in C. When a direct neighbor is inserted or deleted, the goodness must be recomputed. The function GOODNESS can be defined in various ways: as the area of the triangle $\triangle pqr$, as its perimeter, as the length of the segment pr, as the height of the triangle pqr relative to base pr, or even as the angle $\angle q$ in $\triangle pqr$. This yields different variants of the same algorithm. In this section, we shall mainly address the area variant.

3.1 INITIALIZE

The procedure INITIALIZE in Algorithm 1 initializes a balanced binary search tree T and a priority queue H to store the NODE references using two different keys. While points in T are ordered by their polar angles relative to a centroid, the points in H are keyed on their goodness.

Algorithm 1: INITIALIZE(P)

Input : P: The first 3 input points in a data stream S.
Output: T: balanced BST with vertices of conv(P) sorted by angles about centroid c;
H: min-heap of vertices of $conv(P)$ using GOODNESS as priority.

```
1  L ← conv(P)
2  c ← CENTROID(L)
3  (N, W, S, E) ← DIRECTIONALEXTREMA(L)
4  foreach p ∈ L do
5  |  Θ ← POLAR(p, c)
6  |  if p ∈ (N, W, S, E) then
7  |  |  Δ ← ∞
8  |  else
9  |  |  Δ ← GOODNESS(L.PRED(p), p, L.SUCC(p))
10 |  x ← NODE(p, Δ_p, Θ_p, false)
11 |  T.INSERT(Θ_p, x)
12 |  H.INSERT(Δ_p, x)
13 return (T, H, c, k)
```

The structure L in Step 1 is a cyclic array and supports PRED and SUCC operations. The function NODE(p, Δ_p, Θ_p, DELETED) creates a new node

*Department of Computer Science, George Mason University, rrufai@gmu.edu
†SAP Labs, Inc., 111 Rue Duke, Montreal, QC H3C 2M1, Canada, raimi.rufai@sap.com
‡Department of Computer Science, George Mason University, richards@gmu.edu

(a 4-tuple), whose attributes can be accessed using the attribute names POINT, GOODNESS, POLAR, and DELETED. Clearly initialization takes constant time.

3.2 PROCESS

Algorithm 2: PROCESS(T, H, c, k, p)

Input : T: balanced BST with $\leq k$ nodes;
H: min-heap of the nodes of T;
p: new point; k: memory budget
Output: T: a balanced BST updated with p,
H: a min-heap updated with p.

```
1 x ← NODE(p, 0, POLAR(p, c), false)
2 (T, H) ← UPDATEHULL(T, H, c, x)
3 if |T| > k then
4 |   (T, H) ← SHRINKHULL(T, H)
5 return (T, H)
```

Procedure PROCESS is invoked each time a new point arrives. A new node x is created and used to update the current hull by invoking procedure UPDATEHULL. The call to UPDATEHULL(T, H, c, x) in line 2 of Procedure PROCESS updates the structures T and H with x. If the point associated with x falls within the interior of the current convex hull or on its boundary, it is discarded. This test is done in Line 4 of UPDATEHULL. Further, x's goodness is computed and if it is smaller than H.MINIMUM, again x is discarded. Otherwise, the chain of vertices that lie between the two new neighbors of x on the hull are deleted from both T and H. This deletion is done in Lines 8 to 16 of UPDATEHULL. The goodness of x's neighbors are then updated. The directional extrema are also updated if required.

Whenever the number of nodes in T exceeds k, the procedure SHRINKHULL is called to choose one vertex for deletion. This is done by calling DELETEMIN() on the min-heap structure H to obtain the node q that should be deleted. The procedure then updates the GOODNESSes of q's neighbors and deletes q from T.

This algorithm is sensitive to the order in which the points arrive in the stream. Consider the six points A, B, C, D, E, F shown in Figure 1 and Figure 2 below.

3.3 QUERY

Algorithm QUERY is invoked to obtain the current hull at any point in the streaming process. It simply traverses T to return the hull vertices in a cyclic list, and runs in linear time.

3.4 Complexity Analysis

Lemma 1 *Procedure* UPDATEHULL *runs in* $\mathcal{O}(\log k)$ *amortized time on the input stream* S.

Algorithm 3: UPDATEHULL(T, H, c, x)

Input : T: balanced BST with $\leq k$ of conv(S);
H: min-heap of $\leq k$ nodes;
x: new node.
Output: T: balanced BST updated with x;
H: min-heap updated with x.

```
1  T.INSERT(x)
2  y ← T.PRED(x)
3  z ← T.SUCC(x)
4  if not CONTAINS(△ycz, x) then
5  |   (s, t) ← TANGENTS(T, x)
6  |   x.Δ ← GOODNESS(s, x, t)
7  |   if x.Δ ≥ H.MINIMUM() then
8  |   |   w ← T.SUCC(s)
9  |   |   while w ≠ t do
10 |   |   |   w.DELETED ← true
11 |   |   |   H.CHANGEKEY(w, −∞)
12 |   |   |   T.DELETEKEY(w)
13 |   |   |   w ← T.SUCC(s)
14 |   |   q ← H.MINIMUM()
15 |   |   while q.DELETED do
16 |   |   |   q ← H.DELETEMIN()
17 |   |   H.INSERT(x)
18 |   |   H.CHANGEKEY(s, GOODNESS(T.PRED(s), s, x))
19 |   |   H.CHANGEKEY(t, GOODNESS(x, t, T.SUCC(t)))
   |   |     ▷ Update extrema if needed ◁
20 |   |   (N, W, S, E) ← UPDATEEXTREMA(T, c, x)
21 |   |   foreach n ∈ (N, W, S, E) do
   |   |   |   ▷ To prevent the deletion of an extremum ◁
22 |   |   |   H.CHANGEKEY(n, ∞)
23 |   else
24 |   |   T.DELETEKEY(x)
25 else
26 |   T.DELETEKEY(x)
27 return (T, H)
```

Algorithm 4: SHRINKHULL(T, H)

Input : T: BST with $k+1$ vertices of conv(S);
H: min-heap of $k+1$ vertices of conv(S).
Output: T: BST with k vertices of conv(S);
H: min-heap of k vertices of conv(S).

```
1 q ← H.DELETEMIN()
2 p ← T.PRED(q)
3 r ← T.SUCC(q)
4 T.DELETEKEY(q)
5 H.CHANGEKEY(p, GOODNESS(T.PRED(p), p, r))
6 H.CHANGEKEY(r, GOODNESS(p, r, T.SUCC(r)))
7 return (T, H)
```

Proof. The initial steps take $\mathcal{O}(\log k)$ time using standard BST techniques. Step 4 takes $\mathcal{O}(1)$ time. The

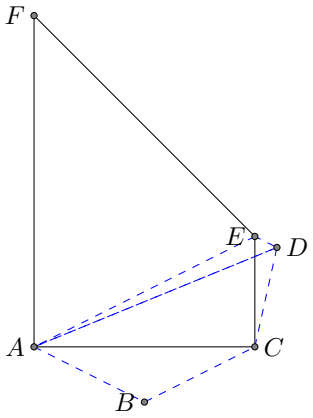


Figure 1: $k = 4$, arrival sequence: A, B, C, D, E, F. D is deleted after E arrives, and B after F.

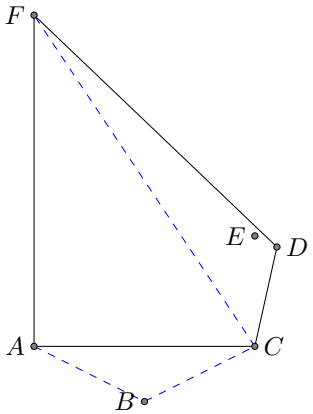


Figure 2: $k = 4$ with arrival sequence: A, B, C, D, F, E. B is deleted after F arrives. E is deleted since it is an interior point.

call to TANGENTS takes $\mathcal{O}(\log k)$ time [5]. The rest of the procedure — Steps 8 through 16 — deletes a vertex chain that no longer belongs to the hull. Since these vertices are only deleted once per point in S, the total cost over all invocations of the procedure UPDATEHULL is $\mathcal{O}(n \log k)$, where n is the length of S. □

Lemma 2 *Procedure* SHRINKHULL *runs in time* $\mathcal{O}(\log k)$.

Proof. Every step of Procedure SHRINKHULL takes $\mathcal{O}(\log k)$. □

Lemma 3 *Procedure* PROCESS *runs in time* $\mathcal{O}(\log k)$ *time.*

Proof. Each invocation of PROCESS makes a single call to UPDATEHULL and at most a single call to SHRINKHULL. Thus PROCESS also runs in $\mathcal{O}(\log k)$ time. □

Lemma 4 *Let* T_{i-1} *be the convex hull computed before invoking Algorithm* UPDATEHULL*, and let* T_i *be the resulting hull after it returns. Then the following invariant holds*

$$|T_{i-1}| \leq |T_i|. \tag{3.1}$$

Proof. Consider the invocation of UPDATEHULL on an arbitrary point p_i. The fate of p_i is one of the following two cases.

Case 1 (p_i lies in the interior of T_{i-1}.) UPDATEHULL *ignores* p_i*, in which case the hull does not grow and* $T_i = T_{i-1}$.

Case 2 (p_i lies in the exterior of T_{i-1}.) UPDATEHULL *expands* T_{i-1} *by adding* p_i *to the hull and therefore* T_i *has a bigger area than* T_{i-1}.

□

Lemma 5 *When* $k \geq |\operatorname{conv}(S)|$ *the algorithm computes the exact convex hull of* S.

Proof. The algorithm then is equivalent to that of Preparata [5]. □

3.5 Error Analysis

We only discuss in this section the relative area error, which is defined as

$$err_{\text{area}}(P, P') = \frac{|\text{area}(P) - \text{area}(P')|}{\text{area}(P)} \tag{3.2}$$

where P denotes the vertex set of the true convex hull, and P' that of the approximate convex hull.

Lemma 6 *Each deletion from a convex* $(k+1)$*-gon by Algorithm* SHRINKHULL *introduces an error no worse than* $\mathcal{O}(1/k^3)$.

Proof. Let $m = k+1$. Let Q be a convex m-gon and let $e_1, e_2, ..., e_m$ be its ears. Let $|e_i|$ denote the area of e_i. Let $Q'_i = Q - e_i$ denote the k-gon that would result if e_i were deleted. Therefore, the ratio $|e_i|/|Q|$ represents the area error that would result from deleting e_i. Further, let R_m denote a regular m-gon with unit area, and let R be the circumradius of R_m.

Renyi and Sulanke [6] proved that

$$\frac{1}{|Q|^m} \prod_{i=1}^{m} |e_i| \leq |r|^m, \tag{3.3}$$

whenever r is an ear of R_m.

By taking logarithms and invoking the mean-value theorem, it is clear that there must exist at least one ear e_j in Q such that $\frac{|e_j|}{|Q|} \leq |r|$. Since

$$|r| = 4R^2 \frac{\pi^3}{m^3}\left[1 - \frac{\pi^2}{m^2} + \mathcal{O}\left(\frac{1}{m^4}\right)\right], \tag{3.4}$$

it follows that

$$\frac{|e_j|}{|Q|} < 4R^2 \frac{\pi^3}{m^3} \tag{3.5}$$

$$= \mathcal{O}\left(\frac{1}{k^3}\right). \tag{3.6}$$

□

Lemma 7 *Let $e_1, e_2, ..., e_m$ denote the sequence of ears deleted by the streaming algorithm. Then*

$$|e_i| \leq |e_{i+1}| < H.\,\text{MINIMUM} \text{ for all } i = 1, 2, ..., m-1. \tag{3.7}$$

Proof. Recall that Algorithm UPDATEHULL only inserts a new node if its goodness is greater than $H.\,\text{MINIMUM}$. By definition, $H.\,\text{MINIMUM}$ increases with each deletion. Before the the ith deletion, $H.\,\text{MINIMUM} = |e_i|$, but becomes $|e_{i+1}|$ afterwards. □

Note that the computed hull consists of four (x-y monotone) chains: from W to N, from N to E, from E to S, and from S to W. Our discussion will only be for the chain from W to N. Suppose that chain is s_1, s_2, ... s_l. Let s_0 be a short vertical side below W and s_{l+1} be a short horizontal side to the right of N. (The reason for these two additional sides is to automatically take into account the fact that all points seen will be in a bounding box, as indicated by the next lemma. If we do not maintain the bounding box, then our chains are not monotone and the definitions below would be more complex.) Let p_i be the vertex common to s_i and s_{i+1}.

Lemma 8 *After processing the points in S, the directional extrema (N, W, S, E), maintained by Algorithm* UPDATEHULL *define an axis-parallel bounding box B that contains* conv(S).

Proof. Note that these directional extrema are extreme over all of S in the four axis-parallel directions. Suppose there were some point p in S not contained in B. Further suppose, without loss of generality, that p lies above B. Then p must be more extreme than N in the positive y direction, a contradiction. □

The *outer ear* for side s_i is the triangle formed by s_i and the extensions of the sides s_{i-1} and s_{i+1}. The *flap* for side s_i is a trapezoidal subset of its outer ear: it is the region $\square p_i p_{i+1} q_1 q_2$ where $\overline{q_1 q_2}$ is parallel to s_i and q_1 and q_2 are on the boundary of the outer ear. The height of the flap, h_i, perpendicular to s_i will be chosen to be the minimum value that maintains an invariant. The h_i is used in the analysis and is not calculated by the algorithm.

We will choose h_i after each deletion that creates the side s_i so that this invariant holds (if s_i was not created by a deletion, then $h_i = 0$).

Invariant 1 *Each deleted point, not in the hull itself, is from one of the flaps. Further, the area of the corresponding ear is contained in the flap.*

When p_i is deleted and a new side $s = \overline{p_{i-1}p_{i+1}}$ created, the corresponding h is calculated: it is minimized subject to the constraint that the new flap includes the flaps from s_{i-1} and s_i. Let h' be the height of p_i in $\triangle p_{i-1} p_i p_{i+1}$. Then $h \leq 2h'$, by similar triangles. Recall that h corresponds to a triangle (ear) chosen because it had minimum area. Hence we get the following lemma.

Lemma 9 *The area of any flap is $\leq 4H.\,\text{MINIMUM}()$.*

Proof. Suppose s_i is the side for a given flap. Let $a = H.\,\text{MINIMUM}()$. The height h satisfies

$$h \leq 2h' \leq 2\frac{2a}{s_i}. \tag{3.8}$$

Since the top of the trapezoid is less than its base, it fits within a parallelogram M of base s_i and height h. □

The following theorem gives an upper bound on the area error for processing $n \gg k$ points.

Theorem 10 *The total area error incurred in the streaming process is bounded above by $\mathcal{O}\left(1/k^2\right)$.*

Proof. A deleted point contributes to the error if it is outside the computed hull. Some or all of its ears may not be in the computed answer. We know that each such ear is from some flap. A single flap may cover many such (overlapping) ears, but the total missed area of all such ears is bounded by the area of that flap. Hence the total area error is bounded by the total area of all the flaps.

By Lemma 6, $H.\,\text{MINIMUM}$ is at most $\mathcal{O}\left(1/k^3\right)$ and since there are k outer ears, the total error is $\mathcal{O}\left(1/k^2\right)$. □

Note that, in general, not all deletions will have an impact on the final k-gon returned after processing all the points in the stream. However, when an adversary could provide a stream of points that all lie on the convex hull, such as the vertices of a regular n-gon, the above error bound, being a worst-case bound, will still apply.

Theorem 11 *(Lopez and Reisner [4]) Given an adversarial input, the total area error accumulated by all the deletions is at least*

$$2\pi^2\left[\frac{1}{k^2}-\frac{1}{n^2}\right]. \tag{3.9}$$

Proof. This bound was obtained by [4], but in their case, they had access to all the vertices offline, as mentioned earlier in Section 2. □

4 Empirical Results

A stream S of 10,000 random points lying on a common circle was generated. We then fed 33 random shuffles of S to the streaming algorithm and computed the mean distance and area relative errors. These were then used to compute the lower and upper bounds as defined in Theorem 10 and Theorem 11. The empirical area error is neatly sandwiched between the two bounds, as expected.

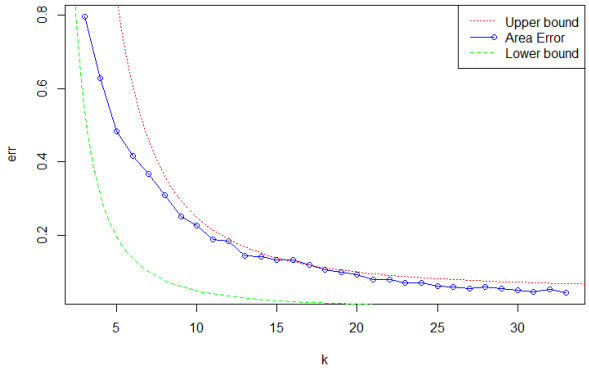


Figure 3: Empirical area error sandwiched between the curves of the lower and upper bounds

The relative distance measure between the set P of vertices of the true convex hull and the set P' of vertices of the approximate hull is defined as

$$err_{\delta,\mathrm{diam}}(P,P') = \delta(P,P')/\operatorname{diam}(P), \tag{4.1}$$

where $\delta(\cdot,\cdot)$ stands for the Hausdorff distance[1].

Figure 4 and Figure 5 show the distance and area relative errors using three goodness measures: the area of an ear, the height of the ear, and the angle made by the ear with the centroid. What is clear from these results is that the measure of goodness based on the area and that based on the distance (height of the ears) are both very effective. The results for the angle of an ear were not as good, indicating that the relation between the measure of goodness and the error measure is important.

[1]The Hausdorff distance between a finite point set P and another Q is defined as $\delta(P,Q) = \max(\max_{p\in P}\min_{q\in Q}\|p-q\|, \max_{q\in Q}\min_{p\in P}\|q-p\|)$.

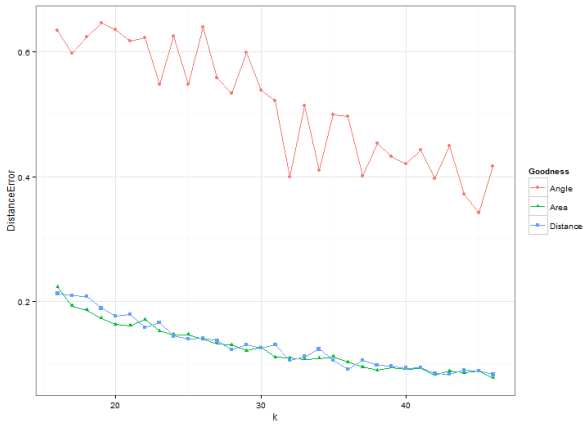


Figure 4: Distance Relative Errors

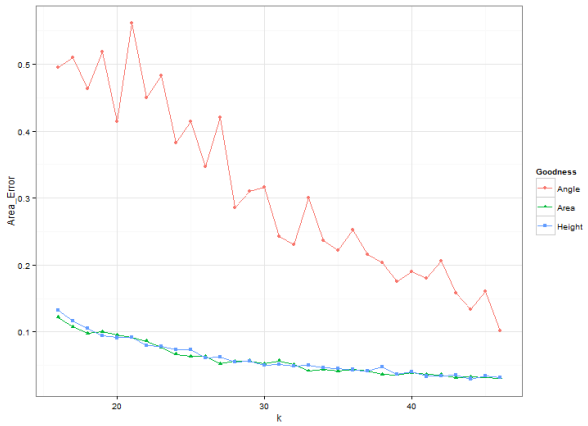


Figure 5: Area Relative Errors

5 A More General Approach

We propose a refinement of Algorithm 2, which uses the idea from Lopez and Reisner [4]. The essential difference is that rather than invoke ShrinkHull every time the k-gon grows into a $(k+1)$-gon, the algorithm waits until the k-gon grows into an mk-gon for some small constant m before invoking ShrinkHull. The algorithm Process2 shows the details. This only works, of course, if the memory constraint allows the use of $(m-1)k$ extra memory for processing. The main benefit of this enhancement is that it reduces the effect of the order of the point sequence (illustrated earlier in

Figure 1 and Figure 2), while keeping the same overall asymptotic time bounds. We hope to analyze this approach both analytically and empirically.

Algorithm 5: PROCESS2(T, H, c, k, p)

Input : T: balanced BST with $\leq k$ of conv(S);
H: min-heap of $\leq k$ of conv(S);
p: new point; k: memory budget
Output: T: a height-balanced BST update with p if on the hull, H: a binary min-heap updated with p if on the hull.

```
x ← NODE(p, 0, POLAR(p, c), false)
(T, H) ← UPDATEHULL(T, H, c, x)
if |T| > mk then
    while |T| > k do
        (T, H) ← SHRINKHULL(T, H)
return (T, H)
```

6 Conclusion

We have presented a new streaming algorithm for the convex hull and analyzed its runtime and error bounds. We have proven that it is optimal for the area error measure. We have empirically shown that it is robust with respect to different goodness and error measures. Further analytic results are being studied.

The generalization of this approach to three or higher dimensions is conceptually straightforward. Each new point that is outside the current hull subtends a volume analogous to an ear, which can be given a goodness measure. Points which can be stored in memory are deleted according to this measure. However, we have not explored the computational complexity of these steps.

Acknowledgement

We thank the anonymous referees for their feedback and comments, which have helped us improve the readability of this paper.

References

[1] J. Hershberger and S. Suri. Convex hulls and related problems in data streams. In *Proc. of the ACM/DIMACS Workshop on Management and Processing of Data Streams*, 2003.

[2] J. Hershberger and S. Suri. Adaptive sampling for geometric problems over data streams. *Computational Geometry*, 39(3):191–208, 2008.

[3] J. Hershberger and S. Suri. Simplified planar coresets for data streams. In *Algorithm Theory—SWAT 2008*, pages 5–16. Springer, 2008.

[4] M. A. Lopez and S. Reisner. Efficient approximation of convex polygons. *International Journal of Computational Geometry & Applications*, 10(05):445–452, 2000.

[5] F. Preparata. An optimal real-time algorithm for planar convex hulls. *Communications of the ACM*, 22:402–405, 1979.

[6] A. Rényi and R. Sulanke. Über die konvexe Hülle von n zufällig gewählten Punkten. *Probability Theory and Related Fields*, 2:75–84, 1963. 10.1007/BF00535300.

Approximation and Streaming Algorithms for Projective Clustering via Random Projections

Michael Kerber* Sharath Raghvendra†

Abstract

Let P be a set of n points in $\mathbb{R}^d$. In the projective clustering problem, given k, q and norm $\rho \in [1, \infty]$, we have to compute a set $\mathcal{F}$ of k q-dimensional flats such that $(\sum_{p\in P} \mathtt{d}(p, \mathcal{F})^\rho)^{1/\rho}$ is minimized; here $\mathtt{d}(p, \mathcal{F})$ represents the (Euclidean) distance of p to the closest flat in $\mathcal{F}$. We let $f_k^q(P, \rho)$ denote the minimal value and interpret $f_k^q(P, \infty)$ to be $\max_{r\in P} \mathtt{d}(r, \mathcal{F})$. When $\rho = 1, 2$ and ∞ and $q = 0$, the problem corresponds to the k-median, k-mean and the k-center clustering problems respectively.

For every $0 < \varepsilon < 1$, $S \subset P$ and $\rho \geq 1$, we show that the orthogonal projection of P onto a randomly chosen flat of dimension $O(((q + 1)^2 \log(1/\varepsilon)/\varepsilon^3) \log n)$ will ε-approximate $f_1^q(S, \rho)$. This result combines the concepts of geometric coresets and subspace embeddings based on the Johnson-Lindenstrauss Lemma. As a consequence, an orthogonal projection of P to an $O(((q + 1)^2 \log((q + 1)/\varepsilon)/\varepsilon^3) \log n)$ dimensional randomly chosen subspace ε-approximates projective clusterings for every k and ρ simultaneously. Note that the dimension of this subspace is independent of the number of clusters k.

Using this dimension reduction result, we obtain new approximation and streaming algorithms for projective clustering problems. For example, given a stream of n points, we show how to compute an ε-approximate projective clustering for every k and ρ simultaneously using only $O((n + d)((q + 1)^2 \log((q + 1)/\varepsilon))/\varepsilon^3 \log n)$ space. Compared to standard streaming algorithms with $\Omega(kd)$ space requirement, our approach is a significant improvement when the number of input points and their dimensions are of the same order of magnitude.

1 Introduction

Consider the *projective clustering problem*: For a set P of n points in $\mathbb{R}^d$, given integers $k, q < n$ and an integer norm $\rho \geq 1$, compute a set $\mathcal{F}$ of k q-dimensional flats (or *q-flats*) such that $(\sum_{p\in P} \mathtt{d}(p, \mathcal{F})^\rho)^{1/\rho}$ is minimized; here $\mathtt{d}(p, \mathcal{F})$ represents the Euclidean distance of p to its closest point on any flat in $\mathcal{F}$. We define

$$f_k^q(P, \rho) := \min_{\mathcal{F}} (\sum_{p\in P} \mathtt{d}(p, \mathcal{F})^\rho)^{1/\rho}$$

and interpret $f_k^q(P, \infty)$ to be $\min_{\mathcal{F}} \max_{p\in P} \mathtt{d}(p, \mathcal{F})$. The projective clustering problem is a generalization of several well-known problems. For example, when $\rho = \infty$, $q = 0$ this problem is the *minimum enclosing ball* (MEB) problem (when $k = 1$) and the *k-center clustering problem* (for arbitrary k). When $\rho = \infty$ and $q = 1$, we get the *minimum enclosing cylinder* (MEC) (for $k = 1$) and the *k-cylinder clustering problem* (for arbitrary k). When $q = 0$, we get the k-median clustering problem (for $\rho = 1$) and the k-means clustering problem (for $\rho = 2$). The projective clustering problem is NP-Hard [5] and, therefore, most research has focused on the design of approximation algorithms. For an error parameter $0 < \varepsilon < 1$, an *ε-approximate projective clustering* is a set of q-flats $\tilde{\mathcal{F}}$ such that $(\sum_{p\in P} \mathtt{d}(p, \tilde{\mathcal{F}})^\rho)^{1/\rho} \leq (1 + \varepsilon) f_k^q(P, \rho)$.

Projective clustering is an important task arising in unsupervised learning, data mining, computer vision and bioinformatics; see [31] for a survey of some of these applications. Given its significance, clustering problems have received much attention leading to new approximation algorithms. The early algorithms for these problems had exponential dependence on d [2, 4] and were well-suited for low-dimensional inputs. However, for many practical problems, the number of input points n and their dimension d are in the same order of magnitude [21].

Badoiu, Indyk and Har-Peled [8] made a breakthrough in the design of high-dimensional clustering algorithms. They designed a *coreset*-based algorithm that quickly constructs a small "most-relevant" subset E of the input points P with the property that an optimal clustering on E is an approximate clustering for P, and use this coreset to compute an approximate clustering. Based on this idea, several coreset-based approximation algorithms for projective clustering were developed, also for the design of *streaming* algorithms for projective clustering[1]; see for example [11, 20, 22]. In recent research, depending on the problem, different definitions of coreset have been used. These definitions vary from weak to strong notions of when a subset is relevant, and therefore yield different size bounds (see for instance [21] for a careful discussion).

Throughout this paper, we use the following definition: a coreset (with respect to ε, q, ρ) is a subset $E \subseteq P$ such that the affine subspace spanned by E contains a

*Max Planck Institute for Informatics, Saarbrücken, Germany, mkerber@mpi-inf.mpg.de

†Virginia Tech, Blacksburg, USA, sharathr@vt.edu

[1] In the streaming setting, algorithms are allowed to make one or few passes over the data and compute an approximate solution using a small workspace.

q-flat F with $(\sum_{p\in P} \mathsf{d}(p,F)^\rho)^{1/\rho} \leq (1+\varepsilon) f_1^q(P,\rho)$. We let $C_\rho(q,\varepsilon)$ denote the worst-case size of such a coreset for approximating $f_1^q(P,\rho)$. This is a comparably weak version of coresets: we only require that the subspace spanned by E contains some ε-approximate solution; we do *not* require that the optimal solution for E is that ε-approximation. For problems such as MEB, MEC, 1-mean, or 1-median, there are coresets whose size is independent of the number of points and the ambient dimension [7, 8, 23, 35].

Another useful tool for the design of high-dimensional clustering algorithms is the *random projection* method [36]. At its heart is the following well-known lemma [26] which says that an orthogonal projection of any point set to a random $O(\log n/\varepsilon^2)$-dimensional flat ε-approximates pairwise distance between all pairs of points; see [16] for an elementary proof.

Theorem 1 (Johnson-Lindenstrauss) *For $0 < \varepsilon < 1$, a set $P \subset \mathbb{R}^d$ of n points, and $m \geq 36\ln(n)/\varepsilon^2$, there is a map $\hat{\pi} : \mathbb{R}^d \to \mathbb{R}^m$ such that*

$$(1-\varepsilon)\|u-v\|^2 \leq \|\hat{\pi}(u)-\hat{\pi}(v)\|^2 \leq (1+\varepsilon)\|u-v\|^2$$

for any $u, v \in P$. Moreover, a randomly chosen map $\hat{\pi}$ of the form $\hat{\pi}(p) = \sqrt{d/m}\cdot\pi(p)$ where π is the orthogonal projection to a m-dimensional subspace of $\mathbb{R}^d$, satisfies that property with probability at least $1/2$.

We abuse notations and refer to $\hat{\pi}$ as above as a *random projection* to an m-dimensional flat.

The Johnson-Lindenstrauss Lemma shows that random projections approximate pairwise distances between points. A natural question is what other geometric and structural properties of high-dimensional point cloud are preserved by random projections, and numerous such properties have been identified [1, 12, 24, 28, 32]. Random projection techniques are widely used for clustering problems: ongoing research focuses in particular to the case of k-means clustering [9, 14, 15], although it has also been used for certain projective clustering problems [8, 30].

Our results. We establish a link between coresets and the random projections for the projective clustering problem in Section 2. We show that, for every $0 < \varepsilon < 1$, $q \geq 0$, and $\rho \geq 1$, a random projection to a $O(((q+1)^2\log((q+1)/\varepsilon)/\varepsilon^3\log n)$-dimensional space ε-approximates $f_1^q(S,\rho)$ for all $S \subseteq P$. The main ingredient of our proof is to show that a random projection to an $O(C_\rho(q,\varepsilon)\log n/\varepsilon^2)$-dimensional subspace "preserves" all flats defined by subsets of size $C_\rho(q,\varepsilon)$.Our argument follows the standard proof technique for subspace embeddings (as sketched in [14, 29]) by approximately preserving the lengths of vectors taken from a sufficiently dense ε-net. For a given q and any $\rho \geq 1$, the existence of small-sized coresets with $C_\rho(q,\varepsilon) = O((q+1)^2/\varepsilon\log((q+1)/\varepsilon))$ is known [35]. This leads to the previously mentioned bound on the dimension of the projected space.

As a consequence, we show that by projecting to the same dimension, also $f_k^q(P,\rho)$ is preserved for all k and $\rho \geq 1$. Note that the dimension of the subspace is independent of k and ρ and is only logarithmic in n. Our results imply that improved bounds on the size of the coreset $C_\rho(q,\varepsilon)$ lead to better bounds on the dimension of the random subspace. Interestingly, unlike previous applications of coresets, we do not require a fast method to compute $C_\rho(q,\varepsilon)$. Therefore, we can shoot for even smaller-sized coresets without being restricted by its computation time (Section 3).

Our results has the following applications (Section 4): For a given q and a stream of n points, we give an algorithm that can compute projective clustering of P for every value of k and ρ using only $O(((q+1)^2\log((q+1)/\varepsilon)/\varepsilon^3)(n+d)\log n)$ space. Almost all known (multi-pass) streaming algorithms for projective clustering problems have a linear dependence on the product of k and d, and therefore, they tend to require $\Omega(nd)$ space for when $k = \Theta(n)$. As opposed to this, our algorithm requires $\tilde{O}(n+d)$ space which is particularly useful when n and d are of the same order of magnitude. Also, in many practical scenarios, the number of clusters k and the norm ρ are not known in advance. Our algorithm is also useful in such cases since our dimension reduction technique works for all values of k and ρ simultaneously.

We also generically improve approximation algorithms for projective clustering problems. Again, we project P onto a randomly chosen subspace and compute an approximate solution in the projected subspace. We obtain a solution in the original d-dimensional space by "lifting" each cluster from the projected space separately. For the approximate k-cylinder problem, our approach yields a bound of $O(n\log n 2^{k\log k/\varepsilon} + \frac{dn\log n}{\varepsilon^3})$ which improves the previously known best $O(nd2^{k\log k/\varepsilon})$ [7]; note that k and d are decoupled in our complexity bound.

Finally, since our results imply that, under random projections, the radius of MEB is approximated for every subset of the input, we immediately get an approximation scheme for a d-dimensional Čech complex in Euclidean space by a Čech complex in lower dimensions. In particular, this result bounds the persistence of high-dimensional homology classes of the original Čech complex. Recently, these results have been proven independently by Sheehy [34].

2 Generalized Johnson-Lindenstrauss Lemma

Recall the definition of $f_1^q(P,\rho)$ as the L_ρ-distance of P to the best fitting q-flat. We show that a random projection to appropriately large subspaces approximately preserves $f_1^q(S,\rho)$ for any subset $S \subseteq P$. What dimension is appropriate for a projection depends on the cor-

responding coreset size $C := C_\rho(q, \varepsilon)$; precisely, picking a $O(C \log(n)/\varepsilon^2)$-dimensional subspace is enough.

We outline the proof of the statement before giving the technical details in the remainder of the section. For a set $S \subset P$, we let $\langle S \rangle$ denote the *span* of S, that is, the subspace spanned by the points in S. We know that any subset of P has a coreset of size C whose span contains an approximately optimal q-flat F. If the distance of F to any $p \in P$ is preserved under the projection, we can guarantee to preserve $f_1^q(S, \rho)$ approximately as well. We ensure this preservation by the stronger property in Lemma 3 that for any $p \in P$, the distance to *any* q-flat in the span of *any* subset of P of cardinality C is preserved. Note that the number of such subspaces is bounded by n^C and therefore polynomial in n.

Lemma 3 in turn follows easily from a generalization of the Johnson-Lindenstrauss lemma that we prove first: for an integer $c > 0$, we show that a random projection to roughly $c \log(n)/\varepsilon^2$ dimensions preserves for *all* subset S of c points the distance between any two points in $\langle S \rangle$. While the proof of this subspace embedding result has been outlined in previous work [14, 29], we are not aware of a formal proof of the statement.

Lemma 2 *For $0 < \varepsilon < 1$, a set $P \subset \mathbb{R}^d$ of n points, an integer $c \geq 0$, and $m \geq \lambda \cdot c \log(n)/\varepsilon^2$ for a suitable constant λ, a random projection $\hat{\pi}$ satisfies with high probability that for any subset $S \subset P$ of cardinality c and for any $u, v \in \langle S \rangle$*

$$(1-\varepsilon)\|u - v\| \leq \|\hat{\pi}(u) - \hat{\pi}(v)\| \leq (1+\varepsilon)\|u - v\|.$$

Proof. The proof of Theorem 2.1 in Dasgupta and Gupta [16] implies the following statement: When projecting a unit vector in $\mathbb{R}^d$ to a fixed $m = O(c \log n/\varepsilon^2)$-dimensional subspace, the probability that its squared length does not lie in $((1-\varepsilon)m/d, (1+\varepsilon)m/d)$ is at most

$$2\exp(-\frac{m\varepsilon^2}{4}) \leq 2\exp(-\frac{\lambda c \log n}{4}) \leq n^{-8c}$$

for a suitable constant λ. As they argue, the same bound applies for a fixed unit vector and a uniformly chosen m-dimensional subspace.

A result by Feige and Ofek [19] (see also [6]), translated in geometric terms, says that by approximately preserving the pairwise squared distances between a set of at most $\exp(c \ln 18)$ sample points belonging to an c-dimensional subspace, we can approximately preserve the squared length of all unit vectors in the subspace, and thus all pairwise distances; see [32, Proof of Cor. 11] for further explanations. Hence, for a fixed subspace, we need to preserve $\exp(2c \ln 18) \leq \exp(6c)$ distances. Moreover, we want to preserve distances in n^c many subspaces, yielding a total of $\exp(6c)n^c \leq n^{7c}$ distances to be preserved. By the union bound, choosing a m-dimensional subspace uniformly at random, the probability of success is at least $1 - \frac{n^{7c}}{n^{8c}} \geq 1 - 1/n^c$. □

The preservation of point-to-flat distances in low-dimensional subspaces is a simple consequence:

Lemma 3 *Let $0 < \varepsilon < 1$, $P \subset \mathbb{R}^d$ a set of n points and $q < c$ positive integers. With high probability, a random projection to an $O(c \log n/\varepsilon^2)$-dimensional flat satisfies for all subsets $S \subset P$ of cardinality c, all q-flats $Q \subset \langle S \rangle$, and all $p \in P$ that*

$$(1-\varepsilon)d(p, Q) \leq d(\hat{\pi}(p), \hat{\pi}(Q)) \leq (1+\varepsilon)d(p, Q).$$

Proof. For any $p \in P$ and any $Q \subset \langle S \rangle$, there exists a space with $c+1$ points that contains both p and Q. Let $t \in Q$ be the point such that $d(p, Q) = \|p - t\|$. Applying Theorem 2 for $c' := c+1$ immediately implies that $d(\hat{\pi}(p), \hat{\pi}(Q)) \leq \|\hat{\pi}(p) - \hat{\pi}(t)\| \leq (1+\varepsilon)d(p, Q)$. The second inequality follows similarly, considering the point $t' \in Q$ that realizes $d(\hat{\pi}(p), \hat{\pi}(Q))$. □

We show our main theorem that random projections preserve $f_1^q(S, \rho)$ for any $S \subseteq P$.

Theorem 4 *Let $0 < \varepsilon < 1$, $P \subset \mathbb{R}^d$ consist of n points, $q \geq 0$ an integer and $\rho \in \mathbb{Z}_{\geq 1} \cup \{\infty\}$. Then with high probability, for $m \geq \lambda \cdot C_\rho(q, \varepsilon/2) \log(n)/\varepsilon^2$ with a suitable constant λ, a random projection $\hat{\pi}$ satisfies for all subsets $S \subseteq P$*

$$(1-\varepsilon)f_1^q(S, \rho) \leq f_1^q(\hat{\pi}(S), \rho) \leq (1+\varepsilon)f_1^q(S, \rho).$$

Proof. Let $S \subseteq P$ arbitrary. We start by showing the second inequality: By the coreset property, there exists a subset $E \subset S$ of $C_\rho(q, \varepsilon/2)$ points such that $\langle E \rangle$ contains a q-flat F that is an $\frac{\varepsilon}{2}$-approximate solution. For $\rho \neq \infty$, applying Lemma 3 with $\varepsilon' = \varepsilon/3$, we get that

$$\begin{aligned} f_1^q(\hat{\pi}(S), \rho) &\leq \left(\sum_{p \in S} d(\hat{\pi}(p), \hat{\pi}(F))^\rho \right)^{1/\rho} \\ &\leq \left(\sum_{p \in S} (1+\varepsilon/3)^\rho d(p, F)^\rho \right)^{1/\rho} \\ &\leq (1+\varepsilon/3)(1+\varepsilon/2) f_1^q(S, \rho) \leq (1+\varepsilon) f_1^q(S, \rho), \end{aligned}$$

where we use $(1+\varepsilon/3)(1+\varepsilon/2) < 1+\varepsilon$ for $0 \leq \varepsilon \leq 1$. For $\rho = \infty$, the proof for $\rho = 1$ directly carries over.

For the first inequality, we apply the coreset property on the set $\hat{\pi}(S)$: let $\hat{\pi}(E')$ be a coreset for $\hat{\pi}(S)$. Let G denote the approximate solution in $\langle \hat{\pi}(E') \rangle$; it holds that $G = \hat{\pi}(F')$ for some q-flat F' in $\langle E' \rangle$. Using again Lemma 3, we have that

$$\begin{aligned} (1-\varepsilon) f_1^q(S, \rho) &\leq (1-\frac{\varepsilon}{2})(1-\frac{\varepsilon}{3}) \left(\sum_{p \in S} d(p, F')^\rho \right)^{1/\rho} \\ &\leq (1-\frac{\varepsilon}{2}) \left(\sum_{p \in S} d(\hat{\pi}(p), G)^\rho \right)^{1/\rho} \\ &\leq (1-\frac{\varepsilon}{2})(1+\frac{\varepsilon}{2}) f_1^q(\hat{\pi}(S), \rho) \leq f_1^q(\hat{\pi}(S), \rho). \end{aligned}$$

Again, the case $\rho = \infty$ is analogue to $\rho = 1$. □

Theorem 4 implies that $f_k^q(P, \rho)$ is preserved for any $k \geq 1$.

Corollary 5 *With the notations of Theorem 4 and $k \geq 1$, a random projection $\hat{\pi}$ satisfies with high probability*

$$(1 - \varepsilon) f_k^q(P, \rho) \leq f_k^q(\hat{\pi}(P), \rho) \leq (1 + \varepsilon) f_k^q(P, \rho).$$

Proof. Let $\mathcal{F} = \{F_1, \ldots, F_k\}$ denote an optimal collection of q-flats, that is, for any $p \in P$, the closest flat in $\mathcal{F}$ has distance at most $f_k^q(P, \rho)$. Let $P_i \subseteq P$ be the set of points closest to F_i, for $i = 1, \ldots, k$. Note that F_i is the optimal q-flat for P_i, in other words, it realizes $f_1^q(P_i, \rho)$.[2] Using Theorem 4 on the subsets P_i, we get for $\rho < \infty$ that

$$f_k^q(\hat{\pi}(P), \rho) \leq \sum_{i=0}^{k} f_1^q(\hat{\pi}(P_i), \rho)$$

$$\leq \sum_{i=0}^{k} (1 + \varepsilon) f_1^q(P_i, \rho) = (1 + \varepsilon) f_k^q(P, \rho),$$

proving the second inequality. The first part follows the same way considering an optimal $\mathcal{F}$ for $\hat{\pi}(P)$. The case $\rho = \infty$ is analogous, replacing all sums by max. □

3 Coresets for Projective Clustering

Recall that $C_\rho(q, \varepsilon)$ is defined as the coreset size for approximating the L_ρ-optimal q-flat, in the sense that there exists a subset of $C_\rho(q, \varepsilon)$ input points whose span contains an ε-approximate optimal q-flat. Because of space restrictions, we omit the (simple) proofs of the results in this section (see Appendix A).

The case of 0-flats For a point set $P \subset \mathbb{R}^d$, we consider the point that minimizes, for a fixed $\rho \in [1, \infty]$,

$$\delta(q) := \left(\sum_{p \in P} d(p, q)^\rho \right)^{1/\rho}$$

over all $q \in \mathbb{R}^d$. We call the minimizer o in $\mathbb{R}^d$ the *optimal center* and note that $\delta(o) = f_1^0(P, \rho)$. We call o' an *ε-approximate center*, if $\delta(o') \leq (1 + \varepsilon)\delta(o)$. Since Theorem 4 only requires a bound on the coreset and no method to compute it, we can free ourselves from algorithmic considerations and concentrate on existential results.

A lower bound of $\Omega(1/\varepsilon)$ can be derived easily by considering the standard simplex. This has been done by Bădoiu and Clarkson [10] for the case $\rho = \infty$.

[2] For $\rho = \infty$, this is not necessarily true for any optimal solution, but we can replace every q-flat with the optimal one wlog

Theorem 6 *There exists a point set such that no subset of less than $1/(2\varepsilon)$ points contains an ε-approximate center, i.e., $C_\rho(0, \varepsilon) = \Omega(1/\varepsilon)$.*

The following result gives an almost tight upper bound for arbitrary ρ. It follows directly from the techniques introduced by Shyamalkumar and Varadarajan [35] for the case of lines through the origin.

Theorem 7 *For any (finite) point set $P \subset \mathbb{R}^d$, there is a set $S \subset P$ of $O(1/\varepsilon \log(1/\varepsilon))$ points such that the subspace spanned by S contains an ε-approximate center. In other words, $C_\rho(0, \varepsilon) = O(1/\varepsilon \log(1/\varepsilon))$.*

Smaller coresets exist for special cases: for $\rho = \infty$, a coreset of size $O(1/\varepsilon)$ (in fact, of size $\lceil 1/\varepsilon \rceil$) exists [10]. It is also known that for $\rho = 2$, the squared distance function $d^2(x, P)$ is a quadratic function in x and can therefore be tackled through sparse greedy optimization in the Frank-Wolfe framework [13, 25].

Theorem 8 *For $\rho = 2$, there is a set of $O(1/\varepsilon)$ points such that their subspace contains an ε-approximate center, i.e., $C_2(0, \varepsilon) = O(1/\varepsilon)$.*

The case of general q The best known bounds for $C_\rho(q, \varepsilon)$ with $q \geq 0$, are again due to Shyamalkumar and Varadarajan. The aforementioned result for lines yields that $C_\rho(1, \varepsilon) = O(1/\varepsilon \log(1/\varepsilon))$, the same bound as for $q = 0$ [35, Lemma 3.2]. They use the line case in an inductive argument to show [35, Lemma 3.3]:

Theorem 9 *For $q \geq 1$, $C_\rho(q, \varepsilon) = O(q^2/\varepsilon \log(q/\varepsilon))$.*

A natural question is to ask about the tightness of the coreset bounds: for the point case $q = 0$, we conjecture that coresets of size $O(1/\varepsilon)$ exist for any norm ρ (currently, this is only established for $\rho \in \{2, \infty\}$). For general q, an improved upper bound of $O(q/\varepsilon)$ would yield a target dimension linear in q in our dimension reduction result.

4 Applications

Streaming algorithms for projective clustering We consider the projective clustering problem in a streaming context. In this setup, we do not return the cluster centers (the q-flats) but only an ε-approximation of $f_k^q(P, \rho)$. We let $S(n, d, q, k, \varepsilon, \rho)$ be the space complexity for this problem. We assume that n, the size of the stream, is known in advance.

Set $m := O((q + 1)^2/\varepsilon^3 \log n \log((q + 1)/\varepsilon))$. In the simplest variant, our streaming initially chooses a $d \times m$ projection matrix uniformly at random, projects every point from the stream to $\mathbb{R}^m$, and stores all points in a set P'. The algorithm, then uses an (offline)-algorithm to approximate $f_k^q(P', \rho)$. The total work space of this

algorithm is $O(dm + nm + S)$, where dm is the space required to store the projection matrix, nm is the size of P', and S is the space required to find approximate clustering of P'. Using any approximation algorithm that computes using linear space, we obtain a streaming algorithm to approximate $f_k^q(P, \rho)$ with a space complexity of $O((q+1)^2(n+d)/\varepsilon^3 \log n \log((q+1)/\varepsilon))$. This is much smaller than the input size of $O(dn)$ and, for small q, not too far from the lower bound of $\Omega(n)$ [3].

In a similar fashion, our results can be used to speed up other streaming approaches: Again, we choose initially a $d \times m$ projection matrix uniformly at random which is stored throughout the algorithm. Furthermore, we maintain the workspace of a streaming algorithm that computes an approximation of the considered projective clustering problem in m dimensions. When a new point $p \in \mathbb{R}^d$ arrives, we compute its projection $\hat{\pi}(p) \in \mathbb{R}^m$ and treat this as an input to the m-dimensional streaming algorithm. We return the output value of the m-dimensional streaming algorithm as our result. The correctness of the approach (with high probability) follows from Corollary 5. The space complexity is $O(dm + S)$, with S the space complexity of the m-dimensional streaming algorithm.

Approximate Projective Clustering. Our technique is also useful for the computation of approximate cluster centers: For a set P of n points in $\mathbb{R}^d$, let $T(n, d, q, k, \varepsilon, \rho)$ denote the time complexity to compute k q-flats $\mathcal{F}$ that ε-approximate the optimal solution, that is, $(\sum_{p\in P}(\mathtt{d}(p, \mathcal{F}))^\rho)^{1/\rho} \leq (1+\varepsilon) f_k^q(P, \rho)$. We design a new algorithm as follows: Set $\varepsilon' := \varepsilon/5$. First, we randomly project the input point set from d to $m := O(C_\rho(q, \varepsilon') \log n/\varepsilon'^2)$ dimensions. Let P' be this set of projected points. Then, we (ε'-approximately) solve the same problem for P' in m dimensions, using some algorithm for this problem as a black box. The computed solution clusters P' in k subsets of points that are closest to a particular q-flat in the solution. We let $P^1, \dots P^k$ be the pre-image of these k clusters and assume wlog that $P^i \cap P^j = \emptyset$. For each P^i, we compute an ε'-approximation of the best fitting q-flat. We return the collection of these k q-flats as solution. Correctness of this approach follows from Theorem 4 and Corollary 5. As an example, we get the k-center problem by setting $\rho = \infty$ and $q = 0$. Using the bounds $C_\infty(0, \varepsilon) = 2/\varepsilon$, $T(n, d, 0, k, \varepsilon, \infty) = O(nd2^{k \log k/\varepsilon})$ and $T(n, d, 0, 1, \varepsilon, \infty) = O(\frac{nd}{\varepsilon^2} + \frac{1}{\varepsilon^5})$ from [7], we get a running time of

$$O(n \log n 2^{k \log k/\varepsilon} + \frac{dn \log n}{\varepsilon^3}).$$

Approximating Čech complexes A standard tool in capturing topological properties of point cloud data is the *Čech complex*[3]. It is usually defined to be the *nerve* of balls of some fixed radius α centered at the points from the sample P, and denoted as $\mathcal{C}_\alpha(P)$. An equivalent definition is that a k-simplex $\{p_0, \dots, p_k\}$ is in $\mathcal{C}_\alpha(P)$ if and only if the radius of $\mathrm{meb}(p_0, \dots, p_k)$ is at most α.

The downside of Čech complexes is the size: Their d-skeleton can consist of up to $O(n^{d+1})$ simplices. Recent work suggests to work instead with an approximation of the Čech complex [27] (or of the closely related Vietoris-Rips complex [33] [17]). "Approximation" in this context means that the persistence diagrams of the modules induced by the Čech filtration and by the approximate filtration are close to each other. Theorem 4 for $q = 0$, $k = 1$ and $\rho = \infty$ implies that the radius of MEBs is preserved for any subset. That implies immediately that Čech complexes can be approximated by Čech complexes in lower dimensions.

Proposition 10 *For $0 < \varepsilon \leq \frac{c-1}{c} < 1$ with $c > 1$ and arbitrary constant, a set $P \subset \mathbb{R}^d$ of n points, and $m = \Theta(\log(n)/\varepsilon^3)$, a random projection $\hat{\pi} : \mathbb{R}^d \to \mathbb{R}^m$ satisfies with high probability that*

$$\mathcal{C}_{(1-c\varepsilon)\alpha}(P) \subseteq \mathcal{C}_\alpha(\hat{\pi}(P)) \subseteq \mathcal{C}_{(1+c\varepsilon)\alpha}(P).$$

An interesting consequence of this statement is that a Čech complex cannot have any significantly persistent features in dimensions higher than m. Independently from our work, Sheehy [34] recently showed a slightly stronger result, projecting to $\Theta(\log(n)/\varepsilon^2)$ dimensions.

Acknowledgments. The authors thank the anonymous referees of an earlier version of this paper whose comments have led to significant simplifications and improvements of our results. The first author acknowledges support by the Max Planck Center for Visual Computing and Communication. The second author acknowledges support by NSF through grant CCF-1464276.

References

[1] P. Agarwal, S. Har-Peled, and H. Yu. Embeddings of surfaces, curves, and moving points in Euclidean space. *SIAM Journal on Computing*, 42(2):442–458, 2013.

[2] P. Agarwal and N. Mustafa. k-means projective clustering. In *Proceedings of the Twenty-third ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 155–165, 2004.

[3] P. Agarwal and R. Sharathkumar. Streaming algorithms for extent problems in high dimensions. *Algorithmica*, pages 1–16, 2013.

[4] P. K. Agarwal, C. M. Procopiuc, and K. R. Varadarajan. Approximation algorithms for a k-line center. *Algorithmica*, 42(3-4):221–230, 2005.

[3] For brevity, we omit a thorough introduction of the topological concepts used in this paragraph. See [18] for more details

[5] D. Aloise, A. Deshpande, P. Hansen, and P. Popat. NP-hardness of euclidean sum-of-squares clustering. *Machine Learning*, 75(2):245–248, 2009.

[6] S. Arora, E. Hazan, and S. Kale. A fast random sampling algorithm for sparsifying matrices. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 4110 of *LNCS*, pages 272–279. 2006.

[7] M. Bâdoiu and K. Clarkson. Smaller core-sets for balls. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 801–802, 2003.

[8] M. Bādoiu, S. Har-Peled, and P. Indyk. Approximate clustering via core-sets. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing*, pages 250–257, 2002.

[9] C. Boutsidis, A. Zouzias, and P. Drineas. Random projections for k-means clustering. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 298–306. 2010.

[10] M. Bădoiu and K. Clarkson. Optimal core-sets for balls. *Computational Geometry: Theory and Applications*, 40:14–22, 2008.

[11] K. Chen. On coresets for k-median and k-means clustering in metric and Euclidean spaces and their applications. *SIAM Journal of Computing*, 39(3):923–947, 2009.

[12] K. Clarkson. Tighter bounds for random projections of manifolds. In *Proceedings of the 24th Symposium on Computational Geometry*, pages 39–48, 2008.

[13] K. Clarkson. Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. *ACM Transactions on Algorithms*, 6(4), 2010.

[14] K. Clarkson and D. Woodruff. Low rank approximation and regression in input sparsity time. In *Proceedings of the 45th Annual ACM Symposium on Theory of Computing*, pages 81–90, 2013.

[15] M. B. Cohen, S. Elder, C. Musco, C. Musco, and M. Persu. Dimensionality reduction for k-means clustering and low rank approximation. *CoRR*, abs/1410.6801, 2014.

[16] S. Dasgupta and A. Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures & Algorithms*, 22:60–65, 2003.

[17] T. Dey, F. Fan, and Y. Wang. Graph induced complex on point data. In *Proceedings of the 29th ACM Symposium on Computational Geometry*, 2013.

[18] H. Edelsbrunner and J. Harer. *Computational Topology, An Introduction.* American Mathematical Society, 2010.

[19] U. Feige and E. Ofek. Spectral techniques applied to sparse random graphs. *Random Structures & Algorithms*, 27:251–275, 2005.

[20] D. Feldman, M. Monemizadeh, C. Sohler, and D. Woodruff. Coresets and sketches for high dimensional subspace approximation problems. In *Proceedings of the 21st Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 630–649, 2010.

[21] D. Feldman, M. Schmidt, and C. Sohler. Turning big data into tiny data: Constant-size coresets for k-means, PCA and projective clustering. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1434–1453, 2013.

[22] S. Har-Peled and S. Mazumdar. On coresets for k-means and k-median clustering. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pages 291–300, 2004.

[23] S. Har-Peled and K. Varadarajan. Projective clustering in high dimensions using core-sets. In *Proceedings of the 18th ACM Symposium on Computational Geometry*, pages 312–318, 2002.

[24] P. Indyk and A. Naor. Nearest-neighbor-preserving embeddings. *ACM Transactions on Algorithms*, 3(3), Aug. 2007.

[25] M. Jaggi. Revisiting Frank-Wolfe: Projection-free sparse convex optimization. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013*, pages 427–435, 2013.

[26] W. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. *Contemporary Mathematics*, 26:189–206, 1982.

[27] M. Kerber and R. Sharathkumar. Approximate Čech complex in low and high dimensions. In *24th International Symposium on Algorithms and Computation*, LNCS 8283, pages 666–676, 2013.

[28] A. Magen. Dimensionality reductions in ℓ_2 that preserve volumes and distance to affine spaces. *Discrete & Computational Geometry*, 38(1):139–153, 2007.

[29] J. Nelson and H. Nguyen. Lower bounds for oblivious subspace embeddings. In *Automata, Languages, and Programming*, volume 8572 of *Lecture Notes in Computer Science*, pages 883–894. Springer Berlin Heidelberg, 2014.

[30] R. Ostrovsky and Y. Rabani. Polynomial time approximation schemes for geometric k-clustering. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, FOCS '00, pages 349–, 2000.

[31] C. Procopiuc. Projective clustering. In C. Sammut and G. Webb, editors, *Encyclopedia of Machine Learning*, pages 806–811. Springer US, 2010.

[32] T. Sarlos. Improved approximation algorithms for large matrices via random projections. In *Foundations of Computer Science*, pages 143–152, 2006.

[33] D. Sheehy. Linear-size approximation to the Vietoris-Rips filtration. In *Proceedings of the 28th ACM Symposium on Computational Geometry*, pages 239–248, 2012.

[34] D. Sheehy. The persistent homology of distance functions under random projection. In *Proceedings of the 30th ACM Symposium on Computational Geometry*, 2014.

[35] N. Shyamalkumar and K. Varadarajan. Efficient subspace approximation algorithms. *Discrete & Computational Geometry*, 47(1):44–63, 2012.

[36] S. Vempala. *The random projection method*, volume 65. AMS Bookstore, 2004.

A Missing proofs of Section 3

Proof of Theorem 6 Consider the standard $(n-1)$-simplex spanned by n points in $\mathbb{R}^n$, namely the points $e_1, \ldots, e_n$. The optimal center for any ρ is the barycenter o given by $(1/n, \ldots, 1/n)$, and we have that

$$\delta(o) = n^{1/\rho}\sqrt{\frac{n-1}{n}}.$$

Choose a subset of size c, w.l.o.g. $e_1, \ldots, e_c$ and let F be the affine subspace spanned by these points. Let o' denote the barycenter of the $(c-1)$-simplex $(e_1, \ldots, e_c)$. Now o' is the point that minimizes $\delta(\cdot)$ over F, since o' is the orthogonal projection on F of any e_i with $i > c$. So, assuming that F contains an ε-approximate center, o' must be an approximate center. On the other hand, $\delta(o')$ is easily computed by noting that $d(o', e_i) = \sqrt{(n-1)/n}$ for any $i \le c$ and $d(o', e_i) = \sqrt{(n+1)/n}$ for any $i > c$. This yields

$$\begin{aligned}&\left(c\left(\sqrt{\frac{c-1}{c}}\right)^\rho + (n-c)\left(\sqrt{\frac{c+1}{c}}\right)^\rho\right)^{\frac{1}{\rho}}\\ =\;& \delta(o') \le (1+\varepsilon)\delta(o) = (1+\varepsilon)n^{1/\rho}\sqrt{\frac{n-1}{n}}.\end{aligned}$$

Raising to the ρ-th power and dividing by n yields that

$$\frac{c}{n}\left(\sqrt{\frac{c-1}{c}}\right)^\rho + \frac{n-c}{n}\left(\sqrt{\frac{c+1}{c}}\right)^\rho \le (1+\varepsilon)^\rho\left(\sqrt{\frac{n-1}{n}}\right)^\rho.$$

Because this bounds has to hold for every n, it must also hold in the limit for $n \to \infty$. That results in

$$\left(\sqrt{\frac{c+1}{c}}\right)^\rho \le (1+\varepsilon)^\rho,$$

and by solving for c yields that $c \ge 1/(2\varepsilon+\varepsilon^2) \ge 1/(3\varepsilon)$ for $\varepsilon \le 1$.

Proof of Theorem 7 We define an iterative procedure which creates points $c_0, c_1, \ldots$ such that c_i is in the subspace spanned by i input points. The initial point c_0 is chosen to be point closest to the optimal center.[4] If some c_i is an ε-approximate center, we are done. Otherwise, we show that we can chose a point c_{i+1} that is significantly closer to o. For that, let s be the point the maximizes

$$\frac{d(c_i, p)}{d(o, p)}$$

over all $p \in P$. By construction, $d(c_i, s) \ge (1+\varepsilon)d(o, s)$ We choose c_{i+1} as the point on the line segment $c_i s$ that is closest to o. It follows easily [35, Lemma 2.1] that $d(c_{i+1}, o) \le (1-\varepsilon/2)d(c_i, o)$. Combined with the triangle inequality and the fact that c_0 is the closest point to o in P, this implies that for any $p \in p$:

$$\begin{aligned}d(c_k, p) &\le d(c_k, o) + d(o, p)\\ &\le (1-\varepsilon/2)^k d(c_0, o) + d(o, p)\\ &\le (1 + (1-\varepsilon/2)^k)d(o, p).\end{aligned}$$

For $k = O(1/\varepsilon \log(1/\varepsilon))$, this means that $d(c_k, p) \le (1+\varepsilon)d(o, p)$ for all $p \in P$, which directly implies that $\delta(c_k) \le (1+\varepsilon)\delta(o)$.

Proof of Theoerem 8 Writing A for the matrix whose columns are the points in P and Δ for the standard simplex with points $e_1, \ldots, e_n$, we can consider the function $g : \Delta \to \mathbb{R}$ defined by

$$\begin{aligned}g(x) &= \sum_{i=1}^{n} \|Ax - Ae_i\|^2\\ &= \sum_{i=1}^{n} (x - e_i)^T A^T A(x - e_i)\\ &= x^T M x + x^T b + a\end{aligned}$$

with $M = nA^TA$, $b = -2\sum A^T A e_i$, and $a = \sum e_i^T A^T A e_i$. Therefore, g is a quadratic function. We apply the Frank-Wolfe optimization on the (convex) function g: this method starts in an arbitrary point x_0 in P and improves the approximation quality in every step by moving towards the point in P which the steepest descent. The obtained sequence of iterates $x_0, x_1, \ldots$ converges to the (unique) minimum of g, and by construction, the iterate x_i lies in the span of i points of P.

A crucial quantity in the convergence behavior of Frank-Wolfe is the quantity C_g: this is a scaled form of the Bregman divergence of the function g, measuring the difference between $g(y)$ and the value at y of the tangent plane of g at x, for all pairs x and y. Since g is a quadratic function, [13, Sec. 4.3] asserts that $C_g \le \operatorname{diam}(P)^2$. Writing r for the radius of the MEB of P, this implies $C_g \le 4r^2$.

Slightly abusing notation, we let $o \in \Delta$ denote the point that minimizes g. Using Theorem 2.3 from [13], after running the Franke-Wolfe optimization for $k := 2\lceil 1/\varepsilon \rceil$ steps, we find an iterate x_k on a k-simplex which satisfies

$$g(x_k) - g(o) \le 4\varepsilon C_f \le 16\varepsilon r^2 \le 16\varepsilon g(o),$$

where the last inequality comes from the fact that with p being the furthest point from o, it holds that $g(o) \ge \|o - p\|^2 \ge r^2$. Therefore, we have that $g(x_k) \le (1+16\varepsilon)g(o)$.

[4] Again, since we only care about existence, we can conveniently assume that the center is known to us.

Fun with Restricted Delaunay Triangulations

Jonathan Richard Shewchuk (University of California at Berkeley)

Abstract

The restricted Delaunay triangulation is a subcomplex of the three-dimensional Delaunay triangulation that serves as a triangulation of a smooth surface embedded in three-dimensional space. It has proven itself as a mathematically powerful tool for surface meshing and surface reconstruction. I discuss two fascinating mathematical twists on these structures. First, I address a question of Bruno Levy: can we constrain them to include specified edges–that is, can we define mathematically well-behaved constrained Delaunay triangulations on smooth surfaces? Second, the restricted Delaunay triangulation can be conceived as an operator that takes as input a smooth surface and a set of points sampled from that surface, and produces as output a triangulation of the surface. What happens if we feed that triangulation back into the operator, replacing the original surface, while retaining the same sample points? Interestingly, the answer leads us to a method for reconstructing 2-manifolds embedded in high-dimensional spaces.

(This work is done jointly with Marc Khoury, Bruno Levy, and Marc van Kreveld.)

Algorithms for Minimizing the Movements of Spreading Points in Linear Domains

Shimin Li* Haitao Wang†

Abstract

We study a problem on spreading points. Given a set P of n points sorted on a line L and a distance value δ, we wish to move the points of P along L such that the distance of any two points of P is at least δ and the maximum movement of all points is minimized. We present an $O(n)$ time algorithm for this problem. Further, we extend our algorithm to solve (in $O(n)$ time) the cycle version of the problem where all points of P are on a cycle C. Previously, only weakly polynomial-time algorithms were known for these problems based on linear programming. In addition, we present a linear-time algorithm for a similar facility-location moving problem, which improves the previous work.

1 Introduction

We consider the following *points-spreading* problem. Given a set P of n points sorted on a line L and a distance value $\delta \geq 0$, we wish to move the points of P along L such that the distance of any two points of P is at least δ and the maximum movement of all points of P is minimized. The above is the *line version*. We also consider the *cycle version* of the problem, where all points of P are given sorted cyclically on a cycle (one may view C as a simple closed curve). We wish to move the points of P on C such that the distance of any two points of P along C is at least δ and the maximum movement of all points of P along C is minimized. Note that since C is a cycle, the distance of any two points of C is defined to be the length of the shortest path on C between the two points.

Both versions of the problem have been studied before. By modeling them as linear programming problems (with n variables and $\Theta(n)$ constraints), Dumitrescu and Jiang [4] gave the first-known polynomial-time algorithms for both problems. Since there only exist weakly polynomial-time algorithms for linear programming [8, 9], it would be interesting to design strongly polynomial-time algorithms for the points-spreading problem. In this paper, we solve both versions of the problem not only in strongly polynomial time but also in $O(n)$ time (which is optimal). Our algorithms are based on a greedy strategy.

In addition, we consider a somewhat related problem, called the *facility-location movement* problem, defined as follows. Suppose we have a set of k "server" points and another set of n "client" points sorted on L. We wish to move all servers and all clients on L such that each client co-locates with a server and the maximum moving distance of all servers and clients is minimized. Dumitrescu and Jiang [4] solved this problem in $O((n+k)\log(n+k))$ time. We present an $O(n+k)$ time algorithm based on their approach.

1.1 Related Work

The points-spreading problem in 2D was proposed by Demaine et al. [3] (called "movement to independence" problem in [3, 4]). The problem in 2D is NP-hard and an approximation algorithm was given in [3]; the algorithm was improved later by Dumitrescu and Jiang [4].

The points-spreading problem is related to the points dispersion problems which involve arranging a set of points as far away from each other as possible subject to certain constraints. For example, Fiala et al. in [6] studied such a problem in which one wants to place n given points, each inside its own, prespecified disk, with the objective of maximizing the distance between the closest pair of these points. The problem was shown to be NP-hard [6]. Approximation algorithms were proposed by Cabello [1]. Dumitrescu and Jiang [5] gave improvement on the approximation algorithms and also presented algorithms for the problem in high-dimensional spaces. In fact, Fiala et al. [6] studied the dispersion problems on a more general problem settings. Another variation of the dispersion problems is to select a subset of facilities from a set of given facilities to maximize the minimum distance (or some other distance function) among all pairs of selected facilities [10, 11]. The problem is generally NP-hard (e.g., in 2D) but polynomial time algorithms are available in the one-dimensional space [10, 11]. In addition, Chandra and Halldórsson [2] studied dispersion problems on other problem settings.

The facility-location movement problem was first introduced by Demaine et al. [3] in graphs, which was proved to be NP-hard. A 2-approximation algorithm

*Department of Computer Science, Utah State University, Logan, UT 84322, USA. shiminli@aggiemail.usu.edu

†Department of Computer Science, Utah State University, Logan, UT 84322, USA. haitao.wang@usu.edu

was presented in [3] for this problem in graphs, and later it was shown that the 2-approximation ratio cannot be improved unless P=NP [7]. Dumitrescu and Jiang [4] studied the geometric version of this problem in the plane, and they showed that the problem is NP-hard to approximate within 1.8279. Fixed parameter algorithms (with k as the parameter) were also given in [4].

1.2 Our Approaches

For solving the line version of the points-spreading problem, essentially we first solve a "one-direction" case of the problem in which points are only allowed to move rightwards, by using a simple greedy algorithm. Suppose d is the maximum movement in the solution of the above one-direction case. Then, we show that an optimal solution to the original problem can be obtained by shifting each point of P leftwards by the distance $d/2$.

For solving the cycle version of the problem, essentially we also first solve a one-direction case in which points are only allowed to move counterclockwise on C. If d is the maximum movement in the solution of the one-direction case, then we also show that an optimal solution to the original problem can be obtained by shifting each point of P clockwise by $d/2$. However, unlike the line version, the one-direction case of the problem becomes more difficult on the cycle. One straightforward idea is to cut the cycle C at a point of P (and extend C as a line) and then apply the algorithm for the one-direction case of the line version. However, the issue is that the last point may be too close to or even "cross" the first point if we put all points back on C. By observations, we show that if such a case happens, we can run the line-version algorithm for another round and the second round is guaranteed to find an optimal solution. Overall, the algorithm is still simple, but it is challenging to show the correctness.

For solving the facility-location movement problem, Dumitrescu and Jiang [4] presented an $O((m+n)\log(m+n))$ time algorithm using dynamic programming. By discovering a monotonicity property on the dynamic programming, we improve Dumitrescu and Jiang's algorithm to $O(n+k)$ time.

The rest of the paper is organized as follows. In Section 2, we present our algorithm for the line version of the points-spreading problem. The cycle version of the problem is solved in Section 3. Section 4 discusses our solution for the facility-location movement problem.

Due to the space limit, proofs of all lemmas and observations in Section 3 are in the appendix.

2 The Points-Spreading Problem on a Line

In the line version, the points of P are given sorted on the line L. Without loss of generality, we assume L is the x-axis and $P = \{p_1, p_2, \ldots, p_n\}$ are sorted by their x-coordinates from left to right. For each $i \in [1, n]$, let x_i denote the location (or x-coordinate) of p_i on L. For any two locations x and x' of L, denote by $|xx'|$ the distance between x and x', i.e., $|xx'| = |x - x'|$.

Our goal is to move each point $p_i \in P$ to a new location x'_i on L such that the distance of any pair of two points of P is at least δ and the maximum moving distance, i.e., $\max_{1\le i\le n} |x_i x'_i|$, is minimized. For simplicity of discussion, we make a general position assumption that no two points of P are at the same location in the input. The degenerate case can also be handled by our techniques but the discussions would be more tedious.

We refer to a *configuration* as a specification of the location of each point p_i of P on L. For example, in the input configuration each p_i is at x_i. Let F_0 denote the input configuration. A configuration is *feasible* if the distance between any pair of points of P is at least δ.

Denote by d_{opt} the maximum moving distance in any optimal solution. If the input configuration F_0 is feasible, then we do not need to move any point, implying that $d_{opt} = 0$. Since the points of P are sorted, we can check whether F_0 is feasible in $O(n)$ time by checking the distance between every adjacent pair of points of P. Below, we assume F_0 is not feasible, and thus $d_{opt} > 0$.

We first present some observations, based on which our algorithm will be developed.

2.1 Observations

For any two indices $i < j$ in $[1, n]$, define

$$w(i, j) = (j - i) \cdot \delta - |x_i x_j|.$$

As discussed in [4], there exists an optimal solution in which the order of all points of P is the same as that in the input configuration F_0. Based on this property, we prove Lemma 1 regarding the value d_{opt}.

Lemma 1 $d_{opt} \ge \max_{1\le i<j\le n} \frac{w(i,j)}{2}$.

Proof. Consider any optimal solution OPT in which the order of all points of P is the same as that in F_0. For each $1 \le i \le n$, let x_i^* be the location of p_i in OPT.

Consider any i and j with $1 \le i < j \le n$. Our goal is to prove $d_{opt} \ge w(i, j)/2$. Since the points of P in OPT have the same order as in F_0, for each k with $i < k \le j$, we have $|x_{k-1}^* x_k^*| \ge \delta$ because OPT is a feasible solution. Hence, $|x_i^* x_j^*| = \sum_{k=i+1}^{j} |x_{k-1}^* x_k^*| \ge (j - i) \cdot \delta$.

If $|x_i^* x_j^*| - |x_i x_j| \le 0$, then $|x_i x_j| \ge |x_i^* x_j^*| \ge (j - i) \cdot \delta$ and $w(i, j) \le 0$. Since $d_{opt} > 0$, $d_{opt} \ge w(i, j)/2$ holds.

If $|x_i^* x_j^*| - |x_i x_j| > 0$, then the difference of $|x_i^* x_j^*|$ and $|x_i x_j|$ are due to the moving of p_i and p_j. It is not difficult to see that $\max\{|x_i x_i^*|, |x_j x_j^*|\} \ge (|x_i^* x_j^*| - |x_i x_j|)/2$ (the equality happens when p_i moves leftwards by distance $(|x_i^* x_j^*| - |x_i x_j|)/2$ and p_j moves rightwards by the same distance). Since $d_{opt} \ge \max\{|x_i x_i^*|, |x_j x_j^*|\}$,

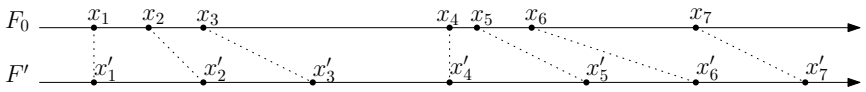


Figure 1: Illustrating our algorithm for computing the configuration F.

it holds that $d_{opt} \geq (|x_i^* x_j^*| - |x_i x_j|)/2$. Due to $|x_i^* x_j^*| \geq (j-i) \cdot \delta$, we obtain that $d_{opt} \geq w(i,j)/2$. □

Lemma 2 *If there exist i and j with $1 \leq i < j \leq n$ and a feasible configuration F' in which each point $p_k \in P$ moves rightwards to x'_k (i.e., $x_k \leq x'_k$) such that $w(i,j) = \max_{1 \leq k \leq n} |x_k x'_k|$, then we can obtain an optimal solution by shifting each point of P in F' leftwards by distance $w(i,j)/2$.*

Proof. Let F'' be the configuration obtained by shifting each point of P in F' leftwards by distance $w(i,j)/2$.

Consider any point $p_k \in P$. Let x''_k denote the location of p_k in F'', i.e., $x''_k = x'_k - w(i,j)/2$. In order to prove that F'' is an optimal solution, by Lemma 1, it is sufficient to show that $|x_k x''_k| \leq w(i,j)/2$, as follows.

Indeed, since $0 \leq x'_k - x_k \leq w(i,j)$, i.e., x'_k is to the right of x_k at most $w(i,j)$, after p_k is moved leftwards by $w(i,j)/2$ to x''_k, x''_k must be within distance $w(i,j)/2$ from x_k. Hence, $|x_k x''_k| \leq w(i,j)/2$. □

We call a feasible configuration that satisfies the condition in Lemma 2 a *canonical configuration* (such as F' in Lemma 2). Due to Lemma 2, to solve the problem in linear time, it is sufficient to find a canonical configuration in linear time, which is our focus below.

2.2 Computing a Canonical Configuration

We present a linear-time algorithm for finding a canonical configuration. Comparing with the original problem, now we only need to consider the rightward movements.

Initially, we set $x'_1 = x_1$. Then we consider the points $p_2, p_3, \ldots, p_n$ from left to right. For each i with $2 \leq i \leq n$, suppose we have already moved p_{i-1} to x'_{i-1}. Then, we set $x'_i = \max\{x_i, x'_{i-1} + \delta\}$, and move p_i to x'_i. Refer to Fig. 1 for an example. The algorithm finishes after all points of P have been considered. Clearly, the algorithm runs in $O(n)$ time. Let F' denote the resulting configuration (i.e., each p_i is at x'_i).

Lemma 3 *F' is a canonical configuration.*

Proof. First of all, based on our way of setting x'_i for $i = 1, 2, \ldots, n$, every two points of P in F' are at least δ away from each other. Thus, F' is a feasible configuration. Note that $x'_i \geq x_i$ for any $i \in [1,n]$. Next, we show that there exist i and j with $1 \leq i < j \leq n$ such that $w(i,j) = d_{max}$, where $d_{max} = \max_{1 \leq k \leq n} |x_k x'_k|$.

Recall that $d_{max} > 0$. Suppose the moving distance of p_j is the maximum, i.e., $d_{max} = |x_j x'_j|$. Let i be the largest index such that $i < j$ and p_i does not move in the algorithm (i.e., $x_i = x'_i$). Note that such a point p_i must exist as $x_1 = x'_1$ and $x'_j > x_j$.

For any point $p_k \in P$, if p_k is moved (rightwards) in F' (i.e., $x_k < x'_k$), then according to our way of setting x'_k, it must hold that $x'_k - x'_{k-1} = \delta$. By the definition of i, for each point p_k with $k \in [i+1, j]$, p_k is moved in F', and thus $x'_k - x'_{k-1} = \delta$. Therefore, we obtain $|x'_i x'_j| = x'_j - x'_i = \sum_{i+1 \leq k \leq j} (x'_k - x'_{k-1}) = (j-i) \cdot \delta$.

Since $x'_i = x_i$ and $x_j < x'_j$, we have $|x_i x'_j| = |x_i x_j| + |x_j x'_j|$. Hence, $d_{max} = |x_j x'_j| = |x_i x'_j| - |x_i x_j| = (j-i) \cdot \delta - |x_i x_j| = w(i,j)$. This proves the lemma. □

Lemmas 2 and 3 together lead to Theorem 4.

Theorem 4 *The line version of the points-spreading problem is solvable in $O(n)$ time.*

Remark: One may verify that our algorithm for computing the canonical configuration F' essentially solves a *one-direction case* of the line version problem: Move the points of P rightwards such that any pair of points of P are at least δ away from each other and the maximum moving distance of all points of P is minimized.

3 The Points-Spreading Problem on a Cycle

In the cycle version, the points of $P = \{p_1, \ldots, p_n\}$ are on a cycle C sorted cyclically, say, in the counterclockwise order. We use $|C|$ to denote the length of C. For any two locations x and x' on C, the distance between x and x', denoted by $|xx'|$, is the length of the shortest path between x and x' on C. Clearly, $|xx'| \leq |C|/2$. For each $i \in [1,n]$, we use x_i denote the location of p_i on C in the input. Our goal is to move each point $p_i \in P$ to a new location x'_i such that the distance of any pair of two points of P on C is at least δ and the maximum moving distance, i.e., $\max_{1 \leq i \leq n} |x_i x'_i|$, is minimized.

We assume $|C| \geq \delta \cdot n$ since otherwise there would be no solution. Again, for simplicity of discussion, we make a general position assumption that no two points of P are at the same location on C in the input.

As before, we refer to a *configuration* as a specification of the location of each point of P on C. A configuration is *feasible* if the distance between any pair of points of P is at least δ. Let F_0 denote the input configuration.

Denote by d_{opt} the maximum moving distance in any optimal solution. If F_0 is feasible, then $d_{opt} = 0$. We can also check whether F_0 is feasible in $O(n)$ time. Below, we assume F_0 is not feasible, and thus $d_{opt} > 0$.

To solve the problem, we extend our algorithm (and observations) for the line version in Section 2. Namely,

we first move all points of P on C counterclockwise to obtain a "canonical configuration", and then shift all points clockwise. However, as will be seen later, the problem becomes much more difficult on the cycle.

Consider any two locations x and x' on C. We define $C(x, x')$ as the portion of C from x to x' counterclockwise. We use $|C(x, x')|$ to denote the length of $C(x, x')$. Note that $|xx'| = \min\{|C(x, x')|, |C(x', x)|\}$.

As in the line version, we first give some observations, based on which our algorithms will be developed.

3.1 Observations

For any two indices $i \neq j$ in $[1, n]$, define

$$w(i, j) = \big[(n + j - i) \mod n\big] \cdot \delta - |C(x_i, x_j)|.$$

In words, if $i < j$, then $w(i, j) = (j-i)\cdot\delta - |C(x_i, x_j)|$; otherwise, $w(i, j) = (n + j - i) \cdot \delta - |C(x_i, x_j)|$. Since $|C| \geq \delta \cdot n$, it can be verified that $w(i, j) \leq |C|$.

As discussed in [4], there exists an optimal solution in which the order of all points of P is the same as that in the input configuration F_0. Using this property, we can prove Lemma 5, which is analogous to Lemma 2.

Lemma 5 $d_{opt} \geq \max_{1\leq i,j\leq n} \frac{w(i,j)}{2}$.

Based on Lemma 5, we obtain the following lemma, which is analogous to Lemma 3 for the line version.

Lemma 6 *If there exist $i \neq j$ in $[1, n]$ and a feasible configuration F' in which each point $p_k \in P$ is at location x'_k such that $w(i, j) = \max_{1\leq k\leq n} |C(x_k, x'_k)|$, then we can obtain an optimal solution by shifting every point of P in F' clockwise by distance $w(i, j)/2$.*

We call a feasible configuration that satisfies the condition in Lemma 6 a *canonical configuration*. In light of Lemma 6, to solve the problem in linear time, it is sufficient to find a canonical configuration in linear time, which is our focus below.

3.2 Computing a Canonical Configuration

We present a linear-time algorithm for finding a canonical configuration. Now we only need to consider the counterclockwise movements.

Recall that the points $p_1, p_2, \ldots, p_n$ are ordered on C counterclockwise in the input configuration F_0. For convenience of discussion, we define coordinates for locations on C in the following way. Define x_1 as the origin with coordinate zero. For any other location $x \in C$, the coordinate of x is defined to be $|C(x_1, x)|$. Hence each location of C has a coordinate no greater than $|C|$.

Our algorithm has two rounds. In the first round, we will use the same approach as for the line version of the problem, and let F_1 denote the resulting configuration. However, the issue is that in F_1 the new location of p_n may be too close to p_1 or p_n may even "cross" p_1, which might make F_1 not feasible. If p_n does not cross p_1 and p_n is at least δ away from p_1 in F_1, then we will show that F_1 is a canonical configuration. Otherwise, we will proceed on the second round, which is to consider all points again from p_1 and use the same strategy to set the new locations of the points. We will show that the configuration F_2 obtained after the second round is a canonical configuration. The details are given below.

3.2.1 The first round

In the first round, we will move each point $p_i \in P$ from x_i along C counterclockwise to a new location x'_i. The way we set x'_i here is similar to that in the line version and the difference is that we have to take care of the cycle situation. Specifically, $x'_1 = x_1$, i.e., p_1 does not move. For each $i \in [2, n]$, suppose we have already moved p_{i-1} to x'_{i-1}, then we define x'_i as follows:

$$x'_i = \begin{cases} x_i & \text{if } x_i \geq x'_{i-1} + \delta \\ (x'_{i-1} + \delta) \mod |C| & \text{if } x_i < x'_{i-1} + \delta. \end{cases} \quad (1)$$

This finishes the first round of our algorithm. Denote by F_1 the resulting configuration.

Note that if $x'_{i-1} + \delta > |C|$, then since $x_i \leq |C|$, by Equation (1), $x'_i = (x'_{i-1} + \delta) \mod |C|$, which is equal to $x'_{i-1} + \delta - |C|$; in this case, we say that the counterclockwise movement of p_i *crosses* the origin x_1.

Lemma 7 *If p_n does not cross x_1 $(= x'_1)$ in the first round of the algorithm and $|C(x'_n, x'_1)| \geq \delta$, then F_1 is a canonical configuration.*

By Lemma 7, if p_n does not cross $x_1 = x'_1$ in the first round and $|C(x'_n, x'_1)| \geq \delta$ in F_1, then we have found a canonical configuration and our algorithm stops. Otherwise, we proceed on the second round, as follows.

3.2.2 The second round

In the second round, we will move each $p_i \in P$ from x'_i counterclockwise to a new location x''_i, as follows.

We first define x''_1. Recall that we proceed on the second round because either p_n crosses $x_1 = x'_1$ in the first round or $|C(x'_n, x'_1)| < \delta$. In either case we define

$$x''_1 = (x'_n + \delta) \mod |C|. \quad (2)$$

Hence, $|C(x'_n, x''_1)| = \delta$.

For each $i = 2, 3, \ldots, n$, suppose p_{i-1} has been moved to x''_{i-1}; then we move p_i from x'_i counterclockwise to x''_i, with

$$x''_i = \max\{x'_i, (x''_{i-1} + \delta) \mod |C|\} \quad (3)$$

This finishes the second round of our algorithm. Let F_2 be the resulting configuration. In the sequel we show that F_2 is a canonical configuration.

Observation 1 *There must be a point p_i with $i \in [2, n]$ such that p_i does not move in the first round of the algorithm (i.e., $x_i = x'_i$).*

Observation 2 *If a point p_i does not move in the second round, then for each point p_j with $j \in [i, n]$, p_j does not move in the second round either.*

Lemma 8 *Suppose k is the largest index such that p_k does not move in the first round of the algorithm; then p_k does not move in the second round of the algorithm either, i.e., $x_k = x'_k = x''_k$.*

Based on the proof of Lemma 8, we have the following two corollaries.

Corollary 9 *The configuration F_2 is feasible.*

Corollary 10 *The total counterclockwise moving distance of each point of P in the two rounds of the algorithm is at most $|C|-\delta$, which implies that $|C(x_i, x''_i)| \leq |C| - \delta$ for each $1 \leq i \leq n$.*

With the previous observations, Lemma 11 finally shows that F_2 is a canonical configuration.

Lemma 11 *F_2 is a canonical configuration.*

Clearly, both rounds of our algorithm run in $O(n)$ time. Combining Lemmas 6, 7, and 11, we have the following result.

Theorem 12 *The cycle version of the points-spreading problem is solvable in $O(n)$ time.*

Remark: One may verify that our algorithm for computing the canonical configuration F_2 essentially solves the following *one-direction case* of the cycle version problem: Move the points of P counterclockwise such that any pair of points of P are at least δ away from each other and the maximum counterclockwise moving distance of all points of P is minimized.

4 The Facility-Location Movement Problem

In this section, we present our linear-time algorithm for the facility-location movement problem. In this problem, we are given a set S of k "server" points and a set Q of n "client" points sorted on a line L, and the goal is to move all servers and clients on L such that each client co-locates with a server and the maximum moving distance of all servers and clients is minimized.

As shown by Dumitrescu and Jiang [4], the problem is equivalent to finding k intervals (i.e., line segments) on L such that each interval contains at least one server, each client is covered by at least one interval, and the maximum length of these intervals is minimized. In the following, we will solve this *interval coverage* problem (also called *constrained k-center* problem in [4]).

Dumitrescu and Jiang [4] presented an $O((n + k)\log(n + k))$ time algorithm using dynamic programming. We discover a monotonicity property on their dynamic programming scheme, and consequently improve their algorithm to $O(n+k)$ time. Below, we first review the algorithm in [4] and then show our improvement.

4.1 Preliminaries

Without loss of generality, we assume L is the x-axis. For any two points p and q on L with p to the left of q, we use $[p, q]$ to denote the interval on L with left endpoint at p and right endpoint at q. An easy observation is that there exists an optimal solution consisting of k intervals in $\{[p, q] \mid p, q \in S \cup P\}$. For any two points p and q on L, let $d(p, q)$ denote the distance between them.

Let $S = \{s_1, s_2, \ldots, s_k\}$ be the set of servers sorted on L from left to right. Let $Q = \{q_1, q_2, \ldots, q_n\}$ be the set of clients sorted on L from left to right. For ease of exposition, we assume no two points in $S \cup Q$ are at the same location.

The servers of S partition the clients of Q into $k + 1$ subsets, defined as follows. For each $i \in [1, k - 1]$, let Q_i be the subset of the clients of Q between s_i and s_{i+1} on L. In addition, we let Q_0 be the subset of the clients of Q to the left of s_1, and let Q_k be the subset of the clients of Q to the right of s_k. Since both S and Q are already given sorted, we can obtain the subsets $Q_0, Q_2, \ldots, Q_k$ in $O(n + k)$ time. In the following, for simplicity of discussion, we assume Q_i is not empty for each $i \in [0, k]$. This implies that the rightmost client q_n is to the right of the rightmost server s_k and the leftmost client q_1 is to the left of the leftmost server s_1. For each $i \in [1, k]$, let $Q'_i = \{s_i\} \cup Q_i$.

4.2 A Dynamic Programming Algorithm [4]

Consider any Q'_i with $1 \leq i \leq k$. Let q be any point in Q'_i. Consider the *subproblem* at q: Finding i intervals on L such that each interval contains at least one server of $\{s_1, s_2, \ldots, s_i\}$, each client to the left of q (including q if $q \neq s_i$) must be covered by at least one interval, and the maximum length of these i intervals is minimized. Define $\alpha(q)$ as the maximum length of the intervals in an optimal solution of the above subproblem at q. Our goal for the interval coverage problem is to solve the subproblem at q_n and compute the value $\alpha(q_n)$.

For any point $q \in S \cup Q$, we use $r(q)$ to denote right neighboring point of q on L in $S \cup Q$ (i.e., the closest point of $S \cup Q$ to q strictly to the right of q). Note that after merging S and Q into one sorted list, we can obtain $r(q)$ for each $q \in S \cup Q$ in constant time.

Initially, for each $q \in Q'_1$, $\alpha(q) = d(q_1, q)$ (recall that q_1 is to the left of s_1). In general, consider any $q \in Q'_i$

for any $2 \leq i \leq k$. It holds that

$$\alpha(q) = \min_{q' \in Q'_{i-1}} \max\{\alpha(q'), d(r(q'), q)\}.$$

In words, in order to solve the subproblem at q, we use the $i-1$ intervals for the subproblem at q' along with an additional interval $[r(q'), q]$. To compute $\alpha(q)$, Dumitrescu and Jiang [4] used the following observation: As we consider the points q' of Q'_{i-1} from left to right, $\alpha(q')$ is monotonically increasing and $d(r(q'), q)$ is monotonically decreasing. Hence, if $\alpha(q')$ for all $q' \in Q'_{i-1}$ are known, $\alpha(q)$ can be computed in $O(\log |Q'_{i-1}|)$ time by binary search.

In this way, $\alpha(q_n)$ can be computed in $O((n+k)\log(n+k))$ time (more precisely, $O((n+k)\log n)$ time) and an optimal solution can be found correspondingly.

4.3 An Improved Implementation

We give an $O(n+k)$ time implementation for the above dynamic programming scheme. To this end, we find a new monotonicity property in Lemma 13.

Consider any point $q \in Q'_i$ such that $r(q)$ is still in Q'_i. For any point $q' \in Q'_{i-1}$, define $f(q') = \max\{\alpha(q'), d(r(q'), q)\}$. Hence, $\alpha(q) = \min_{q' \in Q'_{i-1}} f(q')$. Let $g(q)$ be the point in Q'_{i-1} such that $\alpha(q) = f(g(q))$ (if there is more than one such point, we let $g(q)$ refer to the rightmost one).

Lemma 13 *Either $g(r(q)) = g(q)$ or $g(r(q))$ is strictly to the right of $g(q)$.*

Proof. We only give an "intuitive" proof. Recall that as we consider the points q' of Q'_{i-1} from left to right, $\alpha(q')$ is monotonically increasing and $d(r(q'), q)$ is monotonically decreasing. Intuitively, $g(q)$ corresponds to the intersection of the two functions $\alpha(q')$ and $d(r(q'), q)$ for $q' \in Q'_{i-1}$ (e.g., see Figure 2). Similarly, for the point $r(q)$, which is still in Q'_i, $g(r(q))$ corresponds to the intersection of the two functions $\alpha(q')$ and $d(r(q'), r(q))$ for $q' \in Q'_{i-1}$. An observation is that we can obtain the function $d(r(q'), r(q))$ by shifting $d(r(q'), q)$ upwards by the value $d(q, r(q))$ (e.g., see Fig. 2). This implies that $g(r(q))$ cannot be strictly to the left of $g(q)$. The lemma thus follows. □

Lemma 13 essentially says that if we consider all points $q \in Q'_i$ from left to right, then $g(q)$ in Q'_{i-1} are also sorted on L from left to right. Due to this monotonicity property on $g(q)$, we can compute $g(q)$ and $\alpha(q)$ for all $q \in Q'_i$ in a total of $O(|Q'_{i-1}| + |Q'_i|)$ time by scanning the points of Q'_{i-1} from left to right. More specifically, suppose we have computed $g(q)$ and $\alpha(q)$ for some $q \in Q'_i$; then if $r(q)$ is still in Q'_i, we can compute $g(r(q))$ and $\alpha(r(q))$ by scanning the points of Q'_{i-1} starting from $g(q)$ to the right.

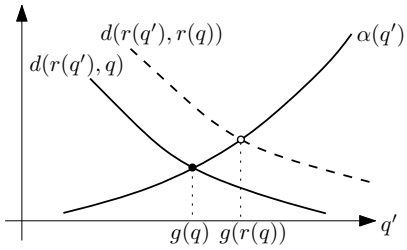


Figure 2: Illustrating the three functions $\alpha(q')$, $d(r(q'), q)$, and $d(r(q'), r(q))$ for $q' \in Q'_{i-1}$.

In this way, the value $\alpha(q_n)$ can be computed in $O(n+k)$ time, and an optimal solution can be found correspondingly. Hence, we have the following theorem.

Theorem 14 *The facility-location movement problem can be solved in $O(n+k)$ time.*

Acknowledgment. The authors would like to thank Minghui Jiang for bringing these problems to them. The research was supported in part by NSF under Grant CCF-1317143

References

[1] S. Cabello. Approximation algorithms for spreading points. *Journal of Algorithms*, 62:49–73, 2007.

[2] B. Chandra and M. Halldórsson. Approximation algorithms for dispersion problems. *Journal of Algorithms*, 38:438–465, 2001.

[3] E. Demaine, M. Hajiaghayi, H. Mahini, A. Sayedi-Roshkhar, S. Oveisgharan, and M. Zadimoghaddam. Minimizing movement. *ACM Transactions on Algorithms*, 5(3), 2009. Article No. 30.

[4] A. Dumitrescu and M. Jiang. Constrained k-center and movement to independence. *Discrete Applied Mathematics*, 159:859–865, 2011.

[5] A. Dumitrescu and M. Jiang. Dispersion in disks. *Theory of Computing Systems*, 51:125–142, 2012.

[6] J. Fiala, J. Kratochvíl, and A. Proskurowski. Systems of distant representatives. *Discrete Applied Mathematics*, 145:306–316, 2005.

[7] Z. Friggstad and M. Salavatipour. Minimizing movement in mobile facility location problems. *ACM Transactions on Algorithms*, 7(3), 2011. Article No. 28.

[8] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[9] L. G. Khachiyan. Polynomial algorithm in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20:53–72, 1980.

[10] S. Ravi, D. Rosenkrantz, and G. Tayi. Heuristic and special case algorithms for dispersion problems. *Operations Research*, 42(2):299–310, 1994.

[11] D. Wang and Y.-S. Kuo. A study on two geometric location problems. *Information Processing Letters*, 28:281–286, 1988.

Maximizing the Minimum Angle with the Insertion of Steiner Vertices

Shankar P. Sastry*

Abstract

We consider the problem of inserting a vertex inside a star-shaped input polygon at the location that maximizes the minimum angle in the resulting triangulation. An existing polynomial-time algorithm solves for the intersection of three polynomial surfaces (a prior paper indicates that these are eighth-degree polynomials) and computes the maxima of the curve of intersection of two such surfaces to solve the problem. We developed a similar technique through the geometric insight that at least two angles (typically, three) of the triangulation have to be identical at the optimal location. We combinatorially process the angles to compute the optimal location in each case. The worst-case complexity of the algorithm remains $O(n^3 \log n)$, but it is much easier to implement partly because our algorithm requires the solutions of an (at most) eighth-degree, *univariate* polynomial for each combination of the angles. We also modified the algorithm to lower the *expected* running time to $O(n^2)$ using a recursive, randomized algorithm for LP-type problems. We extend the algorithm by imposing constraints on the location of the Steiner vertex and solving the constrained optimization problem in a similar manner. We also extend the algorithm to simultaneously insert two vertices by considering all possible topologies and ensuring that the necessary conditions for local maxima are satisfied.

1 Introduction

We consider the problem of positioning a Steiner vertex that is connected to all the vertices of a star-shaped input polygon such that it maximizes the minimum angle in the resulting triangulation. The point has to be inside a convex feasible region so that no triangle lies outside the polygon [6, 7]. The algorithm that solves the problem may be used in Delaunay mesh refinment algorithms to insert addition vertices into a mesh at an optimal location or to carry out smoothing of the mesh vertices to improve the mesh quality.

In the context of mesh smoothing, Freitag and Plassman [6] developed a quadratic programming-based active-set approach to maximize the minimum angle by observing that it is a convex optimization problem with a nondifferentiable objective function. Recent research by Aronov et al. [2] has yielded polynomial-time algorithms to maximize the minimum angle. Their recent attempt [3] solves an LP-type problem by a computing the lower envelope of bivariate functions after solving eighth-degree polynomial equations. Their latest attempt [4] solves for the intersection of three bivariate polynomial surfaces or computes the maxima of the curve of intersection of two such surfaces. The surface intersection represents locations at which two or three angles of the triangulation are equal. These algorithms are described in Section 2. The active-set approach is easy to implement, but it is numerical in nature. The LP-type approach solves the problem exactly, but computing the lower envelope of bivariate functions is not easy. To obtain a maximum of the curve of intersection of two surfaces or a point of intersection of three surfaces is also hard in comparison with solving a univariate polynomial equation.

We use concepts from nondifferentiable optimization to develop our algorithm. We infer that the minimum angle is shared by two or more angles because it is always possible to improve the minimum angle (at the cost of the "better" angles) by appropriately moving the vertex. A brief background on this topic is presented in Section 3. Our approach is combinatorial and is similar to [4]. We take all combinations of possible locations of the minimum angles and return the "best" location of the Steiner vertex. Our algorithm is described in detail in Section 4. We observe that the number of possible combinations is $O(n^3)$, where n is the number of segments in the polygonal cavity, and we determine the optimal location by considering only those points where the gradients are suitably directed. We improve the expected running time to $O(n^2)$ using a recursive, randomized technique adapted from Clarkson's algorithm [5]. We also provide an algorithm for a generalized constrained optimization problem. We also extend the algorithm for the insertion of two vertices. These extensions are discussed in Section 5.

Our work has also provided some insight into local mesh quality improvement by vertex movement. Section 6 includes a discussion about improvements to our algorithm and future work.

*Scientific Computing and Imaging Institute, University of Utah, sastry@sci.utah.edu

2 Related Work

In this section, we discuss two of the most relevant algorithms in the evolution of our algorithm. The first algorithm is by Aronov and Yagnatinsky [4]. Although we developed our algorithm independently, their algorithm is similar to ours. The main difference is the geometric intuition; we believe ours is simpler to comprehend, and, therefore, easier to extend to tougher cases, as we shall see in Section 5. The second algorithm is the active set-based technique for mesh quality improvement [6]. We use concepts from this algorithm to accelerate our algorithm for realistic cases.

2.1 The Bivariate Surface Algorithm

Aronov and Yagnatinsky [4] begin by defining a feasible region in the domain where it is possible to construct a triangulation whose minimum angle is some z. The feasible region is bounded by circular arcs (that circumscribe an edge so that the angle subtended at the circumference is z) and line segments (that emanate at an angle z from the edges). The feasible region may be empty. For each value of z, we have arcs and lines on the domain. When they are lifted in the third dimension for all values of z, they form surfaces. In a prior paper [3], they have used eighth-degree, bivariate polynomials to define similar surfaces. They then claim that the optimal point is at a location where three surfaces intersect at a point or the maxima of curve formed by the intersection of two surfaces. The feasible maximum over the points returns the optimal triangulation. We use the gradients of the objective function to prove the claim in the language of nondifferentiable optimization and compute those points on the 2D plane. Our functions are eighth-degree, univariate polynomials.

2.2 The Active Set Method

Freitag and Plassman [6] solve the problem through vertex movement dictated by a numerical optimization algorithm. At a given location, they define the active set as the set of angle(s) with the minimum value. They use the gradient of the function defining the angle(s) to compute the direction in which the vertex should move in order to optimize it. The descent direction is computed by considering all convex combinations of active set gradients and choosing the one that minimizes the magnitude. If the active set changes during the vertex movement, the gradients and descent direction are recomputed. Similarly, we consider only those combinations of angles in which all the current active-set angle(s) are present.

3 Background

Since our objective function is nondifferentiable, we provide a background on necessary conditions[1] for the constrained optimization of such functions [8], which will also help us extend the algorithm for more complicated cases. In the subsequent sections, we will provide a geometric intuition behind the material presented here.

We assume that our objective function, $f(x)$, is defined as a minimum over a set of functions $f_i(x)$, $1 \leq i \leq n$. We define an active set at a location x as the maximum subset of functions whose value is equal to $f(x)$. A vector, $\vec{g}$ is defined as a subgradient of f at x if $\exists$ an ϵ neighborhood such that $f(x+\epsilon) - f(x) \geq \vec{g}^T \epsilon$. If only one function is present in the active set, the gradient of that function is the subgradient of f at x. If multiple functions are present, any *convex* combination of the gradients of the functions is also the subgradient of f at x. The set of all subgradients is called the subdifferential, $\partial f(x)$, of f at x.

For the unconstrained case, the necessary condition for x^* to be a local optimum is $0 \in \partial f(x^*)$. In other words, some convex combination of gradients of the functions in the active set should vanish, i.e., if $d < n+1$ functions are present in the active set in an $n-$dimensional space, the gradients should lie on some $d-$dimensional hyperplane and span both sides of all other $d-$dimensional hyperplanes at the origin. If $d \leq n+1$, the origin should lie inside the convex hull of the gradient vectors, which is easy to verify for any dimension.

For the constrained case, where $h_i(x) \leq 0$, $i \in [1,k]$, are the set of inequality constraints, a local optimum should satisfy the Karush-Kuhn-Tucker (KKT) conditions for nondifferentiable functions, i.e., $h_i(x^*) \leq 0$, $\lambda_i^* \geq 0$, $\lambda_i^* h_i(x^*) = 0$, and $0 \in \partial f(x^*) - \sum_{i=0}^{(k-1)} \lambda_i \partial h_i(x^*)$. These conditions imply that some convex combination of the gradients of the function(s) of the active set and the gradients of all the active constraint(s) should vanish.

It can also be shown that if some convex combination of m $n-$dimensional vectors vanishes (where $m \geq n+1$), there exists a set of $n+1$ vectors (from the original set of m vectors) for which some convex combination also vanishes. Thus, it is sufficient to consider all possible combinations of $n+1$ vectors rather than considering all combinations of $m > n+1$ vectors.

4 Algorithm

Based on the theory presented in the last section, we need to compute all possible locations where a set of angles are equal and some convex combination of their gradients vanishes. Since this is a 2D problem, we need to

[1] They hold true for a maximum, minimum, and saddle points.

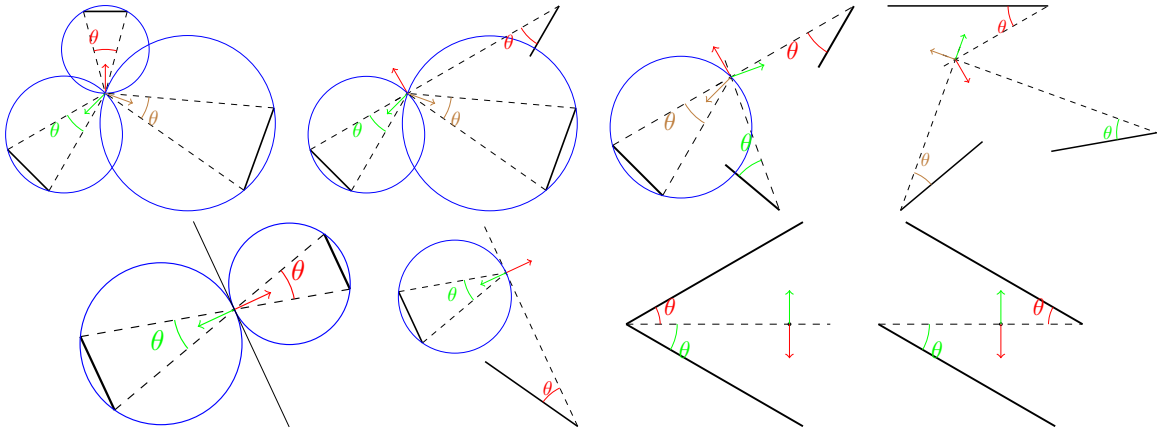


Figure 1: All possible ways in which a local maximum can occur. The thick edges are the input edges, and the dashed edges are the edges in the final triangulation. The circles (and some dashed edges) are potential locations of the Steiner vertex that result in an angle θ at some vertex. The arrows indicate the gradient for each angle. Notice how their convex combination can result in a zero vector.

only consider all possible sets of (no greater than) three angles in order to determine all possible local maxima. We also know that it is a convex optimization problem, so there exists only one local maximum, which is also the global maximum. Our algorithm considers all cases combinatorially and computes the global maximum.

In Fig. 1, we present all possible ways in which a set of three angles in the triangulation can be equal. If the input polygon has n sides, there are $2n$ angles on the edges and n angles at the Steiner vertex. The sets of two and three angles can be composed of either type of angles. All seven (eight, if the last two cases involving two angles on the edges are considered different) ways in which we can achieve equiangular configurations are shown in Fig. 1 along with the gradients for each of the angles.

In the top row of Fig. 1, we consider cases where three angles are equal. Consider the locus of points that results in an angle being θ. For angles at the Steiner vertex, the locus is a circle, and for angles at the edge, the locus is a straight line. The values of θ for which the three curves are concurrent need to be computed. In the bottom row, we consider cases where two angles are equal, where their gradients must be anti-parallel, i.e., the curves must be intersect tangentially.

We will now describe how a univariate polynomial function can be constructed with $\lambda = \cot\theta$ as the only variable. This is also called a rational univariate representation. The solution of the equation gives us the θ values at which a maximum can occur. The actual polynomial, however, is tedious to write down, but it is straightforward, so we leave that as an algebra exercise. First, we need to compute the circle that subtends an angle θ at the chord formed by an edge. The center of the circle should be at a distance λl from the mid point of the edge, where $2l$ is its length (see Fig. 2). The line that is at an angle θ can also easily be computed as shown in the figure. Second, as we now have the equations of the circles and/or lines (as functions of x, y, and λ), we compute their points of intersection in a pairwise manner. The point of intersection of two lines (as a function of λ) can be computed as a function of second-degree polynomials. The points of intersection of a line and a circle can be similarly computed. For two circles, however, we compute the equation of the line on which the two circles intersect.

We will consider each case in Fig. 1 separately, going from left to right (top row first). For the three angles at the Steiner vertex, we have the equations for three lines. We compute their points of intersection in a pairwise manner and equate one of their coordinates to get an eight-degree polynomial in λ. For two angles at the Steiner vertex and one at an edge, we have two lines whose point of intersection can be computed as a function of λ. We solve for the λ at which the angle subtended by one of the edges is θ. For an angle at the vertex and two at the edges, we are given a point. We again solve for λ at which the angle subtended by the edge is θ. When the angles are on three edges, we have three pairwise points of intersection. We equate the coordinates of the points of intersection to get an equation in λ. For the first two cases in the bottom row, the value of λ can be computed by ensuring that the roots of the resulting quadratic equation are equal because they intersect tangentially. The last two cases, however, are degenerate, and therefore, are not considered here; they are considered when three angles are considered.

To determine the maximum, we consider all sets of two and three angles and look for points at which their values are equal *and* some convex combination of their

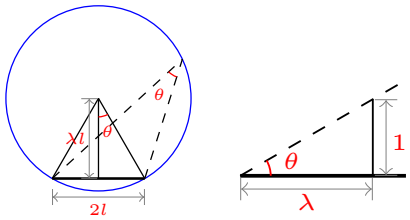


Figure 2: For a given $\lambda = \cot\theta$, the locus of points is either a circle or a line.

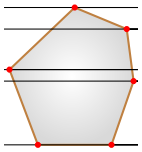


Figure 3: Preprocessing: determine the feasible region [7], and decompose the region into triangles/trapezoids in $O(n)$ time. For each point, determine the potential triangle/trapezoid using the binary search technique in $O(\log n)$ time, and determine if the point is inside or outside the triangle/trapezoid.

gradients vanish, which takes $O(n^3)$ time. Points that are outside the convex feasible region of the star-shaped polygon need to be discarded. It is possible to determine if a point is inside or outside the feasible region in $O(\log n)$ time after a line sweeping-based preprocessing algorithm (see Fig. 3) that is carried out at the beginning. Our global maximum has to occur at one of these locations that has not been discarded. The objective function is convex inside the feasible region [6]. At the points corresponding to smaller λ values (larger θ values), at least one of the angles associated with the largest λ value will be of less than the optimal value. Thus, the largest λ (smallest θ) corresponds to the global maximum. If not, it results in a contradiction. The worst-case complexity is, therefore, $O(n^3 \log n)$.

To improve the expected time, we adapt Clarkson's recursive algorithm [5] for our problem in which our $O(n^3)$ algorithm is a subroutine that terminates the recursion when n is small. In this algorithm, a subset of angles (the size of the subset is proportional to $O(\sqrt{n})$) is randomly chosen, and the optimal vertex location for those angles is recursively found. If the number of angles is small, our algorithm is used to carry out the optimization. For the optimal location found, if the minimum angle among the chosen subset also the minimum angle when all other angles are considered, we are done. If not, $O(\sqrt{n})$ more angles are randomly chosen from the set of angles that are smaller than the optimal angle for the chosen subset, and the algorithm is repeated. Although Clarkson's algorithm takes linear time for linear programming problems, as shown by Amenta et al. [1], the expected running time for our problem is quadratic because computing the minimum angle itself takes linear time.

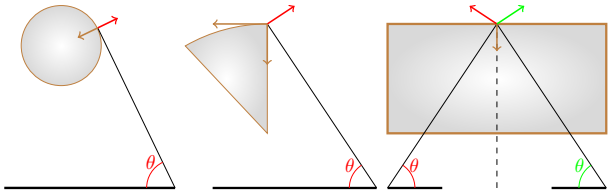


Figure 4: Three possible locations of a local maximum in a constrained problem. The vertex is constrained to be in the gray region. The gradients of the constraint(s) and the angle(s) are also shown.

5 Extensions

5.1 Constrained Optimization

The motivation for solving the constrained optimization problem is to gurantee termination of Delaunay refinement algorithm. Delaunay refinement terminates only if the newly inserted vertex is at least at a certain distance from other vertices in the domain. Thus, our inserted vertex must be sufficiently distant from the edges and vertices of the input polygon. Here, we consider a set of more general geometric constraints. Based on the theory of constrained optimization of nondifferentiable functions, we need to compute the locations where some convex combination of the gradients of the angle(s) and the constraint(s) vanishes. The three ways in which this can happen are depicted in Fig. 4. If the locus of points that results in a constant angle is tangential to a curve representing a constraint, the point of intersection is of interest. Similarly, when two constraint curves intersect and the gradient of an angle is suitably directed, a local maximum is present at the point of intersection. Also, the locus of points where two angles of the resulting triangulation are equal may meet a constraint. If the gradients are suitably directed, we have a local maximum. If there are $O(n)$ angles and $O(m)$ constraints, it takes $O(n^2m + nm^2)$ time to consider all cases. These cases are in addition to the cases for the unconstrained problem. For each case, it takes $O(m)$ time to determine if the location is within the constraints and $O(n)$ time to compute the minimum angle. This can be accelerated using the techniques described in the previous section. If the constraints are nonconvex or disjoint, we have to consider all combinations. We may also cache the sorted order of the angles in previous combinations to quickly discard a potential solution.

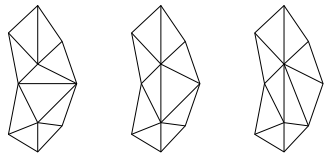


Figure 5: Three of the possible $O(n^2)$ topologies for insertion of two Steiner vertices.

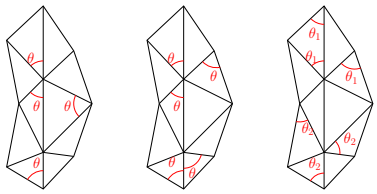


Figure 6: The possible choice of angles for a given topology; (left) underdetermined system of equations; (middle) zero-dimensional system of equations; (right) overdetermined system of equations.

5.2 Insertion of Two Vertices

In the case of inserting just one vertex, the topology of the resulting triangulation (connectivity of the vertices) is fixed, i.e., the Steiner vertex is connected to all the vertices of the input polygon. If two Steiner vertices are to be inserted, we have to consider multiple topologies as shown in Fig. 5. The two Steiner vertices may or may not be connected to each other, and each Steiner vertex is connected to a subset of polygon vertices. Exactly two vertices are present in both subsets (to avoid overlapping trianglesr). The common vertices divide the polygon into two sides, and each Steiner vertex is connected to all polygon vertices on its side. There are $O(n^2)$ such topologies to consider.

For the insertion of just one vertex, we actually solve a system of trivariate polynomials in x, y, and λ that we managed to reduce to a single variable. The system consists of three equations when three angles are being equated. When two angles are being equated, we still have three equations: two from the angles and an additional equation constraining the gradients to be anti-parallel. In the new case, we have a system of equations with five variables (x_1, y_1, x_2, y_2, and λ). Thus, five polynomial equations are needed to solve the problem. We also know that three equations need to be associated with the angles of triangles at each of the two Steiner vertices at the local maxima. Our choice of the angles should respect these conditions.

Consider the case where the Steiner vertices are not connected, which reduces to two instances of the single vertex insertion problem for each such topology. Thus, the optimal locations can be computed in $O(n^5 \log n)$ in the worst case.

Let us now consider the more interesting case where the Steiner vertices are connected. In this case, we need to choose two or three angles in the triangles associated with each of the Steiner vertices. Fig. 6 shows three possible ways this can be achieved. On the left, two of the angles chosen are common to both Steiner vertices, and only four angles have been chosen in total, which results in an underdetermined system of polynomial equations, which have an infinite number of solutions. As a result, the optimal value of λ will come at the cost of some other triangle. In the middle, only one angle is common to the Steiner vertices, and five angles have been chosen altogether. Thus, we have five equations with five variables, which is called a zero-dimensional system because it has a finite number of solutions. We have to consider all such cases in our algorithm. On the right, we have chosen six angles with each Steiner vertex being associated with three of them. This is an overdetermined system if we insist that the value of θ be the same for all angles. If they are different (as in the figure), we are solving a problem that is similar to the case where the two Steiner vertices are not connected. Note that the discussion above also holds when only two angle are chosen for a vertex because we have an additional equation in the form of their gradients being anti-parallel.

In order to determine the computational complexity, we have to analyze the number of ways in which these angles can be chosen. Let the degree of the vertices be d_1 and d_2, $d_1 + d_2 = n + 2$. We do not consider the case on the left in Fig. 6 because it results in an underdetermined system. For the case in the middle, there are only six ways in which the common angle can be chosen. There are $O(d_1^2)$ and $O(d_2^2)$ ways in which the other two angles can be chosen. The number of combinations is $O(n^4)$. The overall complexity of the number of possible cases for a given topology is $O(n^4)$ The number of possible topologies is $O(n^2)$. Thus, we consider $O(n^6)$ cases in total. Since feasible regions are hard to find in the fourth dimension, we carry out a linear search for each case to determine the smallest angle and, subsequently, the global maximum in $O(n^7)$ time. For the case on the right, the angles in the common triangles are not being considered. Therefore, they are two independent subproblems as in the case where the vertices are not connected. If the minimum angle is part of the common triangles, it will be handled in one of our earlier cases. If not, it can be handled in the case where the two vertices are not connected. Therefore, we do not consider this case in our analysis here.

In order to improve the expected running time, we may use a method similar to Clarkson's algorithm that we used for the single vertex insertion case. Note that it has to be used for every possible topology to obtain the most optimal location. We can extend this algorithm to get a "near-optimal" solution by considering changes in the topology (flip edges) only if it improves an existing

triangulation, but we cannot guarantee anything about the approximation of the solution obtained.

It is also possible to construct a univariate polynomial that is equivalent to the system of five pentavariate polynomials described earlier in this section. Consider the two angles of a vertex that do not belong to the common triangles. The locus of points that results in equal angles at the two corners can be described as a parametric equation in λ for both vertices, which fixes the location of the two vertices as a function of λ. We get a polynomial equation by enforcing the condition that the angle in the common triangle is also $\cot^{-1}\lambda$. In the case where only two angles are equated at a vertex, we need an additional parameter, say, t, to fix the location of one of the vertices. It is possible to compute the location of the other vertex as a function of λ and t and impose the angle condition as well as the gradient condition to obtain a system of two bivariate polynomials in λ and t.

6 Conclusion and Future Work

We used the theory of nondifferentiable optimization to develop our algorithm and extend it to tougher problems. Due to the firm theoretical ground, this research can be extended to investigate the complexity of the optimization of angles when an arbitrary number of vertices are inserted in 2D as well as 3D meshes. It can easily be seen that there is a combinatorial explosion of possible mesh topologies as the number of vertices increases, and they require solutions of larger systems of multivariate polynomials. A practical algorithm should account for this and prune the search space appropriately. We think that the prior research in mesh quality improvement [9] will help us in analyzing these problems. Mesh quality improvement techniques typically optimize one vertex at a time. Since we now know that two or three angles in 2D meshes (two, three, or four angles in 3D meshes) are identical in an optimal patch, reordering of vertices over which the optimization is carried out will bring about greater improvement in mesh quality. Prior research [9] has not taken advantage of this property. Our next research will look in this direction.

As mentioned in Section 4, the expected running time of our algorithm is improved by choosing a random subset of angles from the given angles in each subroutine. It is likely that the minimum angle occurs at the angle opposite to the shortest input edge or the smallest angle in the input polygon. Perhaps, assigning a higher probablity for such angles may yield faster results. Also, given an optimal Steiner vertex location for a subset of angles, we may choose additional angles by assigning a higher probablity to those angles (in the original set) that are smaller. A theoretical or emperical analysis of the improvement in the expected running time due to such modifications is also an interesting topic of research.

Acknowledgment

The work of the author was supported in part by the NIH/NIGMS Center for Integrative Biomedical Computing grant 2P41 RR0112553-12 and a grant from ExxonMobil. The author would also like to thank Christine Pickett, an editor at the University of Utah, for proofreading and finding numerous typos in the paper.

References

[1] N. Amenta, M. Bern, and D. Eppstein. Optimal point placement for mesh smoothing. *J. Algorithms*, 30(2):302 – 322, 1999.

[2] B. Aronov, T. Asano, and S. Funke. Optimal triangulations of points and segments with steiner points. *Int. J. Comput. Geom. Ap.*, 20(01):89–104, 2010.

[3] B. Aronov and M. V. Yagnatinsky. How to place a point to maximize angles. In *Proc. of the 25th Canadian Conference on Computational Geometry*, 2013.

[4] B. Aronov and M. V. Yagnatinsky. Quickly placing a point to maximize angles. In *Proc. of the 26th Canadian Conference on Computational Geometry*, 2014.

[5] K. L. Clarkson. Las vegas algorithms for linear and integer programming when the dimension is small. *J. ACM*, 42(2):488–499, Mar. 1995.

[6] L. A. Freitag and P. Plassmann. Local optimization-based simplicial mesh untangling and improvement. *Int. J. Numer. Meth. Engrg.*, 49(1-2):109–125, 2000.

[7] D. T. Lee and F. P. Preparata. An optimal algorithm for finding the kernel of a polygon. *J. ACM*, 26(3):415–421, July 1979.

[8] A. P. Ruszczynski. *Nonlinear optimization*. Princeton University Press, 2006.

[9] S. M. Shontz and P. Knupp. The effect of vertex reordering on 2D local mesh optimization efficiency. In *Proc. of the 17th International Meshing Roundtable*, pages 107–124. Springer Berlin Heidelberg, 2008.

An Output-Sensitive Algorithm for Computing the s-Kernel*

Leonidas Palios†

Abstract

Two points p, q of an orthogonal polygon P are s-visible from one another if there exists a staircase path (i.e., an x- and y-monotone chain of horizontal and vertical line segments) from p to q that lies in P. The s-kernel of P is the (possibly empty) set of points of P from which all points of P are s-visible.

We are interested in the problem of computing the s-kernel of a given orthogonal polygon (on n vertices) possibly with holes. The problem has been considered by Gewali [1] who described an $O(n)$-time algorithm for orthogonal polygons without holes and an $O(n^2)$-time algorithm for orthogonal polygons with holes. The problem is a special case of the problem considered by Schuierer and Wood [5], whose work implies an $O(n)$-time algorithm for orthogonal polygons without holes and an $O(n \log n + h^2)$-time algorithm for orthogonal polygons with $h \geq 1$ holes.

In this paper, we give a simple output-sensitive algorithm for the problem. For an n-vertex orthogonal polygon P that has h holes, our algorithm runs in $O(n + h \log h + k)$ time where $k = O(1 + h^2)$ is the number of connected components of the s-kernel of P. Additionally, a modified version of our algorithm enables us to compute the number k of connected components of the s-kernel in $O(n + h \log h)$ time.

Keywords: s-kernel, visibility, orthogonal polygon, output-sensitive algorithm.

1 Introduction

A polygon is *orthogonal* if its edges are either horizontal or vertical; an edge e of such a polygon is a N-edge (S-edge, E-edge, and W-edge, resp.) if the outward-pointing normal vector to e is directed towards the North (South, East, and West, resp.); see Figure 1(a). Of particular importance are the *dents*, i.e., edges whose endpoints are reflex vertices of the polygon, characterized as N-dents, S-dents, E-dents, and W-dents (see Figure 1(a)); the dents are a measure of non-convexity of an orthogonal polygon.

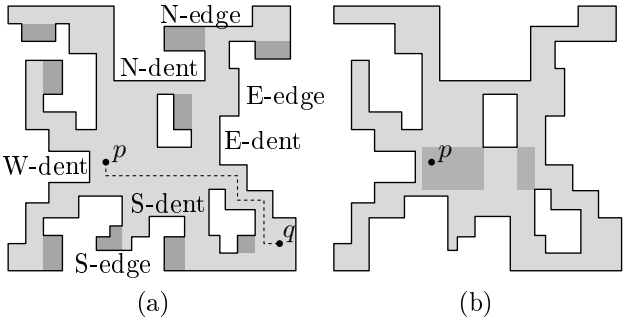


Figure 1: (a) Illustration of the main definitions (the portions of the polygon not s-visible from p are shown dark); (b) the s-visibility polygon of p, which is an s-star, with its s-kernel shown darker.

A set of points is *x-monotone* (*y-monotone*, resp.) if its intersection with any line perpendicular to the x-axis (y-axis, resp.) is a connected set. A *staircase path* is a chain of horizontal and vertical segments that is both x- and y-monotone.

Then, two points p, q of an orthogonal polygon P are *s-visible* from one another if there exists a staircase path from p to q that lies in P (Figure 1(a) shows two such points p and q). The set of points that are s-visible from a point p form the *s-visibility polygon* of p. The *s-kernel* of P is the (possibly empty) set of points of P whose s-visibility polygon is equal to P, i.e., the set of points from which all points of P are s-visible (the s-kernel of the orthogonal polygon in Figure 1(b) is shown darker); note that the s-kernel may be disconnected. An orthogonal polygon is an *s-star* if it has non-empty s-kernel. The orthogonal polygon in Figure 1(b) is an s-star; as can be seen in the figure, an s-star may have holes.

Visibility problems are closely related to reachability and to covering problems. The s-kernel of a polygon is the set of points from which all other points of the polygon can be reached by means of x- and y-monotone paths. So, if a robot restricted to move parallel to the coordinate axes is considered to "guard" a point p in an orthogonal polygon provided that it can get to p along a monotone path, then the polygons that can be "guarded" are those with non-empty s-kernel. Additionally, because the s-stars may be highly non-convex

* This research has been co-financed by the European Union (European Social Fund - ESF) and Greek national funds through the Operational Program "Education and Lifelong Learning" of the National Strategic Reference Framework (NSRF) - Research Funding Program: THALIS UOA (MIS 375891) - Investing in knowledge society through the European Social Fund.

†Department of Computer Science and Engineering, University of Ioannina, Greece, palios@cs.uoi.gr

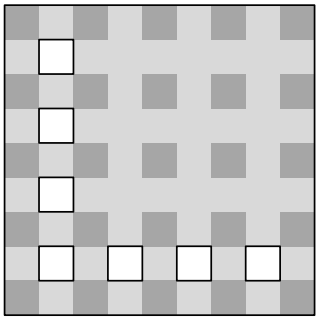


Figure 2: An orthogonal polygon with $\Theta(n)$ holes whose s-kernel (shown darker) has $\Theta(n^2)$ size.

(see Figure 1(b)), a minimum cover of an orthogonal polygon using s-stars (see [3] for an algorithm) is expected to involve a smaller number of pieces compared to other minimum covers. (Note also that in the usual sense of visibility, the kernel of a polygon with holes is empty and that the kernel of an n-vertex polygon can be computed in $O(n)$ time [2].)

Gewali [1] has considered the problem of computing the s-kernel of an orthogonal polygon; he described an $O(n)$-time algorithm for an orthogonal polygon without holes and an $O(n^2)$-time algorithm for orthogonal polygons with holes where n is the number of vertices of the polygon. He also showed that the latter algorithm is worst-case optimal since the s-kernel of an orthogonal polygon with holes may be of $\Theta(n^2)$ size; Figure 2 shows an orthogonal polygon with $\Theta(n)$ holes whose s-kernel has $\Theta(n^2)$ size [1]. Gewali used this result to give an $O(n \log n)$-time algorithm for recognizing whether an orthogonal polygon with holes is an s-star.

Schuierer and Wood [5] studied the notion of $\mathcal{O}$-visibility, that is, visibility along a set $\mathcal{O}$ of orientations and gave an $O(n \log |\mathcal{O}|)$-time algorithm for the computation of the $\mathcal{O}$-kernel of an orthogonal polygon without holes and an $O(n(\log |\mathcal{O}| + \log n) + h(|\mathcal{O}| + h))$-time algorithm for polygons with h holes, respectively. Their algorithms imply $O(n)$-time and $O(n \log n + h^2)$-time algorithms for the s-kernel of orthogonal polygons without holes and of orthogonal polygons with $h \geq 1$ holes, respectively.

In this paper, we present a simple output-sensitive $O(n + h \log h + k)$-time and $O(n)$-space algorithm for computing the s-kernel of an orthogonal polygon having n vertices, $h \geq 0$ holes, and an s-kernel consisting of k connected components; as we will see $k = O(1 + h^2)$. The algorithm also enables us to count the number k of connected components of the s-kernel of such a polygon in $O(n + h \log h)$ time using $O(n)$ space (i.e., without computing the s-kernel), and thus we can determine if an orthogonal polygon is an s-star in the same time and space complexity.

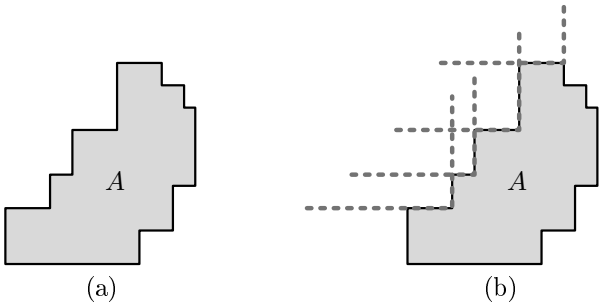


Figure 3: (a) An orthogonal polygon A that is orthogonally convex; (b) some of the quadrants whose union is equal to the complement of A.

2 Theoretical Framework

For an edge e of an orthogonal polygon P, let D_e be a small enough disk centered at the midpoint of e; we define the *in-halfplane* of e as the *closed* halfplane that is defined by the line supporting e and contains the portion of D_e that lies in P.

An orthogonal polygon is *orthogonally convex* if it is both x-monotone and y-monotone. For simplicity and since we deal with orthogonal polygons, in the following, an orthogonally convex orthogonal polygon will be referred to as "orthogonally convex polygon." Clearly, an orthogonally convex polygon cannot have dents. The reverse also works, and we have:

Observation 1 *An orthogonal polygon is orthogonally convex if and only if it has no dents.*

Therefore, the boundary of an orthogonally convex polygon consists of x- and y- monotone chains connecting the leftmost edge of the polygon, to the uppermost edge, to the rightmost edge, to the bottommost edge, and back to the leftmost edge (see Figure 3(a)); any one of these chains may degenerate to a single point. Moreover, it is important to observe that the following lemma holds.

Lemma 1 *Let A be an orthogonally convex polygon having n vertices. Then, the complement of A can be expressed as the union of $\Theta(n)$* open *quadrants.*

The lemma follows from the fact that the complement of an orthogonally convex polygon is equal to the union of as many open quadrants as the polygon's reflex vertices (for a reflex vertex, the corresponding quadrant is the complement of the union of the in-halfplanes of the edges incident on the reflex vertex) plus 4 more (one for each of the leftmost, topmost, rightmost, bottommost edge); Figure 3(b) shows the quadrants belonging to the complement of an orthogonally convex polygon that are associated with the boundary chain from the leftmost

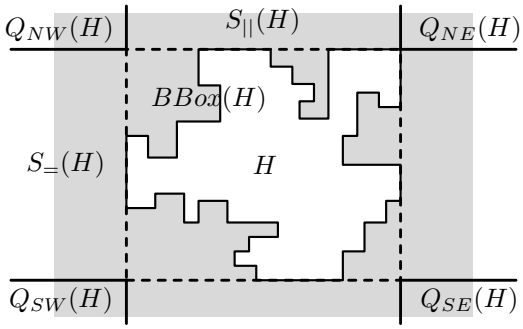


Figure 4: Illustration of $BBox(H)$, $S_{=}(H)$, $S_{||}(H)$, $Q_{NW}(H)$, $Q_{NE}(H)$, $Q_{SE}(H)$, and $Q_{SW}(H)$ for a hole H.

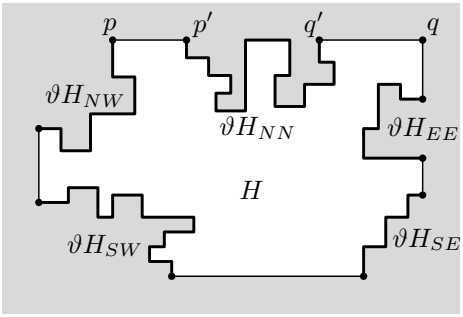


Figure 5: Illustration of the boundary subchain notation for a hole H (the subchain ϑH_{NE} is point q; no $\vartheta H_{WW}, \vartheta H_{SS}$ exist).

to the topmost edge (the remaining three chains contribute additional quadrants in a similar fashion). As a result, the total number of quadrants is nearly half the number of vertices of the polygon.

2.1 The s-kernel of orthogonal polygons without holes

The algorithm of Gewali [1] computes the s-kernel of an orthogonal polygon P without holes by intersecting P with the in-halfplanes of the lowermost N-dent, the rightmost W-dent, the topmost S-dent, and the leftmost E-dent. This implies the following result.

Lemma 2 *Let P be an orthogonal polygon without holes that has n vertices. The s-kernel of P is an orthogonally convex polygon of $O(n)$ size.*

2.2 Notation for orthogonal polygons with holes

Let D be an orthogonal polygon or a hole in an orthogonal polygon. Then, we define:

ϑD**:** the boundary of D;

$BBox(D)$**:** the smallest axes-aligned rectangle containing D.

Additionally, for a hole H, we have:

$S_{=}(H)$**:** the smallest *open* horizontal strip containing the interior of H;

$S_{||}(H)$**:** the smallest *open* vertical strip containing the interior of H;

$Q_{NW}(H)$**:** the *closed* axes-aligned quadrant that is the complement of the union of the interiors of the in-halfplanes of the top and left edges of the rectangle $BBox(H)$ (see Figure 4) — similarly, we define $Q_{NE}(H)$, $Q_{SW}(H)$, and $Q_{SE}(H)$;

ϑH_{NW}**:** the part of the boundary of H in counterclockwise direction from the leftmost among the points of H with maximum y-coordinate to the topmost among the points of H with minimum x-coordinate (see Figure 5) — similarly, we define ϑH_{NE}, ϑH_{SW}, and ϑH_{SE};

ϑH_{NN}**:** let p, q be the leftmost and rightmost, resp., vertices of H with maximum y-coordinate; if p, q are adjacent in H then no ϑH_{NN} exists; otherwise, if p' (q', resp.) is the other endpoint of the horizontal edge incident on p (q, resp.), ϑH_{NN} is the part of the boundary of H connecting p' and q' after the edges pp' and qq' have been removed (see Figure 5) — similarly, we define ϑH_{WW}, ϑH_{SS}, and ϑH_{EE}.

The following lemma provides important properties of the s-kernel of orthogonal polygons with holes.

Lemma 3 *Let H be a hole of an orthogonal polygon P. Then:*

(i) No point of the strips $S_{=}(H)$ and $S_{||}(H)$ belongs to the s-kernel of P.

(ii) If ϑH_{NW} is not a single point, then no point of the quadrant $Q_{SE}(H)$ belongs to the s-kernel of P. Moreover:
if ϑH_{NW} contains a S-dent or an W-dent, then no point of the quadrant $Q_{SW}(H)$ belongs to the s-kernel of P (see Figures 6 and 7);
if ϑH_{NW} contains a N-dent or an E-dent, then no point of the quadrant $Q_{NE}(H)$ belongs to the s-kernel of P;
if ϑH_{NW} contains a N-dent or an W-dent, then no point of the quadrant $Q_{NW}(H)$ belongs to the s-kernel of P.
Similar results hold for the boundary subchains ϑH_{NE}, ϑH_{SW}, and ϑH_{SE}.

(iii) If the boundary of H contains a subchain ϑH_{NN}, then no point of the quadrants $Q_{SW}(H) \cup Q_{SE}(H)$

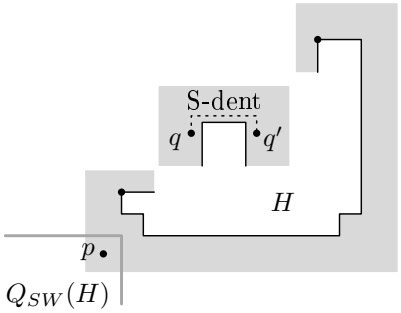


Figure 6: If ϑH_{NW} contains a S-dent, then no point of the quadrant $Q_{SW}(H)$ belongs to the s-kernel.

belongs to the s-kernel of P. Moreover:
if ϑH_{NN} contains a N-dent or an E-dent, then no point of the quadrant $Q_{NE}(H)$ belongs to the s-kernel of P;
if ϑH_{NN} contains a N-dent or an W-dent, then no point of the quadrant $Q_{NW}(H)$ belongs to the s-kernel of P.
Similar results hold for the boundary subchains ϑH_{WW}, ϑH_{SS}, and ϑH_{EE}.

The fact that if ϑH_{NW} contains a S-dent, then no point of the quadrant $Q_{SW}(H)$ belongs to the s-kernel of P (statement (ii) of Lemma 3) follows from the fact that there cannot exist x- and y-monotone paths from any point p of $Q_{SW}(H)$ to both points q, q' on either side of the S-dent; see Figure 6. Figure 7 shows examples of subchains ϑH_{NW} containing a S-dent but no W-dents (at left) and an W-dent but no S-dents (at right).

Lemma 3 implies that for a hole H of the given orthogonal polygon P, points of the s-kernel of P belong to all, some, or none of the four quadrants $Q_{NW}(H)$, $Q_{NE}(H)$, $Q_{SW}(H)$, and $Q_{SE}(H)$.

3 Computing the s-Kernel

Let P be an orthogonal polygon. In [5], the s-kernel of an orthogonal polygon P with h holes is computed as the intersection of the s-kernel A of P after having ignored the holes in P with the *external* s-kernels of all of P's holes. However, as the external s-kernel of each hole contains a horizontal and a vertical strip, the intersection of the external s-kernels may result to computing a partial s-kernel of quadratic (in h) size, most of which may be clipped in the end. So, in order to get a faster algorithm, we need to avoid this. Hence, we process the horizontal strips $S_{=}(\ )$ of the holes separately, computing the horizontal "in"-strips, i.e., the horizontal strips that form the complement of the strips $S_{=}(\ )$; these strips thus contain the entire s-kernel of P (see Lemma 3(i)). We work similarly for the vertical strips $S_{||}(\ )$. Next, we clip the complement of the union U_Q of all the quadrants not containing points of the s-kernel

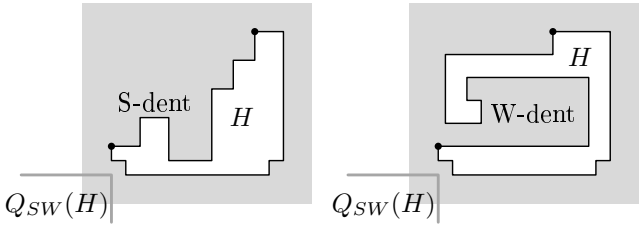


Figure 7: No point of the quadrant $Q_{SW}(H)$ belongs to the s-kernel if ϑH_{NW} contains: (left) a S-dent or (right) an W-dent.

(resulting from the holes as described in Lemma 3(ii) and (iii)) about the polygon A. Finally, we intersect the clipped complement of U_Q with the vertical and horizontal "in"-strips. A detailed description of the algorithm is given in Algorithm s-KERNEL below.

Algorithm s-KERNEL(P)
Input : an orthogonal polygon P possibly with holes
Output: the s-kernel of P

---

1. compute the s-kernel A of the orthogonal polygon bounded only by P's outer boundary component;
   if P has no holes
   then return A as the s-kernel of P;
   exit;
   let $x_{min}, x_{max}, y_{min}, y_{max}$ be the extreme values of x- and y-coordinates of the bounding rectangle $BBox(A)$ of A;

2. process the holes of P to determine the (open) strips and (closed) quadrants that do not contain points of the s-kernel of P (see Lemma 3);
   if all 4 quadrants $Q_{NW}(H)$, $Q_{NE}(H)$, $Q_{SW}(H)$, $Q_{SE}(H)$ of a hole H do not contain points of the s-kernel of P
   then print("The s-kernel of P is empty.");
   exit;
   let $\mathcal{C}_{=}$ ($\mathcal{C}_{||}$, $\mathcal{C}_Q$, resp.) be the set of horizontal strips (vertical strips, quadrants, resp.) not containing points of the s-kernel of P;

3. {*process the strips in $\mathcal{C}_{=}$ and $\mathcal{C}_{||}$*}
   compute the union of the horizontal strips in $\mathcal{C}_{=}$, clip it about the range $[y_{min}, y_{max}]$, and store it in a y-ordered array $M_{=}$ of alternating *closed* "in"-strips (containing points of the s-kernel) and *open* "out"-strips (not containing points of the s-kernel); work similarly for the vertical strips in $\mathcal{C}_{||}$ using the range $[x_{min}, x_{max}]$, producing an x-ordered array $M_{||}$;

4. {*process the quadrants in $\mathcal{C}_Q$*}
   compute the union U_Q of all the quadrants in $\mathcal{C}_Q$, and clip its complement about the boundary of the

polygon A computed in Step 1;
if the clipped complement of the union U_Q of the quadrants in $\mathcal{C}_Q$ is empty
then **print**("The s-kernel of P is empty.");
exit;

5. **for** each polygon B_j in the clipped complement of U_Q in y-order **do**
for each horizontal "in"-strip I intersecting B_j in y-order **do**
compute the boundary $\vartheta B_j(I) = \vartheta B_j \cap I$;
locate a leftmost point of $\vartheta B_j(I)$ in the vertical strips array $M_{||}$;
walk on $\vartheta B_j(I)$ and in $M_{||}$ until a rightmost point of $\vartheta B_j(I)$ is found, printing each polygon (if any) contributed by $B_j \cap I$ and each "in"-strip of $M_{||}$;

(Note that the clipped complement of the union U_Q at the completion of Step 4 does not contain its entire boundary; it contains the edges that resulted from the clipping about A but it does not contain the edges that resulted from the quadrants in $\mathcal{C}_Q$.)

The correctness of Algorithm s-KERNEL follows from Lemma 3 and the fact that the s-kernel of P indeed is the intersection of polygon A (see Step 1) with the complement of the union of the collected strips and quadrants from the holes of P.

Time and Space Complexity. Let n and h be the number of vertices and holes of the input orthogonal polygon P. In the following lemma, we show that the complement of the union of axes-aligned quadrants has some very interesting properties; two polygons are *horizontally* (*vertically*, resp.) *separated* if no horizontal (vertical, resp.) line intersects both them.

Lemma 4 *(i) Each halfline bounding a quadrant in $\mathcal{C}_Q$ contributes at most one edge to the polygons forming the complement of the union U_Q of all the quadrants in $\mathcal{C}_Q$.*

(ii) The complement of U_Q consists of $O(h)$ orthogonally convex polygons that are horizontally and vertically separated and have $O(h)$ total size.

(iii) The clipped complement of U_Q computed upon completion of Step 4 of Algorithm s-KERNEL consists of $O(h)$ horizontally and vertically separated orthogonally convex polygons of $O(n)$ total size.

Lemma 4(iii) and the fact that the intersection of $O(h)$ horizontal strips with $O(h)$ vertical strips consists of $O(h^2)$ connected components of $O(h^2)$ total size imply the following corollary.

Corollary 5 *The s-kernel of an n-vertex orthogonal polygon that has h holes consists of $O(1+h^2)$ orthogonally convex polygons of $O(n+h^2)$ total size.*

The number of orthogonally convex polygons and the size of a s-kernel given in Corollary 5 are tight; a lower bound can be obtained by a generalization of the polygon in Figure 2.

The computation of the s-kernel in Step 1 takes $O(n)$ time [1] and so does the entire Step 1. Step 2 takes $O(n)$ time as well by traversing the boundary of each hole H of P, computing the subchains ϑH_{NW}, ϑH_{NW}, ϑH_{NE}, ϑH_{SW}, ϑH_{SE}, ϑH_{NN}, ϑH_{WW}, ϑH_{SS}, and ϑH_{EE}, determining whether they contain dents, and applying Lemma 3. The processing of the h horizontal strips in $\mathcal{C}_=$ in Step 3 can be completed in $O(h \log h)$ time by sorting them by non-decreasing bottom side and then processing them from bottom to top; similarly, the processing of the vertical strips in $\mathcal{C}_{||}$ takes $O(h \log h)$ time. In Step 4, we sort the quadrants in y-order in $O(h \log h)$ time and compute the right-bounding line of the union of quadrants $Q_{NW}(H_i)$ and $Q_{SW}(H_{i'})$ in $\mathcal{C}_Q$ and the left-bounding line of the union of quadrants $Q_{NE}(H_i)$ and $Q_{SE}(H_{i'})$ in $O(h)$ time. The complement of these unions is clipped about polygon A and by traversing their boundaries from top to bottom we compute the clipped complement of U_Q in $O(n)$ time. In total, Step 4 takes $O(n+h \log h)$ time. For Step 5, let t_j be the number of horizontal "in"-strips intersecting polygon B_j. Because the polygons in the clipped complement of U_Q are horizontally separated (Lemma 4(iii)), then any other polygon may be intersected only by the topmost or bottommost of these t_j "in"-strips. Then, the number of pairs of polygons and "in"-strips considered is $\sum_j t_j = \sum_j 2 + \sum_j (t_j - 2) = O(h)$ since the total number of polygons B_j (see Lemma 4(iii)) and the total number of "in"-strips are both $O(h)$. Thus, if the s-kernel of P has k connected components, Step 5 takes $O(n + h \log h + k)$ time by using binary search in the x-sorted array $M_{||}$ for locating leftmost points. Therefore:

Theorem 6 *Let P be an orthogonal polygon having n vertices and $h = O(n)$ holes. Algorithm s-KERNEL computes the s-kernel of P in $O(n + h \log h + k)$ time using $O(n)$ space where k is the number of connected components of the s-kernel of P.*

4 Computing the Number of Components of the s-Kernel

Algorithm s-KERNEL can be modified to help us compute the number k of connected components of the s-kernel of a given orthogonal polygon P; it suffices to modify Step 1 so that if P has no holes it returns 0 if A is empty and 1 otherwise, Steps 2 and 4 to return

0 if the s-kernel is found empty, and Step 5 as follows: for each polygon B_j and each horizontal "in"-strip I intersecting B_j, we compute a leftmost point a and a rightmost point z of the boundary of B_j in I, and locate them in the vertical strips array $M_{||}$ using binary search; then, depending on the indices of the strips to which a, z belong and whether they are "in"- or "out"-strips, we compute the number $\kappa(B_j, I)$ of "in"-strips (if any) between (and including) the strips of a and of z. The total number of components of the s-kernel of P is the sum of all the computed $\kappa(B_j, I)$.

The correctness of the modified algorithm follows immediately from the fact that for each polygon B_j and each horizontal "in"-strip I, each"in"-strip between (and including) the strips containing a and z contributes a separate component to the s-kernel of P. The complexity analysis of Step 5 of Algorithm s-KERNEL and the fact that $\kappa(B_j, I)$ can be computed in constant time after the strips containing a and z have been determined imply that the modified Step 5 takes $O(n+h\log h)$ time. Recall that the number k of connected components of the s-kernel may be as large as $\Theta(1 + h^2)$; see Corollary 5.

Therefore, we have:

Theorem 7 *Let P be an orthogonal polygon having n vertices and $h = O(n)$ holes. The described modified algorithm computes the number of connected components of the s-kernel of P in $O(n + h\log h)$ time using $O(n)$ space.*

5 Recognizing s-Stars

The modified algorithm of Section 4 to recognize whether a polygon P is an s-star (i.e., its s-kernel consists of at least 1 component) or not. A simpler version that does not compute the number k of components simply checks in Step 5 whether a and z fall in the same vertical "out"-strip of $M_{||}$; if they don't, then there exists a point in $B_j \cap I$ belonging to the s-kernel of P and hence P is an s-star (the algorithm can be augmented to return such a point as a certificate of its decision). If the above condition for a, z does not hold for all polygons B_j and "in"-strips I, then clearly the s-kernel of P is empty, and hence P is not an s-star.

Theorem 8 *Let P be an orthogonal polygon having n vertices and $h = O(n)$ holes. It can be decided whether P is an s-star in $O(n + h\log h)$ time using $O(n)$ space.*

6 Concluding Remarks

In this paper, we presented a simple output-sensitive algorithm for computing the s-kernel of an orthogonal polygon possibly with holes. The algorithm runs in $O(n + h\log h + k)$-time using $O(n)$ space, where n and h are the numbers of vertices and holes, respectively, of the input polygon, and k is the number of connected components of the computed s-kernel. Modifications of our algorithm enable us to compute the number k of connected components and to recognize if an orthogonal polygon is an s-star in $O(n + h\log h)$ time using $O(n)$ space.

Schuierer and Wood [5] mention that Rawlins in his PhD thesis [4] showed that the computation of the kernel of a multiply connected polygon under restricted orientation visibility has a lower bound of $\Omega(n\log n)$. This may imply that our s-kernel algorithm is optimal.

It is interesting to investigate the complexity status of the s-star recognition problem, i.e., can there be an algorithm running in $o(n + h\log h)$ time or is there an $\Omega(n + h\log h)$ lower bound? Additionally, it would be interesting to study extensions of the problem to 3-dimensional space.

References

[1] L.P. Gewali. Recognizing s-Star Polygons. *Pattern Recognition* 28(7), 1019-1032, 1995.

[2] D.T. Lee and F.P. Preparata. An Optimal Algorithm for Finding the Kernel of a Simple Polygon. *J. ACM* 26, 415-421, 1979.

[3] R. Motwani, A. Raghunathan, and H. Saran. Covering Orthogonal Polygons with Star Polygons: the Perfect Graph Approach. *J. Comput. Systems Science* 40, 19-48, 1990.

[4] G. Rawlins. *Explorations in Restricted-Orientation Geometry.* PhD Thesis, University of Waterloo, 1987.

[5] S. Schuierer and D. Wood. Generalized Kernels of Polygons with Holes. *Proc. 5th Canadian Conf. on Computational Geometry*, 222-227, 1993.

On the Inverse Beacon Attraction Region of a Point

Bahram Kouhestani David Rappaport Kai Salomaa*

Abstract

Motivated by routing in sensor networks, Biro et al. [2] introduced the notion of beacon attraction and inverse attraction as a new variant of visibility in a simple polygon. A beacon b is a point inside a polygon P that can induce an attraction that moves a target point p greedily towards it in a trajectory that always reduces distance from p to b. The trajectory of p may require sliding p along the boundary of an obstacle. The attraction region of b is the set of all points that eventually reach b. The inverse attraction region of p is the set of points that can attract p. We present algorithms to efficiently compute the inverse attraction region of a point for simple, monotone, and terrain polygons with respective time complexities $O(n^3)$, $O(n \log n)$ and $O(n)$.

1 Introduction

Biro et al. [2] introduced a novel variation of the art gallery problem motivated by geographical greedy routing in sensor networks. A guard is a fixed point, called a beacon, that induces a force of attraction within the environment. The attraction of a beacon moves objects (represented by points) greedily towards the beacon. A point is attracted (covered) by a beacon if it eventually reaches the beacon. It is a common practice in sensor networks that message sending is performed by greedy routing where a node sends or passes the message to its neighbour that is closest to the destination. Depending on the geometry of the network and the location of the sender and receiver, greedy routing may fail. This introduces the interesting problem to determine whether messages can be exchanged between sender and receiver using greedy routing.

Biro et al. [2] studied the combinatorics of guarding a polygon with beacons and showed that $\lfloor \frac{n}{2} \rfloor - 1$ beacons are sometimes necessary and always sufficient to route between any pair of points in a simple polygon. They also proved that it is NP-hard to find a minimum cardinality set of beacons to cover a simple polygon. In 2013, Biro et al. [3] presented a polynomial time algorithm for routing between two fixed points using a discrete set of candidate beacons in a simple polygon and a 2-approximation algorithm where the beacons are placed with no restrictions. For polygons with holes, Biro et al. [4] showed that $\lfloor \frac{n}{2} \rfloor - h - 1$ beacons are sometimes necessary and $\lfloor \frac{n}{2} \rfloor + h - 1$ beacons are always sufficient to guard a polygon with h holes. For other results on beacons see [1].

In this paper we present algorithms to compute the inverse attraction region of a point inside an n-gon. We show that the inverse attraction region of a point can be computed in $O(n^3)$ time in a simple polygon. For monotone polygons we present a simple $O(n \log n)$ time algorithm to compute the inverse attraction region, and for terrain polygons we can further reduce the complexity to $O(n)$ time.

2 Preliminaries

Let P be simple polygon in the plane with the vertices $v_1, v_2, ..., v_n$ in counter-clockwise order. P is *monotone* with respect to the line L if every line orthogonal to L intersects P in at most one connected component. Throughout this paper, without loss of generality, we assume that L is the x-axis. Let u and v be the first and last vertices of the monotone polygon M in lexicographic order. The upper (lower) chain of M is the ordered set of edges from u to v in clockwise (counter-clockwise) order. We define a *terrain polygon*[1] as a monotone polygon with one of its chains consisting of a single line segment.

Let p and q be two points inside P. The Euclidean shortest path (geodesic path) between p and q, $SP(p, q)$ is a path inside P that connects p and q and among all such paths it has the smallest length. The union of Euclidean shortest paths from p to all vertices of P is called the shortest path tree of p and is denoted by $SPT(p)$. Guibas et al. presented a linear time algorithm to compute $SPT(p)$ [6]. It is worth mentioning that $SP(p, q)$ turns only at reflex vertices of P and the angle facing the exterior of P at a turn is convex (the outward convex property of the shortest path). The parent of a node $u \neq p$ in $SPT(p)$ is the last reflex vertex on $SP(p, u)$ which is not u. For proofs and details on shortest paths, see [7, Ch. 3].

*School of Computing, Queen's University, Kingston, Ontario, Canada. {kouhesta,daver,ksalomaa}@cs.queensu.ca

[1] A terrain polygon is sometimes called a "monotone mountain".

A *beacon* is a stationary point inside a simple polygon P that can induce a force of attraction within P. When beacon b is activated, points in P move greedily towards b and monotonically decrease their Euclidean distance to b. Furthermore, points are allowed to slide on the boundary of the environment in order to get closer to b, thus, the movement of a point alternates between moving straight towards b and sliding on the boundary of P (Fig. 1). Let e be an edge of P and let L be the supporting line of e. Let h be the orthogonal projection of b on L (Fig. 2a). As h is the point with the shortest distance to b among all points on L, sliding on e is always towards h. If h is located on e a point sliding on e will reach h and remain on h. Otherwise, it slides all the way to an endpoint of e. Then the point will move straight towards b if that is possible. Otherwise, depending on the location of the orthogonal projection of b on the supporting line of the adjacent edge, the point either slides on the new edge or remains stationary on the endpoint (Fig. 2).

Eventually a moving point either reaches b or becomes stuck on a boundary point of P. The path from the original position of a point p to its final position is called the *attraction trajectory* of p. A point in P is *attracted* by b if its Euclidean distance to b is eventually decreased to 0. The *attraction region* of a beacon b is the set of all points in P that b can attract and can be computed in linear time [1]. In the case that the point does not reach b, its final location is called a *dead point*. The *dead region* relative to a dead point d is the set of all points that end up on d. The boundary between the attraction region and a dead region or two dead regions is called a *split edge*. We denote a split edge that separates the attraction region of the beacon from a dead region as a *separation edge*. In contrast to conventional visibility, beacon attraction is not symmetric. For example in Fig. 1 a beacon located on p cannot attract a point on b. The *inverse attraction region* of a point q is defined as the set of beacon locations in P that attract q.

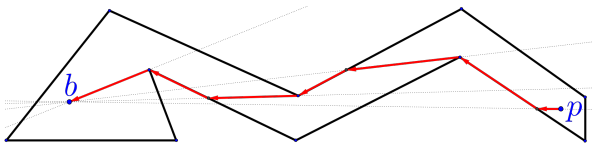


Figure 1: The movement of a point alternates between moving straight towards the beacon and sliding on the boundary.

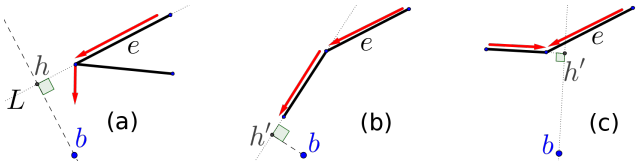


Figure 2: Three cases when a point slides to an endpoint of e. (a) It moves straight towards b. (b) It slides on the adjacent edge. (c) It get stuck on the endpoint. Here h' is the orthogonal projection of b on the supporting line of the adjacent edge of e.

Let r be a reflex vertex incident to edges e_1 and e_2. Let H_1 and H_2 be half-planes perpendicular to e_1 and e_2 emanating from r which include the outside of P in a small neighbourhood of r. The *dead wedge* of r is defined as the intersection of H_1 and H_2 (Fig 3). Let b be a beacon inside the dead wedge of r and to the left of r. Consider h, the orthogonal projection of b on the supporting line of e_2. Note that a point on e_2 close to r will slide away from r. Let Γ be the ray from r and in the direction of $\overrightarrow{br}$ and let s be the line segment between r and the first intersection of Γ with the boundary of P. The attraction of b to a point just to the right of s moves the point to e_2 and slides it towards h, while a point just to the left of s avoids e_2 and passes r. In other words the final destination of those two points will be different and therefore s is a split edge of b and we have the following lemma.

Lemma 1 *A reflex vertex r introduces a split edge for the beacon b if and only if b is inside the dead wedge of r.*

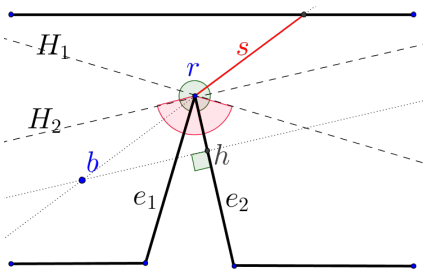


Figure 3: The dead wedge of a reflex vertex r is the intersection of half planes H_1 and H_2 designated by the red arc.

Next we address the problem of computing the inverse attraction region, that is:
Given a simple polygon P and a point q inside P, find the set of all beacon locations in P that attract q.

3 Inverse attraction region in simple polygons

Biro presented an algorithm for computing the inverse attraction region of a point in a simple polygon [1]. Unfortunately his $O(n^2)$ time and space algorithm has a flaw. The algorithm begins by constructing an arrangement A_P of lines to partition P with the idea that for any two points inside a particular region, either both or none attract q.

The arrangement A_P contains three types of lines: 1) lines through edges of P, 2) lines through a reflex vertex and perpendicular to one of the edges incident to this reflex vertex (i.e lines supporting edges of the dead wedge of reflex vertices), and 3) lines through q and each reflex vertex of P.

As far as we know Biro's proof [1] of the following property for A_P, is correct.

Property 1: If b_1 and b_2 belong to the same region of A_P and the reflex vertex r is a split vertex relative to b_1 (i.e. r introduces a split edge for b_1) then r is also a split vertex relative to b_2 [1].

Biro used Property 1 to conclude that all points in a particular region behave the same with respect to q (all or none attract q). The example in Fig. 4 illustrates that property 1 is not sufficient to guarantee that points in the same region have the same attraction behaviour with respect to q. Consider the line L going through the reflex vertices r_1 and r_2 and let s and t be two points close to and on opposite sides of L. Even though r_2 introduces a split edge for both s and t, it is easy to see that s cannot attract q while t can. This example suggests that additional lines need to be added to the arrangement.

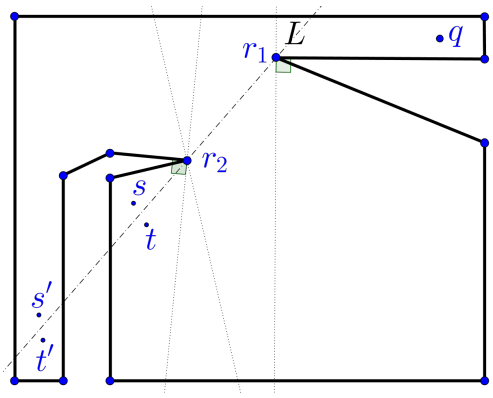


Figure 4: An example where the arrangement in [1] does not work. Point s cannot attract q while t can. Also observe that point s' can attract q while t' cannot.

The example in Fig. 4 implies that it is necessary to add some of the lines going through pairs of reflex vertices of P to the arrangement. As a polygon may have $O(n)$ reflex vertices, this adds an additional $O(n^2)$ lines with an arrangement of $O(n^4)$ regions.

We construct a new arrangement of $O(n^2)$ complexity which correctly groups together points in P.

The arrangement A_P uses three types of lines:
1) Lines through edges of $SPT(q)$.
2) Lines through the edges of the dead wedge of a reflex vertex of p.
3) Lines through edges of the polygon.

Note that lines of the third type are added to A_P to distinguish points that are inside or outside of P.

Lemma 2 *If b_1 and b_2 belong to the same region of A_P then either both or neither attract q.*

Proof. For the sake of contradiction and without loss of generality assume b_1 attracts q while b_2 does not attract q. Let r be the split vertex that separates q from the attraction region of b_2 (i.e. r introduces a split edge for b_2 that separates q from the attraction region of b_2). Without loss of generality let us assume that q is to the left of this split edge, s_2 (Fig. 5). As b_1 and b_2 are in the same region of A_p, b_1 is also in the dead wedge of r and r introduces a split edge for b_1. As b_1 attracts q, q lies to the right of this split edge s_1.

We have two cases:
1) Both s_1 and s_2 have an (upper) endpoint on a common edge e (Fig. 5a). In this case q lies in the triangle formed by s_1, s_2 and e. This triangle is contained in P, and therefore q sees r and the line segment connecting r and q is in $SPT(q)$. Therefore, the line $\overline{qr}$ forces b_1 and b_2 to be in two different regions of A_P, a contradiction.
2) Edges s_1 and s_2 have (upper) endpoints on different edges of P (Fig. 5b). Let the endpoint of s_1 lie on e and the endpoint of s_2 lie on e'. Notice that the left endpoint of e' is located between s_1 and s_2. Now consider the shortest path between q and r. If q and r see each other directly then the supporting line of the line segment $\overline{qr}$ belongs to A_P and similar to the previous case we have a contradiction. If q and r cannot see each other directly then there exists a reflex vertex r' between s_1 and s_2 such that the shortest path between q and r passes through r'. Now by the construction the line $\overline{rr'}$ is in A_P which forces b_1 and b_2 to be located in two different regions, a contradiction. □

Theorem 3 *The inverse attraction region of a point in a simple polygon can be computed in $O(n^3)$ time and $O(n^2)$ space.*

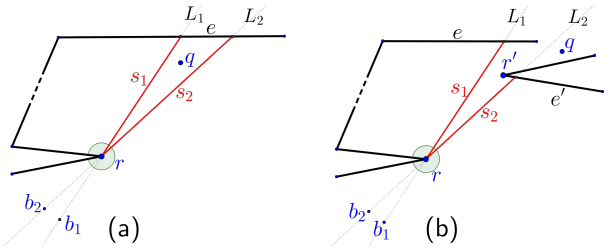


Figure 5: Split edges of b_1 and b_2.

Proof. There are $O(n)$ lines in the arrangement. Therefore the number of regions in the arrangement is $O(n^2)$ and for each region we can check whether a candidate point can attract q in linear time, resulting in the O(n^3) time complexity. □

4 Inverse attraction region in a monotone polygon

In the previous section we showed that lines passing through edges of SPT(q) and through edges of dead wedges form the boundaries between regions that attract q and those that don't. For the case of monotone polygons we show that a much smaller subset of these boundary edges suffice.

Let M be a monotone polygon and let q be a point in M. We begin by studying the effect of a single reflex vertex on the inverse attraction region of q. Let v be a reflex vertex of M with e_l and e_r the left and right adjacent edges of v, respectively. Let $q \in M$ be a point to the right of v. Our goal is to distinguish all beacon placements to the left of v that do not attract q because they are blocked by an edge incident to v. To do so, first we assume that there are no reflex vertices between q and v (i.e no reflex vertex exists simultaneously to the left of q and to the right of v) and find points to the left of v that cannot move q past a vertical line through v.

We show that a ray passing through v can be used to bound a subpolygon of M so that any beacon placed within that subpolygon cannot attract the point q. This ray can be defined in one of two ways yielding what we call a *blocking ray*. The two cases of blocking rays are described as follows:

Case 1 blocking ray: $q_1 \in M$ is a point to the right of the reflex vertex v and below the line L_1 orthogonal to e_r at v. Observe that L_1 passes through the left edge of the dead wedge of v. According to attraction properties (Fig. 2), a beacon in M below L_1 and to the left of v cannot attract q_1 past the vertical line through v, and therefore it does not attract q_1. Thus we can express the effect of v by a ray Γ_1 emanating from v extending to the left along L_1. No point below Γ_1 can attract q_1. We call Γ_1 the *blocking ray* of v relative to q_1.

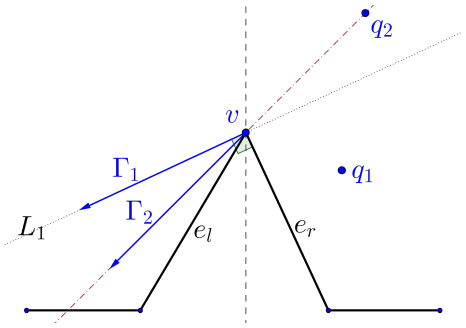


Figure 6: The effect of a single reflex vertex on the inverse attraction region of a point. Case 1 blocking ray: q_1 lies below L_1: beacons below Γ_1 cannot attract q_1. Case 2 blocking ray: q_2 lies above L_1 and and below L_2: beacons below Γ_2 cannot attract q_2.

Case 2 blocking ray: $q_2 \in M$ is a point to the right of v and above L_1. Let Γ_2 be the ray emanating from v extending to the left along the line $\overline{q_2 v}$. Note that Γ_2 is in the dead wedge of v. Consider a beacon b to the left of v. If b is to the right of Γ_2 then the attraction path of q_2 will intersect e_r and by considering the orthogonal projection of b on the supporting line of e_r, we see that b cannot pass q_2 over v. Now assume b is to the left of Γ_2. Here the line segment $\overline{q_2 b}$ will not intersect e_r and therefore b can move q past over v. Here Γ_2 is the blocking ray of v relative to q_2.

We define the *blocking region* of a reflex vertex v relative to q as points of M which are below the blocking ray of v relative to q. Informally, the blocking region of v is the set of beacon locations that cannot attract q due to v. Note that a point in the blocking region of v (in both cases) is in the dead wedge of v.

We can now present an algorithm to compute the inverse attraction region of a point in a monotone polygon.

Algorithm InverseAttractionRegion
Input. Monotone polygon M and a point $q \in M$.
Output. Inverse attraction region of q, that is, beacon locations in P that attract q.

1: Compute $SPT(q)$, the shortest path tree from q to each vertex of M.
2: **for** each reflex vertex r that sees q **do**
3: Discard points in the blocking region of r relative to q
4: **end for**

5: **for** each pair of consecutive reflex vertices v, v' in $SPT(q)$ (v = parent(v')) **do**
6: Discard points in the blocking region of v' relative to v.
7: **end for**
8: **return** The remaining polygon

Theorem 4 *Algorithm InverseAttractionRegion correctly computes the inverse attraction region of an input point q in a monotone polygon.*

Proof. Suppose p is discarded by the algorithm due to the edge $s = vv'$, where $v = parent(v')$ in SPT(q). We claim that p cannot attract any point on s (see appendix). We show that p cannot attract q as well. We consider two cases:

1) v and v' lie on different chains of M. Here, s partitions M into two sub-polygons and p and q are in different sub-polygons. Let π be the attraction trajectory of q to p. As p and q are on different sides of s, π crosses s. Let x be the intersection of π and s. As p cannot attract x, we conclude that it cannot attract q.

2) v and v' are on the same monotone chains (Fig. 8). Let w be the first intersection point of the ray $\overrightarrow{v'v}$ with M to the right of v. Note that as the shortest path is outward convex, the parent of v in $SPT(q)$ lies in the sub-polygon to the right of the line segment $\overline{vw}$. Therefore, $\overline{vw}$ partitions M into two sub-polygons where p and q are in different sub-polygons. As p cannot attract v, we can show that it cannot attract any point on $\overline{vw}$ (see appendix). If p attracts q then the attraction trajectory must intersect $\overline{vw}$ which is a contradiction.

Now suppose p is a point that cannot attract q. Let t be the separation edge of the attraction region of p such that p and q are in different sides of t. Let v' be the reflex vertex that introduces t and M_1 be the sub-polygon that contains q (Fig. 7). Observe that v = parent(v') in SPT(q) is in M_1 because the shortest path is outward convex. Therefore, p does not attract v and p lies in the blocking region of v' relative to v. With our construction when the pair $(v, v') \in SPT(q)$ is processed, p will be discarded. □

We use a result of Hershberger [5] that computes the upper envelope of a set S of n non-vertical line segments in $O(n \log n)$ time. The upper envelope of S is defined as the portion of the segments in S visible from $y = +\infty$. The lower envelope is defined symmetrically.

Lemma 5 *The time complexity of the Algorithm InverseAttractionRegion is $O(n \log n)$.*

Proof. In order to achieve an $O(n \log n)$ time complexity, we first collect all blocking rays and then discard points in blocking regions. Let B be an axis aligned bounding box of M. By intersecting the blocking rays with B (and adding the top and bottom edges of B) we have a collection of blocking line segments. If the blocking line segment originated from a reflex vertex of the lower (upper) chain, then we need to discard points of M that are vertically below (above) this line segment. Using Hershberger's algorithm [5], we construct the upper (lower) envelope of blocking line segments of vertices of the lower (upper) chain in $O(n \log n)$ time and obtain two monotone polygons. The intersection of these two polygons with M is the set of points below all upper chain blocking rays and above all lower chain blocking rays. As the intersection of monotone polygons can be computed in linear time, the total complexity is $O(n \log n)$. □

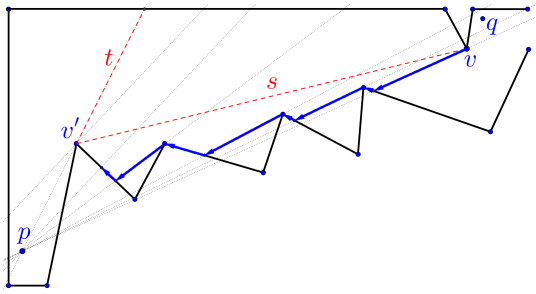


Figure 7: Attraction trajectory of v. Here, p cannot attract v.

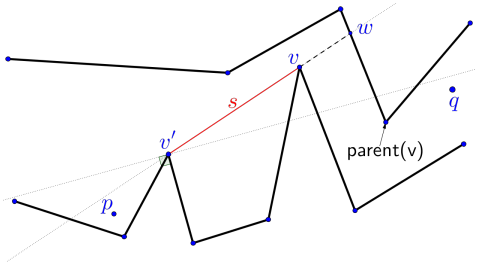


Figure 8: No point on $\overline{vw}$ can be attracted by p. Therefore p cannot attract q.

5 Inverse attraction in a terrain polygon

Let M be a terrain polygon and let L be a vertical line through q. L partitions M into two terrain polygons. We consider each of these polygons separately and discard points that cannot attract q in each polygon. Here we explain how this is done for M_1, the polygon to the left of L. Let R_1 be all rays of R that extend from left to right. We present a linear time algorithm to discard points below the rays in R_1. The algorithm starts by traversing M_1 from right to left. Events are reflex vertices with a blocking ray that extends to the

left. The algorithm preserves the invariant that at each event point the computed polygon to the right is the set of points in M_1 vertically above all current blocking rays $\Gamma_1, \Gamma_2, ..., \Gamma_i$. Furthermore, the algorithm stores and updates a convex set C which is the upper envelope of current rays intersected by a bounding box of the polygon.

Algorithm DiscardingBelowRays
Input. A terrain polygon M_1. A set $R = \Gamma_1, \Gamma_2, ..., \Gamma_m$ of blocking rays all extending to the left.
Output. A polygon P obtained by discarding points in M_1 vertically below the rays in R.

1: Order R such that Γ_i is the blocking ray of the reflex vertex r_i and r_i is to the left of r_{i+1} for all $i = 1, 2, ..., m-1$.
2: Let C be an axis aligned bounding box of M_1.
3: Let V_i be the vertical line through r_i and H_i be the half-plane to the left of V_i.
4: Let polygon P be the subset of M_1 between V_1 and a vertical line through q.
5: **for** $i = 1$ to m **do**
6: $C = C \cap H_i$
7: **if** r_i is in C **then**
8: Intersect C with the half-plane above the supporting line of Γ_i by traversing the lower edges of C and finding the first edge of C that intersects Γ_i.
9: Add to P all points of M_1 between V_i and V_{i+1} that are also in C
10: **end if**
11: **end for**
12: **return** P

The algorithm computes the upper envelope of rays $\Gamma_1, \Gamma_2, ..., \Gamma_i$ between V_i and V_{i+1} and intersects the result with the portion of M_1 between V_i and V_{i+1} (Fig. 9). Therefore, the output are points of M above all blocking rays. Before we analyze the time complexity of the algorithm, we show that it is safe to ignore rays of reflex vertices that start below some current blocking regions (step 6).

Lemma 6 *If $r_i \notin C$ then Γ_i does not contribute to P.*

Proof. See appendix. □

Lemma 7 *Algorithm DiscardingBelowRays runs in $O(n)$ time.*

Proof. We use a sequential search in both step 7 intersecting $C \cap H_i$ and in step 9 intersecting the upper envelope of Γ_i with C. In each case once we step over an edge we eliminate it forever. Thus the overall complexity of the algorithm is O(n). □

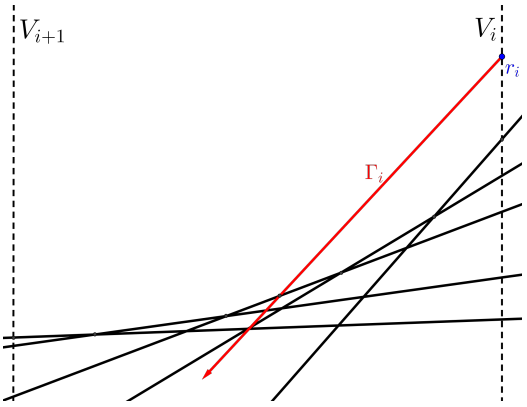


Figure 9: Discarding points below rays.

6 Conclusion

In this paper, we presented algorithms to efficiently compute the inverse attraction region of a point for simple, monotone, and terrain polygons. Currently we are developing a more efficient algorithm for simple polygons using the ideas of chapter 4. We believe that we can design an $O(n \log n)$ time algorithm which can be shown to be optimal.

References

[1] M. Biro. Beacon-based Routing and Guarding. PhD Dissertation, Stony Brook University, 2013.

[2] M. Biro and J. Gao, J. Iwerks, I. Kostitsyna and J. S. B. Mitchell. Beacon-based routing and coverage. Proceedings of the 21st Fall Workshop on Computational Geometry, 2011.

[3] M. Biro, J. Iwerks, I. Kostitsyna and J. S. B. Mitchell. Beacon-Based Algorithms for Geometric Routing. Proceedings of the 13th International Symposium on Algorithms and Data Structures, WADS, 2013.

[4] M. Biro, J. Gao, J. Iwerks, I. Kostitsyna and J. S. B. Mitchell. Combinatorics of beacon routing and coverage. Proceedings of the 25th Canadian Conference on Computational Geometry, 2013.

[5] J. Hershberger. Finding the upper envelope of n line segments in $O(n \log n)$ time. Information Processing Letters, 33(4):169-174, 1989.

[6] L. Guibas, J. Hershberger, D. Leven, M. Sharir, R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. Algorithmica, 2:209-233, 1987.

[7] S. Ghosh. Visibility Algorithms in the Plane. Cambridge University Press, 2007.

Appendix

Although our goal is to compute the inverse attraction region of a fixed point, it is useful to compare the blocking regions of two points relative to a particular reflex vertex.

Lemma 8 *Let v be a reflex vertex of M. Let q and q' be two points of M such that q' is on the open line segment $\overline{qv}$. If there are no reflex vertices between v and q, then the blocking regions of q and q' (relative to v) are equal.*

Proof. Consider the cases in Fig. 6. It is easy to verify that when q' lies on the (open) line segment $\overline{qv}$, the blocking rays of q and q' are the same. Therefore, their blocking regions are equal. □

Lemma 9 *Let q be a point close to the left edge of v. Consider the clockwise rotation of q around v to a vertical position. During the rotation, the blocking region of v relative to q never increases.*

Proof. During the rotation, as long as q is below L_1 (see Fig. 6), the blocking region of q remains the same. While q is rotated from L_1 to a vertical position, the blocking ray of v relative to q will rotate clockwise from L_1 to a vertical downward ray. During this time the blocking region of q (i.e. points in M below the blocking ray) monotonically gets smaller until it is empty. Therefore, during the rotation the blocking region of q is non increasing. □

Next we consider the effect of two reflex vertices on the inverse attraction region of a point. Let v and v' be the only two reflex vertices of M. If a point $q \in M$ is located between v and v' then any attraction trajectory of q can at most have one of v and v' on its path and therefore the effect of v and v' can be considered separately. Therefore we focus on the inverse attraction region of the point q which lies to the right of both reflex vertices. Without loss of generality assume v is on the lower chain and v' is on the upper chain of M.

Case 1) If q is visible to both v and v', we claim that any attraction trajectory of q can at most pass through one of these reflex vertices. The attraction trajectory of q to a beacon b passes through v only if b is below the ray $\overrightarrow{qv}$ and passes through v' only if b is above the ray $\overrightarrow{qv'}$ (Fig. 10). As q sees both v and v' there does not exist a beacon both below $\overrightarrow{qv}$ and above $\overrightarrow{qv'}$. Therefore at most one reflex vertex can affect the attraction trajectory and in the computation of the inverse attraction v and v' are considered separately. We conclude that a point inside the blocking regions of v or v' cannot attract q.

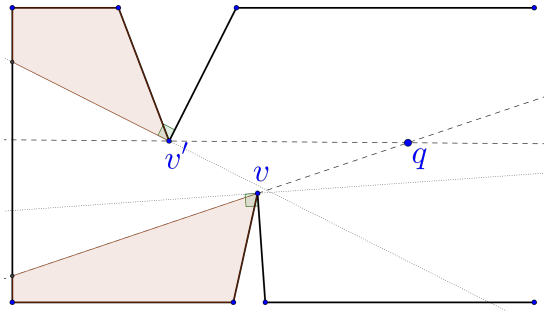


Figure 10: If q sees both v and v', no attraction trajectory of q can intersect both v and v' and in the computation of the inverse attraction region, v and v' are considered separately. Here points that cannot attract q are shaded.

Case 2) Otherwise, without loss of generality assume that q can see v but not v' (Fig. 11). We classify the points to the left of v' into two groups: i) points above the ray $\overrightarrow{qv}$ and ii) points below $\overrightarrow{qv}$. Let p be a point in group i. Consider π the attraction trajectory of q in the attraction of p. As p is located above $\overrightarrow{qv}$, π does not intersect the adjacent edges of v. We conclude that p can attract q if and only if p is not in the blocking region of v' (relative to q). Now assume that p is a point in group ii. In this case π will intersect v or the right edge of v. Therefore, p attracts q if and only if p can move q from its initial position to v (i.e. p is above the blocking ray of v relative to q) and p can attract v (i.e p is below the blocking ray of v' relative to v).

Next we show how to combine the two groups of case 2.

Lemma 10 *If q sees v but not v' then points in the blocking region of v relative to q and points in the blocking region of v' relative to v are the only points that cannot attract q.*

Proof. It is obvious that a point in the blocking region of v relative to q does not attract q, because it cannot move q past over v. So we only need to argue about points to the left of v'. Let p be a point in group ii (i.e. p is a point to left of v' and below $\overrightarrow{qv}$). By the previous argument p attracts q if and only if p can move q from its initial position to v and p can attract v. Therefore, p cannot lie in the blocking region of v (relative to q) and it cannot lie in the blocking region of v' relative to v and so the lemma follows.

Now let p be a point in group i (i.e. p is to the left of v' and above $\overrightarrow{pv}$). Note that as q does not see v', p also lies above the line $\overline{vv'}$ (see Fig. 11). Recall our case analysis in Fig. 6. If the relative position of v with respect to v' lies in case 1 (which is the case in Fig. 11), then the

blocking region of v' relative to v is all points in the left side of the dead wedge of v'. The attraction trajectory of q in the attraction of a point in group i intersects the right edge of v'. Therefore a point in group i can attract q if it is not located on the left side of the dead wedge of v'. This is precisely the blocking region of v' relative to v.

Now assume that the relative position of v to v' lies in case 2 of Fig. 6. Recall that the blocking ray of v' relative to v is the ray from v' in the direction of the vector $\overrightarrow{vv'}$. As points above the $\overline{qv}$ are also above $\overline{vv'}$, all points of group i reside in the blocking region of v' relative to v and lemma follows. □

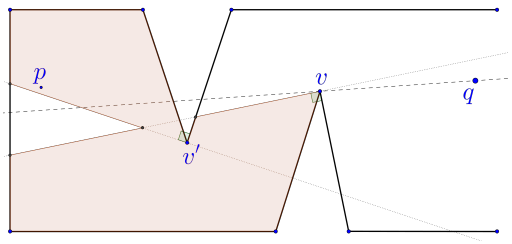


Figure 11: Points that cannot attract q are shaded.

Theorem 4 *The algorithm InverseAttractionRegion correctly computes the inverse attraction region of a given point in a monotone polygon.*

Proof. We use proof by contradiction. First that assume p is a point that can attract q and is discarded by the algorithm. Without loss of generality we assume p is to the left of q. If p is discarded in step 3 of the algorithm then let v be the rightmost reflex vertex responsible for discarding p. Note that q and v see each other, and p is in the blocking region of v therefore it is also in the dead wedge of v. As p is also to the left of v, p cannot attract any points on the right adjacent edge of v. Since p attracts q, the attraction trajectory of q to p must pass above v. Here in order for q to pass above v, there must exist an edge e between v and q such that q slides on e and moves above the line $\overline{pv}$. This implies that e blocks the visibility of v and q, which is a contradiction.

Assume p is discarded in step 6 due to s, where s is the directed open edge of SPT(q) from v to v'. Note that due to the monotonicity of M both v and v' are to the right of p and to the left of q. Consider π_{pv} the attraction trajectory of v to p (Fig. 7). As p is discarded when the pair (vv') is processed, in the absence of other reflex vertices p cannot attract v. Since v and v' are visible, no attraction trajectory (towards p) can slide through s. By lemma 8 the blocking region of all points on s are equal and by lemma 9 no points below s can have a blocking region smaller than the blocking region of v. Therefore (even in the presence of other reflex vertices) no points on π_{pv} can be attracted by p and thus p does not attract v. Now we show that p cannot attract q as well. We consider two cases:

1) v and v' lie on different chains of M. Here, s partitions M into two sub-polygons and p and q are in different sub-polygons. Note that by lemma 8 the blocking region of v relative to any point on s is precisely the blocking region of v relative to v'. This implies that p cannot attract any point on s. Let π be the attraction trajectory of q to p. As p and q are on different sides of s, π crosses s. Let x be the intersection of π and s. As p cannot attract x, we conclude that it cannot attract q.

2) v and v' are on the same monotone chains. Let w be the first intersection point of the ray $\overrightarrow{vv'}$ with M to the right of v (Fig. 8). Note that as the shortest path is outward convex, the parent of v in $SPT(q)$ lies in the sub-polygon to the right of the line segment $\overline{vw}$. Therefore, $\overline{vw}$ partitions M into two sub-polygons where p and q are in different sub-polygons. By lemma 8 the relative blocking region of v' relative to any point on $\overline{vw}$ is exactly the blocking region of v' relative to v. As p cannot attract v, it cannot attract any point on $\overline{vw}$. If p attracts q then the attraction trajectory must intersect uw which is a contradiction.

Now suppose p is a point that cannot attract q and is not discarded by the algorithm. Let t be the separation edge of the attraction region of p such that p and q are in different sides of t. Let v' be the reflex vertex that introduces t and M_1 be the sub-polygon that contains q (Fig. 7). Observe that $v = \text{parent}(v')$ in SPT(q) is in M_1 because the shortest path is outward convex. Therefore, p does not attract v and p lies in the blocking region of v' relative to v. With our construction when the pair $(v, v') \in SPT(q)$ is processed, p will be discarded. □

Lemma 6 *If $r_i \notin C$ then Γ_i does not contribute to P.*

Proof. Let Γ_i be the blocking ray of r_i and $r_i \notin C$. Let Γ_j $(j < i)$ be the leftmost ray above r_i. Consider the parent of r_i in $SPT(q)$. If r_j is the parent of r_i then the blocking ray of r_i relative to r_j will be on or under the ray $r_j r_i$, therefore all points in the blocking region of r_i are also in the blocking region of r_j. Now assume $w \neq r_j$ is the parent of r_i and therefore w lies above the ray $r_i r_j$. Consider the blocking ray of r_i relative to w. It lies on or below the line $r_i w$ and so below the line $r_i r_j$. Therefore in both cases the blocking region of r_i can be ignored. □

A Combinatorial Bound for Beacon-based Routing in Orthogonal Polygons

Thomas C. Shermer*

Abstract

Beacon attraction is a movement system whereby a robot (modeled as a point in 2D) moves in a free space so as to always locally minimize its Euclidean distance to an activated beacon (also a point). This results in the robot moving directly towards the beacon when it can, and otherwise sliding along the edge of an obstacle. When a robot can reach the activated beacon by this method, we say that the beacon *attracts* the robot. A *beacon routing* from p to q is a sequence $b_1, b_2, \ldots, b_k$ of beacons such that activating the beacons in order will attract a robot from p to b_1 to $b_2 \ldots$ to b_k, and where a beacon placed at q will attract b_k. A *routing set of beacons* is a set B of beacons such that any two points p, q in the free space have a beacon routing with the intermediate beacons $b_1, b_2, \ldots b_k$ all chosen from B. Here we address the question of "how large must such a B be?" in orthogonal polygons, and show that the answer is "sometimes as large as $\lfloor \frac{n-4}{3} \rfloor$, but never larger."

1 Background

Beacon attraction has come to the attention of the community recently as a model of greedy geographical routing in dense sensor networks. In this application, each node of the network has a location, and each communication packet knows the location of its destination. Nodes having a packet to deliver forward the packet to their neighbor that is the closest (using Euclidean distance) to the packet's destination [5, 7].

In the abstract geometric setting, the destination point is called a beacon, and the message is considered to be a point (or robot) that greedily moves towards the beacon. The robot, under this motion, may or may not reach the beacon—if it does reach the beacon, we say that the beacon *attracts* the robot's starting point. The attraction relation between points has the flavor of a visibility-type relation, with the interesting twist that it is asymmetric: if point p attracts point q, then it does not follow that point q attracts p. In a series of publications, Biro, Gao, Iwerks, Kostitsyna, and Mitchell have studied various visibility-type questions for beacon attraction, such as computing attraction (and inverse-attraction) regions for points, computing attraction kernels, guarding, and routing [4, 3, 2]. Bae, Shin, and Vigneron addressed beacon-attraction guarding in orthogonal polygons [1].

In beacon-based routing, the goal is to route from a source p to a destination q through a series of intemediate points $b_1, b_2, \ldots b_k$ where b_1 attracts q, b_2 attracts b_1, b_3 attracts b_2, etc., and finally q attracts b_k. The idea is that we activate the beacons $b_1, b_2, \ldots b_k$ individually in turn, and then activate a beacon at q, and we will have attracted p all of the way to q. In the application setting, this corresponds to using greedy geographical routing for each hop in a multi-hop routing for the packet; beacons correspond to *landmark* or *backbone* nodes of the network [8]. Ad-hoc networks (and to some extent, sensor networks) expect to see messages from many different p's to many different q's. Thus it is natural to ask whether we can find some set B of backbone nodes (beacons) such that one can route from *any* p to *any* q using only backbone nodes chosen from B.

We call such a set B a *routing set of beacons*. Biro et al.[3] studied the problem of finding minimum-cardinality routing beacon sets in simple polygons. They established that it is NP-hard to find such a minimum-cardinality B, and that such a B can be as large as, but never exceed, $\lfloor \frac{n-2}{2} \rfloor$. Biro [2] also conjectured that, in orthogonal polygons, such a B could be as large as, but never exceed, $\lfloor \frac{n-4}{4} \rfloor$. In this paper, we disprove this conjecture, pinning this maximum minimum size at $\lfloor \frac{n-4}{3} \rfloor$ instead.

In this paper, we omit many details, lemmas, and proofs due to size constraints. Full details are available in the arXiv preprint [9].

2 Preliminaries

2.1 Routing segments

If p and q are points in a polygon with a beacon routing from p to q, then by a *routing segment* we mean any maximal section of the beacon-routing path during which a point travelling the path is attracted by a single beacon (or by the destination point q). If the beacon routing from p to q starts at p, proceeds to beacon b_1, then to beacon b_2, then to q, then the routing segments are the part from p to b_1, the part from b_1 to b_2, and the part from b_2 to q.

*School of Computing Science, Simon Fraser University, shermer@sfu.ca

2.2 Decomposition and neighboring rectangles

Let P be an orthogonal polygon of n vertices in general position; handle special-position instances with the usual perturbation technique. Construct the *vertical decomposition* (or *trapezoidation* [6]) of P by creating a vertical chord from every reflex vertex (see Figure 1).

Because of our restriction to general position, there are $\frac{n-4}{2}$ verticals, decomposing the polygon into $\frac{n-2}{2}$ axis-aligned rectangles. Each such rectangle has between one and four neighboring rectangles. If we form a graph of the neighbor relation on the rectangles, then we have the *dual tree* (or *weak dual*) of the decomposition, as shown in Figure 1.

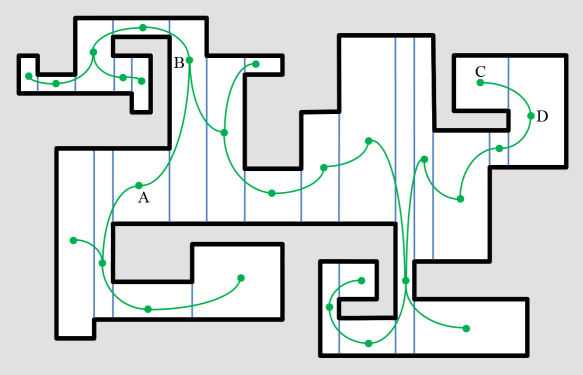


Figure 1: The vertical decomposition of a polygon, with its dual tree.

We classify the different types of neighbors of a rectangle R in 3 primary ways: *left* vs. *right*, depending on the side of R they are on; *top* vs. *bottom*, depending on whether the neighbor and R have the same polygon edge as their top or bottom; and *short* vs. *tall*, depending on whether the neighbor is shorter or taller than R. We combine these classifications: for instance, in Figure 1, A is a short bottom left neighbor of B, and D is a tall top right neighbor of C.

Observation 1 *If a rectangle R is a tall left (or right) neighbor of S, then it is the* only *left (or right, respectively) neighbor of S.*

Observation 2 *If a rectangle R is a short left (or right) neighbor of S, then it is either the* only *left (or right, respectively) neighbor of S (which we call a* solo *neighbor), or there is one other short left (or right, respectively) neighbor of S (in which case we call R a* paired *neighbor of S).*

We generally divide the different cases of a neighboring rectangle's type into into tall, solo, and paired. Figure 2 shows these three types of neighbors.

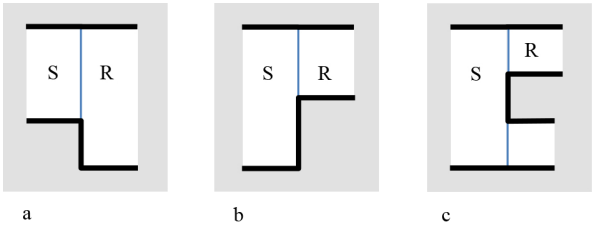


Figure 2: The three types of top right neighbor R of a rectangle S: (a) tall, (b) solo, (c) paired.

2.3 Beacon coverage

If a point p in a polygon attracts a point q, and q attracts p, then we say that p *covers* q. If p covers every point in some region C, then we say that p *covers* C. And if there is a set of points B in the polygon such that for every point q in C, there is a b in B that attracts q, and a b' in B that q attracts, then we say that B covers C. Typically, the point set B will be our set of beacons, and C will be the entire polygon, or a small region of it.

To build a set of beacons we need to know which regions an individual beacon will cover. Fortunately, for our purposes it will mainly suffice to know which rectangles of the decomposition a beacon covers.

First, a beacon b will cover any rectangle of the decomposition it is in. (If b is on a vertical then it will be in two such rectangles.) The lemmas in this section establish some beacon placements that cover rectangles other than their containing rectangles. To save space, in this paper we ignore details about issues of closure that affect the analysis only at reflex vertices. Also, in this section we will omit the proofs of the lemmas but leave the corresponding figures to illustrate definitions and to give the reader a hint at the proofs.

Lemma 1 *If rectangle S is a solo neighbor of rectangle R in the decomposition of a polygon, then any point of R covers S, and any point of S covers R.*

See Figure 3.

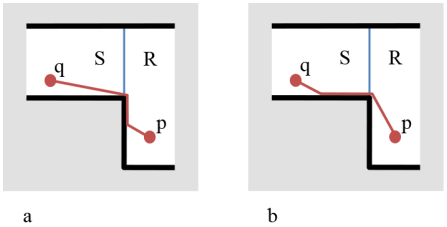


Figure 3: S is a solo neighbor of R. (a) p is attracted into the left wall of R. (b) q is attracted into the bottom wall of S.

Next we look at a rectangle with paired neighbors.

Let R have paired neighbors on the left; we define the *left center* of R as the closed rectangle that is the full width of R and has the vertical span of the polygon edge on the left of R (as illustrated in Figure 4a).

We similarly define the *right center* if R has paired neighbors on the right. See Figure 4.

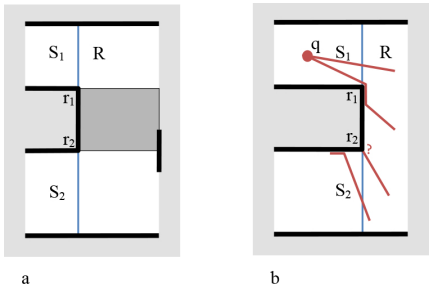


Figure 4: (a) the left center of R is shown shaded. (b) If p is attracted to the left side of R at or above r_1, it proceeds into S_1 (and directly to q). If p is attracted to the left wall of R between r_2 and r_1, it is pulled up the wall and at r_1 will enter S_1 and then will reach q. If p is attracted to the left wall at the point r_2, the behavior is indeterminate. If p is attracted to the left side below r_2, it proceeds into S_2 and does not reach q.

Lemma 2 *If rectangles S_1 and S_2 are paired left (right) neighbors of rectangle R in the decomposition, then any point in the left (right, respectively) center of R covers S_1 and S_2.*

We will mostly be applying Lemma 2 with the point in the center of R being either $r_1 + \varepsilon\hat{x}$ or $r_2 + \varepsilon\hat{x}$, where $\hat{x}$ is the unit vector in the x-direction.

3 Trapping and repair

3.1 Locality

We will call a routing segment *local* if there are (at most) three rectangles of the vertical decomposition whose union contains the segment. We will similarly call a routing path local if all of its segments are local, and a routing beacon set local if it supports a local routing path between every pair of points in the polygon. The routing beacon sets that we construct will all be local.

We let the *local attraction relation* be the attraction relation restricted to those ordered pairs of points (p, q) where p attracts q via a local routing segment.

3.2 Trapped paths

In the inductive step of our proof, we will be removing a few rectangles from the polygon P_k by cutting the polygon along a vertical V of the decomposition. Let C denote the (closed) region that is removed; it will consist of a few rectangles. The (closed) polygon remaining is denoted P_{k+1}. In P_{k+1}, the vertical V is part of the polygon boundary, but in P_k it is not.

To form a beacon set B_k for P_k, we would like to take the beacon set B_{k+1} for P_{k+1} (which inductively exists) and add a few beacons to it. We could use B_{k+1} for routing between pairs of points in P_{k+1} (as a subset of P_k), and then just worry about routing the points of C (to each other, and into and out of P_k). However, this simple strategy does not work, because in P_k, the beacons B_{k+1} may not be a routing set for the region P_{k+1}. This happens because the points of V have changed status from boundary to non-boundary.

We will call the rectangle of C that contains the vertical V the *detachment rectangle*, and the rectangle of P_{k+1} containing V the *base rectangle*. By considering whether the detachment rectangle is a tall, solo, or paired neighbor of the base (analysis omitted in this paper), we find the only problematic case is when it is paired.

In this case, the beacon routing of P_{k+1} may have segments dependent on V being boundary: a routing path segment may encounter the wall of P_{k+1} at a point on V, and then be pulled along that wall containing V until it reaches the reflex vertex (and then leaves the wall; see Figure 5a). In P_k, the corresponding attraction path, upon encountering V, would continue into C and become *trapped*, not reaching the beacon, as shown in Figure 5b.

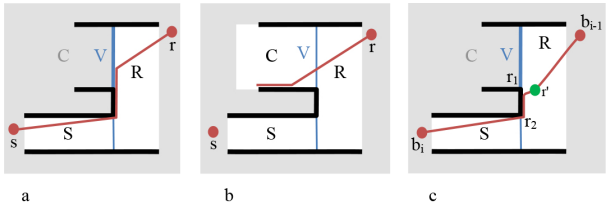


Figure 5: A trapped path. (a) a path segment from r to s hits a wall in P_{k+1}. (b) the attraction path from r towards s continues into C in P_k. (c) repairing a segment from b_i to b_{i-1} with r'.

3.3 Repair of trapped paths

To fix the problem of trapped paths, we will have to devote a new beacon to *repair* such trapped path segments, as suggested in Figure 5c.

Lemma 3 *Let B_{k+1} be a local routing set of beacons in P_{k+1}. If a left (or right) paired neighbor Q has been cut from rectangle R in P_k to form P_{k+1}, we can add the point $r + \varepsilon\hat{x}$ (or $r - \varepsilon\hat{x}$) to B_{k+1} to obtain a beacon set*

that supports local routing between any pair of points in the subpolygon P_{k+1} of P_k, where r is the reflex vertex of P_k common to Q and R.

The proof of this lemma, lengthy and omitted here, relies crucially on the locality of path segments. If a segment is trapped, then locality allows us to contain that segment in the union of S and R (as in Figure 5), and one other rectangle. The other rectangle must be some sort of neighbor of S or R, and we treat each such possibility in a case analysis.

We use the term *repair position* to refer to the placement of the new beacon (point) in the previous lemma.

4 Upper bound

We will prove the theorem by induction on the size of the dual tree of the vertical decomposition. We first root the dual tree at an arbitrary leaf. At each step, we will examine the structure of the vertical decomposition at and around a deepest node in the rooted tree. We will place some beacons and remove some rectangles/dual tree nodes; we will place at most two beacons per three rectangles removed. We stop and consider basis cases when the depth of the dual tree reaches 0, 1, or 2.

We start with a tree T_0 that is the entire dual tree of the polygon P (which we also denote by P_0). After step k, we will have a tree T_k which is a subgraph of T_0, with the rectangles corresponding to its vertices forming a single polygon P_k which is a subpolygon of P. We call each induction step from T_k and P_k to T_{k+1} and P_{k+1} a *reduction*.

Since the case analysis that will follow gets tedious, we first establish easily-verified sufficient (but not necessary) conditions to form an beacon routing set by inductively cutting off a region C from P_k to yield P_{k+1}. We use these conditions for most but not all of our cases.

Lemma 4 *If the following conditions hold, then $B_k = B_{k+1} \cup B'$ is a routing beacon set for P_k.*

1. *The beacons given (B') cover the region $C = P_k \setminus P_{k+1}$, using local paths.*
2. *B' induces a strongly connected graph in the graph of the local attraction relation.*
3. *At least one element b' of B' is in P_{k+1}.*
4. *If the base rectangle is taller than the detachment rectangle, then b' is in repair position.*

Assume we are after step k, having tree T_k and polygon P_k remaining. If T_k is of height 1 or 2, we stop. Otherwise, let L be a deepest node in the dual tree, let A_1 be its direct ancestor (parent), and in general let A_j be the direct ancestor of A_{j-1}. The grandparent A_2 of L exists. In general, we will try to reduce the size of T_k by removing the dual tree nodes of A_1's subtree, cutting the polygon between A_1 and A_2. In some cases, we must consider alternatives to this cutting location.

The figures used in the case analysis obey the following visual conventions: Parts of the figure boundary *known* to be boundary of P_k are shown with thick black lines. Parts without may or may not be boundary of P_k. Beacon placements are shown as green dots, and rectangles removed in the reduction are shaded.

We assume without loss of generality (by symmetry) that A_2 is an upper right neighbor of A_1. With respect to A_1, the neighbor A_2 is either tall, solo, or paired. We first examine the case when A_2 is taller than A_1.

In this paper, we will only outline this case, giving just one proof; we will furthermore completely omit the three sections for the other cases.

4.1 Case 1: A_2 is a tall neighbor of A_1

In this case, A_1 must have at least one child (the deepest leaf L) and can have at most two children. All of A_1's children are left children.

Lemma 5 *If A_2 is a tall upper right neighbor of A_1, and A_1 has two children, then P_k can be reduced by 3 rectangles at a cost of 2 beacons.*

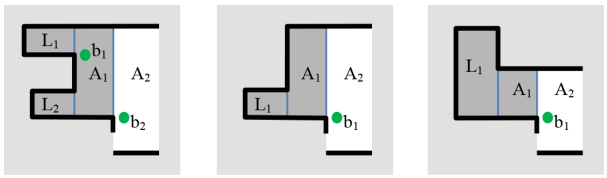


Figure 6: A_2 is a tall neighbor of A_1. (a) A_1 has two children L_1 and L_2. (b) A_1 has a solo lower-left child. (c) A_1 has a tall lower-left child.

Proof. The two children L_1 and L_2 must be left paired children, as shown in Figure 6a.

In this situation, we remove 3 rectangles (L_1, L_2, and A_1) at a cost of placing 2 beacons (b_1 and b_2). Now we show that, if P_{k+1} has a set B_{k+1} of beacons that allows a routing, then P_k has a set of beacons $B_k = B_{k+1} \cup \{b_1, b_2\}$ that allows a routing.

Let $C = P_k \setminus P_{k+1}$, i.e. C is the union of the rectangles L_1, L_2, and A_1. Also let $B = \{b_1, b_2\}$. Now the conditions of Lemma 4 are seen to be satisfied: b_1 covers the cut-off rectangles L_1, L_2, and A_1 (by Lemma 2); b_1 and b_2 are visible, so B' is strongly connected in the attraction graph, and b_2 is in repair position in P_{k+1}. □

Lemma 6 *If A_2 is a tall upper right neighbor of A_1, and A_1 has one lower-left child, then P_k can be reduced by 2 rectangles at a cost of 1 beacon (see Figure 6b and 6c).*

Lemma 7 *If A_2 is a tall upper right neighbor of A_1, and A_1 has one short upper-left child, then P_k can be reduced by 2 rectangles at a cost of 1 beacon (see Figure 7a).*

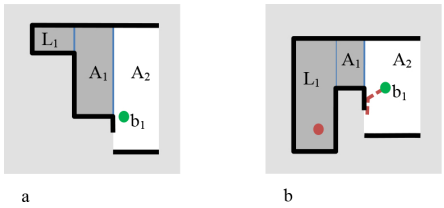


Figure 7: A_2 is a tall neighbor of A_1. (a) A_1 has a short upper-left child L_1. (b) A_1 has a tall upper-left child L_1; the point b_1 is not attracted by the point in L_1.

Figure 7b shows the situation when L_1 is a tall upper-left child of A_1. Here a beacon at b_1 would not suffice, as any point of L_1 below b_1 would not attract b_1. The technique we use to handle this case involves analyzing A_2 and all of its descendants.

4.2 The induction basis

The basis cases are when there are only one to three levels in the dual tree. If it is one level, the polygon is a rectangle. If it is two levels, the polygon is a 6-vertex "L" shape. In both of these cases, every point in the polygon attracts every other point in the polygon (see Lemma 1). Thus, there are no intermediate beacons required and the smallest beacon routing set is of size 0. We omit the analysis for a three-level dual tree.

4.3 The result

Theorem 8 *Any orthogonal polygon of n vertices has a local beacon routing set of at most $\lfloor \frac{n-4}{3} \rfloor$ beacons.*

Proof. Let r be the number of rectangles in the vertical decomposition of the polygon. Since $n = 2r+2$, the floor in the theorem is equivalent to $\left\lfloor \frac{(2r+2)-4}{3} \right\rfloor = \lfloor \frac{2r-2}{3} \rfloor$. We proceed to prove that there is a beacon set no larger than this, by induction on r.

Our basis has $r = 1$ to 6, with each case having a local beacon routing set of 0, 1, or 2 beacons, as discussed above. The number of beacons in each of the cases satisfies $b \leq \lfloor \frac{2r-2}{3} \rfloor$.

For our inductive step, assume $r \geq 3$ and we have rooted the dual tree at a leaf, so the depth of the dual tree is at least 2. One of the lemmas from the case analysis will apply, giving a reduction of 2 rectangles for 1 beacon, 3 rectangles for 2 beacons, 4 rectangles for 2 beacons, or 5 rectangles for 3 beacons. In each of these cases, we show that the local beacon routing set has at most $\lfloor \frac{n-4}{3} \rfloor$ beacons.

Take the first case: here we reduce P by 2 rectangles to construct a P' with $r' = r - 2$ rectangles. By induction P' has a local beacon routing set of at most $\left\lfloor \frac{2r'-2}{3} \right\rfloor = \left\lfloor \frac{2(r-2)-2}{3} \right\rfloor = \lfloor \frac{2r-6}{3} \rfloor$ beacons. To construct the beacon set for P, we add 1 beacon to that, and so we have at most $\lfloor \frac{2r-6}{3} \rfloor + 1 = \lfloor \frac{2r-3}{3} \rfloor \leq \lfloor \frac{2r-2}{3} \rfloor$ beacons.

The other cases proceed in an identical fashion, and the theorem follows. □

5 Lower bound

Here we establish that, for infinitely many n, there are orthogonal polygons that require at least $\lfloor \frac{(n-4)}{3} \rfloor$ beacons in any routing set. The examples are geometrically simple: each is an orthogonal spiral polygon with a "corridor width" of 1. Bae, Shin, and Vigneron have independently developed similar orthogonal lower-bound examples for the beacon-based art gallery problem [1].

Our polygons will spiral outwards clockwise as one moves through the reflex chain when walking counterclockwise around the polygon (i.e. left hand on interior). Call the reflex vertices of the polygon $r_1, r_2, \ldots r_{(n-2)/2}$ in this counterclockwise order, and let r_0 and $r_{n/2}$ denote the convex vertices adjacent to r_1 and $r_{(n-2)/2}$, respectively. Let c_k be the convex vertex just outside of (and closest to) r_k (refer to Figure 8). Let e_k be the edge from r_k to $r_k + 1$, and l_k be the length of e_k.

Now let C_k be the "corner" 1 by 1 square in P with vertices r_k and c_k, and H_k be the "hallway" rectangle (with dimensions 1 by l_k) between C_{k-1} and C_k.

If m_k^{in} is the midpoint of r_{k-1} and r_k, and m_k^{out} is the midpoint of c_{k-1} and c_k, we can partition the "hallway" H_k into two halves H_k^+ and H_k^- by splitting with its bisector $m_k^{\text{in}} m_k^{\text{out}}$. Let H_k^+ be the half adjoining C_k, and let that half (and not H_k^-) contain the points on the segment $m_k^{\text{in}} m_k^{\text{out}}$.

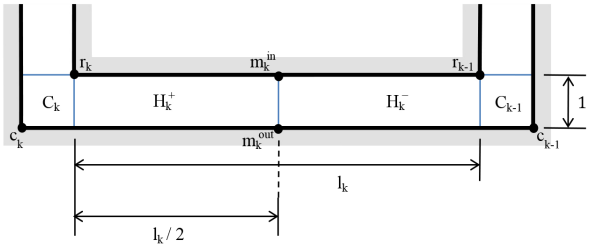


Figure 8: Notation for an orthogonal spiral.

We will construct polygons for $n = 6r + 4$ for some r; these polygons are specified simply by giving the lengths $l_1, l_2, \ldots l_{3r+1}$ of the $3r + 1$ "hallway" rectangles. Provided we have $l_j > l_{j-2} + 2$ for all $3 \leq j \leq 3r$, the polygon will spiral outward and not self-intersect.

We specify r sections $S_1, S_2, \ldots S_r$ of the polygon, by letting S_i be the union of $H^+_{3i-2}, C_{3i-2}, H_{3i-1}, C_{3i-1}, H_{3i}, C_{3i}$, and H^-_{3i+1} (see Figure 9). Note that no point of P is contained in more than one section, and there are points at either end of the spiral (in H^-_1 and H^+_{3r+1}) that are in no section.

Now consider a set of beacons B that can route in such a polygon P. We claim that $|B| \geq 2r$. If this were not the case, then by the pigeonhole principle some section S_i would contain fewer than two beacons.

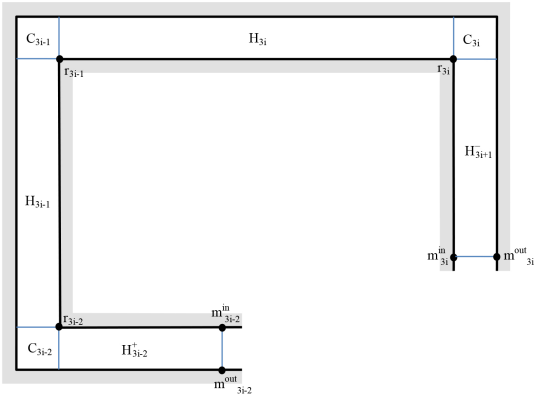


Figure 9: A section of an orthogonal spiral.

In the full paper, we proceed to show that if S_i contains only one beacon, then this beacon must lie in C_{3i-1}. In order to route from points "before" the section to points "after" it, and vice-versa, the beacon must lie in the shaded region in Figure 10, above the line $r_{3i-1}m^{\text{out}}_{3i+1}$ and below the line $r_{3i-1}m^{\text{out}}_{3i-2}$ (directions relative to the figure). By making l_{3i} (the vertical corridor on the right) long enough, we can cross these lines, leaving the reflex vertex r_{3i-1} as the only possibility for the beacon location.

Showing that this reflex vertex cannot properly be attracted to points both before and after section S_i is a purely *definitional* problem. A robot on a reflex vertex is a peculiar thing. There are many possible ways to define what happens when one pulls it towards the exterior: the robot path is indeterminate, the robot follows the wall to the left when it faces the direction of pull, the robot follows the horizontal wall, etc. The model of attraction must address this question somehow.

However, for reasonable, simple models, including those above, such a point r_{3i-1} cannot be successfully attracted both to points before and points after S_i. Thus, in these models, we have a contradiction;

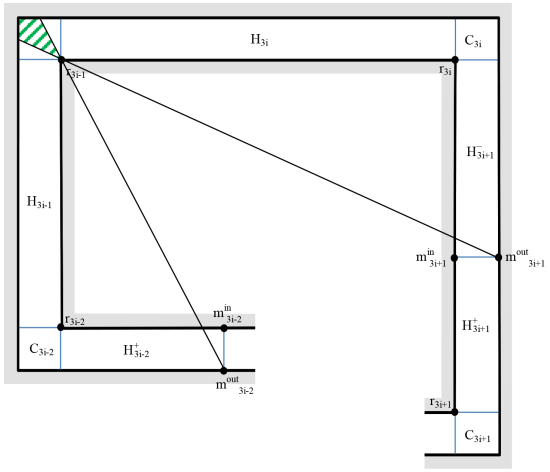


Figure 10: The beacon in S_i must lie in the shaded area.

each section must contain at least two beacons. Hence $|B| \geq 2r$. Since $n = 6r + 4$, $2r = \frac{n-4}{3}$, and we have:

Theorem 9 *For all $n \equiv 4 \pmod 6$, there are orthogonal spiral polygons requiring at least $\frac{n-4}{3}$ beacons in a routing beacon set.*

The constraint on the length of the spiral corridors in Section 5 works out to:

$$l_{3i+1} > \frac{4l_{3i}(l_{3i-1} + 1)}{l_{3i-2}},$$

whose solution is $l_k \in 2^{\Theta(k^2)}$. This growth rate is quite high, leaving us unable to provide figures illustrating these polygons. It would be interesting to try to develop alternative examples that do not have exponentially-growing coordinates.

References

[1] S. W. Bae, C.-S. Shin, and A. Vigneron. Improved bounds for beacon-based coverage and routing in simple rectilinear polygons. *arXiv preprint arXiv:1505.05106*, 2015.

[2] M. Biro. *Beacon-based routing and guarding*. PhD thesis, State University of New York at Stony Brook, 2013.

[3] M. Biro, J. Gao, J. Iwerks, I. Kostitsyna, and J. S. Mitchell. Combinatorics of beacon routing and coverage. In *Proceedings of the 25th Canadian Conference on Computational Geometry*, Waterloo, Ontario, 2013.

[4] M. Biro, J. Iwerks, I. Kostitsyna, and J. S. Mitchell. Beacon-based algorithms for geometric routing. In F. Dehne, R. Solis-Oba, and J.-R. Sack, editors, *Algorithms and Data Structures*, pages 158–169. Springer, 2013.

[5] P. Bose, P. Morin, I. Stojmenovic, and J. Urrutia. Routing with guaranteed delivery in ad hoc wireless networks. *Wireless Networks*, 7(6):609–616, 2001.

[6] A. Fournier and D. Y. Montuno. Triangulating simple polygons and equivalent problems. *ACM Transactions on Graphics (TOG)*, 3(2):153–174, 1984.

[7] B. Karp and H. T. Kung. Gpsr: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, MobiCom '00, pages 243–254, New York, NY, USA, 2000. ACM.

[8] A. Nguyen, N. Milosavljevic, Q. Fang, J. Gao, and L. Guibas. Landmark selection and greedy landmark-descent routing for sensor networks. In *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*, pages 661–669, May 2007.

[9] T. C. Shermer. A combinatorial bound for beacon-based routing in orthogonal polygons. *arXiv preprint (number pending)*, 2015.

Guarding Orthogonal Terrains

Stephane Durocher* Pak Ching Li* Saeed Mehrabi*

Abstract

A *1.5-dimensional terrain* T with n vertices is an x-monotone polygonal chain in the plane. A point guard p on T guards a point q of T if the line segment connecting p to q lies on or above T; p is a *vertex guard* if it is a vertex of T. In the *Optimal Terrain Guarding (OTG) problem* on T, the objective is to guard the vertices of T by the minimum number of vertex guards. King and Krohn [9] showed that the OTG problem is NP-hard on arbitrary terrains, and Gibson et al. [6] gave a PTAS for this problem. In this paper, we introduce *directed visibility* in which the visibility is directed only at adjacent vertices. We give an $O(n)$-time algorithm that solves the OTG problem exactly on *orthogonal* terrains under directed visibility.

1 Introduction

A *1.5-dimensional terrain* T is an x-monotone polygonal chain in the plane, where $V(T) = \{v_1, \ldots, v_n\}$ is the set of vertices of T ordered from left to right, and $E(T) = \{e_1 = (v_1, v_2), \ldots, e_{n-1} = (v_{n-1}, v_n)\}$ is the set of edges of T induced by the vertex set $V(T)$. Terrain T is called an *orthogonal terrain* if each edge $e \in E(T)$ is either horizontal or vertical. Let p be a point guard on T; p is called a *vertex* guard if $p \in V(T)$. A point q on T is seen/guarded by p (or, p sees/guards q) if and only if every point of the line segment $\overline{pq}$ lies either on or above T.

Given a (not necessarily orthogonal) terrain T, two common types of guarding problems are defined on T. In the *continuous* terrain guarding problem, the objective is to find a minimum-cardinality set S of points on T that *guards* T; that is, for every point $p \in T$, either p is in S or p is guarded by at least one point in S. In the *discrete* terrain guarding problem, on the other hand, two sets P and G of points on T are given along the terrain T as input and the objective is to find a subset $G' \subseteq G$ of minimum cardinality such that G' guards the points in P.

Related Work. The terrain guarding problem belongs to the well-known family of art gallery problems. The objective of the art gallery problem is to guard the interior of a polygon using the minimum number of point guards. The problem was first introduced by Klee in 1973 [12] and Chvátal [2] was the first to answer Klee's art gallery question by giving an upper bound proving that $\lfloor n/3 \rfloor$ point guards are always sufficient and sometimes necessary to guard a simple polygon with n vertices. The orthogonal art gallery problem was first studied by Kahn et al. [7] who proved that $\lfloor n/4 \rfloor$ guards are always sufficient and sometimes necessary to guard the interior of a simple orthogonal polygon with n vertices. In terms of the complexity of the art gallery problem, Lee and Lin [11] showed that the art gallery problem is NP-hard on simple polygons. Moreover, the problem is also NP-hard on simple orthogonal polygons [13] and it remains NP-hard even for monotone polygons [10]. Eidenbenz et al. [3] proved that the art gallery problem is APX-hard on simple polygons. They also showed that if the input polygon is allowed to have holes, then the problem cannot be approximated by a polynomial-time algorithm with factor $((1-\epsilon)/12)\ln n$ for any $\epsilon > 0$, where n is the number of the vertices of the polygon.

Ben-Moshe et al. [1] gave the first constant-factor approximation algorithm for the terrain guarding problem and left the complexity of the problem open. King and Krohn [9] showed that both continuous and discrete versions of the terrain guarding problem are NP-hard on arbitrary terrains. A 4-approximation algorithm for the terrain guarding problem was given by Elbassioni et al. [4], and Katz and Roisman [8] gave a 2-approximation algorithm for the OTG problem on orthogonal terrains. Gibson et al. [6] gave a polynomial-time approximation scheme (PTAS) for the discrete version of the terrain guarding problem, and a PTAS for the continuous version of the problem was recently given by Friedrichs et al. [5]. To the best of our knowledge, however, the complexity of the OTG problem on orthogonal terrains remains open. We note that the hardness result of King and Krohn [9] does not apply to the OTG problem on orthogonal terrains due to a number of essential differences between arbitrary and orthogonal terrains (e.g., see Lemma 4).

Problem Definition and Our Result. In this paper, we consider the discrete terrain guarding problem on an orthogonal terrain T under *directed visibility* such that $P = G = V(T)$; let $n = |V(T)|$. Directed visibility is defined as follows.

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

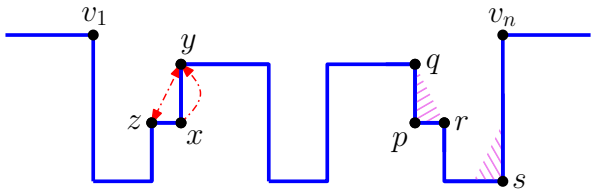


Figure 1: An orthogonal terrain T; throughout the paper, we assume that the leftmost and rightmost edges of T are two horizontal rays starting from v_1 and v_n, respectively. (a) An illustration of directed visibility: neither vertex y nor z can see vertex x under directed visibility, but they can see each other. The vertex x can see vertex y, but it cannot see vertex z because the line segment $\overline{xz}$ is horizontal. (b) The vertices q and r are reflex while the vertices p and s are convex. Moreover, q and r are both right reflex, p is left convex and s is right convex; vertex y is a left reflex vertex.

Definition 1 ***(Directed Visibility).*** *Let u be a vertex of T. 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 $\overline{uv}$ 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 non-horizontal line segment that lies on or above T.*

It is possible, under directed visibility, that a vertex u of T sees a vertex v, but vertex v cannot see u; see Figure 1(a) for an example. Therefore, we consider the following problem:

Definition 2 ***(The Directed Terrain Guarding (DTG) Problem on Orthogonal Terrains).*** *Given an orthogonal terrain T, compute a subset $S \subseteq V(T)$ of minimum cardinality that guards the vertices of T under directed visibility. That is, for every vertex $u \in V(T)$, either $u \in S$ or u is guarded by at least one other vertex in S under directed visibility.*

We give an $O(n)$-time algorithm for the DTG problem on orthogonal terrains under directed visibility. To this end, we first reduce the DTG problem to two subproblems such that an exact solution for the DTG problem reduces to the union of exact solutions of the two subproblems. We then give an $O(n)$-time greedy algorithm for solving each of the subproblems. To the best of our knowledge, this is the first exact algorithm for a non-trivial instance of the art gallery problem on terrains and partially answers a question posed by Ben-Moshe et al. [1] for orthogonal terrains.

1.1 Paper Organization

The paper is organized as follows. Section 2 presents preliminaries and some definitions. In Section 3, we give a characterization for an exact solution of the DTG problem: we define two subproblems and show that an exact solution for the DTG problem reduces to the union of the exact solutions of the subproblems. In Section 4, we show how to solve each subproblem in $O(n)$ time by a simple greedy algorithm. We conclude the paper in Section 5.

2 Preliminaries and Definitions

We denote the x- and y-coordinates of a point p on an orthogonal terrain T by $x(p)$ and $y(p)$, respectively. We use terms "terrain" and "guard" to refer to an orthogonal terrain and a vertex guard, respectively, unless otherwise stated. Moreover, we simply use "guarding" to mean "guarding under directed visibility" unless otherwise stated.

A vertex u of T is convex if the angle formed by the edges incident to u above T is $\pi/2$ degrees, otherwise u is reflex. We partition the vertices of T into 4 equivalences classes right or left endpoints of a horizontal edge of T, and whether the vertex is reflex or convex. We use $V_{LC}(T)$, $V_{RC}(T)$, $V_{LR}(T)$ and $V_{RR}(T)$ to respectively denote the left convex, right convex, left reflex, and right reflex subsets of the vertices of T. See Figure 1(b) for an example of these definitions.

For consistency, we assume that the leftmost and rightmost edges of T are two horizontal rays starting from v_1 and v_n, respectively; see Figure 1 for an illustration. For a reflex vertex u of T, we denote the convex vertex directly below u by $B(u)$. We say that a subset M of vertices of T guards a subset M' of vertices of T, where $M \cap M' = \emptyset$, if every vertex in M' is guarded by at least one vertex in M. We first describe some properties of orthogonal terrains.

Observation 1 *Let u and v be two reflex vertices of T. If vertex u sees $B(v)$, then u must also see v; see Figure 2 for an illustration.*

Let u and v be two convex vertices of T. If $y(u) = y(v)$, then clearly u and v cannot see each other under directed visibility because the line segment $\overline{uv}$ is horizontal. If $y(u) \neq y(v)$, then depending on the x-coordinates of u and v the line segment $\overline{uv}$ will pass through the region below the horizontal edge incident to either u or v and, therefore, u and v cannot see each other. This leads to the following lemma.

Lemma 1 *No two convex vertices of T can see each other under directed visibility.*

Observation 2 *Let u be a reflex vertex of a terrain T. If u is right reflex and sees a right convex vertex v of T, then $x(u) < x(v)$ and $y(u) > y(v)$. Similarly, if u is left reflex and sees a left convex vertex v of T, then $x(u) > x(v)$ and $y(u) > y(v)$.*

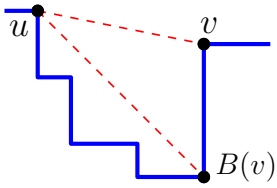


Figure 2: If a reflex vertex u sees $B(v)$, for some reflex vertex v, then u must also see vertex v itself.

Since directed visibility imposes a constraint relative to the standard visibility, the visibility graph of the vertices of T under directed visibility is a subgraph of that of the vertices of T under standard visibility. Therefore, the following property, called *the order claim*, still holds under directed visibility:

Lemma 2 (Ben-Moshe et al. [1]) *Let p, q, r and s be four vertices of a terrain T such that $x(p) < x(q) < x(r) < x(s)$. If p sees r and q sees s, then p sees s.*

Lemma 3 *Let u be a reflex vertex of a terrain T. If u is right reflex (resp., left reflex), then u cannot see any left convex (resp., right convex) vertex of T.*

Proof. We prove the lemma for when u is right reflex; the other case is proved by a symmetric argument. Let v be a left convex vertex of T. If $x(v) = x(u)$, then $v = B(u)$ and, therefore, u cannot see v under directed visibility. If $x(v) \neq x(u)$, then there are three cases.

- If $y(v) = y(u)$, then v is the adjacent vertex to the left of u and so u cannot see v under directed visibility.
- If $y(v) > y(u)$, then the line segment $\overline{uv}$ passes through the region below the horizontal edge incident to v and, therefore, vertex u cannot see v.
- If $y(v) < y(u)$, then there are two cases: (i) if $x(v) < x(u)$, then the line segment $\overline{uv}$ passes through the region below the horizontal edge incident to u and, therefore, vertex u cannot see v. (ii) If $x(v) > x(u)$, then the line segment $\overline{uv}$ passes through the region to the left of the vertical edge incident to v and, therefore, vertex u cannot see v.

The three cases above complete the proof of the lemma. □

In an arbitrary terrain, it is possible that a reflex vertex can guard both a left and a right convex vertex. For orthogonal terrains, however, this is not the case. This property is stated in the following lemma.

Lemma 4 *Let u be a right convex vertex and v be a left convex vertex of a terrain T. Then, there is no reflex vertex of T that sees both u and v under directed visibility.*

Proof. By Lemma 3, (i) no left reflex vertex of T can see u, and (ii) no right reflex vertex of T can see v. Therefore, no reflex vertex of T can see both u and v. This completes the proof of the lemma. □

3 An Exact Algorithm for the DTG Problem

In this section, we present our exact $O(n)$-time algorithm for the DTG problem on orthogonal terrains. Let T be an orthogonal terrain with n vertices. To solve the DTG problem on T, we first show that the DTG problem on T can be reduced to two subproblems such that an exact solution for the DTG problem is equivalent to the union of the exact solutions for the two subproblems. The subproblems are defined as follows.

Definition 3 ***(The Left-Convex Guarding (LCG(M)) Problem).*** *Given a set $M \subseteq V_{LC}(T)$, the objective of the LCG(M) problem is to compute a minimum-cardinality set $M' \subseteq V(T)$ such that for every vertex $u \in M$, either $u \in M'$ or u is guarded by at least one vertex in M'.*

Definition 4 ***(The Right-Convex Guarding (RCG(M)) Problem).*** *Given a set $M \subseteq V_{RC}(T)$, the objective of the RCG(M) is to compute a minimum-cardinality set $M' \subseteq V(T)$ such that for every vertex $u \in M$, either $u \in M'$ or u is guarded by at least one vertex in M'.*

To compute an exact solution for the DTG problem on T, we first show that we can restrict our attention to solutions that are in standard form. A feasible solution S to the DTG problem on T is in *standard form* if and only if every reflex vertex in S sees at least one convex vertex of T.

Lemma 5 *For any orthogonal terrain T, there exists an exact solution S for the DTG problem on T that is in standard form.*

Proof. Take any exact solution S_0 for the DTG problem on T. We construct a feasible solution S from S_0 such that $|S| \leq |S_0|$ and S is in standard form. To this end, for each reflex vertex $u \in S_0$ that does not see any convex vertex of T, replace u with $B(u)$ (i.e., the convex vertex directly below u). Let S be the resulting set. Clearly, $|S| \leq |S_0|$ and every reflex vertex in S sees at least one convex vertex of T. We now show that S is a feasible solution for the DTG problem on T. Consider a reflex vertex $u \in S_0$ that was replaced by $B(u)$ in S and let $Vis(u)$ be the set of vertices of T that are seen by u. We next prove that every vertex in $Vis(u)$ is still guarded by at least one vertex in S. First, note that every vertex in $Vis(u)$ is a reflex vertex. Let $v \in Vis(u)$ and consider $B(v)$. If $B(v) \in S$, then v is guarded by at least one vertex in S (i.e., the vertex

$B(v)$). If $B(v) \notin S$, then there must be a reflex vertex $w \in S_0$ that guards $B(v)$ because no two convex vertices of T can guard each other by Lemma 1. We note that $w \in S$ because w sees at least one convex vertex of T and so we have not replaced it with $B(w)$ in S. By Observation 1, vertex $w \in S$ guards v and, therefore, S is a feasible solution. Since $|S| \leq |S_0|$, the set S is an exact solution for the DTG problem on T that is in standard form. This completes the proof of the lemma. □

The following lemma, whose proof is given in Appendix A due to space constraints, states a necessary and sufficient condition for solving the DTG problem on T.

Lemma 6 *Let S be a feasible solution for the DTG problem on T. The set S is an exact solution if and only if there exists a partition $\{S_L, S_R\}$ of S such that (i) the set S_L is an exact solution for the LCG($V_{LC}(T)$) problem on T, and (ii) the set S_R is an exact solution for the RCG($V_{RC}(T)$) problem on T.*

By Lemma 6, we have the following theorem.

Theorem 7 *To solve the DTG problem on T, it is sufficient to solve the LCG($V_{LC}(T)$) and the RCG($V_{RC}(T)$) problems on T.*

4 Solving the LCG($V_{LC}(T)$) Problem

In this section, we present an $O(n)$-time exact algorithm for the LCG($V_{LC}(T)$) problem on T; an exact algorithm for the RCG($V_{RC}(T)$) problem can be derived analogously. First, by Lemma 1 (no convex vertex of T can see one other convex vertex of T) and Lemma 3 (no left reflex vertex of T can see a right convex vertex of T), we have the following result.

Lemma 8 *If M is a feasible solution for the LCG($V_{LC}(T)$) problem on T, then $M \subseteq \{V_{LC}(T) \cup V_{LR}(T)\}$.*

Next, we show that we can restrict our attention to solutions that are in a standard form. A feasible solution M for the LCG($V_{LC}(T)$) problem on T is in *standard form* if and only if a left convex vertex u is in M if and only if no reflex vertex of T can see u.

Lemma 9 *For any orthogonal terrain T, there exists an exact solution M for the LCG($V_{LC}(T)$) problem on T that is in standard form.*

Proof. Take any exact solution M_0 for the LCG($V_{LC}(T)$) problem on T. We construct a feasible solution M from M_0 such that $|M| \leq |M_0|$ and M is in standard form. For every left convex vertex $u \in M_0$ that is seen by at least one left reflex vertex v of T, replace u with v; let M be the resulting set. Clearly, $|M| \leq |M_0|$. Moreover, M is a feasible solution for the LCG($V_{LC}(T)$) problem on T because (i) the vertex u is now guarded by v, and (ii) the vertex u, which is left convex, cannot see any other left convex vertex of T. Therefore, every left convex vertex of T is still guarded by at least one vertex in M. Since $|M| \leq |M_0|$ and no left convex vertex of T that is in M is seen by a left reflex vertex of T, we conclude that M is an exact solution for the LCG($V_{LC}(T)$) problem on T that is in standard form. □

4.1 A Characterization

To solve the LCG($V_{LC}(T)$) problem on T, we give a characterization for an exact solution of the LCG($V_{LC}(T)$) problem on T. The following lemma, whose proof is given in Appendix B due to space constraints, is similar to the one given in Lemma 6 for the DTG problem.

Lemma 10 *Let M be a feasible solution for the LCG($V_{LC}(T)$) problem on T. The set M is an exact solution if and only if there exists a partition $\{A, B\}$ of M such that (i) $u \in A$ if and only if u is a left convex vertex and no reflex vertex of T can see u, and (ii) $B = M \setminus A$ is a minimum-cardinality subset of $V_{LR}(T)$ that guards $V_{LC}(T) \setminus A$.*

A similar result can be derived for an exact solution of the RCG($V_{RC}(T)$) problem analogously.

Lemma 11 *Let M be a feasible solution for the RCG($V_{RC}(T)$) problem on T. The set M is an exact solution if and only if there exists a partition $\{P, Q\}$ of M such that (i) $u \in P$ if and only if u is a right convex vertex and no reflex vertex of T can see u, and (ii) $Q = M \setminus P$ is a minimum-cardinality subset of $V_{RR}(T)$ that guards $V_{RC}(T) \setminus P$.*

By Lemma 10 and Lemma 11, we have the following theorem.

Theorem 12 *To solve the LCG($V_{LC}(T)$) problem on T, it is sufficient to first find the subset A of $V_{LC}(T)$, where $u \in A$ if and only if no reflex vertex of T can see u, and then compute a minimum-cardinality subset B of $V_{LR}(T)$ that guards $V_{LC}(T) \setminus A$. Similarly, to solve the RCG($V_{RC}(T)$) problem on T, it is sufficient to first find the subset P of $V_{RC}(T)$, where $u \in P$ if and only if no reflex vertex of T can see u, and then compute a minimum-cardinality subset Q of $V_{RR}(T)$ that guards $V_{RC}(T) \setminus P$.*

4.2 A Greedy Algorithm

In this section, we show how to compute an exact solution for the LCG($V_{LC}(T)$) problem on T; an exact solution for the RCG($V_{RC}(T)$) problem on T can be

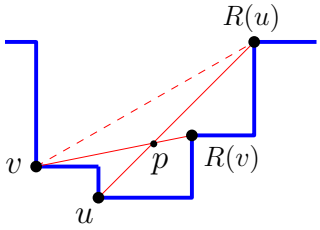


Figure 3: An illustration in support for the proof of Lemma 13.

computed analogously. By Theorem 12, we first compute the set A, where $u \in A$ if and only if u is a left convex vertex and it is not seen by any reflex vertex of T. In Section 4.3, we give a linear-time algorithm for computing $R(u)$ for all the left convex vertices of u, where $R(u)$ is the rightmost left reflex vertex of T that sees u (see Lemma 14). Therefore, we can use that algorithm to determine whether a left convex vertex u of T is seen by any reflex vertex of T at all and, therefore, the set A can be computed in $O(n)$ time overall. Now, let $C = V_{LC}(T) \setminus A$. In the following, we give an $O(n)$-time greedy algorithm for the problem of guarding C with the minimum-cardinality subset B of $V_{LR}(T)$.

For each left convex vertex $u \in C$, let $R(u)$ be the rightmost left reflex vertex of T (i.e., the rightmost vertex in $V_{LR}(T)$) that sees u. Consider the left convex vertices of C from right to left: for each left convex vertex u in order, if u is not yet guarded by a reflex vertex in B, then we add $R(u)$ into B. Clearly, the set B is a feasible solution for guarding the vertices in C. Let B' be the set of convex vertices that force the algorithm to add a new guard into B. Clearly, $|B'| = |B|$. We now show that no left reflex vertex of T can see two vertices in B', which proves that the set B is an exact solution. Suppose for a contradiction that there exists a left reflex vertex v that sees two vertices w_i and w_j in B'. Without loss of generality, assume that $x(w_i) > x(w_j)$; that is, vertex w_i is guarded before vertex w_j in the ordering. Since v sees w_i, we must have that $x(R(w_i)) \geq x(v)$. Note that $x(R(w_i)) \neq x(v)$ because otherwise we would have not added a new guard for w_j. Therefore, we have the ordering $x(w_j) < x(w_i) < x(v) < x(R(w_i))$ such that w_j sees v and w_i sees $R(w_i)$. But, by Lemma 2, this means that w_j is seen by $R(w_i)$ which is a contradiction. This proves that no left reflex vertex of T can see two convex vertices in B' and so the set B is an exact solution for guarding the vertices in C.

4.3 Algorithmic Details

In this section, we show how to implement the algorithm in time linear in n, the number of vertices of T. Our implementation of the algorithm uses the following result.

Lemma 13 *Let u and v be two left convex vertices of T such that $x(v) < x(u)$. Then, the line segments $\overline{uR(u)}$ and $\overline{vR(v)}$ do not intersect at an interior point.*

Proof. Suppose for a contradiction that the line segments $\overline{uR(u)}$ and $\overline{vR(v)}$ intersect at an interior point p. Since $x(v) < x(u)$, we must have that $x(R(v)) < x(R(u))$. Therefore, we have the ordering $x(v) < x(u) < x(R(v)) < x(R(u))$; see Figure 3 for an example. By Lemma 2, the vertex v must see vertex $R(u)$, which is a contradiction to the fact that $R(v)$ is the rightmost left reflex vertex of T that sees v. This completes the proof of the lemma. □

Consider the left convex vertices of T from right to left and let u and v be two left convex vertices such that $x(v) < x(u)$. By Lemma 13, vertex $R(v)$ cannot lie between the vertices u and $R(u)$; that is, vertex $R(v)$ is either $R(u)$ or a vertex to the right of $R(u)$, or it is a vertex to the left of vertex u. This property leads us to a linear-time algorithm for computing $R(u)$ for all the left convex vertices u in C as follows. Consider the vertices in $\{C \cup V_{LR}(T)\}$ from right to left in order. Note that the first vertex must be a left reflex vertex r. Moreover, we assume that the second vertex is also left reflex; otherwise, we set $R(u)$ to r for every visited left convex vertex until we reach to a left reflex vertex s; we push r and s into a stack `S` in the order they have been visited. In the following, let s and r be the vertices on top of the stack `S`. Moreover, let t be the next visited vertex and let α be the angle formed by the line segments $\overline{ts}$ and $\overline{sr}$ that faces above T:

- if t is left reflex, then we pop the two vertices s and r from `S`. If $\alpha > \pi$, then we push the three vertices r, s and t into the stack `S`; otherwise, we ignore vertex s and push only vertex r into `S`. Now, we repeat the same procedure with the current two top vertices s' and r' of `S` until α becomes greater than π in which case we push the three vertices r', s' and t into `S`.

- if t is left convex, then we pop the two vertices s and r from `S`. If $\alpha > \pi$, then we set $R(t)$ to s and push vertices r and s back into the stack `S`; otherwise, we ignore vertex s and push only vertex r into `S`. Now, we repeat the same procedure with the current two top vertices s' and r' of `S` until α becomes greater than π in which case we set $R(t)$ to s' and push r' and s' into the stack `S`.

See Figure 4 for an example of the algorithm. Let u be a left reflex vertex of T. If $\alpha > \pi$, then we process u in $O(1)$ time and move to the next vertex. If $\alpha \leq \pi$, then one vertex is removed from the stack `S` and we then repeat the same procedure which may consist of removing further vertices from `S`. Therefore, at each left reflex vertex u, either we perform an $O(1)$-time operation or

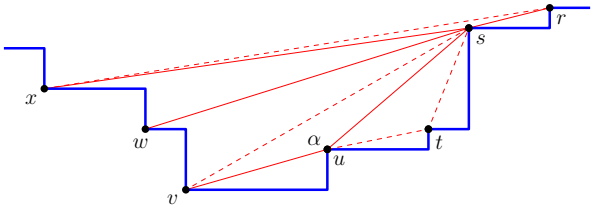


Figure 4: An example illustrating the computation of $R(v)$, $R(w)$ and $R(x)$. After processing vertex u, the status of the stack S from top to bottom is: $[u, s, r]$. When processing vertex v, vertex u is removed from S since $\alpha < \pi$ for the line segments $\overline{vu}$ and $\overline{us}$; then $R(v)$ is set to s. Vertex $R(w)$ is also set to s because $\alpha > \pi$ for the line segments $\overline{ws}$ and $\overline{sr}$. Finally, vertex s is removed from S and $R(x)$ is set to r. The final status of S is: $[r]$.

we remove a set S_u of vertices from S permanently. Note that by Lemma 13, the vertices in S_u will not be pushed back into S in the future. We can show using an analogous argument that at each left convex vertex, either we perform an $O(1)$-time operation or we remove a set of vertices from S permanently.

Although this procedure was described for computing $R(u)$ for all the left convex vertices in C, in fact it can be used to compute $R(u)$ for all the left convex vertices of T in $O(n)$ time. This leads us to the following lemma:

Lemma 14 *Given an orthogonal terrain T, the overall procedure of computing $R(u)$ for all the left convex vertices u of T can be completed in $O(n)$ time, where $|V(T)|$.*

By Lemma 14, we have the following theorem.

Theorem 15 *The LCG($V_{LC}(T)$) problem on T can be solved exactly in $O(n)$ time, where $n = |V(T)|$.*

We note that the RCG($V_{RC}(T)$) problem on T can be solved analogously in $O(n)$ time. Let S_1 and S_2 be the exact solutions for the LCG($V_{LC}(T)$) and the RCG($V_{RC}(T)$) problems on T, respectively. By Theorem 7, the set $S = \{S_1 \cup S_2\}$ is an exact solution for the DTG problem on T. Therefore, by Theorem 15, we have the main result of this paper.

Theorem 16 *There exists an $O(n)$-time exact algorithm for the DTG problem on any orthogonal terrain T with n vertices.*

5 Conclusion

In this paper, we considered the problem of guarding the vertices of an orthogonal terrain T with the minimum number of vertex guards under directed visibility (i.e., the DTG problem). We showed that the DTG problem on T is linear-time tractable by first reducing the problem to two subproblems (i.e., the $LCG(V_{LC}(T))$ and $RCG(V_{RC}(T))$ problems) and then solving each subproblem by a greedy algorithm that runs in $O(n)$ time, where n is the number of the vertices of T. Our algorithm assumes the directed visibility and it does not apply to the DTG problem under standard visibility. The complexity of the problem remains open without the directed visibility constraint.

References

[1] B. Ben-Moshe, M. J. Katz and J. S. B. Mitchell. A constant-factor approximation algorithm for optimal terrain guarding. In *SODA*, 515–524, 2005.

[2] V. Chvátal. A combinatorial theorem in plane geometry. In *Journal of Combinatorial Theory, Series B*, 18:39–41, 1975.

[3] S. Eidenbenz, C. Stamm and P. Widmayer. Inapproximability Results for Guarding Polygons and Terrains In *Algorithmica*, 31(1):79–113, 2001.

[4] K. M. Elbassioni, E. Krohn, D. Matijevic, J. Mestre and D. Severdija. Improved Approximations for Guarding 1.5-Dimensional Terrains. In *Algorithmica*, 60(2):451–463, 2011.

[5] S. Friedrichs, M. Hemmer and C. Schmidt. A PTAS for the continuous 1.5D Terrain Guarding Problem. In *CCCG*, 2014.

[6] M. Gibson, G. Kanade, E. Krohn and K. R. Varadarajan. An Approximation Scheme for Terrain Guarding. In *APPROX-RANDOM*, 140–148, 2009.

[7] J. Kahn, M. M. Klawe and D. J. Kleitman. Traditional galleries require fewer watchmen. In *SIAM Journal on Algebraic Discrete Methods*, 4(2):194–206, 1983.

[8] M. J. Katz and G. S. Roisman. On guarding the vertices of rectilinear domains. In *Comput. Geom.*, 39(3):219–228, 2008.

[9] J. King and E. Krohn. Terrain Guarding is NP-Hard. In *SIAM J. Comput.*, 40(5):1316–1339, 2011.

[10] E. Krohn and B. J. Nilsson. Approximate Guarding of Monotone and Rectilinear Polygons. In *Algorithmica*, 66(3):564–594, 2013.

[11] D. T. Lee and A. K. Lin. Computational complexity of art gallery problems. In *IEEE Transactions on Information Theory*, 32(2):276–282, 1986.

[12] J. O'Rourke. Art gallery theorems and algorithms. *Oxford University Press, Inc.*, 1987.

[13] D. Schuchardt and H.-D. Hecker. Two NP-hard art-gallery problems for ortho-polygons. In *Mathematical Logic Quarterly*, 41(2):261–267, 1995.

Appendix A: Proof of Lemma 6

Proof. ($\Rightarrow$) Let S be an exact solution for the DTG problem on T; by Lemma 5, we assume that S is in standard form. Let $S_L \subseteq S$ such that $u \in S_L$ if and only if u is either a left convex vertex or it is a left reflex vertex of T. Similarly, let $S_R \subseteq S$ such that $v \in S_R$ if and only if v either is a right convex vertex or it is a right reflex vertex of T that sees at least one right convex vertex. Since S is in standard form, $\{S_L, S_R\}$ is a partition of S.

We first prove that S_L is a feasible solution for the LCG($V_{LC}(T)$) problem on T. Let a be a left convex vertex of T. If $a \in S$, then $a \in S_L$. If $a \notin S$, then by Lemma 3 and the fact that no convex vertex can see another convex vertex (see Lemma 1), we conclude that there must be a left reflex vertex $b \in S$ that guards a and, therefore, $b \in S_L$. This means that for every left convex vertex a of T, we have either $a \in S_L$ or a is guarded by at least one vertex in S_L. Therefore, S_L is a feasible solution for the LCG($V_{LC}(T)$) problem on T. By an analogous argument, we can show that S_R is a feasible solution for the RCG($V_{RC}(T)$) problem on T.

We next prove that S_L is an exact solution for the LCG($V_{LC}(T)$) problem on T. Suppose for a contradiction that there exists a feasible solution S'_L for the LCG($V_{LC}(T)$) problem on T such that $|S'_L| < |S_L|$. In the following, we prove that the set $\{S'_L \cup S_R\}$ is a feasible solution for the DTG problem on T, which is a contradiction to the fact that S is an exact solution for the DTG problem on T because $|S'_L \cup S_R| \leq |S'_L| + |S_R| < |S_L| + |S_R| = |S|$ (the last equality follows from the fact that $\{S_L, S_R\}$ is a partition of S). Let u be a vertex of T. If u is left convex, then u is either in S'_L or it is guarded by a left reflex vertex in S'_L because S'_L is a feasible solution for the LCG($V_{LC}(T)$) problem on T. Similarly, if u is a right convex vertex, then u is either in S_R or it is guarded by a right reflex vertex in S_R because S_R is a feasible solution for the RCG($V_{RC}(T)$) problem on T. Now, suppose that u is a reflex vertex that is not in $S'_L \cup S_R$. Then, consider the vertex $B(u)$. If $B(u) \in \{S'_L \cup S_R\}$, then u is guarded by at least one vertex in $S'_L \cup S_R$ (i.e., the vertex $B(u)$). If $B(u) \notin \{S'_L \cup S_R\}$, then it must be guarded by a reflex vertex $w \in \{S'_L \cup S_R\}$ because no two convex vertices of T can see each other by Lemma 1. By Observation 1, vertex w must also guard the vertex u. This proves that every vertex of T that is not in $S'_L \cup S_R$ is guarded by at least one vertex in $S'_L \cup S_R$ and, therefore, $S'_L \cup S_R$ is a feasible solution for the DTG problem on T. By an analogous argument, we can show that S_R is an exact solution for the RCG($V_{RC}(T)$) problem on T.

($\Leftarrow$) Suppose that there exists a partition $\{S_L, S_R\}$ of S such that S_L is an exact solution for the LCG($V_{LC}(T)$) problem on T and S_R is an exact solution for the RCG($V_{RC}(T)$) problem on T. We now prove that $S = \{S_L \cup S_R\}$ is an exact solution for the DTG problem on T. Suppose for a contradiction that there exists a feasible solution S' for the DTG problem on T such that $|S'| < |S|$; by Lemma 5, we assume that S' is in standard form. Let X be a subset of S' such that $u \in X$ if and only if u is either a left convex vertex or it is a left reflex vertex of T. Similarly, let Y be a subset of S' such that $v \in Y$ if and only if v is either a right convex vertex or it is a right reflex vertex of T. Since S' is in standard form, $\{X, Y\}$ is a partition of S'. Since $|S'| < |S|$, we must have $|X| < |S_L|$ or $|Y| < |S_R|$. Without loss of generality, assume that $|X| < |S_L|$. In the following, we show that X is a feasible solution for the LCG($V_{LC}(T)$) problem on T, which is a contradiction to the fact that S_L is an exact solution for the LCG($V_{LC}(T)$) problem on T. To show the feasibility of X, let x be a left convex vertex of T. If $x \in S'$, then $x \in X$. If $x \notin S'$, then we conclude by Lemma 3 that there must be a left reflex vertex $y \in S'$ that guards x. Since y guards at least one left convex vertex of T, we have $y \in X$. This means that every left convex vertex of T is either in X or it is guarded by at least one left reflex vertex in X. Therefore, the set X is a feasible solution for the LCG($V_{LC}(T)$) problem on T.

We have proved that it is not possible that $|S'| < |S|$ and, therefore, the set S is an exact solution for the DTG problem on T. This completes the proof of the lemma. □

Appendix B: Proof of Lemma 10

Proof. ($\Rightarrow$) Suppose that M is an exact solution for the LCG($V_{LC}(T)$) problem on T; by Lemma 9, we assume that M is in standard form. Let A be the subset of M such that $u \in A$ if and only if u is a left convex vertex of T, and let $B = M \setminus A$. Clearly, $\{A, B\}$ is a partition of M. Also, no reflex vertex of T can see a vertex in A because M is in standard form and, by Lemma 8, we have that $B \subseteq V_{LR}(T)$. Moreover, since M is a feasible solution for the LCG($V_{LC}(T)$) problem, every left convex vertex of T that is not in A is guarded by at least one left reflex vertex in B. Therefore, it only remains to show that B has minimum cardinality among all subsets of $V_{LR}(T)$ that guard $V_{LC}(T) \setminus A$. Suppose for a contradiction that $B' \subseteq V_{LR}(T)$ guards $V_{LC}(T) \setminus A$ such that $|B'| < |B|$. Then, $\{A \cup B'\}$ is a feasible solution for the LCG($V_{LC}(T)$) problem on T, but $|A \cup B'| \leq |A| + |B'| < |A| + |B| = |M|$ (the last equality is due to the fact that $\{A, B\}$ is a partition of M); this is a contradiction to the fact that M is an exact solution for the LCG($V_{LC}(T)$) problem on T.

($\Leftarrow$) Suppose that there exists a partition $\{A, B\}$ of M such that (i) $u \in A$ if and only if u is a left convex vertex and no reflex vertex of T can see u, and (ii) $B = M \setminus A$ is a minimum-cardinality subset of $V_{LR}(T)$ that guards $V_{LC}(T) \setminus A$. We now show that $M = \{A \cup B\}$ is an exact solution for the LCG($V_{LC}(T)$) problem on T. Suppose for a contradiction that there exists a feasible solution M' for the LCG($V_{LC}(T)$) problem on T such that $|M'| < |M|$. By Lemma 8, we have that $M' \subseteq \{V_{LC}(T) \cup V_{LR}(T)\}$. Partition M' into two sets X and Y such that $x \in X$ if and only if x is a left convex vertex that is not seen by any left reflex vertex of T, and let $Y = M' \setminus X$. We can assume that $Y \subseteq V_{LR}(T)$ because otherwise we can replace every left convex vertex y in Y with a left reflex vertex of T that sees y.[1] Recall that if $x \in X$, then no left reflex vertex of T can see x and, by Lemma 3, no right reflex vertex of T can see x. Therefore, $x \in A$ because no reflex vertex of T can see x and M is a feasible solution for the LCG($V_{LC}(T)$) problem on T. By an analogous argument, we can show that if $x \in A$, then

[1] Note that at least one such left reflex vertex of T exists because otherwise we would have added y into X.

$x \in X$. Therefore, $X = A$. This means that Y is a subset of $V_{LR}(T)$ that guards $V_{LC}(T) \setminus X = V_{LC}(T) \setminus A$. Since $X = A$ and $|M'| < |M|$, we must have that $|Y| < |B|$, which is a contradiction to the fact that B is a minimum-cardinality subset of $V_{LR}(T)$ that guards $V_{LC}(T) \setminus A$. This completes the proof of the lemma. □

Open Problems from CCCG 2014

Sue Whitesides *

Abstract

This report provides the problems posed by the participants at the open problem session of the 26^{th} Canadian Conference on Computational Geometry.

This well-attended session was held Tuesday, August 12, 2014, as a scheduled session of the conference. Six participants presented a total of seven problems. All presenters kindly agreed to provide written versions of their problems, including references and attributions. The problems appear in the sections below. The references appear at the end. The text is essentially the same, modulo minor editing, as the text provided by the presenters. This material is not refereed.

1 Guarding Orthogonal Terrains

presented by: Giovanni Viglietta[1]

Partition the plane into finitely many (possibly unbounded) orthogonal polygons, and extrude them in 3D, obtaining a set of "orthogonal skyscrapers" of different heights. Let n be the total number of vertices of the orthogonal polygons. We ask to find the minimum number (as a function of n) of vertex guards for the terrain induced by the skyscrapers. In other words, we seek to select a minimum number of "guards" among the vertices of the skyscrapers such that each point in 3-space lying "above" some skyscraper is visible to some guard, where lines of sight must not intersect a skyscraper's top face or a side face.

The best known lower bound is given by a row of k equal cuboidal skyscrapers, where $n = 8k$. In this case $k + 1$ vertex guards are needed, which yields a lower bound of $(n/8) + 1$ vertex guards. We conjecture $n/8 + O(1)$ guards to be sufficient for all orthogonal terrains on n vertices (observe that an L-shaped skyscraper on 12 vertices needs three guards). To our knowledge, the problem is open even in the case of a single "tower" made of nested orthogonal prisms of increasing height, or a single "well".

For background, see [1].

2 Flows on Terrains

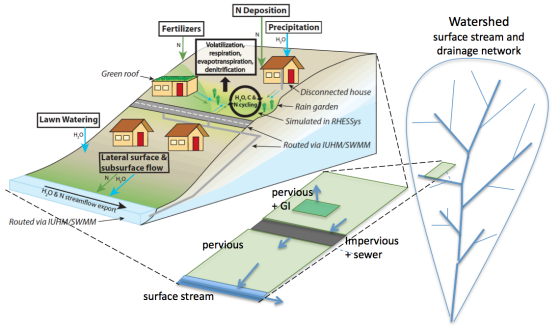
presented by: Jack Snoeyink[2]

What local actions can make a general difference for flow of water, nutrients, and pollutants in a terrain? This is more of an open application area for computational geometry techniques than an open problem.

Consider a real-world terrain with patches having different soil types (e.g., different absorbency properties) together with a network of streams, house gutters, parking lot drains, and underground sewers. There are rain gauges reporting rainfall in cm/hr at some points and flow meters reporting liters/min profiles on some waterways. (These are increasingly common in the "internet of things.")

If we model a rainfall, do we see the measured flows? If not, can we suggest where our information about the flow network is incomplete or inaccurate? If we don't like, say, the surge of flow in the sewers from a rainfall, can we suggest where rain gardens could most effectively delay the flow? At what scale should these questions be asked based on the sensors we have?

There are many simulations that are used [2, 3], but the ideas of computational geometry (like continuous Dijkstra for paths in weighted regions [4], or partitioning terrain into catchments and capturing flow in equilibrium [5]) can be used to preprocess the terrain for more efficient exploration of modifications that would produce the observed or desired flow profiles.



*Professor, Department of Computer Science, U. of Victoria, Canada; email: sue@uvic.ca

[1]Postdoctoral Fellow, U. of Ottawa and School of Computer Science, Carleton U., Canada; email: viglietta@gmail.com

[2]Professor, Dept. of Computer Science, U. North Carolina; email: snoeyink@cs.unc.edu

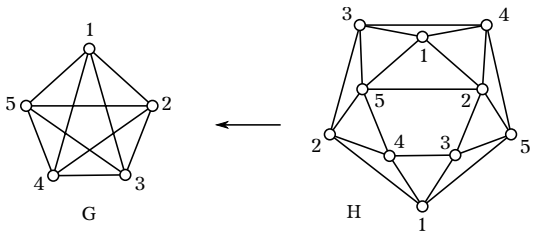


Figure 1: $H \simeq K_5$ and its finite planar emulator H.

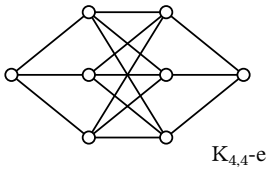


Figure 2: $K_{4,4} - e$, one of the minor-minimal obstructions for the projective plane, where the existence of a finite planar emulator is open.

3 Finding a Shared Delaunay Triangle in Linear Time

presented by: Michael Biro[3]

Let $P = \{p_1, p_2, \ldots, p_n\}$ be a set of n points in the plane and s, t be two query points. The problem is to determine, in linear time, whether or not s and t lie in the same face of the Delaunary triangulation of P. The problem was posed by Joseph S. B. Mitchell in personal correspondence.

The problem can be solved trivially in time $O(n \log n)$ by constructing the Delaunay triangulation and performing point location queries. However, constructing the full Delaunay triangulation has a lower bound of $\Omega(n \log n)$ so this approach cannot be used to determine the answer in linear time. Thus the question is asking, in essence, if we can quickly find local information about a Delaunay triangulation without first having to construct the entire triangulation.

One reason to expect the answer to be affirmative is that the dual question of determining if s and t lie in the same face of the Voronoi diagram of P is trivial to answer in time $O(n)$: simply find the nearest neighbors of s and t, respectively. The two points s and t share nearest neighbors if and only if they are in the same face of the Voronoi diagram of P.

Jack Snoeyink proposed a linear-time solution by lifting the set P to a paraboloid in 3D and locating the lifted points s and t on faces of the convex hull.

4 Finite Planar Emulators

presented by: Martin Derka[4]

A graph G has a finite *planar emulator* H if H is a planar graph and there is a graph homomorphism $\varphi : V(H) \to V(G)$ where φ is locally surjective, i.e. for every vertex $v \in V(H)$, the neighbours of v in H are mapped surjectively onto the neighbours of $\varphi(v)$ in G. We also say that such a G is *planar-emulable*. If we insist on φ being locally bijective, we get H a *planar cover*.

The concept of planar emulators was proposed in 1985 by M. Fellows [11], and it tightly relates (although it is of independent origin) to the better known *planar cover conjecture* of Negami [12]. Fellows also raised the main question: What is the class of graphs with finite planar emulators?

Soon thereafter, he conjectured that the class of planar-emulable graphs coincides with the class of graphs with finite planar covers (conjectured to be the class of projective graphs by Negami [12]—still open at present). This was later restated as follows:

Conjecture 1 *[M. Fellows, falsified in 2008] A connected graph has a finite planar emulator if and only if it embeds in the projective plane.*

It is known that if a graph embedds in the projective plane, it has a finite planar emulator (which takes form of its finite planar cover). The conjecture fails in the converse. Rieck and Yamashita [13], and Chimani et al. [6] constructed finite planar emulators of all the minor minimal obstructions for the projective plane with the exception of those that have been shown non-planar-emulable already by Fellows (the $K_{3,5}$ and "two disjoint k-graphs" cases), and with the exception of $K_{4,4} - e$. The graph $K_{4,4} - e$ is the only forbidden minor for the projective plane where the existence of a finite planar emulator remains open. For more examples of planar emulators and for some graphs that are not planar-emulable, see [6].

5 Colored Radial Orderings

presented by: Ruy Fabila-Monroy[5]

Let S be a set of n points in general position in the plane. Let p be a point not in S such that $S \cup \{p\}$ is in general position; we call p an *observation point*. A *radial ordering* of S with respect to p is a clockwise circular ordering of the points in S by their angle around p. If every point in S is assigned one of two colors, say red and blue, then a *colored radial ordering* of S with respect to p is a circular clockwise ordering of the colors of the points in S by their angle around p. Let

[3]Visiting Asst. Professor, Mathematics and Statistics, Swarthmore College PA; email: michael.j.biro@gmail.com
[4]Ph.D. Student, U. Waterloo; email: mderka@uwaterloo.ca
[5]Professor, Dept. of Mathematics, Cinvestav-IPN, Mexico; email: ruyfabila@math.cinvestav.edu.mx

$\rho(S)$ be the number of distinct radial orderings of S with respect to every observation point in the plane. Likewise, let $\operatorname{col}\rho(S)$ be the number of distinct colored radial orderings of S with respect to every observation point in the plane. Define the following functions:

$$g(n) := \min\{\rho(S) : S \text{ is a set of } n \text{ points}\}$$
$$g_{\text{col}}(n) := \min\{\rho_{\text{col}}(S) : S \text{ is a set of } m \text{ red and } m \text{ blue points, and } n = 2m\}$$

Open Problems

1. Give a tight asymptotic bound for $g(n)$.
2. Give a tight asymptotic bound for $g_{\text{col}}(n)$.

In [14] it is shown that $g(n) \geq \Omega(n^3)$ and it is conjectured that $g(n) = \Theta(n^4)$. For the colored case in the same paper they showed that $g_{\text{col}}(n) = \Omega(n)$ and gave an example of a set of n red and n blue points with $O(n^2)$ colored radial orderings.

6 Finite Simplicial Complexes

two problems presented by: Tamal Dey[6]

Problem 1: Let $K := K(P)$ be a finite simplicial complex linearly embedded in $\mathbb{R}^d$ with vertex set P. Denote by $f_d^k(K, P)$ the number of k-simplices in K. Consider the following quantity:

$$f_d^k(n) = \max_{K, |P|=n} f_d^k(K, P).$$

What is the correct bound on $f_d^k(n)$ in terms of n, k, d? We know that $f_2^1(n) = \Theta(n)$ because planar graphs have at most $3n$ edges and clearly there are planar graphs with $\Omega(n)$ edges. Next question is: what is f_3^2, that is, how many triangles with a total of n vertices can be linearly embedded in $\mathbb{R}^3$? It was proved in [15] that $f_3^2 = O(n^2)$ and a tight lower bound of $\Omega(n^2)$ exists because cyclic polytopes with n vertices in $\mathbb{R}^3$ have a triangulation with $\Omega(n^2)$ triangles. Actually, the lower bound generalizes, that is,

$$f_d^k(n) = \Omega(n^{\min\{k+1, \lceil \frac{d}{2} \rceil\}})$$

because of the known lower bounds for triangulations of cyclic polytopes in $\mathbb{R}^d$. For example, in $\mathbb{R}^4$, of course there could be all possible $n(n-1)/2 = \Theta(n^2)$ edges, but all possible $\binom{n}{3} = \Theta(n^3)$ triangles cannot be linearly embedded. In fact, the following bound is known [16]

$$f_4^3(n) = O(n^{3-\frac{1}{3}}).$$

conjecture: $f_d^k(n) = \Theta(n^{\min\{k+1, \lceil \frac{d}{2} \rceil\}})$

Problem 2: Let K be a finite simplicial complex linearly emedded in $\mathbb{R}^3$. Let C be any given 1-cycle in K. We are interested in detecting if C is *trivial* in the first homology group, that is, if there is a set of triangles in K whose boundaries when summed over Z_2 give C. This problem can be solved in $O(M(n))$ time by first reducing the boundary matrix of K (triangle-edge matrix) to Echelon form and then reducing a column corresponding to C to see if it becomes an empty column or not. Here $M(n)$ is the matrix multiplication time whose current best bound is $O(n^{2.37..})$.

conjecture: Let K be a finite simplicial complex linearly embedded in $\mathbb{R}^3$ with a total of n simplices. Given a 1-cycle C in K, one can detect if C is trivial in the first homology group (with Z_2 coefficient) in $O(n^2)$ time.

If K is a 2-manifold, the detection can be performed in $O(n)$ time by a simple depth-first walk in K. If K is a 3-manifold, the algorithm in [17] can be modified to accomplish the task in $O(n^2)$ time. The question remains open for general simplicial complexes. Although the conjecture is posed here for K embedded in $\mathbb{R}^3$ and for a 1-cycle C, it can be posed for a finite simplicial complex embedded linearly in $\mathbb{R}^d$ and a given p-cycle C in it.

7 Acknowledgements

Thanks to all those who presented problems at the open problem session and later provided written versions for this report. Thanks to the attendees at the session for their participation and comments. Thanks to Norman Zeh and Meng He for the opportunity to chair the session, and thanks to them and also to Zahed Rahmati for technical assistance with the files.

References

[1] P. Bose, T. Shermer, G. Toussaint, and B. Zhu. Guarding Polyhedral Terrains. *Computational Geometry*, v. 7, 173–185, 1997.

[2] C. L. Tague and L. E. Band. RHESSys: Regional Hydro-Ecologic Simulation System—An Object-Oriented Approach to Spatially Distributed Modeling of Carbon, Water, and Nutrient Cycling. *Earth Interactions*, American Meteorological Society, v. 8, no . 19, 1–42, 2004.

[3] United States Environmental Protection Agency (EPA). Storm Water Management Model

[6] Professor, Dept. of Computer Science and Engineering, The Ohio State U.; email: tamaldey@cse.ohio-state.edu

(SWMM). URL http://www2.epa.gov/water-research/storm-water-management-model-swmm last checked Oct. 14, 2014.

[4] Joe Mitchell and Christos Papadimitriou. The Weighted Region Problem: Finding Shortest Paths Through a Weighted Planar Subdivision. *J. ACM*, ACM, New York, v. 38, no. 1, 18–73, 1991.

[5] Yuanxin Liu and Jack Snoeyink. Flooding Triangulated Terrain. *Developments in Spatial Data Handling*, Springer Berlin Heidelberg, 137–148, 2005.

[6] M. Chimani, M. Derka, P. Hliněný, M. Klusáček. How Not to Characterize Planar-emulable Graphs. *Advances in Applied Mathematics*, v. 50, 46–68, 2013.

[7] M. Derka. Planar Graph Emulators: Fellow's Conjecture. Bc. Thesis, Masaryk University, Brno, 2010.

[8] M. Derka. New Challenges in Planar Emulators. Master's Thesis, Masaryk University, Brno, 2013.

[9] M. Derka. Towards Finite Characterization of Planar-emulable Non-projective Graphs. *Congressus Numerantium*, v. 207, 33–68, 2011.

[10] M. Derka, P. Hliněný. Planar Emulators Conjecture Is Nearly True for Cubic Graphs. *European J. of Combinatorics*, to appear.

[11] M. Fellows, *Encoding Graphs in Graphs*. Ph.D. Dissertation, Univ. of California, San Diego, 1985.

[12] S. Negami. Enumeration of Projective-planar Embeddings of Graphs. *Discrete Math*, v. 62, no. 3, 299–306, 1986.

[13] Y. Rieck and Y. Yamashita. Finite Planar Emulators for $K_{4,5} - 4K_2$ and $K_{1,2,2,2}$ and Fellow's Conjecture. *European J. Combinatorics*, v. 31, 903–907, 2010.

[14] J.M. Dáz-Bañez, R. Fabila-Monroy, and P. Pérez-Lantero. On the Number of Radial Orderings of Planar Point Sets. In A. Márquez, P. Ramos, and J. Urrutia, *eds.*, *Computational Geometry: XIV Spanish Meeting on Computational Geometry*, EGC 2011, Springer Berlin Heidelberg, v. 7579 of Lecture Notes in Computer Science (LNCS), 109-118, 2012.

[15] T.K. Dey and H. Edelsbrunner. Counting Triangle Crossings and Halving Planes. *Discrete Comput. Geom.*, v. 12, 281–289, 1994.

[16] T. K. Dey. On Counting Triangulations in D Dimensions. *Computational Geometry: Theory and Applications*, v. 3, 315–325, 1993.

[17] T. K. Dey, F. Fan, and Y. Wang. An Efficient Computation of Handle and Tunnel Loops via Reeb Graphs. *ACM Trans. Graphics (Siggraph 2013)*, v. 32, no. 4, 2013.

One of Ferran Hurtado's favorite topics - Flips

Prosenjit Bose (Carleton University)

Abstract

Reconfiguring graphs via small local changes was one of Ferran Hurtado's favorite research topics. In fact, he had a web page dedicated to his results in the area which he called "The flips corner". In this talk, we will give an overview of the results in the area and highlight some of the impact Ferran has had. Along the way, we will also point out some of the techniques used to prove the main results and mention a few of the challenges remaining in this area.

Weighted Minimum Backward Fréchet Distance

Amin Gheibi*†1, Anil Maheshwari†1, and Jörg-Rüdiger Sack*†1

[1]School of Computer Science, Carleton University, Ottawa, ON, Canada
[agheibi,anil,sack]@scs.carleton.ca

Abstract

The minimum backward Fréchet distance (MBFD) problem is a natural optimization problem for the weak Fréchet distance, a variant of the well-known Fréchet distance. In this problem, a threshold ε and two polygonal curves, T_1 and T_2, are given. The objective is to find a pair of walks on T_1 and T_2, which minimizes the union of the portions of backward movements while the distance between the moving entities, at any time, is at most ε. In this paper, we generalize this model to capture scenarios when the cost of backtracking on the input polygonal curves is not homogeneous. More specifically, each edge of T_1 and T_2 has an associated non-negative weight. The cost of backtracking on an edge is the Euclidean length of backward movement on that edge multiplied by the corresponding weight. The objective is to find a pair of walks that minimizes the sum of the costs on the edges of the curves, while guaranteeing that the curves remain at weak Fréchet distance ε. We propose an exact algorithm whose run time and space complexity is $\mathcal{O}(n^3)$, where n is the maximum number of the edges of T_1 and T_2.

1 Introduction

Measures for similarity between two polygonal curves have been studied in areas such as computational geometry, Geographical Information Systems (GIS), pattern recognition, shape matching, and robotics. Finding measures that capture the requirements of a particular domain remains challenging, both in practice and theory. One of the widely used measures for similarity between curves is the Fréchet distance which takes into account global features of the curves [1]. In some applications (such as map matching) a global approach as taken e.g., by the Fréchet distance, achieves a better result than a local approach [2].

The Fréchet distance is typically illustrated via the person-dog metaphor. Assume that a person wants to walk along one curve and his/her dog on another. Each curve has a starting and an ending point. The person and the dog walk, from the starting point to the ending point, along their respective curves. The standard Fréchet distance is the minimum leash length required for the person to walk the dog without backtracking. A variant of the standard Fréchet distance is the weak Fréchet distance, also known as non-monotone Fréchet distance [1]. In this variant, backtracking is allowed during the walks. In [1], Alt and Godau proposed algorithms to compute the weak Fréchet distance in $\mathcal{O}(n^2 \log n)$ time, where n is the maximum number of segments in the input polygonal curves. The time complexity is improved by Har-Peled and Raichel [6]. They proposed an algorithm with quadratic time complexity for computing a generalization of the weak Fréchet distance. In some applications, the weak Fréchet distance is preferable to the standard Fréchet distance (see [2]).

In [4], Gheibi et al. introduced and solved an optimization problem on the weak Fréchet distance, called the *minimum backward Fréchet distance (MBFD)* problem. Their problem is to determine the minimum total length of backward movements on both input polygonal curves, required for the walks to achieve the given leash length. In that paper, it is assumed that the cost (i.e., weight) of backward movement is uniform and depends only on the Euclidean distance traveled on each of the input polygonal curves. They proposed an algorithm with time complexity $\mathcal{O}(n^2 \log n)$ and space complexity $\mathcal{O}(n^2)$, to solve MBFD exactly. Here, in this paper, we generalize this model to capture scenarios when the cost of backtracking on the input polygonal curves is not homogeneous. These weights could represent, for example, the cost of moving against a flow, or the cost for a moving entity (e.g., a human) to move backwards because of the entity's physiology [3]. Thus, in the new model, each edge of the input polygonal curves has an associated non-negative weight for backward movement. Then, the cost of backtracking on an edge is the Euclidean length of backward movement on that edge multiplied by the corresponding weight. The objective is to design an algorithm that a) finds a pair of walks that minimizes the sum of the costs on the edges of the curves, while guaranteeing that the leash length is at most ε, b) halts

*Research supported by High Performance Computing Virtual Laboratory and SUN Microsystems of Canada

†Research supported by Natural Sciences and Engineering Research Council of Canada

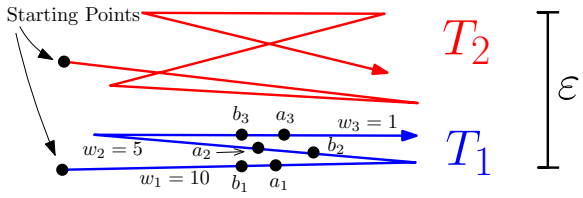


Figure 1: Moving backwards from a_3 to b_3 allows to walk on T_1 and T_2 and keeping the distance between moving entities at most ε during the walks while the cost is minimized.

with the answer of no feasible solution if such a pair of walks does not exist for the given leash length. We call this problem, the *weighted minimum backward Fréchet distance (WMBFD) problem*. Note that if the standard Fréchet distance between the input curves is already at most ε, then no backtracking is necessary and the optimal solution is identical to a pair of walks that realizes the Fréchet distance.

Figure 1 shows an example. In this figure, two polygonal curves, T_1 and T_2, and a length ε are drawn. The person walks on T_1 and the dog walks on T_2. The weights, w_i, $i = 1, 2, 3$, for segments of T_1 are given. For this illustration, we let the cost of backtracking on all segments of T_2 be 1. In this example, it is impossible to walk from the starting point to the end and maintain the leash length at most ε, without moving backwards. Six points, a_1, b_1, a_2, b_2, a_3, and b_3 are specified on T_1. If the person moves backwards, either from a_1 to b_1, or from a_2 to b_2, or from a_3 to b_3, then the curves are at weak Fréchet distance ε. In this example, the Euclidean length of $\overline{a_1b_1}$ is less than the Euclidean length of $\overline{a_3b_3}$. However, the weight of moving backwards on the first segment is 10, while that on the third one is 1. Therefore, the pair of walks that minimizes the cost is as follows: the dog and the person move forwards together from the starting point, until the dog reaches the end of the third segment of T_2 and the person reaches the point a_3 on T_1. Then, the dog keeps moving forwards until the end of the fourth segment of T_2, while the person moves backwards from a_3 to b_3. Finally, they move forwards again together until the end of the respective curves. The cost of this pair of walks is the Euclidean length of $\overline{a_3b_3}$ multiplied by 1.

This paper is organized as follows. In Section 2, we discuss preliminaries and define the problem formally. In Section 3, we propose a polynomial time algorithm to solve the problem exactly. Then, in Section 4, we design an algorithm with improved time and space complexity. At the end, we conclude the paper.

2 Preliminaries and Problem Definition

In this section, first, preliminary concepts are discussed. Then, the WMBFD problem is defined formally. A geometric path in $\mathbb{R}^2$ is a sequence of points in the Euclidean space, $\mathbb{R}^2$. A discrete geometric path, or a polygonal curve, is a geometric path, sampled by a finite sequence of points (i.e., vertices), which are connected by line segments (i.e., edges) in order. Let $T_1 : [0, n] \to \mathbb{R}^2$ and $T_2 : [0, m] \to \mathbb{R}^2$ be two polygonal curves of complexity (number of segments) n and m, respectively. W.l.o.g., assume that $m \leq n$. A vertex of T_1 (resp. T_2) is denoted by $T_1(i)$ (resp. $T_2(j)$), $i = 0, \ldots, n$ (resp. $j = 0, \ldots, m$). An edge of T_1 (resp. T_2) between two vertices $T_1(i-1)$ and $T_1(i)$ (resp. $T_2(j-1)$ and $T_2(j)$) is denoted by e_i (resp. e_j), $i = 1, \ldots, n$ (resp. $j = 1, \ldots, m$). Furthermore, each edge, e_i (resp. e_j), $i = 1, \ldots, n$ (resp. $j = 1, \ldots, m$), of T_1 (resp. T_2) has an associated non-negative weight (or cost) $w_i \in \mathbb{R}$ (resp. $w_j \in \mathbb{R}$). A parameterization of a polygonal curve, $T_1 : [0, n] \to \mathbb{R}^2$, is a continuous function $f : [0, 1] \to [0, n]$, where $f(0) = 0$ and $f(1) = n$ ($[0, 1]$ is a time interval). If f is non-decreasing, then the parameterization is monotone. The weak Fréchet distance, $\delta_w(T_1, T_2)$, is defined as Formula 1, where $d(.,.)$ is the Euclidean distance and f and g are two parameterization of $[0, n]$ and $[0, m]$, respectively. Note that f and g are not necessarily monotone. However, for the standard Fréchet distance, they must be monotone.

$$\delta_w(T_1, T_2) = \inf_{f,g} \max_{t\in[0,1]} d(T_1(f(t)), T_2(g(t))) \qquad (1)$$

Weighted Quality. For a parameterization, f, of a polygonal curve, T_1, let $\mathcal{B}_{f,i} \subseteq [0, 1]$ be the closure of the set of times in which $f(t)$ is decreasing (i.e., the movement is backward), and $f(t) \in [i-1\ i)$ (it is on edge e_i of T_1). $\mathcal{B}_{g,j} \subseteq [0, 1]$ is defined analogously for a parameterization, g, of T_2. For a pair of parameterizations, f and g, of two polygonal curves, T_1 and T_2, we define the weighted quality by Formula 2, where $\|.\|$ is the Euclidean length.

$$\begin{aligned}\mathcal{WQ}_{f,g}(T_1, T_2) := &\sum_{i=1}^{n} ||T_1(f(t))||_{t\in\mathcal{B}_{f,i}} \cdot w_i \\ &+ \sum_{j=1}^{m} ||T_2(g(t))||_{t\in\mathcal{B}_{g,j}} \cdot w_j\end{aligned} \qquad (2)$$

Problem Definition. We formally define the WMBFD problem as follows. For a pair of weighted polygonal curves, T_1 and T_2, and a given leash length, ε, we are looking for a pair of optimal parameterizations, (f, g), as defined in Formula 3. We consider only pairs of parameterizations that guarantee to maintain the leash length at most ε, during the walks.

$$\mathcal{WQ}^{\varepsilon}(T_1, T_2) = \inf_{f,g} \mathcal{WQ}_{f,g}(T_1, T_2) \qquad (3)$$

Weighted Deformed free-space diagram. A useful structure to decide whether the Fréchet distance between two polygonal curves is upper bounded by a given ε, is the free-space diagram [1]. For two polygonal curves, T_1 with n vertices and T_2 with m vertices,

and two corresponding parameterizations, f and g, the *free-space* is defined formally by Formula 4.

$$W = \{(t_1, t_2) \in [0,1]^2 \mid d(T_1(f(t_1)), T_2(g(t_2))) \leq \varepsilon\} \quad (4)$$

The *free-space diagram* is the rectangle $[0,1]\times[0,1]$, partitioned into n columns and m rows. It consists of nm parameter cells $C^{i,j}$, for $i = 1, ..., n$ and $j = 1, ..., m$, whose interiors do not intersect with each other. The cell $C^{i,j}$ represents the multiplication of two subranges of $[0,1]$ that are mapped to the edge between vertices $T_1(i-1)$ and $T_1(i)$ and the edge between vertices $T_2(j-1)$ and $T_2(j)$. For each parameter cell $C^{i,j}$, there exists an ellipse such that the intersection of the area bounded by this ellipse with $C^{i,j}$ is equal to the free-space region of that cell. The boundary of this ellipse and the boundary of the cell, $C^{i,j}$, intersect at most eight times (i.e., at most two intersections per side of $C^{i,j}$). These intersection points form at most four intervals on the boundary of $C^{i,j}$. These intervals could be empty or contains only one point. In addition, two adjacent cells have the same interval on the shared side between the cells. The union of all cells' free-space builds the free-space (or white-space) of the diagram and is denoted by W. The complement of W is the forbidden-space (or black-space) of the diagram and is denoted by B. In this paper, we stretch and compress the columns and rows of the free-space diagram, such that their widths and heights are equal to the lengths of the corresponding segments of T_1 and T_2, respectively. Also, each cell, $C^{i,j}$, has two associated weights, w_x^i and w_y^j. The weight w_x^i is the weight of the edge between vertices $T_1(i-1)$ and $T_1(i)$ and the weight w_y^j is the weight of the edge between vertices $T_2(j-1)$ and $T_2(j)$. The resulting diagram is called the *weighted deformed free-space diagram* and is denoted by $\mathcal{F}$. The bottom left corner of $\mathcal{F}$ represents the starting points of T_1 and T_2 and is denoted by s. The top right corner of $\mathcal{F}$ represents the ending points of T_1 and T_2 and is denoted by t. For the given polygonal curves and ε in Figure 1, the corresponding weighted deformed free-space diagram is shown in Figure 2. As the diagram illustrates, to be able to walk on T_1 and T_2 with a leash length at most ε, there must be a backward movement on the polygonal curves (since there is no xy-monotone path from s to t in W). However, the possible walks are not unique. We are looking for a pair of walks that has the minimum backward movement cost, as we discussed in Section 1. In Figure 2, the red solid polygonal chain, called Π', is a path in W that realizes an optimal pair of walks on T_1 and T_2.

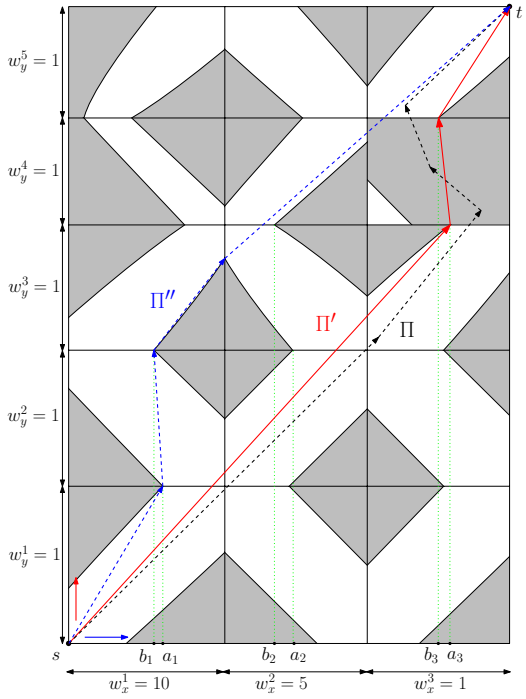


Figure 2: The corresponding weighted deformed free-space diagram of the given polygonal curves in Figure 1 is drawn. Π (the black dashed path) is an arbitrary path in W. $\Pi' \subset \mathcal{G}_w$ (the red solid path) is a path in W that realizes a pair of parameterizations which gives an optimal solution for WMBFD. Π'' (the blue dashed path) is a path in W that realizes the optimal solution for MBFD.

3 Algorithm

In this section, we propose a polynomial time algorithm, by transforming the WMBFD problem to a shortest path problem on a weighted directed graph, $\mathcal{G}_w = \langle V, E\rangle$, defined as follows.

Let $\mathcal{F}$ be the weighted deformed free-space diagram and W (resp. B) be the corresponding free-space (resp. forbidden-space) of $\mathcal{F}$. The vertices of W are the end points of the intervals on the boundary of the cells in $\mathcal{F}$ (i.e., at most 4 intervals per cell). The set of vertices, V, of $\mathcal{G}_w$, is the set of all vertices of W. Each vertex, v, has a x-coordinate (resp. y-coordinate), denoted by v_x (resp. v_y). Also, V contains s and t. We say two points are visible if it is possible to link them by a line segment in W. Every two visible vertices, v_1 and v_2, are linked by two directed edges in E, from v_1 to v_2, $\langle v_1, v_2\rangle$, and vice versa, $\langle v_2, v_1\rangle$. The weight of a directed edge $e = \langle v_1, v_2\rangle \in E$ is a function of its direction, the x- and y-coordinates of v_1 and v_2, and the associated weights of the cells that e intersects. The weight function is defined as follows: suppose e intersects a sequence of k cells, $\langle C^{\sigma(1),\sigma'(1)}, C^{\sigma(2),\sigma'(2)}, \ldots, C^{\sigma(k),\sigma'(k)}\rangle$, of $\mathcal{F}$, where σ (resp. σ') is a function that maps the set $\{1, 2, \ldots, k\}$ to a sub-sequence (or a reversed sub-sequence) of the index sequence $\langle 1, 2, \ldots, n\rangle$ (resp. $\langle 1, 2, \ldots, m\rangle$). The line segment e enters a cell, $C^{\sigma(i),\sigma'(i)}$, $i = 1, \ldots, k$, at point $a^{\sigma(i),\sigma'(i)}$ and exits that cell at point $b^{\sigma(i),\sigma'(i)}$.

Note that $a^{\sigma(1),\sigma'(1)}$ (resp. $b^{\sigma(k),\sigma'(k)}$) is identical to v_1 (resp. v_2). The x-coordinate (resp. y-coordinate) of a point, a, is denoted by a_x (resp. a_y). Note that each cell, $C^{\sigma(i),\sigma'(i)}$, has two associated weights, $w_x^{\sigma(i)}$ and $w_y^{\sigma'(i)}$.
Let $|e|_{w_x} = \sum_{i=1}^{k} |a_x^{\sigma(i),\sigma'(i)} - b_x^{\sigma(i),\sigma'(i)}| \cdot w_x^{\sigma(i)}$ and $|e|_{w_y} = \sum_{i=1}^{k} |a_y^{\sigma(i),\sigma'(i)} - b_y^{\sigma(i),\sigma'(i)}| \cdot w_y^{\sigma'(i)}$. The weight of e, $|e|_w$, is calculated by the following function.

- If e is xy-increasing (i.e., it is non-decreasing from v_1 to v_2 in both x and y axes), then $|e|_w = 0$.
- If e is only x-increasing (resp. y-increasing), then $|e|_w = |e|_{w_y}$ (resp. $|e|_w = |e|_{w_x}$).
- Otherwise, $|e|_w = |e|_{w_x} + |e|_{w_y}$.

Finding an Optimal Solution. By construction of $\mathcal{G}_w$, both s and t are vertices in V. If either s or t is not in W, or there is no path from s to t in $\mathcal{G}_w$, then there is no solution for the given leash length. Otherwise, we prove that a shortest path from s to t, in $\mathcal{G}_w$, gives an optimal solution. Note that a vertex of the graph also corresponds to a point in $\mathcal{F}$. Therefore, the geometric embedding of a path in $\mathcal{G}_w$ is constructed by connecting the consecutive vertices of the path by line segments.

Observation 1 *Let $\Pi : [0,1] \to [0,n] \times [0,m]$ be a path in the free-space W, from s to t. Π is equivalent to a pair of parameterizations, $f : [0,1] \to [0:n]$ and $g : [0,1] \to [0:m]$, of the two polygonal curves, that maintains the leash length at most ε, for all $t \in [0,1]$.*

In this paper, we use norms in two spaces: (1) the Euclidean space of the input polygonal curves, called the input space, (2) the weighted deformed free-space diagram, called the configuration space. In the input space, we use the Euclidean length of a polygonal curve T and denote it by $||T||$. In the configuration space, a path from s to t in W, is denoted by its vertices, $\Pi : \langle s = p_1, p_2, \ldots, p_k = t \rangle$. All segments in Π are directed, $\overrightarrow{p_i p_{i+1}}$, $i = 1, \ldots, k-1$. The weighted length (or simply length), $|.|_w$, of each segment of Π is calculated by the weight function that we explained earlier in this section, for computing the weight of a directed edge in the graph. The weighted length (or simply length) of a path, $|\Pi|_w$, is the sum of the length of its segments. In addition, the notation Π_i is used to denote the sub-path of Π from p_1 to p_i.
Correctness. Lemma 3 is at the heart of the correctness proof. In order to prove that lemma, we need Lemmas 1 and 2. Their proofs are provided in the Appendix. This section is concluded by a corollary to Lemma 3 and Observation 1, that is, in order to find an optimal pair of parameterizations in our problem setting, it suffices to find a shortest path from s to t in $\mathcal{G}_w$.

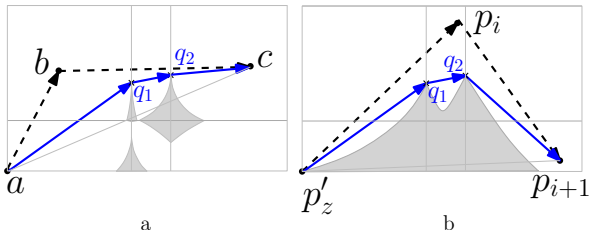


Figure 3: a) The visibility chain from a to c (the blue solid polygonal chain), $CC_a^c = \langle a, q_1, q_2, c \rangle$. b) The visibility chain from p'_z to p_{i+1}, $CC_{p'_z}^{p_{i+1}}$ (see Algorithm 1).

Definition 1 *A path $\Pi \in W$ is x-monotone (resp. y-monotone), if and only if, any vertical (resp. horizontal) line intersects it at most once. Π is xy-monotone, if and only if, it is both x- and y-monotone.*

Observation 2 *Let a and b be two points in W such that $a_x \neq b_x$ and $a_y \neq b_y$. Suppose Π is a xy-monotone path from a to b. In addition, let $R(a,b)$ be the axes-aligned rectangle uniquely determined by a and b as corners. Π lies inside $R(a,b)$.*

Lemma 1 *Let Π_1 and Π_2 be two xy-monotone paths in W, from a to b, where $a, b \in W$. Then, $|\Pi_1|_w = |\Pi_2|_w$. Furthermore, if Π_3 is an arbitrary path in W from a to b, then $|\Pi_1|_w \leq |\Pi_3|_w$.*

Definition 2 *Let a, b, and c be three distinct non-collinear points in W such that $\overrightarrow{ab} \in W$, $\overrightarrow{bc} \in W$ and $\overrightarrow{ac} \notin W$. We define the visibility chain from a to c, denoted by CC_a^c, as follows (see Figure 3a). Let B_{abc} denote the portion of B (the black-space) inside the triangle $\triangle abc$. Let CH be the convex hull of B_{abc} and the points a and c. Then, CC_a^c is defined to be the chain comprising the boundary of CH from a to c that lies inside $\triangle abc$. The visibility chain is directed from a to c, $CC_a^c = \langle a, q_1, \ldots, q_{last}, c \rangle$.*

Lemma 2 *Let $a, b, c \in W$ be three distinct non-collinear points that $\overrightarrow{ab}, \overrightarrow{bc} \in W$ and $\overrightarrow{ac} \notin W$. If $\triangle abc$ lies in $R(a,c)$, then CC_a^c is xy-monotone and $|CC_a^c|_w = |\overrightarrow{ab}|_w + |\overrightarrow{bc}|_w$ (Figure 3a).*

Lemma 3 *For any path $\Pi : \langle s = p_1, p_2, \ldots, p_{k_1} = t \rangle$ in W, there is a path $\Pi' : \langle s = p'_1, p'_2, \ldots, p'_{k_2} = t \rangle$ in W such that $\Pi' \subset \mathcal{G}_w$ and $|\Pi'|_w \leq |\Pi|_w$.*

Proof. We prove this lemma by designing an algorithm that constructs the path $\Pi' \subset \mathcal{G}_w$, through a transformation of path Π. Initially, Π' contains only $s = p'_1 = p_1$ and $p'_z = s$. In this algorithm, p'_z is the latest vertex appended to the tail of Π'. Π' is constructed as follows. When considering the i-th vertex of Π, p_i, the algorithm tests if $\overrightarrow{p'_z p_{i+1}} \in W$. If so, p_i is skipped and Π' remains unchanged. Otherwise, the visibility chain from p'_z to

p_{i+1} is constructed (Figure 3b) and its vertices from q_1 to q_c are appended to the tail of Π'. The algorithm for constructing Π' is stated in Algorithm 1. The correctness of this algorithm is given in the Appendix, Lemma 5. The output, Π', of this algorithm is a path from s to t, such that $\Pi' \subset \mathcal{G}_w$ and $|\Pi'|_w \leq |\Pi|_w$. □

Algorithm 1 Constructing $\Pi' \in \mathcal{G}_w$

Input: The free-space W, A path $\Pi = \langle p_1, p_2, \ldots, p_{k_1} \rangle$, where $s = p_1$ and $p_{k_1} = t$.
Output: A path $\Pi' = \langle p'_1, p'_2, \ldots, p'_{k_2} \rangle$, where $s = p'_1$ and $p'_{k_2} = t$, such that $\Pi' \subset \mathcal{G}_w$ and $|\Pi'|_w \leq |\Pi|_w$.

$\Pi' := \langle s \rangle$;
$p'_z = s$;
for i=2 to $k_1 - 1$ **do**
  if $\overrightarrow{p'_z p_{i+1}} \notin W$ **then**
    Compute the visibility chain from p'_z to p_{i+1}, $CC^{p_{i+1}}_{p'_z} = \langle p'_z, q_1, \ldots, q_c, p_{i+1} \rangle$, in $\triangle p'_z p_i p_{i+1}$;
    Append q_j, $j = 1, ..., c$, to Π';
    $p'_z = q_c$;
Append t to Π';
return Π';

Corollary 1 *There is a path from s to t in $\mathcal{G}_w$ with minimum weighted length in the free-space W.*

Proof. Assume Π is a path from s to t with minimum weighted length in the free-space W. If Π is not a subset of $\mathcal{G}_w$, then, by Lemma 3 there is a path from s to t, Π' in W such that $\Pi' \subset \mathcal{G}_w$ and $|\Pi'|_w \leq |\Pi|_w$. Since Π has minimum weighted length, $|\Pi'|_w = |\Pi|_w$. □

Theorem 1 *Let T_1 and T_2 be two polygonal curves and ε be a given leash length. Each segment of T_1 and T_2 has an associated weight, corresponding to the backward movement on that segment. A pair of parameterizations of T_1 and T_2 that minimizes the weighted sum of the backward movements during the walks can be found in polynomial time and space.*

Proof. It follows from Observation 1 and Corollary 1 that a shortest path in $\mathcal{G}_w$ yields an optimal pair of parameterizations for the WMBFD problem. Since $\mathcal{F}$ has a complexity of $\mathcal{O}(n^2)$, the number of edges of $\mathcal{G}_w$ is $\mathcal{O}(n^4)$. We construct the topology of the graph in $\mathcal{O}(n^4)$ time by the method in [5]. Since the weight of each edge of $\mathcal{G}_w$ is computed based on the projections onto x and y axes, it is possible to compute it in constant time using prefix sums [7]. We find a shortest path in the graph in $\mathcal{O}(n^4)$ time by Dijkstra's algorithm. Therefore, the total time complexity is $\mathcal{O}(n^4)$. □

4 Improved Algorithm

In Section 3, we showed that the weighted graph $\mathcal{G}_w = \langle V, E \rangle$ contains a path that yields an optimal pair of parameterizations for the WMBFD problem. In this section, we will discuss that it is sufficient to compute only a subgraph of $\mathcal{G}_w$ to obtain an optimal solution. Let $\mathcal{G}_w{}' = \langle V, E' \rangle$ be a sub-graph of $\mathcal{G}_w$ such that $E' = \{e' \in E \mid e' \text{ lies completely within a row or within a column of } \mathcal{F}\}$, where $\mathcal{F}$ is the weighted free-space diagram.

Lemma 4 *There is a path in $\mathcal{G}_w{}'$ that realizes an optimal pair of parameterizations for our problem setting.*

Proof. We will show that, for any directed edge $e = \langle u_1, u_2 \rangle \in E$ that is not in E', we can construct a xy-monotone path from u_1 to u_2, π_{u_1,u_2}, in $\mathcal{G}_w{}'$ (see Figure 4). Then, by Lemma 1, $|\pi_{u_1,u_2}|_w = |e|_w$. By Theorem 1, a shortest path, Π' in $\mathcal{G}_w$ yields an optimal solution. Therefore, if for any directed edge, $e = \langle u_1, u_2 \rangle$, of Π', π_{u_1,u_2} exists in $\mathcal{G}_w{}'$, then there is a path in $\mathcal{G}_w{}'$ that realizes an optimal pair of parameterizations.

Now, we prove that π_{u_1,u_2} exists in $\mathcal{G}_w{}'$, for any directed edge $e = \langle u_1, u_2 \rangle \in E$. If e stays completely within a row or a column of $\mathcal{F}$, then $\pi_{u_1,u_2} = e$. Otherwise, e crosses several rows and columns. There are four cases, depending on the orientation of e: a) xy-increasing b) x-increasing and y-decreasing c) y-increasing and x-decreasing d) xy-decreasing. We prove this lemma for the last case. The proofs for the other 3 cases are analogous. Assume that e is xy-decreasing. The edge e intersects a sequence of intervals on the boundary of the cells of $\mathcal{F}$. We partition e into sub-edges so that each sub-edge is contained within a row or within a column of $\mathcal{F}$, as follows (Figure 4).

We traverse e from u_1 to u_2. The point $p_1 \in e$ is the point where we exit the row and the column that contain u_1. Therefore, any point after p_1 on e during the traversal is not in the row or the column that contains u_1. We continue the traversal from p_1 to u_2. The point $p_2 \in e$ is defined analogously. It is the point where we exit the row and the column that contain p_1. We define p_i, $i = 3, \ldots, z$, analogously with respect to p_{i-1}. Then, the sequence of sub-edges of e is $\langle \overrightarrow{u_1 p_1}, \overrightarrow{p_1 p_2}, \ldots, \overrightarrow{p_z u_2} \rangle$.

Denote the interval that contains p_i, $i = 1, \ldots, z$, by I_i. Let $u_1 = p_0 \in I_0$ and $u_2 = p_{z+1} \in I_{z+1}$. Note that I_i and I_{i+1}, $i = 0, \ldots, z$, are on the boundary of a row or a column. We say a point $q = (q_x, q_y)$ dominates a point $p = (p_x, p_y)$, if $p_x \leq q_x$ and $p_y \leq q_y$. In this proof, the endpoint of I_i that dominates p_i is denoted by q_i. We construct π_{u_1,u_2}, in two phases. In the first phase, we construct a xy-monotone path, π'_{u_1,u_2}, from u_1 to u_2 (the green dashed polygonal chain in Figure 4). Then, in the second phase, we transform it to a path, π_{u_1,u_2}, in $\mathcal{G}_w{}'$ (the blue dotted polygonal chain in Figure 4).

In the first phase, we start from $p_0 = u_1$. For each sub-edge of e, $\overrightarrow{p_i p_{i+1}}$, $i = 0, \ldots, z$, if we construct a xy-monotone path, $\pi'_{p_i,q_{i+1}}$, from p_i to q_{i+1}, then the concatenation of $\pi'_{p_i,q_{i+1}}$ and $\overrightarrow{q_{i+1} p_{i+1}}$ is a xy-monotone path from p_i to p_{i+1}, $\pi'_{p_i,p_{i+1}}$, because q_{i+1} dominates

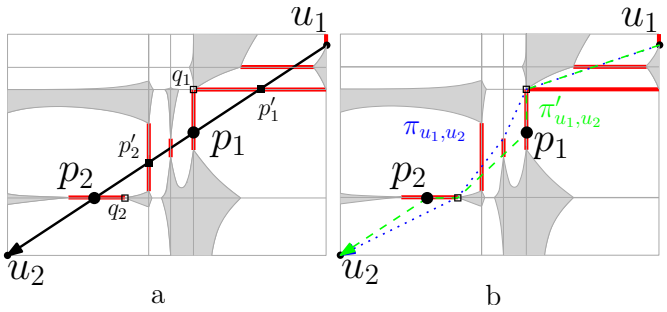


Figure 4: Illustration of proof of Lemma 4.

p_{i+1}. Then, the concatenation of $\pi'_{p_i,p_{i+1}}$, $i = 0, \ldots, z$ is a xy-monotone path, π'_{u_1,u_2}, from u_1 to u_2. Now, we explain how to construct $\pi'_{p_i,q_{i+1}}$. If $\overrightarrow{p_iq_{i+1}} \in W$, then $\pi'_{p_i,q_{i+1}} = \overrightarrow{p_iq_{i+1}}$. It is obviously xy-monotone. If $\overrightarrow{p_iq_{i+1}} \notin W$, then $\pi'_{p_i,q_{i+1}}$ is the visibility chain, $CC_{p_i}^{q_{i+1}}$, from p_i to q_{i+1}, in $\triangle p_ip'_{i+1}q_{i+1}$, where p'_{i+1} is a point, defined as follows. Let I'_{i+1} be the last interval that $\overrightarrow{p_ip_{i+1}}$ intersects before intersecting I_{i+1} and p'_{i+1} be the intersection point of I'_{i+1} and $\overrightarrow{p_ip_{i+1}}$. The point p'_{i+1} dominates q_{i+1}. Therefore, $\triangle p_ip'_{i+1}q_{i+1}$ lies in $R(p_i, q_{i+1})$, the axes-aligned rectangle that is determined by p_i and q_{i+1} as opposite corners. Thus, by Lemma 2, $CC_{p_i}^{q_{i+1}}$ is xy-monotone.

In the second phase, we transform π'_{u_1,u_2} to a xy-monotone path, π_{u_1,u_2}, in $\mathcal{G}_w{}'$. This transformation is done by replacing the edges in π'_{u_1,u_2} that are not in E'. These edges are $\overrightarrow{q_ip_i}$ and $\overrightarrow{p_i\mathcal{S}(p_i)}$, $i = 1, \ldots, z$, where $\mathcal{S}(.)$ is the successor operation and $\mathcal{S}(p_i)$ is the vertex after p_i in π'_{u_1,u_2}. These are the edges that connect p_i, $i = 1, \ldots, z$, to the previous and next vertex of p_i in π'_{u_1,u_2}. Note that $\mathcal{S}(p_i)$ is a vertex in V and could be identical to q_{i+1}. If $\overrightarrow{q_i\mathcal{S}(p_i)} \in W$, then the two edges, $\overrightarrow{q_ip_i}$ and $\overrightarrow{p_i\mathcal{S}(p_i)}$, are replaced by $\overrightarrow{q_i\mathcal{S}(p_i)} \in E'$. It is obviously xy-monotone. If $\overrightarrow{q_i\mathcal{S}(p_i)} \notin W$, then the two edges, $\overrightarrow{q_ip_i}$ and $\overrightarrow{p_i\mathcal{S}(p_i)}$, are replaced by the visibility chain, $CC_{q_i}^{\mathcal{S}(p_i)}$, from q_i to $\mathcal{S}(p_i)$, in $\triangle q_ip_i\mathcal{S}(p_i)$. Since π'_{u_1,u_2} is a xy-monotone path, the concatenation of $\overrightarrow{q_ip_i}$ and $\overrightarrow{p_i\mathcal{S}(p_i)}$ is also a xy-monotone path. Therefore, $\triangle q_ip_i\mathcal{S}(p_i)$ lies in $R(q_i, \mathcal{S}(p_i))$. Thus, by Lemma 2, $CC_{q_i}^{\mathcal{S}(p_i)}$ is xy-monotone. Also, $CC_{q_i}^{\mathcal{S}(p_i)} \subset \mathcal{G}_w{}'$ since the vertices of this visibility chain belong to one column or one row of $\mathcal{F}$. By repeating this process for every p_i, $i = 1, \ldots, z$, the resulting path, denoted by π_{u_1,u_2}, is in $\mathcal{G}_w{}'$. Since all sub-paths of π_{u_1,u_2} are xy-monotone, π_{u_1,u_2} is also xy-monotone. □

Theorem 2 *Assume we are given two polygonal curves, T_1 and T_2, and a leash length, ε. Each segment of T_1 and T_2 has an associated weight, corresponding to the backward movement on that segment. A pair of parameterizations of T_1 and T_2 that minimizes the weighted sum of the backward movements during the walks can be found in $\mathcal{O}(n^3)$ time and space, where n is the number of segments in the input polygonal curves.*

Proof. The correctness follows from Lemma 4. $\mathcal{F}$ has $\mathcal{O}(n^2)$ cells and each vertex of $\mathcal{G}_w{}'$ on the boundary of a cell is connected to at most $\mathcal{O}(n)$ vertices of $\mathcal{G}_w{}'$ that are in the same row or column. Therefore, the number of edges of $\mathcal{G}_w{}'$ is $\mathcal{O}(n^3)$. It is possible to find all the edges of $\mathcal{G}_w{}'$ that lie in a column or row of $\mathcal{F}$ in $\mathcal{O}(n^2)$ time by the method proposed in [5]. In addition, to compute the weight of the edges that are in one row or column, $\mathcal{O}(n^2)$ time and $\mathcal{O}(n)$ space suffice (by using prefix sums, see [7]). Using Dijkstra's algorithm, we find a shortest path in $\mathcal{G}_w{}'$ in $\mathcal{O}(n^3)$ time. Therefore, both time and space complexities of our algorithm are $\mathcal{O}(n^3)$. Note that if the representing nodes for s and t in $\mathcal{G}_w{}'$ are not in a connected component of $\mathcal{G}_w{}'$, then there is no feasible walk with the leash length of ε. □

5 Conclusion

In this paper, we generalized the MBFD problem by capturing weighted scenarios. We established that this problem setting is dual to a weighted shortest path problem in a weighted deformed free-space diagram, $\mathcal{F}$. We proposed an exact algorithm to solve the problem in $\mathcal{O}(n^3)$ time and space. We are currently working on improving the time complexity.

References

[1] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geometry Appl.*, 5:75–91, 1995.

[2] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk. On map-matching vehicle tracking data. *31st VLDB*, pp. 853–864, 2005.

[3] T. Flynn, S. Connery, M. Smutok, R. Zeballos, and I. Weisman. Comparison of cardiopulmonary responses to forward and backward walking and running. *Med. Sci. Sports Exerc.*, 26(1):89–94, 1994.

[4] A. Gheibi, A. Maheshwari, J.-R. Sack, and C. Scheffer. Minimum Backward Fréchet Distance. *22nd ACM SIGSPATIAL*, pp. 381–388, 2014.

[5] S. K. Ghosh and D. M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM J. Comput.*, 20(5):888–910, 1991.

[6] S. Har-Peled and B. Raichel. The Fréchet Distance Revisited and Extended. *27th ACM SoCG*, pp. 448–457, 2011.

[7] G. E. Blelloch. Prefix Sums and Their Applications. *Synthesis of Parallel Algorithms*, Morgan Kaufmann, 1990.

6 Appendix

Lemma 1 *Let Π_1 and Π_2 be two xy-monotone paths in W, from a to b, where $a, b \in W$. Then, $|\Pi_1|_w = |\Pi_2|_w$. Furthermore, if Π_3 is an arbitrary path in W from a to b, then $|\Pi_1|_w \leq |\Pi_3|_w$.*

Proof. If $a_x = b_x$ or $a_y = b_y$, then Π_1 and Π_2 are identical and the proof is trivial. Otherwise, by Observation 2, Π_1 and Π_2 lie in $R(a, b)$. Since Π_1 (also Π_2) is xy-monotone, its orthogonal projections onto x and y axes are not overlapping and equal to the width and height of $R(a, b)$, respectively. Because Π_1 and Π_2 have identical projections onto x and y axes and the weighted length of a path is defined based on its projection, then $|\Pi_1|_w = |\Pi_2|_w$. Also, any xy-monotone path from a to b has minimum weighted length among all paths from a to b, because its orthogonal projections onto x- and y-axis are non-overlapping. □

Lemma 2 *Let $a, b, c \in W$ be three distinct non-collinear points that $\overrightarrow{ab}, \overrightarrow{bc} \in W$ and $\overrightarrow{ac} \notin W$. If $\triangle abc$ lies in $R(a, c)$, then CC_a^c is xy-monotone and $|CC_a^c|_w = |\overrightarrow{ab}|_w + |\overrightarrow{bc}|_w$ (Figure 3a).*

Proof. Since $\triangle abc$ lies in $R(a, c)$, the path that consists of $\overrightarrow{ab}$ and $\overrightarrow{bc}$ is a xy-monotone path from a to c. If we show that CC_a^c is also a xy-monotone path from a to c, then by Lemma 1, $|CC_a^c|_w = |\overrightarrow{ab}|_w + |\overrightarrow{bc}|_w$.

To prove this, we need to define the angle of a vector. Suppose a directed segment in the free-space is a vector from the origin of the Cartesian coordinate system. The angle of a vector is defined as the angle between that vector and the positive direction of x-axis. Let α (resp. β) be the angle of $\overrightarrow{ab}$ (resp. $\overrightarrow{bc}$). Since $\triangle abc$ lies in $R(a, c)$, $|\alpha - \beta| = \pi/2$. In addition, since $R(a, c)$ is axes-aligned, precisely one of the four following cases is true: $0 \leq \alpha, \beta \leq \pi/2$, $\pi/2 \leq \alpha, \beta \leq \pi$, $\pi \leq \alpha, \beta \leq 3\pi/2$, $3\pi/2 \leq \alpha, \beta \leq 2\pi$. We denote the angles of segments, $\overrightarrow{aq_1}, \overrightarrow{q_1q_2}, \ldots, \overrightarrow{q_{last}c}$, of $CC_a^c = \langle a, q_1, \ldots, q_{last}, c\rangle$ by θ_μ, $\mu = 1, \ldots, last + 1$. Since CC_a^c is a convex chain, the sequence of θ_μ, $\mu = 1, \ldots, last + 1$, is in a sorted order (either increasing or decreasing), between α and β. Therefore, all θ_μ, $\mu = 1, \ldots, last + 1$, are in one of the four mentioned quadrants. Thus CC_a^c is xy-monotone. This proves the lemma. □

Lemma 5 *Algorithm 1 is correct.*

Proof. We show that prior to the execution of the i-th iteration, $i = 2, \ldots, k_1 - 1$, of the **for**-loop in Algorithm 1 the following invariant holds:

I1. $\overrightarrow{p'_z p_i} \in W$

I2. $\Pi'_z \subset \mathcal{G}_w$ and

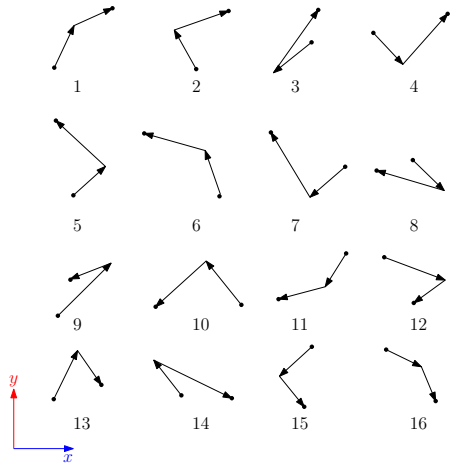


Figure 5: There are 16 cases for the combination of two directed segments.

I3. $|\Pi'_z|_w + |\overrightarrow{p'_z p_i}|_w \leq |\Pi_i|_w$.

We prove this by induction on i, the index of the vertices of Π (and index of the **for**-loop in Algorithm 1). The base case is $i = 2$. In this case, p'_z is equal to $p'_1 = s$. Clearly, $\overrightarrow{p'_1 p_2} \in W$, $\Pi'_1 \subset \mathcal{G}_w$ and $|\Pi'_1|_w + |\overrightarrow{p'_1 p_2}|_w = |\Pi_2|_w$, because $\overrightarrow{p'_1 p_2} = \overrightarrow{p_1 p_2}$. The induction hypothesis is that the invariant holds for all loop iterations before the i-th iteration of the **for**-loop. In the following, it is proved that it also holds before $(i + 1)$-th iteration of the **for**-loop.

In each iteration of the **for**-loop in Algorithm 1, we distinguish between the two cases: *a)* $\overrightarrow{p'_z p_{i+1}} \in W$, *b)* $\overrightarrow{p'_z p_{i+1}} \notin W$.

Case a) In Case *a*, p_i is skipped and Π' thus remains unchanged. Therefore, I1 and I2 hold, due to the induction hypothesis. In addition, since $\overrightarrow{p'_z p_{i+1}}$ is a segment in W and thus trivially xy-monotone, by Lemma 1, $|\overrightarrow{p'_z p_{i+1}}|_w \leq |\overrightarrow{p'_z p_i}|_w + |\overrightarrow{p_i p_{i+1}}|_w$. By induction hypothesis, we have $|\Pi'_z|_w + |\overrightarrow{p'_z p_i}|_w \leq |\Pi_i|_w$. By adding $|\overrightarrow{p_i p_{i+1}}|_w$ to the both sides of the inequality, we obtain $|\Pi'_z|_w + |\overrightarrow{p'_z p_{i+1}}|_w \leq |\Pi_i|_w + |\overrightarrow{p_i p_{i+1}}|_w = |\Pi_{i+1}|_w$. Therefore, I3 remains true after i-th iteration (i.e., before $(i + 1)$-th iteration).

Case b) In Case *b*, the **then** part of the **if** statement of the algorithm is entered and the visibility chain from p'_z to p_{i+1} is constructed. It is denoted by $CC_{p'_z}^{p_{i+1}}$: $\langle p'_z, q_1, \ldots, q_c, p_{i+1}\rangle \in W$, where $q_j \in V$, $j = 1, \ldots, c$. Then, the q_j, from $j = 1$ to $j = c$, is appended to the tail of Π'. Finally, p'_z is updated to q_c. In the remaining, it is proved that the invariant holds.

Since $CC_{p'_z}^{p_{i+1}}$ is the visibility chain, it is easy to see that all q_i, $i = 1, \ldots, last$, are represented by a node in the graph, $\mathcal{G}_w$, because they are vertices of W. Therefore, I1 and I2 hold. In order to check if I3 holds, we

need to analyze different cases. Each directed segment in W is of one of the following types: 1. xy-increasing 2. x-increasing and y-decreasing 3. y-increasing and x-decreasing 4. xy-decreasing. Therefore, there are 16 cases for the combination of two segments, $\overrightarrow{p'_z p_i}$ and $\overrightarrow{p_i p_{i+1}}$ (Figure 5). In all 16 cases, the orthogonal projection of $CC^{p_{i+1}}_{p'_z}$ onto the x-axis (resp. y-axis) is not longer than the sum of the orthogonal projections of $\overrightarrow{p'_z p_i}$ and $\overrightarrow{p_i p_{i+1}}$ onto the x-axis (resp. y-axis). Therefore, $|CC^{p_{i+1}}_{p'_z}|_w \leq |\overrightarrow{p'_z p_i}|_w + |\overrightarrow{p_i p_{i+1}}|_w$. Here we only show the proofs for two cases of Figure 5 as the proofs for the other cases are analogous.

Consider the case when both $\overrightarrow{p'_z p_i}$ and $\overrightarrow{p_i p_{i+1}}$ are y-increasing and x-decreasing (see Case 6 in Figure 5). In this case, since $\triangle p'_z p_i p_{i+1}$ lies in $R(p'_z, p_{i+1})$, by Lemma 2, $|CC^{p_{i+1}}_{p'_z}|_w = |\overrightarrow{p'_z p_i}|_w + |\overrightarrow{p_i p_{i+1}}|_w$. By inductive hypothesis we have $|\Pi'_z|_w + |\overrightarrow{p'_z p_i}|_w \leq |\Pi_i|_w$. Add now $|\overrightarrow{p_i p_{i+1}}|_w$ to both sides of the inequality. We obtain $|\Pi'_z|_w + |CC^{p_{i+1}}_{p'_z}|_w \leq |\Pi_i|_w + |\overrightarrow{p_i p_{i+1}}|_w$. It follows that $|\Pi'_{z+c}|_w + |\overrightarrow{q_c p_{i+1}}|_w \leq |\Pi_{i+1}|_w$, where q_c is the latest inserted vertex to the tail of Π' and Π'_{z+c} is the sub-path of Π' from index 1 to index $z+c$. Therefore, I3 holds. The proofs for cases 1,11 and 16 are similar.

Now consider Case 9, when $\overrightarrow{p'_z p_i}$ is xy-increasing and $\overrightarrow{p_i p_{i+1}}$ decreases in both x and y axes (illustrated in Figure 6). In this case, $|\overrightarrow{p'_z p_i}|_w + |\overrightarrow{p_i p_{i+1}}|_w = 0 + |\overrightarrow{p_i p_{i+1}}|_w$.

The vertical line that passes through p_{i+1} is denoted by $L^{\perp}_x$. The horizontal line that passes through p_{i+1} is denoted by $L^{\perp}_y$. Suppose these lines are directed toward $+\infty$. The following two properties hold. First, any directed segment of $CC^{p_{i+1}}_{p'_z}$ that lies on the left of $L^{\perp}_x$ is increasing in x. Therefore, they have a weighted length zero in the x-dimension. Second, any directed segment of $CC^{p_{i+1}}_{p'_z}$ that lies below $L^{\perp}_y$ is increasing in y. Therefore, they have a weighted length of zero in the y-dimension. By these two properties, any directed segment of $CC^{p_{i+1}}_{p'_z} : \langle p'_z, q_1, \ldots, q_c, p_{i+1}\rangle$ that lies on the left of $L^{\perp}_x$ and below $L^{\perp}_y$ is xy-increasing and has the weighted length zero.

Suppose $\overrightarrow{q_r q_{r+1}}$ is the first line segment in $CC^{p_{i+1}}_{p'_z}$ on the right side of $L^{\perp}_x$ that is x-decreasing. Since $CC^{p_{i+1}}_{p'_z}$ is a convex chain and is inside the triangle $\triangle p'_z p_i p_{i+1}$, the sub-chain $\langle q_r, \ldots, q_c, p_{i+1}\rangle$ is x-monotone and its weighted length in x-dimension is less than or equal to the weighted length of $\overrightarrow{p_i p_{i+1}}$ in x-dimension. Therefore, the weighted length of $CC^{p_{i+1}}_{p'_z}$ in x-dimension is less than or equal to the weighted length of $\overrightarrow{p_i p_{i+1}}$ in x-dimension.

It is analogous for the y-dimension. Suppose $\overrightarrow{q_u q_{u+1}}$ is the first line segment in $CC^{p_{i+1}}_{p'_z}$ above $L^{\perp}_y$ that is y-decreasing. Since $CC^{p_{i+1}}_{p'_z}$ is a convex chain and is inside the triangle $\triangle p'_z p_i p_{i+1}$, the sub-chain $\langle q_u, \ldots, q_c, p_{i+1}\rangle$ is y-monotone and its weighted length in y-dimension is less than or equal to the weighted length of $\overrightarrow{p_i p_{i+1}}$ in y-dimension. Therefore, the weighted length of $CC^{p_{i+1}}_{p'_z}$ in y-dimension is less than or equal to the weighted length of $\overrightarrow{p_i p_{i+1}}$ in y-dimension.

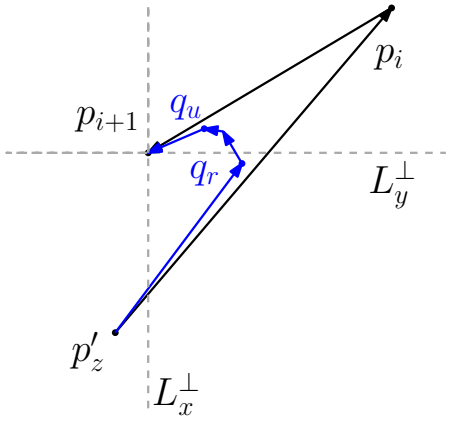


Figure 6: The segment from q_r to q_{r+1} is the first line segment in $CC^{p_{i+1}}_{p'_z}$ on the right side of $L^{\perp}_x$ that is x-decreasing. The segment from q_u to q_{u+1} is the first line segment in $CC^{p_{i+1}}_{p'_z}$ above of $L^{\perp}_y$ that is y-decreasing.

To conclude, the weighted length of $CC^{p_{i+1}}_{p'_z}$, which is the sum of the weighted length of $CC^{p_{i+1}}_{p'_z}$ in x- and y-dimensions, is less than or equal to the weighted length of $\overrightarrow{p_i p_{i+1}}$, which is the sum of the weighted length of $\overrightarrow{p_i p_{i+1}}$ in x- and y-dimensions. Thus, $|CC^{p_{i+1}}_{p'_z}|_w \leq |\overrightarrow{p_i p_{i+1}}|_w = |\overrightarrow{p'_z p_i}|_w + |\overrightarrow{p_i p_{i+1}}|_w$. By inductive hypothesis, $|\Pi'_z|_w + |\overrightarrow{p'_z p_i}|_w \leq |\Pi_i|_w$. By adding $|\overrightarrow{p_i p_{i+1}}|_w$ to the both sides of the inequality, we conclude $|\Pi'_z|_w + |CC^{p_{i+1}}_{p'_z}|_w \leq |\Pi'_z|_w + |\overrightarrow{p'_z p_i}|_w + |\overrightarrow{p_i p_{i+1}}|_w \leq |\Pi_i|_w + |\overrightarrow{p_i p_{i+1}}|_w$. It follows that $|\Pi'_{z+c}|_w + |\overrightarrow{q_c p_{i+1}}|_w \leq |\Pi_{i+1}|_w$. Therefore, I3 also holds for this case. The proofs for the other remaining cases are similar. □

Squeeze-Free Hamiltonian Paths in Grid Graphs

Alexandru Damian* Robin Flatland†

Abstract

Motivated by multi-robot construction systems, we introduce the problem of finding squeeze-free Hamiltonian paths in grid graphs. A Hamiltonian path is *squeeze-free* if it does not pass between two previously visited vertices lying on opposite sides. We determine necessary and sufficient conditions for the existence of squeeze-free Hamiltonian paths in staircase grid graphs. Our proofs are constructive and lead to linear time algorithms for determining such paths, provided that they exist.

1 Introduction

We introduce a problem motivated by collective construction systems in which a large number of simple, autonomous robots build complex structures using modular building blocks. Such systems are inspired by the decentralized construction methods of termites and bees in which global structure emerges from the efforts of individual insects following seemingly simple rules and using environmental cues. They are robust to failure because damaged robots are easily replaced, making them suitable in uncertain and inhospitable environments.

In the TERMES collective construction system introduced in [6], the modular building blocks are cubes that are placed on a regular grid to form lattice-based structures. Robots move from cell to cell on the grid while carrying a block, which they can attach to the structure at an adjacent cell. Physical limitations restrict the class of structures that the robots can build and the order in which they can attach the blocks. For example, it is impossible for a robot to carry blocks down a corridor one block wide, or to place a block directly between two others. Inappropriate intermediate configurations that can no longer be traversed by the robots should therefore be avoided by proper robot coordination.

Coordination in the TERMES system is achieved by precomputing a path that all robots follow while adding blocks to the structure. The path starts at a grid cell on the boundary of the final structure, visits each grid cell of the final structure exactly once, and satisfies the restriction that the path may not "squeeze" into a cell that has two previously visited cells adjacent to it on opposite sides. We call such a path a *squeeze-free Hamiltonian path*. The problem thus reduces to finding a squeeze-free Hamiltonian path in a grid graph. See Figure 1.

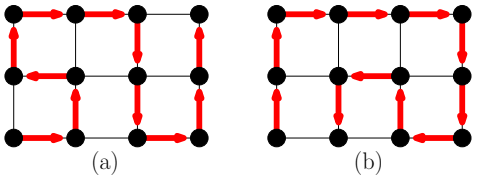


Figure 1: (a) A squeeze-free path, and (b) a path in which the last three vertices are squeezed between two previously visited vertices on opposite sides.

To our knowledge we are the first to study the algorithmic complexity of the squeeze-free Hamiltonian path problem. In [6], their main focus is on engineering the robots rather than algorithms, so they use an exponential time backtracking algorithm to compute the paths. Computing Hamiltonian cycles in general grid graphs is known to be NP-Complete [4], although the problem can be solved in polynomial time for specialized classes of grid graphs [5], [2]. See [1] for a survey and new results on Hamiltonicity of square, triangular, and hexagonal grid graphs. In other related work, [3] presents an $O(n^2)$ algorithm for computing a partial ordering on the placement of blocks subject to the squeeze-free constraint for 2D structures with holes. Here we take a first step to understanding the complexity of the squeeze-free Hamiltonian path problem by providing an $O(t)$ algorithm that determines for any *staircase* grid graph G with t steps, if G has a squeeze-free Hamiltonian path that starts at a boundary vertex located on a step of G.

2 Notation and Definitions

A grid graph is a graph induced by a finite subset of the vertices of a square tiling of the plane. In this paper we consider *staircase* grid graphs which consist of the edges and vertices bounding a set of tiles whose union forms the shape of a staircase extending rightwards and upwards from the bottom leftmost vertex.

For a staircase grid graph G, let ∂G denote the portion of the staircase boundary that extends clockwise from the bottom leftmost vertex to the top rightmost vertex of G. Refer to Figure 2. For any pair of vertices $a, b \in \partial G$, let $\partial G[a, b]$ denote the portion of ∂G extending from a to b. For any vertex x, let x_s, x_w, x_n

*Department of Computing Sciences, Villanova University, Villanova, PA, USA. adamian@villanova.edu

†Department of Computer Science, Siena College, Loudonville, NY, USA. flatland@siena.edu

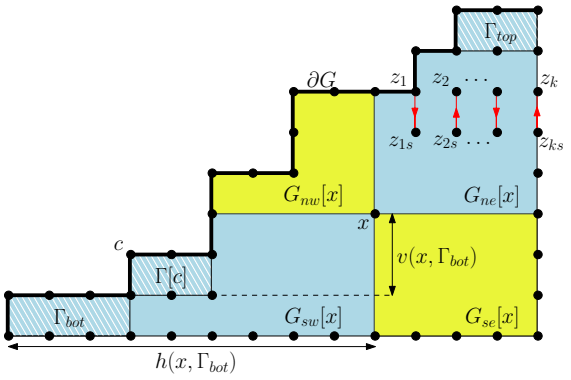


Figure 2: Staircase grid graph. Only some of the interior grid vertices and edges are illustrated.

and x_e refer to the vertices adjacent to x that lie south, west, north and east of x, respectively (if such a vertex exists). We sometimes double the subscript to refer, for instance, to the vertex south of x_e as x_{es}. At each vertex $x \in G$, the coordinate axes with origin x partition the plane into four quadrants. Let $G_{ne}[x]$, $G_{nw}[x]$, $G_{sw}[x]$ and $G_{se}[x]$ denote the subgraph of G that lies entirely in the first, second, third and fourth quadrant, respectively. We assume that each quadrant is closed, so it includes the points on the bounding axes. Define $G_n[x] = G_{ne}[x] \cup G_{nw}[x]$, and similarly for $G_s[x]$, $G_e[x]$ and $G_w[x]$. Let H be a (directed) Hamiltonian path in G. For any two vertices $a, b \in G$, such that a is visited by H before b, let $H[a, b]$ denote the directed subpath of H from a to b. A vertex $v \in H$ is *squeezed* if v_e and v_w are both visited before v, or if v_n and v_s are both visited before v.

A *corner* is a boundary vertex on G with interior angle $\pi/2$ (if convex) or $3\pi/2$ (if reflex). For any convex corner $c \in \partial G$, let $\Gamma[c]$ denote the closed rectangle bounded by the two line segments extending from c to the next and previous corners located clockwise and counterclockwise (respectively) from c. We refer to $\Gamma[c]$ as the *step* with corner c. The height of $\Gamma[c]$ is the height of the corresponding rectangle. The case where G consists of a single step is trivial, so we assume that G has at least two steps. We refer to the highest step of G as Γ_{top} and the lowest step as Γ_{bot}.

For any two vertices $a, b \in G$, let $h(a, b)$ denote the horizontal extent of the line segment ab, and let $v(a, b)$ denote the vertical extent of ab. For any vertex $x \in G$ and any staircase step Γ of G, let $h(x, \Gamma)$ denote the horizontal distance from x to the left side of Γ, and let $v(x, \Gamma)$ denote the vertical distance from x to the top side of Γ. The following definition (depicted in Figure 2 for $j = s$) will play an important role in our discussion.

Definition 1 *Let Z be a sequence of consecutive grid points $z_1, \ldots, z_k$ lying on a horizontal (vertical) grid segment. For a fixed $j \in \{s, n\}$ ($j \in \{e, w\}$), we say that the grid segments $\overline{z_1 z_k}$ and $\overline{z_1 j z_k j}$ form a* zigzag sequence *in H if, for each $i = 1, \ldots, k$, $\overrightarrow{z_i z_{ij}} \in H$ if i is odd, and $\overrightarrow{z_{ij} z_i} \in H$ if i is even.*

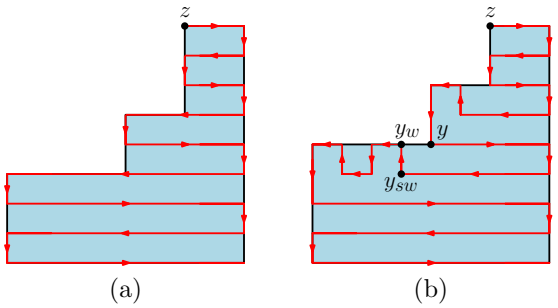


Figure 3: (a) *pattern1* and (b) *pattern2*.

We call a zigzag sequence *separating* if it extends between two boundary edges.

2.1 Hamiltonian Patterns

Let $G' \subseteq G$ be an arbitrary staircase subgraph of G. We define two distinct Hamiltonian path patterns H_1 and H_2 on G', which will later be used in stitching a Hamiltonian path H for G. Each of these patterns is defined for a fixed orientation for the first edge – say east – and grows in a particular direction – say south – with the understanding that the pattern can undergo rotations and reflections as necessary to construct H. Each of H_1 and H_2 begins at a vertex on $\partial G'$. The first pattern H_1, which we refer to as *pattern1*, includes straight horizontal path segments with orientations alternating east and west on each row, and extending between boundary points of G'. See Figure 3a. The second pattern H_2, which we refer to as *pattern2*, is identical to the first pattern, with the only difference that, for each reflex corner y, the straight horizontal subsegment extending west from y_{sw} is replaced by a subpath of unit height that includes the zigzag sequence starting with $\overrightarrow{y_{sw} y_w}$ and extending west. See Figure 3b. Observe that both patterns are squeeze-free.

3 Preliminaries

Here we prove some properties of squeeze-free Hamiltonian paths in G, if it has any. Therefore, throughout this section, assume there exists a squeeze-free Hamiltonian path in G, and H is one such path.

Lemma 1 *At any time during the traversal of H, there can be no unvisited vertices between any two vertices a and b on a grid line that have already been visited.*

Proof. If there were unvisited vertices along the line segment ab, then the vertex last visited among these vertices would cause a squeeze. □

Let p be a directed, simple path in G from a boundary point x to a boundary point y. A vertex $v \in G$ is said to be *left* (*right*) of p if $v \notin p$ and v is on or to the left (right) of the oriented closed curve consisting of p followed by the path counterclockwise (clockwise) along the boundary of G from y to x.

Lemma 2 *Let x be any boundary vertex of G, and let y be a vertex of ∂G such that $y \in G_{ne}[x]$ and H visits x before y. At the time y is visited, all vertices that are left of $H[x, y]$ and also in $G_e[x]$ are visited, with the possible exception of those vertices to the left of y on its horizontal grid line.*

Proof. For contradiction, suppose there is an unvisited vertex z to the left of $H[x, y]$ and in $G_e[x]$ that is not left of y on y's horizontal grid line. E.g., in Figure 4a, z may be any vertex in the shaded regions. Let d be the intersection of an upward ray from x and a leftward ray from y. (Note that d need not be in G, as illustrated in Figure 4a.) To reach z from y, H must eventually either traverse an edge $\overrightarrow{vv_s}$ with v on $\overline{dy}$, or it must traverse an edge $\overrightarrow{vv_e}$ with v_e on $\overline{dx}$. (These edges, which are the only edges taking H into the shaded regions containing z, are depicted in Figure 4a.) In the first case, there must be a previously visited vertex of $H[x, y]$, call it u, located below v_s on its vertical grid line. Thus there is a time during the traversal of H in which v_s is unvisited and between visited vertices v and u in the same vertical grid line, which contradicts Lemma 1. In the second case there is a previously visited vertex of $H[x, y]$ located to the right of v_e on its horizontal grid line, which similarly contradicts Lemma 1. □

Lemma 3 *The end point e of H is one of the following vertices: (i) a top corner of Γ_{top}, (ii) a left corner of Γ_{bot}, or (iii) the lowest rightmost corner of G.*

Proof. Clearly e must be a convex corner, or else it is squeezed between previously visited vertices. Suppose for contradiction that e is a convex corner of a step that is not Γ_{top} or Γ_{bot}. Then the top left vertex x of Γ_{bot} and the top left vertex y of Γ_{top} are both visited before e. Without loss of generality, assume x is visited before y. (If y is visited before x, just rotate the staircase by 90^o and reflect it across the vertical, thus reversing the roles of x and y.) By Lemma 2, e must be visited before y, a contradiction. □

Lemma 4 *Let $x \in G$ be the first vertex visited by H in $G_{ne}[x]$ and let $\overrightarrow{xx_n} \in H$. Then $\overrightarrow{x_{ne}x_e} \in H$ and x_{ne} is the first vertex visited by H in $G_{se}[x_{ne}]$.*

Proof. Because x is the first vertex visited in $G_{ne}[x]$, H traverses $\overrightarrow{xx_n}$ before visiting x_e. Therefore H cannot enter x_e from the east (because x_e would be squeezed between vertices x and x_{ee} which would both be visited before it) or the west (because $\overrightarrow{xx_n} \in H$). It also cannot enter it from the south via $\overrightarrow{x_{se}x_e}$ because H, which begins on the boundary of G in a quadrant other than $G_{ne}[x]$, would have to circle clockwise from x_n around to x_{se}. In doing so, it crosses x_e's vertical grid line L at a vertex above x_e before reaching x_{se} below it, thus leaving x_e unvisited between two visited vertices on L. This contradicts Lemma 1. Therefore, $\overrightarrow{x_{ne}x_e} \in H$.

By Lemma 1, at the time $\overrightarrow{x_{ne}x_e}$ is traversed, no vertex south of x_{ne} on its vertical grid line is visited. Also by Lemma 1, at the time x_{ne} is visited, no vertex east of it on its horizontal grid line is visited (because x_n is already visited). Therefore, x_{ne} is the first vertex visited by H in $G_{se}[x_{ne}]$. □

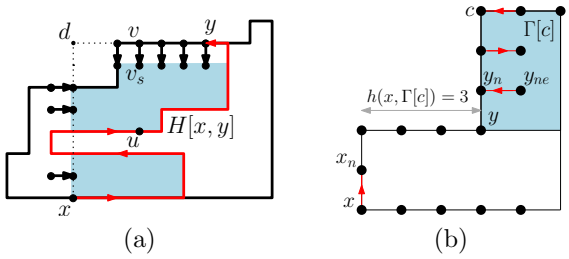


Figure 4: (a) Lemma 2 (b) Lemma 7.

Lemma 5 *Let $i \in \{n, s\}$ and $j \in \{e, w\}$. Let $x \in G$ be the first vertex visited by H in $G_{ij}[x]$ and $\overrightarrow{xx_j} \in H$ ($\overrightarrow{xx_i} \in H$). Then the two parallel grid lines containing x and x_j (x_i) in $G_{ij}[x]$ form a zigzag sequence in H.*

Proof. The case ($i = n$, $j = e$, $\overrightarrow{xx_n} \in H$) follows immediately from Lemma 4, by induction on the number of vertices to the right of x. The other cases are similar by symmetry of rotations and reflections. □

Lemma 6 *If $\overrightarrow{vw}$ is the first edge visited in a zigzag sequence σ, the zigzag edges to each side of $\overrightarrow{vw}$ are visited in sequential order starting with $\overrightarrow{vw}$. (E.g., the edge adjacent to (furthest from) $\overrightarrow{vw}$ on a side is visited first (last) among all the edges on that side). If σ is separating, then (i) if the last visited edge in σ points north (west), then H ends in Γ_{top} (Γ_{bot}), and (ii) if the last visited edge in σ points south (east) then H does not end in Γ_{top} (Γ_{bot}).*

Proof. The first claim of this lemma follows immediately from Lemma 1. By Lemma 3, H must end in either a top corner vertex of Γ_{top}, a left corner vertex of Γ_{bot}, or the rightmost bottom corner of G. If the last visited edge in σ points north, there is no way for H to return to any of these end points other than the ones in Γ_{top}. The other claims follow from similar arguments. □

Due to space considerations, the proof of the following lemma can be found in Appendix 7.

Lemma 7 *Let x be a vertex (interior or boundary) of G. If x is first visited by H, from among all vertices of $G_{ne}[x]$, then the following properties hold:*

(1) *If $\overrightarrow{xx_n} \in H$, then for each step $\Gamma[c] \subset G_{ne}[x]$ of odd height, either $h(x, \Gamma[c])$ is even, or c is the end point of H.*
(2) *If $\overrightarrow{xx_s} \in H$, then for each step $\Gamma[c] \subset G_{ne}[x]$ of odd height, either $h(x, \Gamma[c])$ is odd, or c is the end point of H.*

The following lemma follows immediately from Lemma 7 (by symmetry of rotations and reflections).

Lemma 8 *Let x be a vertex (interior or boundary) of G. If x is first visited by H, from among all vertices of $G_{sw}[x]$, then the following properties hold:*

(1) *If $\overrightarrow{xx_w} \in H$, then for each step $\Gamma[c] \subset G_{sw}[x]$ of odd width, either $v(x, \Gamma[c])$ is even, or c is the end point of H.*
(2) *If $\overrightarrow{xx_e} \in H$, then for each step $\Gamma[c] \subset G_{sw}[x]$ of odd width, either $v(x, \Gamma[c])$ is odd, or c is the end point of H.*

4 Existence of a Squeeze-Free Hamiltonian Path

An algorithm we call VisitWest(G) constitutes a key ingredient in our Hamiltonian path algorithm. It can be applied on any staircase subgraph G that satisfies the condition that, if x is the top right corner of G, then $v(x, \Gamma)$ is even for any step Γ of G of odd width. It constructs a squeeze-free Hamiltonian path H that starts at x and moves in the direction $\overrightarrow{xx_w}$. From x_w H follows *pattern*1 until it reaches a step of G at odd vertical distance from x, then it follows *pattern*2 until it reaches a step of G at even vertical distance from x, then repeats. See Figure 5a for an example and Appendix 8 for more details.

In building a Hamiltonian path for an arbitrary staircase graph G, we will use three other variations of the VisitWest algorithm on various subgraphs of G – namely VisitEast, VisitNorth and VisitSouth. The algorithm VisitEast is similar to VisitWest, with the only difference that the starting point is at the top left corner and H begins by moving east. One may view the path produced by VisitWest(G_{sw}[x]) as composed of the subpath extending from x to the horizontally opposite corner y (see Figure 5a), the edge $\overrightarrow{yy_s}$, and the path produced by VisitEast($G_{sw}[x_s]$). The precondition for VisitEast(G) is that $v(x, \Gamma)$ is odd for each step Γ of G of odd width.

The algorithm VisitNorth is identical to VisitWest, when operating on copy of G rotated clockwise by 90° and then reflected vertically. In this case, the first edge in H is $\overrightarrow{xx_n}$, where x is the lower left corner of G. The precondition for VisitNorth(G) is that $h(x, \Gamma)$ is even for each step Γ of G of odd height. The algorithm VisitSouth is similar to VisitNorth, with the only difference that the starting point is at the top left corner of the lowest stair and H begins by moving south. The precondition for VisitSouth(G) is that $h(x, \Gamma)$ is odd for each step Γ of G of odd height. Thus we have the following lemma.

Lemma 9 *Let G be a staircase graph that satisfies the preconditions of the* VisitWest(East, North, South) *algorithm. The path H produced by running* VisitWest(East, North, South) *on G is a squeeze-free Hamiltonian path for G.*

We now prove our main result in Theorems 10, 11, and 12. The proofs of Theorems 11 and 12 are similar to Theorem 10, so we leave their details for Appendix 9.

Theorem 10 *Let x be a vertex on a horizontal segment of ∂G that is not the top left corner of a step. There is a squeeze-free Hamiltonian path H that starts at x and includes $\overrightarrow{xx_w}$ if and only if the following three conditions hold:*

(1) *For each step Γ_e of odd height lying east of x, $h(x, \Gamma_e)$ is odd. If the width of $G_{se}[x]$ is odd, then Γ_{top} is exempt from this condition.*
(2) *For each step Γ_w of odd width lying west of x, $v(x, \Gamma_w)$ is even. If the height of $G_{se}[x]$ is even, then Γ_{bot} is exempt from this condition.*
(3) *If the height of $G_{se}[x]$ is even, then the width of $G_{se}[x]$ is also even.*

Proof. For the if direction, assume that the three conditions hold. Our goal is to find a squeeze-free Hamiltonian path H in G.

If-Case 1. Consider first the case where the height of $G_{se}[x]$ is odd. (See Figure 5b.) By condition (2), any step Γ_w of odd width lying west of x (including Γ_{bot}) satisfies the restriction that $v(x, \Gamma_w)$ is even. This implies that $G_{sw}[x]$ satisfies the precondition of the VisitWest algorithm, so we begin with H =VisitWest($G_{sw}[x]$). (See Figure 5c). By Lemma 9 all vertices of $G_{sw}[x]$ have been visited by this method.

Let z be the vertex at the intersection between the bottom boundary segment of G and the vertical line through x. Because the height of $G_{se}[x]$ is odd, H points east into z. We let H take another step east (so $\overrightarrow{zz_e} \in H$), then proceed depending on the parity of the width of $G_{se}[x]$. If the width of $G_{se}[x]$ is even, by condition (1) each step Γ_e lying east of z_e (including Γ_{top}) satisfies the restriction that $h(z_e, \Gamma_e)$ is even (because $h(z, \Gamma_e)$ is odd). This implies that $G_{ne}[z_e]$ satisfies the precondition of the VisitNorth algorithm, so we append to H the path produced by VisitNorth($G_{ne}[z_e]$).

The case where x is a reflex corner needs special attention, because in this case the left side xy of the step

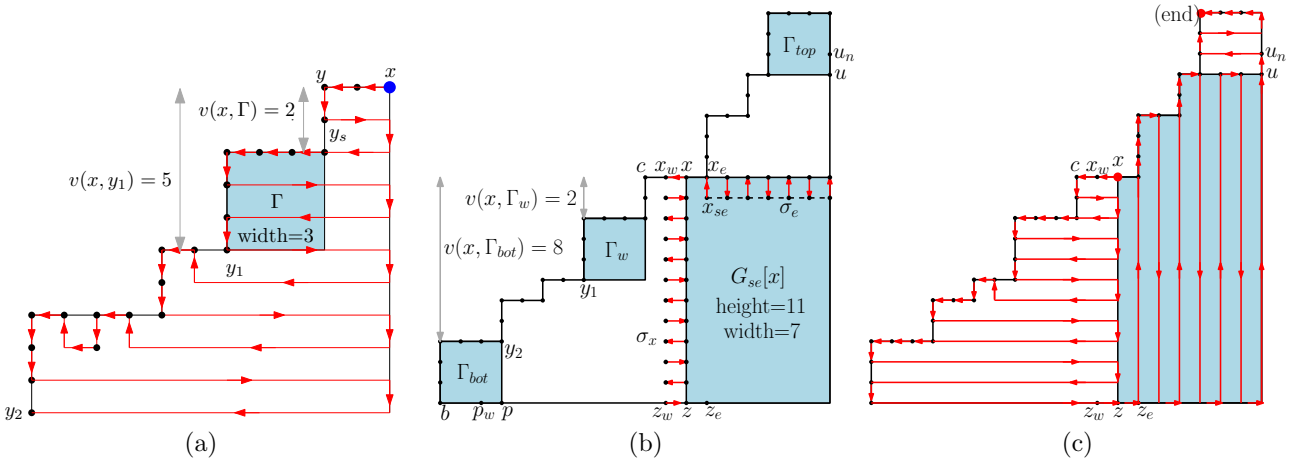


Figure 5: (a) VisitWest($G_{sw}[x]$). (b) Theorem 10: $G_{se}[x]$ has odd height and width. (c) Hamiltonian path.

$\Gamma[y]$ with lower left corner x is not in $G_{ne}[z_e]$. In this case $h(x, \Gamma[y]) = 0$ is even, and by condition (1) the height of $\Gamma[y]$ is even. This allows us to replace the vertical segment in H running alongside xy by a zigzag subpath of unit width that includes all vertices on the segment xy (similar to the *pattern*2 procedure). This along with Lemma 9 guarantees that, at the end of this procedure, all vertices of G have been visited.

If the width of $G_{se}[x]$ is odd (as in Figure 5c), condition (1) allows $h(x, \Gamma_{top})$ to be even and Γ_{top} to be of odd height. In this case $h(z_e, \Gamma_{top})$ is odd and $G_{ne}[z_e]$ does not satisfy the precondition imposed by VisitNorth. We handle this situation by restricting our attention to the subgraph $G'_{ne}[z_e]$ obtained from $G_{ne}[z_e]$ after eliminating Γ_{top}, with the exception of the lowest row of vertices in Γ_{top}. (The subgraph $G'_{ne}[z_e]$ is shaded in Figure 5c.) Note that $G'_{ne}[z_e]$ satisfies the precondition of VisitNorth, so we append to H the path produced by VisitNorth($G'_{ne}[z_e]$). By Lemma 9, at the end of this procedure all vertices of $G'_{ne}[z_e]$ have been visited. If x is a reflex corner, H absorbs the vertices along the vertical boundary segment sitting on x as described above. At this point, H is a squeeze-free Hamiltonian path for $G_{sw}[u]$, where u is the lower right corner u of Γ_{top}.

Because the width of $G_{se}[x]$ is odd, at the end of VisitNorth H points north into u. We add $\overrightarrow{uu_n}$ and $\overrightarrow{u_n u_{ne}}$ to H, then let H follow *pattern1* (reflected vertically) across Γ_{top} until all vertices of G have been visited. The result is a squeeze-free Hamiltonian path for G.

If-Case 2. The case when $G_{se}[x]$ has even height is similar and omitted for space considerations. See Appendix 9.

For the only-if direction, assume that there is a squeeze-free Hamiltonian path H in G. We next show that the three theorem conditions hold. We begin with the following two observations. Refer to Figure 5b.

(a) Because x is the start point of H and $\overrightarrow{xx_w} \in H$ (by the theorem statement), by Lemma 5 the two rightmost columns in $G_{sw}[x]$ form a separating zigzag sequence σ_x.

(b) By Lemma 1, $\overrightarrow{x_{ee}x_e} \notin H$. In addition, $\overrightarrow{x_{ne}x_e} \notin H$, because such an edge could only exist in H if x or x_e were a reflex corner, and in either case it would require that $H[x, x_{ne}]$ intersect a vertical line L through x_e both above and below x_e, which contradicts Lemma 1. Therefore $\overrightarrow{x_{se}x_e} \in H$. We claim that none of the vertices in $G_{ne}[x_{se}]$ has been visited at the time x_{se} is visited. Otherwise, if there were such a vertex $y \in G_{ne}[x_{se}]$ already visited at the time x_{se} is visited, then $H[x, y]$ would have to intersect the horizontal line L passing through x_{se} in two vertices on either side of x_{se}, leaving x_{se} unvisited between two visited vertices on L. This contradicts Lemma 1. Thus we are in the context of Lemma 5, with x_{se} being first visited among all vertices of $G_{ne}[x_{se}]$ and $\overrightarrow{x_{se}x_e} \in H$, therefore the two bottom rows of $G_{ne}[x_{se}]$ form a zigzag sequence σ_e.

Condition (1). By observation (b) above, x_{se} is the first visited by H among all vertices of $G_{ne}[x_{se}]$, and $\overrightarrow{x_{se}x_e} \in H$. Thus we can use the result of Lemma 7 on $G_{ne}[x_{se}]$ to show that, for each step Γ_e of odd height other than Γ_{top}, $h(x_{se}, \Gamma_e)$ is even and $h(x, \Gamma_e) = 1 + h(x_{se}, \Gamma_e)$ is odd. If the width of $G_{se}[x]$ is even, then the rightmost edge in σ_e points south. By Lemma 6, the end point of H lies outside of Γ_{top}. Thus Lemma 7 applies to show that, if the height of Γ_{top} is odd, then $h(x, \Gamma_{top})$ is odd. Thus condition (1) holds.

Condition (2). Because x is the start point of H and $\overrightarrow{xx_w} \in H$, we can use the result of Lemma 8 on $G_{sw}[x]$ to show that, for each step Γ_w of odd width other than Γ_{bot}, $v(x, \Gamma_w)$ is even. If the height of $G_{se}[x]$ is odd (see Figure 5b), then the lowest edge in σ_x points east.

By Lemma 6, the end point of H is not in Γ_{bot}. Thus Lemma 7 applies to show that, if the width of Γ_{bot} is odd, then $v(x, \Gamma_{bot})$ is even. Thus condition (2) holds.

Condition (3). Assume that the height of $G_{se}[x]$ is even. Because it is even, the lowest edge in σ_x points west. By Lemma 6, the end point of H lies in Γ_{bot}. This implies that the rightmost edge in σ_e points south (otherwise H would end in Γ_{top}). Because $\overrightarrow{x_{se}x_e} \in \sigma_e$ points north (see observation (b)), this is possible only if σ_e has odd length. This implies that $G_{se}[x]$ has even width, so condition (3) holds. This completes the proof. □

Theorem 11 *Let $x \in \partial G$ be a vertex on a horizontal segment of a step $\Gamma[c]$ of G, such that removing xx_s does not disconnect G. Let $\Gamma[c_1]$ be the first odd width step that lies west of x (if one exists). There is a squeeze-free Hamiltonian path H that starts at x and includes $\overrightarrow{xx_s}$ if and only if the following conditions hold:*

(1) *For each step Γ_e of odd height lying east of x, $h(x, \Gamma_e)$ is odd. If the width of $G_{se}[x]$ is odd, then Γ_{top} is exempt from this restriction.*
(2) *If c_1 exists, then for each step Γ_w of odd width lying west of $\Gamma[c_1]$, $v(c_1, \Gamma_w)$ is even. If the height of $G_{se}[c_1]$ is even, then Γ_{bot} is exempt from this restriction.*
(3) *If c_1 exists and the height of $G_{se}[c_1]$ is even, then the width of $G_{se}[x]$ is also even.*
(4) *If x does not lie on Γ_{bot}, then $|xc|$ is even.*
(5) *If $|xc|$ is odd, then the width of $G_{se}[x]$ is even.*

Theorem 12 *Let $x \in \partial G$ be a vertex on a horizontal segment of a step $\Gamma[c]$ of G, such that removing xx_e does not disconnect G. Let $\Gamma[c_2]$ be the first odd height step that lies east of x (if one exists). There is a squeeze-free Hamiltonian path H that starts at x and includes $\overrightarrow{xx_e}$ if and only if the following conditions hold:*

(1) *For each step Γ_w of odd width lying west of x, $v(x, \Gamma_w)$ is odd. If the height of $G_{se}[x]$ is odd, then Γ_{bot} is exempt from this restriction.*
(2) *If c_2 exists, then for each step Γ_e of odd height lying east of $\Gamma[c_2]$, $h(c_2, \Gamma_e)$ is even. If the width of $G_{se}[c_2]$ is even, then Γ_{top} is exempt from this restriction.*
(3) *If c_2 exists and the width of $G_{se}[c_2]$ is even, then the height of $G_{se}[x]$ is also even.*
(4) *If x does not lie on Γ_{bot}, then $|xc|$ is even.*
(5) *If $|xc|$ is odd, then the height of $G_{se}[x]$ is odd.*
(6) *If c_2 exists and $h(x, c_2) = 0$, then $\Gamma[c_2] = \Gamma_{top}$, $|xc|$ is even, and the height of $G_{se}[x]$ is even.*

The case where the first edge in H is $\overrightarrow{xx_n}$ is symmetric to the case where the first edge in H is $\overrightarrow{xx_w}$ (subject to a 90° clockwise rotation and a vertical reflection). Similarly, the case where the start point x of H is on a vertical staircase segment is symmetric to the case where x is on a horizontal staircase segment.

5 Complexity of the Decision Problem

Using a sweep line algorithm described in Appendix 10, we have the following result.

Theorem 13 *Given a staircase grid graph G represented as a sequence of t pairs of numbers indicating the height and width of each step in order from left to right, there is an $O(t)$ algorithm that decides whether G admits a squeeze-free Hamiltonian path starting from a vertex on ∂G.*

6 Conclusions

In this paper we give an $O(t)$ algorithm for deciding if a staircase grid graph with t steps has a squeeze-free Hamiltonian path starting at a boundary vertex on a step. Although not included here, we can use the same proof techniques to determine similar necessary and sufficient conditions for the existence of such paths starting from the bottom or right side of the staircase. We conjecture though that if there exists a squeeze-free Hamiltonian path starting at the bottom or right side, then there also exists a squeeze-free Hamiltonian path starting from a vertex on a step.

References

[1] E.M. Arkin, S.P. Fekete, K. Islam, H. Meijer, J.S.B. Michell, Y. Núñez-Rodríguez, V. Polishchuk, D. Rappaport, H. Xiao. Not being (super) thin or solid is hard: A study of grid Hamiltonicity. *Comp. Geom.: Theory and Applications*, 42(6-7), 582-605, 2009.

[2] E. M. Arkin, M. A. Bender, E. Demaine, S. P. Fekete, J. S. B. Mitchell, and S. Sethia. Optimal covering tours with turn costs. In *Proc. of the ACM-SIAM Symp. on Discrete Algs.*, 138-147, 2001.

[3] Z. Fitzsimmons and R. Flatland Algorithms for Collective Construction of 2D Block Structures with Holes *Am. J. of Undergraduate Research*, 10(2): 1-10, 2011.

[4] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamiltonian Paths in Grid Graphs. *SIAM J. on Computing*, 11(4): 676-686, 1982.

[5] C. Umans and W. Lenhart. Hamiltonian Cycles in Solid Grid Graphs. In *Proc. of the IEEE Symp. on Foundations of Comp. Sci.*, 496-507, 1997.

[6] J. Werfel, K. Petersen, and R. Nagpal. Distributed multi-robot algorithms for the TERMES 3D collective construction system. In *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, 2011.

Appendix

7 Proof of Lemma 7 from Section 3

This section contains the proof of Lemma 7 that was omitted from the body of the paper due to space considerations. We begin with Lemma 14 which is referenced in Lemma 7.

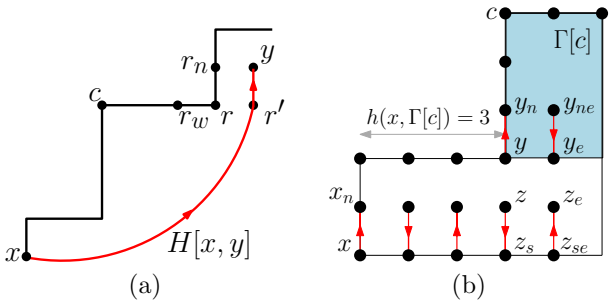


Figure 6: (a) Lemma 14. (b) Lemma 7

Lemma 14 *Let x be the start point of H. For any reflex corner vertex $r \in G_{ne}[x]$ ($r \in G_{sw}[x]$), r is first visited among all vertices in $G_{ne}[r]$ ($G_{sw}[r]$).*

Proof. Let $r \in G_{ne}[x]$ be an arbitrary reflex corner vertex, and let $\Gamma[c]$ be the step with top right corner vertex r. Because r is a reflex corner, r_n and r_w exist. No vertex $y \in \partial G$ above r can be visited before r by Lemma 2. Therefore r_n is visited after r. So assume for contradiction that there is some other vertex $y \in G_{ne}[r]$ that is visited by H prior to r. See Figure 6a. Then there is a vertex r' to the right of r, at the intersection between the horizontal through r and $H[x,y]$. By Lemma 1, r_w is not visited at the time r is visited (because r' is already visited). But $H[x,r]$, which links two boundary vertices, splits G into two pieces, and H cannot visit both unvisited vertices r_n (to the right of $H[x,r]$) and r_w (to the left of $H[x,r]$) without crossing itself. The arguments for the case $r \in G_{sw}[x]$ are symmetric. □

Lemma 7 *Let x be a vertex (interior or boundary) of G. If x is first visited by H, from among all vertices of $G_{ne}[x]$, then the following properties hold:*

(1) *If $\overrightarrow{xx_n} \in H$, then for each step $\Gamma[c] \subset G_{ne}[x]$ of odd height, either $h(x,\Gamma[c])$ is even, or c is the end point of H.*

(2) *If $\overrightarrow{xx_s} \in H$, then for each step $\Gamma[c] \subset G_{ne}[x]$ of odd height, either $h(x,\Gamma[c])$ is odd, or c is the end point of H.*

Proof. For (1), assume for contradiction that there exists a step $\Gamma[c] \subset G_{ne}[x]$ of odd height such that $h(x,\Gamma[c])$ is odd and c is not the end point of H. Refer to Figure 4b. Let y be the left bottom corner of $\Gamma[c]$. Consider the three possible directions from which H might visit y_n. Observe that $\overrightarrow{y_{nn}y_n} \notin H$, because if it were then subpath $H[x,y_{nn}]$ would have to intersect the vertical line L passing through y_n in two vertices on either side of y_n, leaving y_n unvisited between two visited vertices on L. This contradicts Lemma 1. So consider the case when $\overrightarrow{y_{ne}y_n} \in H$. Then y_{ne} is the first vertex visited in $G_{nw}[y_{ne}]$ (otherwise, if there were a vertex $y' \in G_{nw}[y_{ne}]$ visited prior to y_{ne}, then the subpath $H[x,y']$ would cross the vertical through y_{ne} at two vertices on either side of y_{ne}, contradicting Lemma 1). Thus we can apply Lemma 5 to show that $\overrightarrow{y_{ne}y_n}$ is the start of a zigzag sequence in $G_{nw}[y_{ne}]$, and because $|yc|$ is odd, the last edge in the zigzag is directed into corner c, thus H ends at c. This contradicts our assumption that H does not end at c, and thus $\overrightarrow{y_{ne}y_n} \notin H$.

Before completing the proof of property (1), first observe that by Lemma 5, the bottom two rows of $G_{ne}[x]$ form a zigzag sequence. Refer to Figure 6b. Because $h(x,\Gamma[c])$ is odd and the zigzag starts with the upward directed edge $\overrightarrow{xx_n}$, the zigzag edge in c's vertical grid line points downward. Label this edge $\overrightarrow{zz_s}$. Now consider the last case for property (1) which is when $\overrightarrow{yy_n} \in H$. By Lemma 14, y is first visited among all vertices in $G_{ne}[y]$. By Lemma 5, $\overrightarrow{y_{ne}y_e} \in H$, which points in a direction opposite to that of $\overrightarrow{z_{se}z_e}$, contradicting Lemma 1.

For property (2), note that because $\overrightarrow{xx_s} \in H$, no vertex below x on its vertical line is visited before x. This combined with the fact that x is the first vertex visited in $G_{ne}[x]$ shows that x is also the first vertex visited in $G_{se}[x]$. Thus by Lemma 5, the upper two rows of $G_{se}[x]$ form a zigzag sequence, with the first edge, $\overrightarrow{xx_s}$, pointing downward. The rest of the proof is analogous to the proof of property (1). □

8 Additional Details For VisitWest from Section 4

Algorithm 1 details the VisitWest algorithm. See Figure 5a for an example.

Algorithm 1: VisitWest(**staircase** G)

Precondition: Let t be the upper right corner of G. For each step Γ of G of odd width, $v(t,\Gamma)$ is even.

Initialize $x \leftarrow t$ and $H \leftarrow \{\overrightarrow{xx_w}\}$. Let b be the lower left corner of G.

repeat

- Let y_1 be the reflex corner west of x closest to x, such that $v(x,y_1)$ is odd.
- If no such vertex exists, then $y_1 \leftarrow b$.
- Let y_2 be the reflex corner west of y_1 closest to y_1, such that $v(y_1,y_2)$ is odd.
- If no such vertex exists, then $y_2 \leftarrow b$.
- From x, let H follow *pattern*1 south-west until it meets y_1.
- From y_1, let H follow *pattern*2 south-west until it meets y_2.
- Reset $x \leftarrow y_2$.

until *all vertices of G have been visited*;

Output H.

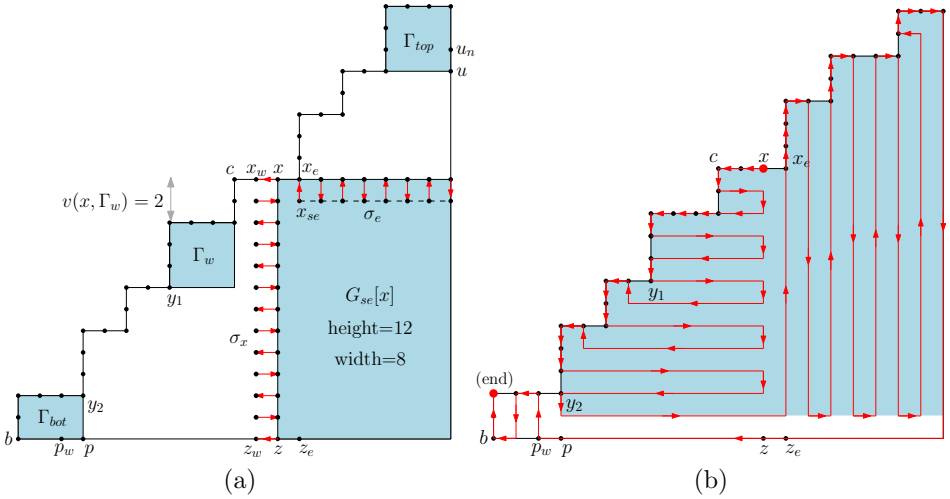


Figure 7: (a) Theorem 10: $G_{se}[x]$ has even height and even width. (b) Hamiltonian path.

9 Proofs of Theorems 10,11, and 12 from Section 4

We begin by supplying the portion of the proof of Theorem 10 that was omitted in the body of the text.

Omitted Portion of Theorem 10 proof: If-Case 2. Consider now the case where the height of $G_{se}[x]$ is even, as depicted in Figure 7a. By condition (3), the width of $G_{se}[x]$ is also even. Let G' be the graph obtained from G after eliminating all vertices on the lowest boundary segment of G, along with all vertices in Γ_{bot}, with the exception of those lying on the right boundary segment of Γ_{bot}. (G' is shown shaded in Figure 7b.) Thus $G'_{se}[x]$ is of odd height and even width. We trace H across $G'_{se}[x]$ using the same procedure as described above for the case where $G_{se}[x]$ was of odd height and even width. It can be verified that, at the end of this procedure, H points south into the lower right corner of G'. At this point H takes a unit step south, then continues west along the bottom boundary segment of G up to the vertex p_w, where p is the lower right corner of Γ_{bot}. From p_w H follows *pattern1* (rotated counterclockwise by 90° and reflected vertically) until all vertices of Γ_{bot} have been visited. At that point, all vertices of G have been visited, and H is a squeeze-free Hamiltonian path for G.

We now provide complete proofs of Theorems 11 and 12.

Theorem 11 *Let $x \in \partial G$ be a vertex on a horizontal segment of a step $\Gamma[c]$ of G, such that removing xx_s does not disconnect G. Let $\Gamma[c_1]$ be the step of odd width closest to x that lies west of x (if one exists). There is a squeeze-free Hamiltonian path H that starts at x and includes $\overrightarrow{xx_s}$ if and only if the following five conditions hold:*

(1) *For each step Γ_e of odd height lying east of x, the horizontal distance $h(x,\Gamma_e)$ is odd. If the width of $G_{se}[x]$ is odd, then Γ_{top} is exempt from this restriction.*

(2) *If c_1 exists, then for each step Γ_w of odd width lying west of $\Gamma[c_1]$, the vertical distance $v(c_1,\Gamma_w)$ is even. If the height of $G_{se}[c_1]$ is even, then Γ_{bot} is exempt from this restriction.*

(3) *If c_1 exists and the height of $G_{se}[c_1]$ is even, then the width of $G_{se}[x]$ is also even.*

(4) *If x does not lie on Γ_{bot}, then $|xc|$ is even.*

(5) *If $|xc|$ is odd, then the width of $G_{se}[x]$ is even.*

Proof. We consider each of the two directions (if, and only if) in turn.

If direction. For the if direction, assume that the five conditions hold. Our goal is to find a squeeze-free Hamiltonian path H in G. Let b be the lower left corner of Γ_{bot}.

If-Case 1. Consider first the case where either of the following is true: (i) c_1 does not exist and $|xc|$ is even, and (ii) c_1 exists and the height of $G_{se}[c_1]$ is odd. In the latter case x may not lie on Γ_{bot} (due to the existence of c_1), and by condition (4) $|xc|$ is even.

Define $d = c_1$ if c_1 exists (see Figure 8), and $d = b_n$ otherwise (see Figure 9a). Let y be the intersection point between the vertical through x and the horizontal through d. In either case, y_s exists. (Note that in case (i) when x is the top left corner of $\Gamma[bot]$, $G_{nw}[y]$ degenerates to a vertical line segment.) Otherwise, by the definition of c_1 and the fact that $|xc|$ is even, every step in $G_{nw}[y]$ has even width. In either case *pattern*1 (starting with $\overrightarrow{xx_s}$) can be used to visit all vertices of $G_{nw}[y]$ (regardless of the existence of c_1). We let H follow this path until it reaches the lower left corner d_1 of $G_{nw}[y]$, coming from north (so $\overrightarrow{d_{1n}d_1} \in H$). If d_{1w} exists, H continues west as far as it can go (up to c_1 in Figure 8b). Next H takes a step south.

Let z be the intersection point between the vertical through x and the bottom boundary segment of G. If c_1 does not exist, $z = y_s$ and H continues east up to z_e (see Figure 9a). If c_1 exists, H visits all vertices of $G_{sw}[y_s]$ on its way to z_e as follows. By condition (2), any step Γ_w of odd width lying west of c_1 (including Γ_{bot}) satisfies the restriction that $v(c_1,\Gamma_w)$ is even. This implies that $v(c_{1s},\Gamma_w)$ is odd, so the precondition of VisitEast restricted to the subgraph

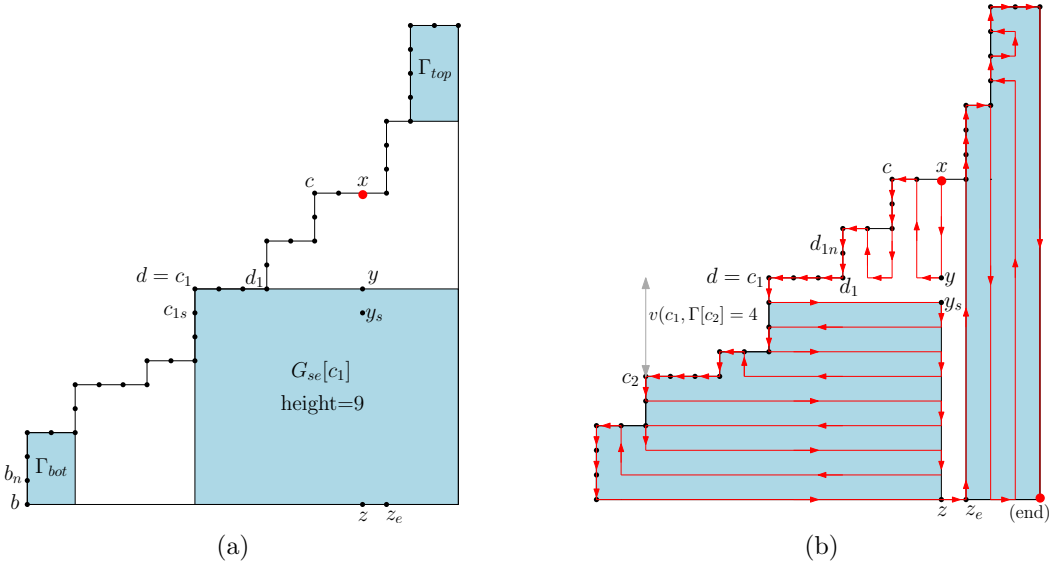


Figure 8: Theorem 11: (a) $G_{se}[c_1]$ has odd height (b) Hamiltonian path H.

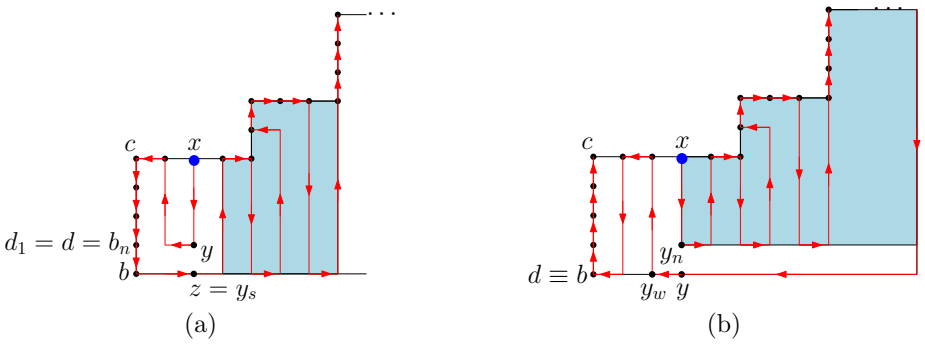


Figure 9: Theorem 11, $x \in \Gamma_{bot}$ (a) $|xc|$ even (b) $|xc|$ odd.

$G_{sw}[y_s]$ (see left shaded subgraph in Figure 8b) is satisfied. We append to H the path produced by VisitEast($G_{sw}[y_s]$). Using the result of Lemma 9, it can be verified that at this point H is a squeeze-free Hamiltonian path in $G_{nw}[z]$. Let H take another step $\overrightarrow{zz_e}$.

If the width of $G_{se}[x]$ is even, or if Γ_{top} has even height, then by condition (1) $h(x, \Gamma_e)$ is odd for every step Γ_e of odd height lying east of x. Then $h(z_e, \Gamma_e)$ is even and therefore the precondition of VisitNorth is satisfied when restricted to $G_{ne}[z_e]$ (see right shaded subgraph in Figure 8b). We append to H the path produced by VisitNorth($G_{ne}[z_e]$).

The case where x is a reflex corner needs special attention, because in this case the left side xy of the step $\Gamma[y]$ with lower left corner x is not in $G_{ne}[z_e]$. In this case $h(x, \Gamma[y]) = 0$ is even, and by condition (1) the height of $\Gamma[y]$ is even. Then we can replace the vertical segment in H running alongside xy by a zigzag subpath of unit width that includes all vertices on the segment xy (similar to the *pattern*2 procedure). This along with Lemma 9 guarantees that, at the end of this procedure, all vertices of G have been visited.

Finally, consider the situation where the width of $G_{se}[x]$ is odd and Γ_{top} has odd height. Let G' be the subgraph of G obtained by removing the top row of vertices in Γ_{top}. Then the top step in G' has even height. This along with condition (1) shows that $G'_{ne}[z_e]$ satisfies the precondition of VisitNorth, so we let H follow the path produced by VisitNorth($G'_{ne}[z_e]$). If x is a reflex corner of G, we adjust H as described above so that it visits all vertices on the vertical boundary segment incident on x. This along with Lemma 9 guarantees that H is a squeeze-free Hamiltonian path for G'. Because $G_{se}[x]$ has odd width, H ends up pointing north into top right corner of G'. Let H take another step north, then west all the way to the top left corner of Γ_{top}. The resulting path is a squeeze-free Hamiltonian path for G.

If-Case 2. Consider now the case where either of the following is true: (i) c_1 does not exist and $|xc|$ is odd, and (ii) c_1 exists and the height of $G_{se}[c_1]$ is even. In either case, conditions (3) and (5) guarantee that the width of $G_{se}[x]$ is even. If $|xc|$ is odd, by condition (4) x lies on Γ_{bot}.

Define $d = c_1$ if c_1 exists (see Figure 10), otherwise $d = b$ (see Figure 9b). Let y be the intersection point between the vertical through x and the horizontal through d. By

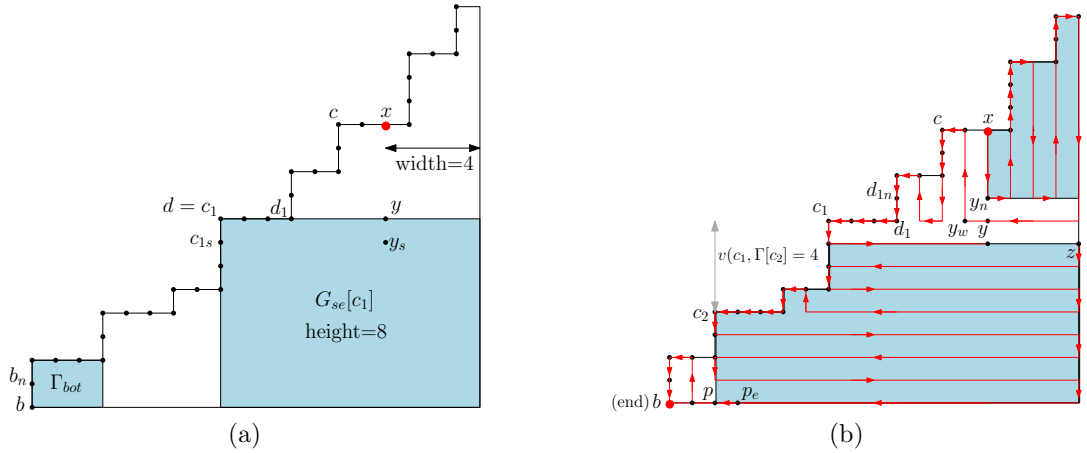


Figure 10: Theorem 11: (a) $G_{se}[c_1]$ has even height (b) Hamiltonian path H.

condition (1), for any step Γ_e of odd height lying east of x (including Γ_{top}), $h(x, \Gamma_e)$ is odd. Thus the precondition of VisitSouth restricted to the subgraph $G_{ne}[y_n]$ (see top shaded subgraph in Figures 10b and 9b) is satisfied. Let H follow the path produced by VisitSouth($G_{ne}[y_n]$). By Theorem 9, H is a squeeze-free Hamiltonian path for $G_{ne}[y_n]$. Because the width of $G_{se}[x]$ is even, H ends up pointing south into the lower right corner of $G_{ne}[y_n]$. Let H take another step south, then west all the way to y_w. As in the previous case, from y_w the path H follows *pattern*1 (starting with $\overrightarrow{y_w y_{nw}}$) restricted to $G_{nw}[y_w]$. If x lies on Γ_{bot}, *pattern*1 completes a squeeze-free Hamiltonian path for G (see Figure 9b).

If x does not lie on Γ_{bot}, the width of $G_{nw}[y_w]$ is even and therefore H arrives at the top right corner d_1 of $\Gamma[c_1]$ coming from north. Refer to Figure 10b. At this point H is a squeeze-free Hamiltonian path of $G_{ne}[d_1]$. From d_1 H continues west until it reaches c_1, then takes a step south. If $\Gamma[c_1]$ is identical to Γ_{bot}, then VisitEast($G_{se}[c_{1s}]$) completes a squeeze-free Hamiltonian path for G.

Assume now that $\Gamma[c_1]$ is not identical to Γ_{bot}. Let G' be the subgraph of G obtained by removing the vertices in Γ_{bot}, with the exception of the rightmost vertex column in Γ_{bot}. Let z be the intersection point between the horizontal through c_{1s} and the right boundary segment of G. From c_{1s}, H follows the path produced by VisitEast($G'_{sw}[z]$), up to the lower right corner p of Γ_{bot}. Condition (2) guarantees that the precondition of VisitEast is satisfied, so at this point H is a squeeze-free Hamiltonian path of $G_{ne}[p]$ (by Lemma 9). Because the height of $G'_{sw}[z]$ is odd, H arrives at p from east, so $\overrightarrow{p_e p} \in H$. We let H take another step east, then trace *pattern*1 (rotated counterclockwise by 90° and reflected vertically) across Γ_{bot} to complete a squeeze-free Hamiltonian path for G.

Only if direction. For the only-if direction, assume that there is a squeeze-free Hamiltonian path H in G. We next show that the five lemma conditions hold. We begin with two observations:

(a) Because x is the starting point of H and $\overrightarrow{xx_s} \in H$ (by the theorem statement), we can use Lemma 5 to show that the top two rows in $G_s[x]$ form a separating zigzag sequence σ_x (see Figure 11).

(b) Assuming that c_1 exists, let d be the top right corner of $\Gamma[c_1]$. By Lemma 14 d is first visited among all vertices in $G_{sw}[d]$. If $\overrightarrow{dd_w} \in H$, by Lemma 5 the rightmost two columns of $G_{sw}[d]$ form a separating zigzag sequence σ_d (see Figure 11a).
If $\overrightarrow{dd_w} \notin H$, then $\overrightarrow{d_{sw}d_w} \in H$ (as the only way to reach d_w). Also note that d_{sw} is first visited among all vertices in $G_{nw}[d_{sw}]$. Otherwise, if there were a vertex $y \in G_{nw}[d_{sw}]$ already visited at the time d_{sw} is visited, then $H[x, y]$ would have to intersect the horizontal line L passing through d_{sw} in two vertices on either side of d_{sw}, leaving d_{sw} unvisited between two visited vertices on L. This contradicts Lemma 1. By Lemma 5, the two rows of $G_{nw}[d_{sw}]$ form a zigzag sequence σ_w (see Figure 11b). Because the width of $\Gamma[c_1]$ is odd (by definition), the leftmost edge in σ_w is $\overrightarrow{c_{1s}c_1}$, therefore c_1 is the end point of H.

Condition (1). Let $\Gamma_e = \Gamma[c_e]$ be an arbitrary step of odd height lying east of x (if no such step exists, there is nothing to prove). By Lemma 7, either $h(x, \Gamma_e)$ is odd or c_e is the end point of H. If $\Gamma_e \neq \Gamma_{top}$, then by Lemma 3 c_e cannot be the end point of H, so condition (1) holds. Assume now that $\Gamma_e = \Gamma_{top}$ and the width of $G_{se}[x]$ is even. In this case the rightmost edge in σ_x points south, and by Lemma 6 H many not end in Γ_{top}. Thus Lemma 7 applies again to show that $h(x, \Gamma_e)$ is odd, so condition (1) holds.

Condition (2). Assume that c_1 exists and let $\Gamma_w = \Gamma[c_w]$ be an arbitrary step of odd width lying west of c_1 (if no such step exists, there is nothing to prove). Note that $\Gamma[c_1] \neq \Gamma_{top}$ because of the existence of $\Gamma[c]$, and there is nothing to prove if $\Gamma[c_1] = \Gamma_{bot}$. By Lemma 3, H may not end at c_1. This along with observation (b) above implies that $\overrightarrow{dd_w} \in H$ (otherwise H would end at c_1). By Lemma 8 either $v(c_1, \Gamma_w)$ is even or c_w is the end point of H. If $\Gamma_w \neq \Gamma_{bot}$, then by Lemma 3 c_w cannot be the endpoint of H, so condition (2) holds. Assume now that $\Gamma_w = \Gamma_{bot}$ and that the height of

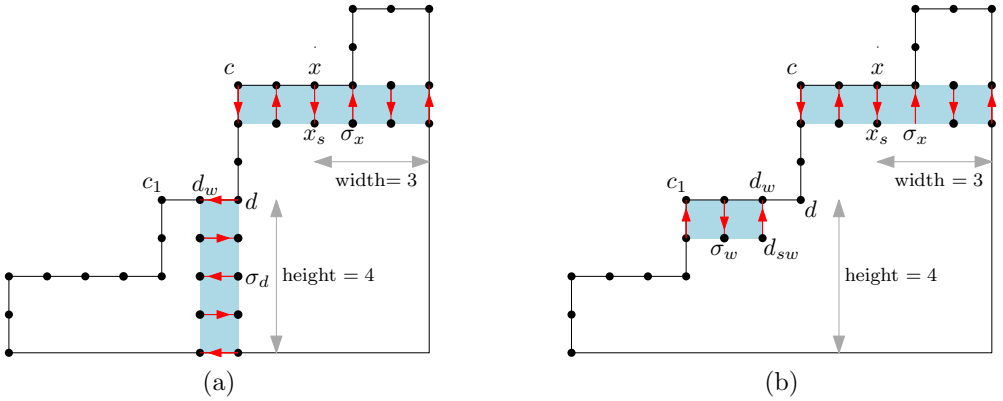


Figure 11: Theorem 11, only-if direction (a) $\overrightarrow{dd_w} \in H$ (b) $\overrightarrow{dd_w} \notin H$

$G_{se}[c_1]$ is odd. In this case the lowest edge in σ_x points east, and by Lemma 6 H cannot end in Γ_{bot}. This implies that c_w is not the endpoint of H, so condition (2) holds.

Condition (3). Assume that c_1 exists and the height of $G_{se}[c_1]$ is even. Note that $\Gamma[c_1] \neq \Gamma_{top}$ due to the existence of $\Gamma[c]$. We prove by contradiction that the width of $G_{se}[x]$ is even. Assume to the contrary that the width of $G_{se}[x]$ is odd. Then the rightmost edge in σ_x points north (see Figure 11a), and by Theorem 6 H must end in Γ_{top}. This along with observation (b) above implies that $\overrightarrow{dd_w} \in H$ (otherwise H would end at c_1). Because $\overrightarrow{dd_w} \in H$ and the height of $G_{se}[c_1]$ is even, the lowest edge in σ_d points west. By Lemma 6 H must end in Γ_{bot}, a contradiction. We conclude that the width of $G_{se}[x]$ is even.

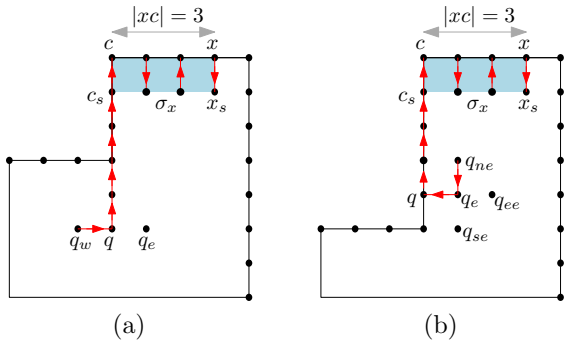


Figure 12: Theorem 11, only-if direction, $x \in \Gamma_{top}$ (a) q_w exists (b) q_w does not exist.

Condition (4). Assume that $x \notin \Gamma_{bot}$. We prove by contradiction that $|xc|$ is even. Assume to the contrary that $|xc|$ is odd. In this case $\overrightarrow{c_s c} \in H$ is the leftmost edge in σ_x, so H ends at c. Because $\Gamma[c] \neq \Gamma_{bot}$, Lemma 3 implies that c is the top left corner of Γ_{top}.

Let $q \in G$ be such that qc is the longest vertical subpath of H that ends at c. Refer to Figure 12. Then either $\overrightarrow{q_w q} \in H$ or $\overrightarrow{q_e q} \in H$ (as the only way to reach q). Note that $\overrightarrow{q_w q}$ would create a squeeze at q, because H must have visited q_e prior to q. Thus $\overrightarrow{q_w q} \notin H$ and therefore $\overrightarrow{q_e q} \in H$. This implies that q_w does not exist (otherwise H would create a squeeze at q). Because G has at least two steps (by our problem statement) we conclude that q lies above the lower left corner of Γ_{top}, and q_{se} exists. Note that all three vertices q_{se}, q_{ne} and q_{ee} must have been visited prior to q_e, so any of $\overrightarrow{q_{se} q_e}$, $\overrightarrow{q_{ne} q_e}$ and $\overrightarrow{q_{ee} q_e}$ would create a squeeze of q_e. This means that H has no way of reaching q_e, contradicting the fact that H is Hamiltonian. We conclude that $|xc|$ is even.

Condition (5). Assume that $|xc|$ is odd. By condition (4), $x \in \Gamma_{bot}$. In this case $\overrightarrow{c_s c} \in \sigma_x$, so H ends at c. If the width of $G_{se}[x]$ is odd, then the rightmost edge in σ_x points north, and by Lemma 6 H ends in Γ_{top}, a contradiction. Thus the width of $G_{se}[x]$ is even and condition (5) holds. This completes the proof. □

Theorem 12 *Let $x \in \partial G$ be a vertex on a horizontal segment of a step $\Gamma[c]$ of G, such that removing xx_e does not disconnect G. Let $\Gamma[c_2]$ be the step of odd height closest to x that lies east of x (if one exists). There is a squeeze-free Hamiltonian path H that starts at x and includes $\overrightarrow{xx_e}$ if and only if the following six conditions hold:*

(1) *For each step Γ_w of odd width lying west of x, the vertical distance $v(x, \Gamma_w)$ is odd. If the height of $G_{se}[x]$ is odd, then Γ_{bot} is exempt from this restriction.*

(2) *If c_2 exists, then for each step Γ_e of odd height lying east of $\Gamma[c_2]$, the horizontal distance $h(c_2, \Gamma_e)$ is even. If the width of $G_{se}[c_2]$ is even, then Γ_{top} is exempt from this restriction.*

(3) *If c_2 exists and the width of $G_{se}[c_2]$ is even, then the height of $G_{se}[x]$ is also even.*

(4) *If x does not lie on Γ_{bot}, then $|xc|$ is even.*

(5) *If $|xc|$ is odd, then the height of $G_{se}[x]$ is odd.*

(6) *If c_2 exists and $h(x, c_2) = 0$, then $\Gamma[c_2] = \Gamma_{top}$, $|xc|$ is even, and the height of $G_{se}[x]$ is even.*

Proof. We consider each of the two directions (if, and only if) in turn.

If direction. For the if direction, assume that the five conditions hold. Our goal is to find a squeeze-free Hamiltonian path H in G. Let z be the intersection point between the

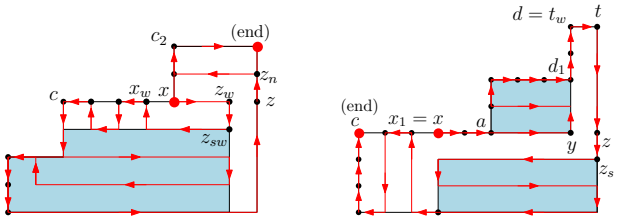


Figure 13: Theorem 12 (a) $h(x, c_2) = 0$ (b) c_2 does not exist and $x \in \Gamma_{bot}$.

horizontal through x and the right boundary segment of G.

If-Case 1. We begin with the simplest case where c_2 exists and $h(x, c_2) = 0$. Refer to Figure 13a. By condition (6) $|xc|$ is even and the height of $G_{se}[x]$ is even. In this case H proceeds east until it reaches z_w, then takes one step south. By condition (1) $v(x, \Gamma_w)$ is odd for every step Γ_w of odd width lying west of x. Thus $v(z_{sw}, \Gamma_w)$ is even and therefore the precondition of VisitEast is satisfied when restricted to $G_{sw}[z_{sw}]$ (see shaded subgraph in Figure 13a). We append to H the path produced by VisitEast($G_{sw}[z_{sw}]$). If x_w exists, we replace the segment in H running along $x_w c$ by a zigzag subpath of unit height (starting with $\overrightarrow{x_{sw}x_w}$) that includes all vertices on $x_w c$. Because $|xc|$ is even, this subpath ends with $\overrightarrow{cc_s}$ and attaches seamlessly to the subpath of H starting at c_s. This along with Lemma 9 guarantees that, at this point, all vertices of $G_{sw}[z_w]$ have been visited. Because the height of $G_{se}[x]$ is even, H ends pointing east into the lower right corner of $G_{sw}[z_w]$. We let H take another step west to meet the the right boundary segment of G, then north up to z_n. Because $h(x, c_2) = 0$, by condition (6) $\Gamma[c_2] = \Gamma_{top}$, so we let H follow *pattern*1 (starting with $\overrightarrow{z_n z_{nw}}$) to complete a squeeze-free Hamiltonian path for G.

If-Case 2. Consider now the case where either c_2 does not exist, or c_2 exists and $h(x, c_2) > 0$. Define the following points: b be the lower left corner of G; t is the upper right corner of G; $d = d_1 = t_w$ if c_2 does not exist (see Figure 13b), else $d = c_2$ and d_1 is the lower reflex corner of $\Gamma[c_2]$ (see Figure 14a); and y is the intersection point between the horizontal through x and the vertical through d.

If x is not a reflex corner, H proceeds east from x until it reaches the first reflex vertex a. By the definition of c_2, every step in $G_{nw}[y]$ has even height See the top left shaded subgraph $G_{nw}[y]$ in Figure 14a for an example. This implies that *pattern*1 (starting with $\overrightarrow{aa_e}$) can be used to visit all vertices of $G_{nw}[y]$ (regardless of the existence of c_2). We let H follow this path until it reaches d_1 coming from the west (so $\overrightarrow{d_{1w}d_1} \in H$). If d_{1n} exists, H continues north as far as it can go (up to d). Next H takes a step east, then continues south all the way down to y_e. From here the path taken by H depends on the parity of the width of $G_{se}[c_2]$.

If-Case 2a. Assume first that the width of $G_{se}[c_2]$ is odd. Note that if c_2 does not exist, y_e coincides with z and H points south into z (see Figure 13b). Otherwise, if y_e does not coincide with z, then y_{ee} exists and is not on the boundary of G, because the width of $G_{se}[c_2]$ is odd. In this case H visits all vertices of $G_{ne}[y_{ee}]$ on its way to z as follows. By condition (2), any step Γ_e of odd height lying east of c_2 (including Γ_{top}) satisfies the restriction that $h(c_2\Gamma_e)$ is even. This implies that $h(y_{ee}, \Gamma_e)$ is also even, so the precondition of VisitNorth restricted to the subgraph $G_{ne}[y_{ee}]$ (see top right shaded subgraph in Figure 14a) is satisfied. We append to H the path produced by VisitNorth($G_{ne}[y_{ee}]$). Using the result of Lemma 9, it can be verified that at this point H is a squeeze-free Hamiltonian path in $G_{nw}[z]$. Because the width of $G_{se}[c_2]$ is odd (by our assumption), H arrives at z from north, so $\overrightarrow{z_n z} \in H$. Let H take another step $\overrightarrow{zz_s}$.

If the height of $G_{se}[x]$ is even, H continues along the path produced by VisitEast($G_{sw}[z_s]$). Arguments similar to the ones used in the first case show that at the end of this visit, H is a squeeze-free Hamiltonian path for G.

Assume now that the height of $G_{se}[x]$ is odd, as depicted in Figure 14a. Let $x_1 = x$ if x is on Γ_{bot} (see Figure 13b), otherwise x_1 is the top right corner of Γ_{bot} (see Figure 14a). Let G' be the subgraph obtained by removing from G all vertices left of the vertical through x_1. Condition (1) guarantees that, for each step Γ_w of odd width lying west of x, $v(x, \Gamma_w)$ is odd. This implies that $G'_{sw}[z_s]$ satisfies the precondition of VisitEast, so we let H follow the path produced by VisitEast($G'_{sw}[z_s]$). If x is not on Γ_{bot}, by condition (4) $|xc|$ is even. In this case we adjust H to incorporate all vertices on the horizontal boundary segment $x_w c$ (if such vertices exist), as in the first case discussed above. At this point H is a squeeze-free Hamiltonian path for G'. Because $G_{se}[x]$ has odd height, H ends up pointing west into the lower left corner of G'. Let H take another step west, then follow *pattern*1 across the vertices in $G \setminus G'$. The resulting path H is a squeeze-free Hamiltonian path for G.

If-Case 2b. Assume now that the width of $G_{se}[c_2]$ is even. Refer to Figure 14b. By condition (3) the height of $G_{se}[x]$ is also even. By condition (5) $|xc|$ is even. By condition (1), for each step Γ_w of odd width lying west of x (including Γ_{bot}), $v(x, \Gamma_w)$ is odd. This implies that $v(y_{se}, \Gamma_w)$ is even, so the precondition of VisitWest restricted to the subgraph $G_{sw}[y_{se}]$ (see lower left shaded subgraph in Figure 14b) is satisfied. Let H follow the path produced by VisitWest($G_{sw}[y_{se}]$). If $|xc| > 0$, we adjust H to incorporate all vertices on the horizontal boundary segment $x_w c$, as discussed above. Because $|xc|$ is even, this adjustment is possible. Because the height of $G_{se}[x]$ is even, H ends up pointing east into the lower right corner u of $G_{sw}[y_e]$. At this point H is a squeeze-free Hamiltonian path for $G_{nw}[u]$. Let H take another step east, so $\overrightarrow{uu_e} \in H$.

Let G' be the graph obtained by removing from G all vertices above the bottom row in Γ_{top}. By condition (2), for each step Γ_e of odd height in G' lying east of c_2, $h(c_2, \Gamma_e)$ is even. Then $h(u_e, \Gamma_e)$ is also even. This implies that the precondition of VisitNorth restricted to the subgraph $G'_{ne}[u_e]$ (see right shaded subgraph in Figure 14b) is satisfied. Let H follow the path produced by VisitNorth($G'_{ne}[u_e]$). Be-

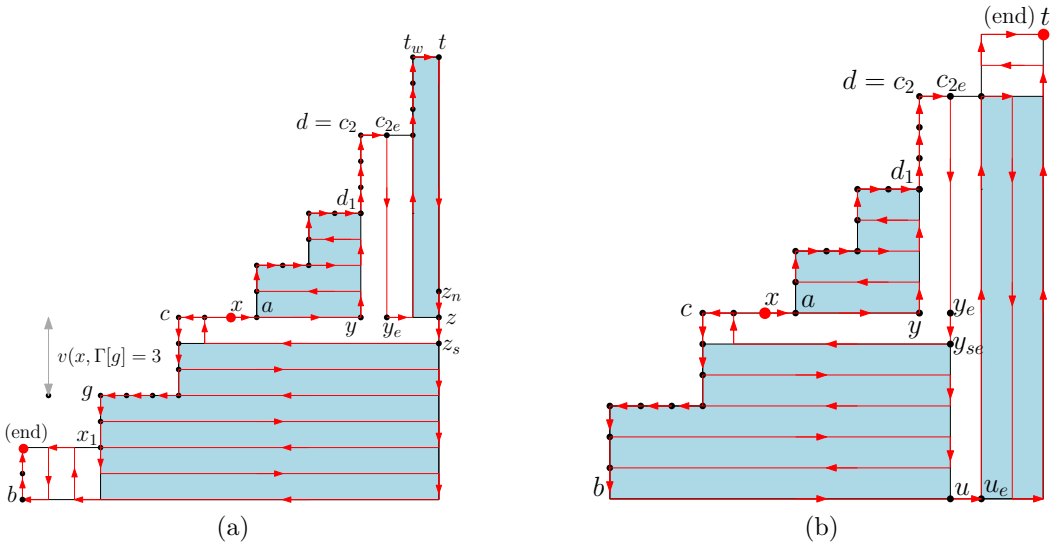


Figure 14: Path H from Theorem 12; the width of $G_{se}[c_2]$ is (a) odd (b) even.

cause the width of $G_{se}[c_2]$ is even, the width of $G'_{ne}[u_e]$ is also even, therefore H ends up pointing north into the lower right corner of Γ_{top}. Let H take another step north, then follow *pattern*1 across the vertices in $G \setminus G'$. The resulting path H is a squeeze-free Hamiltonian path for G.

Only if direction. For the only-if direction, assume that there is a squeeze-free Hamiltonian path H in G. We next show that the six theorem conditions hold. We begin with three observations:

(a) Because x is the starting point of H and $\overrightarrow{xx_e} \in H$ (by the theorem statement), we can use Lemma 5 to show that the two leftmost columns in $G_e[x]$ form a separating zigzag sequence σ_x. Refer to Figure 15.

(b) If x_w exists, then $\overrightarrow{x_{sw}x_w} \in H$. This is because $\overrightarrow{xx_e} \in H$ (by the theorem statement), and by Lemma 1 H cannot arrive at x_w from the left, therefore it must reach it coming from south. Also note that x_{sw} is first visited among all vertices in $G_{nw}[x_{sw}]$. Otherwise, if there were a vertex $y \in G_{nw}[x_{sw}]$ already visited at the time x_{sw} is visited, then $H[x, y]$ would have to intersect the horizontal line L passing through x_{sw} in two vertices on either side of x_{sw}, leaving x_{sw} unvisited between two visited vertices on L. This contradicts Lemma 1. By Lemma 5 the two rows in $G_{nw}[x_{sw}]$ form a zigzag sequence σ_w (provided that x_w exists).

(c) If c_2 exists, let d be the lower left corner of $\Gamma[c_2]$. By Lemma 14 d is first visited among all vertices in $G_{ne}[d]$. If $\overrightarrow{dd_n} \in H$, by Lemma 5 the bottom two rows of $G_{ne}[d]$ form a separating zigzag sequence σ_d (see Figure 15a). If $\overrightarrow{dd_n} \notin H$, then $\overrightarrow{d_{ne}d_n} \in H$ (as the only way to reach d_n). Arguments similar to the ones used in observation (b) above show that d_{ne} is first visited among all vertices in $G_{nw}[d_{ne}]$. By Lemma 5, the two columns of $G_{nw}[d_{ne}]$ form a zigzag sequence σ_n (see Figure 15b). Because $\Gamma[c_2]$ is of odd height (by definition), the topmost edge in σ_n is $\overrightarrow{c_{2e}c_2}$, therefore c_2 is the end point of H.

Condition (1). Because $\overrightarrow{xx_e} \in H$ and x is the start point of H, we can use the result of Lemma 8 to show that, for each step Γ_w of odd width other than Γ_{bot} lying west of x, $v(x, \Gamma_w)$ is odd. If the height of $G_{se}[x]$ is even, then the lowest edge in σ_x points east. By Lemma 6, H does not end in Γ_{bot}. Thus Lemma 8 applies to show that, if the height of $G_{se}[x]$ is even, then $v(x, \Gamma_{bot})$ is odd.

Condition (2). Assume that c_2 exists and let $\Gamma_e = \Gamma[c_e]$ be an arbitrary step of odd height lying east of c_2 (if no such step exists, there is nothing to prove). Note that $\Gamma[c_2] \neq \Gamma_{bot}$ because of the existence of $\Gamma[c]$, and there is nothing to prove if $\Gamma[c_2] = \Gamma_{top}$. By Lemma 3, H may not end at c_2. This along with observation (c) above implies that $\overrightarrow{dd_n} \in H$ (otherwise H would end at c_2). Lemma 7 applied on $G_{ne}[d]$ tells us that either $h(d, \Gamma_e) = h(c_2, \Gamma_e)$ is even, or c_e is the end point of H. If $\Gamma_e \neq \Gamma_{top}$, then by Lemma 3 c_e cannot be the endpoint of H, so condition (2) holds. Assume now that $\Gamma_e = \Gamma_{top}$ and that the width of $G_{se}[c_2]$ is odd. In this case the rightmost edge in σ_d points south, and by Lemma 6 H cannot end in Γ_{top}. This implies that c_e is not the endpoint of H, so condition (2) holds.

Condition (3). Assume that c_2 exists and the width of $G_{se}[c_2]$ is even. Note that $\Gamma[c_2] \neq \Gamma_{bot}$ due to the existence of $\Gamma[c]$. We prove by contradiction that the height of $G_{se}[x]$ is also even. Assume to the contrary that the height of $G_{se}[x]$ is odd. In this case the lowest edge in σ_x points west, and by Lemma 6 the end point of H lies in Γ_{bot}. This along with observation (c) above implies that $\overrightarrow{dd_n} \in H$ (otherwise H would end at c_2). Because $\overrightarrow{dd_n} \in H$ and the width of $G_{se}[c_2]$ is even (by the case statement), the rightmost edge in σ_d points north (see Figure 15a). By Lemma 6 the end

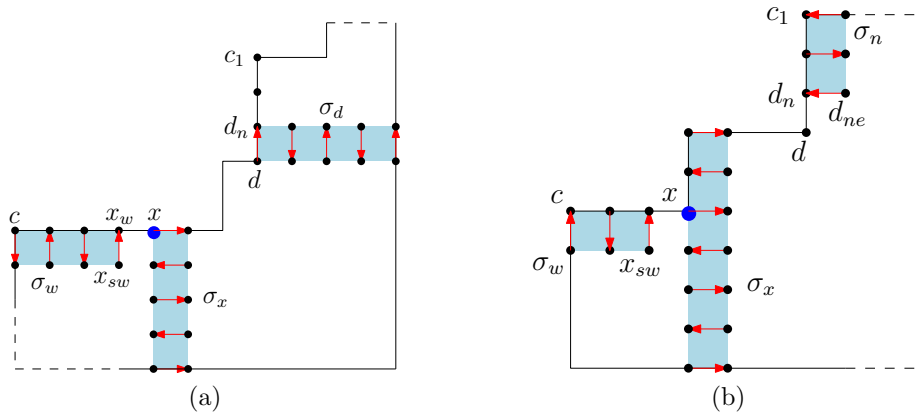


Figure 15: Theorem 12, only-if direction (a) $\overrightarrow{dd_n}$ exists (b) $\overrightarrow{dd_n}$ does not exist.

point of H lies in Γ_{top}, contradicting the fact (established above) that H ends in Γ_{bot}.

Condition (4). This proof that $|xc|$ is odd is identical to the proof for condition (4) in Theorem 11.

Condition (5). Assume that $|xc|$ is odd. By condition (4), $\Gamma[c] = \Gamma_{bot}$. Because $|xc|$ is odd, $\overrightarrow{c_s c} \in \sigma_w$, so H ends at c. If the height of $G_{se}[x]$ is even, then the lowest edge in σ_x points east, and by Lemma 6 H does not end in Γ_{bot}, a contradiction. It follows that $G_{se}[x]$ has odd height.

Condition (6). Assume that c_2 exists and $h(x, c_2) = 0$, meaning that x is the lower left corner of $\Gamma[c_2]$. Because the height of $\Gamma[c_2]$ is odd, the highest edge in σ_x is $\overrightarrow{c_{2e}c_2}$, so H ends at c_2. If the lowest edge in σ_x points west, then by Lemma 6 H ends in Γ_{bot}, a contradiction. So the lowest edge in σ_x points east, which by the definition of a zigzag sequence is possible only if the height of $G_{se}[x]$ is even. So condition (6) of the theorem holds. This completes the proof. □

10 Running Time Details from Section 5

We show that there is an $O(t)$ time algorithm for deciding whether a staircase grid graph G with t steps contains a squeeze-free Hamiltonian path that starts at a boundary vertex of ∂G. We assume G is represented as a sequence of pairs of numbers indicating the height and width of each step in order from left to right.

Theorems 10 through 12 list the conditions necessary and sufficient for the existence of a squeeze-free Hamiltonian path H that starts at a given vertex x on a horizontal staircase segment, and begins in the west, east or south direction. The cases with H beginning north from x, or with x on a vertical staircase segment, are symmetric. Therefore it suffices to show that we can determine in $O(t)$ time whether or not the conditions listed by Theorems 10 through 12 hold for at least one vertex x located on a horizontal staircase segment.

We begin by introducing two decision variables that will play a critical role in our decision procedure. For a fixed vertex $x \in \partial G$, define

$$v(x) = \begin{cases} 0 & \text{if } v(x, \Gamma_w) \text{ is even for each step } \Gamma_w \neq \Gamma_{bot} \\ & \text{of odd width lying west of } x \\ 1 & \text{if } v(x, \Gamma_w) \text{ is odd for each step } \Gamma_w \neq \Gamma_{bot} \\ & \text{of odd width lying west of } x \\ -1 & \text{otherwise} \end{cases}$$

Similarly, define

$$h(x) = \begin{cases} 0 & \text{if } h(x, \Gamma_e) \text{ is even for each step } \Gamma_e \neq \Gamma_{top} \\ & \text{of odd height lying east of } x \\ 1 & \text{if } h(x, \Gamma_e) \text{ is odd for each step } \Gamma_e \neq \Gamma_{top} \\ & \text{of odd height lying east of } x \\ -1 & \text{otherwise} \end{cases}$$

Observe that Theorems 10 through 12 are concerned with the parity of distances, not with the actual distances. We will keep track of these parities by taking all distances modulo 2 (with 0 representing even and 1 representing odd). In addition to the two variables defined above for all vertices x (including the special corner vertices c_1 from Theorem 11 and c_2 from Theorem 12), Theorems 10 through 12 employ the following decision variables: $h(x, \Gamma_{top})$; $v(x, \Gamma_{bot})$; parity of width and height of $G_{se}[x]$; $x \in \Gamma_{bot}$; parity of $|xc|$; existence and location of c_1 and c_2; height of $G_{se}[c_1]$; width of $G_{se}[c_2]$; and $\Gamma[c_2] = \Gamma_{top}$.

We begin by determining the values of these decision variables for the top left corner vertex of each step and then show that this is sufficient to determine if any vertex on a horizontal staircase segment satisfies the conditions of Theorems 10 through 12. Our method of computing these variables consists of two stages: a preprocessing stage, and an incremental update stage. The preprocessing stage initializes all variables corresponding to the top left corner of Γ_{bot}. In addition, it sets up some helper variables that will be used in the incremental update stage. In the incremental update stage, the top left corners of the steps are processed from left to right, and the variable values for the current corner are determined from the values of the previous corner in constant time.

Preprocessing

Let $\Gamma[x_1] = \Gamma_{bot}, \Gamma[x_2], \ldots, \Gamma[x_t] = \Gamma_{top}$ be the steps in order from left to right. Imagine a vertical line sweeping left-to-right across the steps, stopping at each corner vertex $x_1, x_2, \ldots, x_t$. Let o_1 be the top right corner of the first step of odd width encountered after Γ_{bot}. At each step $\Gamma[x_i]$ of odd width north of o_1, check the vertical distance from x_i to the previously visited step of odd width. If even, continue; if odd, let $p_1 = x_i$ and halt the sweeping process. Note that, for any vertex x at or above p_1, $v(x) = -1$; and for any vertex x above o_1 and strictly below p_1, $v(x)$ is either 0 or 1. For vertex o_1 and all vertices west of it, $v(x)$ is undefined because there is no step of odd width lying west of these vertices (except possibly Γ_{bot}).

Similarly, imagine a vertical line sweeping right-to-left across the steps. Let o_2 be the bottom left corner of the first step of odd height encountered after Γ_{top}. At each step $\Gamma[x_i]$ of odd height south of o_2, check the horizontal distance from x_i to the previously visited step of odd height. If even, continue; if odd, let $p_2 = x_i$ and halt the sweeping process. Note that, for any vertex x left of p_2, $h(x) = -1$; and for any vertex x at or to the right of p_2, $h(x)$ is either 0 or 1.

We create a list of all steps of odd height, to be used in determining the existence and position of c_2. We also determine the values for all decision variables corresponding to x_1. Note that every part of this preprocessing stage can be easily implemented in $O(t)$ time.

Incremental Update

Imagine a vertical line starting at x_2 and sweeping left-to-right across the steps, stopping at each top left corner vertex and initializing its decision variables. Let $width(\Gamma[x_i])$ and $height(\Gamma[x_i])$ be the width and height of step $\Gamma[x_i]$. At each corner x_i encountered by the sweep line, we initialize a selection of its decision variables as follows:

- $h(x_i)$: If o_2 does not exist, or if the sweep line has already passed o_2, this variable is undefined (since there are no steps of odd height east of x_i). If x_i is left of p_2, then $h(x_i) = -1$. If $x_i = p_2$, initialize $h(x_i) = 1$ (since the horizontal distance from p_2 to the closest step of odd height lying east of p_2 is odd, by the definition of p_2). If x_i is strictly right of p_2, update $h(x_i) = (h(x_{i-1}) + width(\Gamma[x_{i-1}])) \mod 2$.
- $v(x_i)$: If o_1 does not exist, or if the sweep line has not yet reached o_1, this variable is undefined (since there are no steps of odd width west of x). If the sweep line is at o_1, initialize $v(x_i) = height(x_i) \mod 2$. If the sweep line has passed p_i, $v(x_i) = -1$; otherwise, update $v(x_i) = (v(x_{i-1}) + height(\Gamma[x_i])) \mod 2$.
- $h(x_i, \Gamma_{top})$ and width of $G_{se}[x_i]$: set to the value of the variable for x_{i-1} incremented by $width(x_{i-1})$ (modulo 2).
- $v(x_i, \Gamma_{bot})$ and height of $G_{se}[x_i]$: set to the value of the variable for for x_{i-1} incremented by $height(x_i)$ (modulo 2).

Next we restart the sweeping process to update the remaining decision variables in O(1) time (per step). Note that testing if $\Gamma[x_i] = \Gamma_{bot}$ and $\Gamma[c_2] = \Gamma_{top}$ can be easily determined in constant time, and $|x_i c|$ is zero for all top left corner step vertices. The only decision variables left concern the existence and location of c_1 and c_2, which are initialized for each x_i in $O(1)$ time as follows:

- existence of c_1: true if o_1 exists and the sweep line has passed it, false otherwise. If x_i is the corner vertex of a step of odd width, update a temporary copy $c_1 = x_i$, to become permanent once a new corner vertex is encountered.
- existence of c_2: true if o_2 exists and the sweep line has not reached it yet, false otherwise. We maintain a pointer to the step $\Gamma[c_2]$ (and the associated width of $G_{se}[c_2]$) in the list of steps of odd height. If x_i coincides with c_2, advance the pointer.

Having determined c_1 and c_2, we can access in constant time the values $v(c_1)$, $v(c_1, \Gamma_{bot})$, height of $G_{se}[c_1]$, $h(c_2)$, $h(\Gamma_{top})$, and the width of $G_{se}[c_2]$, computed in the previous sweep stage.

Running Time

The preprocessing stage and incremental update stage for computing the values of the decision variables for each step's top left corner run in $O(t)$ time. Observe that for each decision variable, its value is either the same for all the vertices on a stair's horizontal top segment or its value alternates between 0 and 1 (as the distance of the vertex from the top left corner of the step alternates between even and odd). Therefore, for any step $\Gamma[c]$, the values of the decision variables for c are the same as the values for c_{ee}, and the values for c_e are the same as the values for c_{eee}, and so on... This means it is only necessary to check the conditions listed by Theorem 10 for vertices c_e and c_{ee}, because the variable values for all the other vertices on the step will be the same as for one of these two vertices. For Theorems 11 and 12, it is only necessary to check the conditions for vertices c and c_e. After computing the values of the variables for c using the algorithm described, the variable values for c_e, c_{ee} can easily be determined in constant time because, depending on the variable, they are either the same or the opposite value as that for c.

Thus, once the variable values are calculated for each step's top left corner, we can determine in constant time per step if there is any vertex on its top horizontal segment that satisfies the conditions of Theorems 10- 12. This gives us the result in Theorem 13 from Section 5.

Strongly Connected Spanning Subgraph for Almost Symmetric Networks

A. Karim Abu-Affash* Paz Carmi† Anat Parush Tzur‡

Abstract

In the *strongly connected spanning subgraph* ($SCSS$) problem, the goal is to find a minimum weight spanning subgraph of a strongly connected directed graph that maintains the strong connectivity. In this paper, we consider the $SCSS$ problem for two families of geometric directed graphs; *t-spanners* and *symmetric disk graphs*. Given a constant $t \geq 1$, a directed graph G is a t-spanner of a set of points V if, for every two points u and v in V, there exists a directed path from u to v in G of length at most $t \cdot |uv|$, where $|uv|$ is the Euclidean distance between u and v. Given a set V of points in the plane such that each point $u \in V$ has a radius r_u, the symmetric disk graph of V is a directed graph $G = (V, E)$, such that $E = \{(u, v) : |uv| \leq r_u \text{ and } |uv| \leq r_v\}$. Thus, if there exists a directed edge (u, v), then (v, u) exists as well.

We present $\frac{3}{4}(t+1)$ and $\frac{3}{2}$ approximation algorithms for the $SCSS$ problem for t-spanners and for symmetric disk graphs, respectively. Actually, our approach achieves a $\frac{3}{4}(t+1)$-approximation algorithm for all directed graphs satisfying the property that, for every two nodes u and v, the ratio between the shortest paths, from u to v and from v to u in the graph, is at most t.

1 Introduction

A directed graph is said to be *strongly connected* if it contains a directed path from every node to any other node. Given a directed graph $\overrightarrow{G}$, a *spanning subgraph* of $\overrightarrow{G}$ is a subgraph of $\overrightarrow{G}$ that contains all nodes of $\overrightarrow{G}$. In the *strongly connected spanning subgraph* ($SCSS$) problem, one has to find a minimum weight spanning subgraph of a strongly connected directed graph that maintains the strong connectivity. The $SCSS$ problem is a basic network design problem [6] and is known to be NP-hard [4, 8]. The NP-hardness can be shown by a simple reduction from the Hamiltonian cycle problem.

For unweighted directed graphs (i.e., all edges have weight 1), Khuller et al. [10,11] proposed a polynomial-time 1.61-approximation algorithm for the $SCSS$ problem. Later, Vetta [16] presented a polynomial-time approximation algorithm achieving an approximation ratio of 3/2. Zhao et al. [17] gave a linear-time 5/3-approximation algorithm. For weighted directed graphs, Frederickson and JáJá [6] studied the $SCSS$ problem and presented a linear-time algorithm achieving an approximation ratio of 2.

Given a set V of points in the plane such that each point $u \in V$ has a radius r_u, the *symmetric disk graph* of V is a directed graph $\overrightarrow{G} = (V, \overrightarrow{E})$, such that $\overrightarrow{E} = \{(u, v) : |uv| \leq r_u \text{ and } |uv| \leq r_v\}$, where $|uv|$ is the Euclidean distance between u and v. The weight of an edge $(u, v) \in \overrightarrow{E}$ (denoted by $wt(u, v)$) is some polynomial function on $|uv|$. This weight function is typically used in wireless networks, where $wt(u, v) = |uv|^\alpha$, for $1 \leq \alpha \leq 5$.

Given a set V of points in the plane and a constant $t \geq 1$, a directed graph $\overrightarrow{G}$ is a (geometric) *t-spanner* of V if, for every two points u and v in V, there exists a directed path from u to v in $\overrightarrow{G}$ of length at most $t \cdot |uv|$.

In this paper, we focus on the $SCSS$ problem for symmetric disk graphs and t-spanners. We present a $\frac{3}{2}$-approximation algorithm for the $SCSS$ problem for symmetric disk graphs. Then, we extend this algorithm to obtain a $\frac{3}{4}(t+1)$-approximation algorithm for the $SCSS$ problem for t-spanners. Our approximation algorithms are based on Christofides' algorithm for the traveling salesman (TSP) problem.

Actually, our approach provides a $\frac{3}{4}(t+1)$-approximation algorithm for the $SCSS$ problem for an extended family of directed graphs, which is called *t-symmetric*. For a weighted directed graph $\overrightarrow{G}$, let $\delta_{\overrightarrow{G}}(u, v)$ denote a minimum weight path from u to v in $\overrightarrow{G}$. A weighted directed graph $\overrightarrow{G}$ is called a *t-symmetric* directed graph, for a given constant $t \geq 1$, if, for each pair of nodes u and v in $\overrightarrow{G}$, the weight of $\delta_{\overrightarrow{G}}(u, v)$ is at most t times the weight of $\delta_{\overrightarrow{G}}(v, u)$. Given a t-symmetric directed graph $\overrightarrow{G}$ that is strongly connected, the goal is to find a minimum weight strongly connected spanning subgraph of $\overrightarrow{G}$.

The TSP is defined as follows. Given a weighted complete graph on n nodes, the goal is to find a tour, i.e., a

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

†Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel, carmip@cs.bgu.ac.il. The research is partially supported by the Lynn and William Frankel Center for Computer Science and by grant 680/11 from the Israel Science Foundation (ISF).

‡Department of Computer Science, Ben-Gurion University, Beer-Sheva 84105, Israel, parusha@cs.bgu.ac.il.

simple cycle spanning all the nodes, of minimum weight. Shani and Gonzalez [14] proved that the TSP problem is NP-Complete. In the metric TSP, the weight function of the edge set forms a metric, i.e., the weight function satisfies the triangle inequality; despite this restriction the problem remains NP-hard. A 2-approximation algorithm based on utilizing a minimum spanning tree was proposed in [13]. Christofides [1] improved the algorithm by also utilizing a minimum weight perfect matching, and achieved a 3/2-approximation algorithm.

A connected graph $G = (V, E)$ is called *k-edge-connected* if, for each subset $E' \subseteq E$ of size at most $k-1$, the graph $G' = (V, E \setminus E')$ is also connected. In the *k-edge-connectivity* problem, the goal is to find a minimum weight spanning subgraph of G that is k-edge-connected. The k-edge-connectivity problem has applications in network reliability (besides its theoretical interest), since it ensures that even when $k-1$ links fail , the network remains connected.

The 2-edge-connectivity problem is known to be MAX-SNP-hard [2, 5], as is the unweighted version in which the objective is to minimize the number of edges of the subset. For unweighted graphs, Vempala and Vetta [15] presented a 4/3-approximation algorithm for the 2-edge-connectivity problem. Jothi et al. [9] improved this result by describing a 5/4-approximation algorithm for the 2-edge-connectivity problem. The 3-approximation algorithm for the 2-edge-connectivity problem in weighted graphs that follows from the approximation algorithm of Frederickson and JáJá [6] for the bridge augmenting connectivity problem, was afterwards improved to 2 by Khuller and Vishkin [12]. For weighted complete graphs whose cost function satisfies the triangle inequality, Frederickson and JáJá [7] presented 3/2-approximation algorithm for the 2-edge-connectivity problem. For complete Euclidean graphs in $\mathbb{R}^d$ this problem admits a PTAS [3].

At first glance, the $SCSS$ problem in symmetric disk graphs looks equivalent to the 2-edge-connectivity problem in undirected graphs, since any solution for the 2-edge-connectivity problem is also a solution for the $SCSS$ problem. However, the weight of an optimal solution for the 2-edge-connectivity problem can be $\Omega(n^{\alpha-1})$ times the weight of an optimal solution for the $SCSS$ problem, where the weight of an edge (u, v) is $|uv|^\alpha$ and $\alpha \geq 1$., as illustrated in Figure 1.

The rest of this paper is organized as follows. In Section 2, we give a $\frac{3}{2}$-approximation algorithm for the $SCSS$ problem in symmetric disk graphs. Then, in Section 3, we extend this algorithm to obtain a $\frac{3}{4}(t+1)$-approximation algorithm for the $SCSS$ problem in t-spanners.

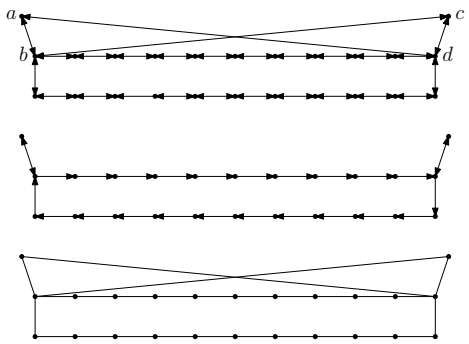


Figure 1: Top, a symmetric disk graph H of n nodes, where all nodes have radius 1 except nodes a, b, c and d that have radius $(n-2)/2$. Middle, an optimal solution for the $SCSS$ problem in H of weight $n+2$. Bottom, the unique solution to the 2-edge-connectivity problem for the undirected version of H, of weight $n + 2(\frac{n-2}{2})^\alpha$, where $wt(u, v) = |uv|^\alpha$.

2 The $SCSS$ problem in symmetric disk graphs

Given a strongly connected symmetric disk graph $\vec{G}=(V, \vec{E})$, in the $SCSS$ problem, the goal is to find a minimum weight set $R^* \subseteq \vec{E}$, such that $G_{R^*} = (V, R^*)$ is strongly connected. Let OPT denote the weight of R^*, i.e., the total weight of the edges in R^*. In this section, we present an algorithm that computes a set $R \subseteq \vec{E}$, such that the graph $G_R = (V, R)$ is strongly connected and the weight of R is at most $\frac{3}{2} \cdot OPT$.

A pair of nodes u and v in a strongly connected graph $\vec{G}$ is called a *cut pair* if the edges (u, v) and (v, u) are in G and their removal separates $\vec{G}$ into two subgraphs. Thus, if $\vec{G}$ contains a cut pair, then this pair separates the $SCSS$ problem into two independent $SCSS$ subproblems that can be approximated by the proposed algorithm. Moreover, cut pairs must be in any feasible solution for the $SCSS$ problem, and, in particular, in any optimal solution. Therefore, from now on, we assume that no cut pairs exist in $\vec{G}$.

Let $\delta_{\vec{G}}(u, v)$ denote a minimum weight path from u to v in $\vec{G}$, and let $wt(\delta_{\vec{G}}(u, v))$ denote the weight of $\delta_{\vec{G}}(u, v)$. The Shortest Paths Graph of $\vec{G}$ (denoted by $SPG(\vec{G})$), is an undirected complete graph over V, where the weight of an edge $\{u, v\}$ is $wt(\delta_{\vec{G}}(u, v))$. Notice that, since $\vec{G}$ is a symmetric disk graph, $wt(\delta_{\vec{G}}(u, v)) = wt(\delta_{\vec{G}}(v, u))$, and therefore, the weight function of the $SPG(\vec{G})$ is well defined, and it forms a metric.

Our algorithm applies the well known Christofides' algorithm (for the TSP problem) on $SPG(\vec{G})$.

Christofides' algorithm finds two edge sets, a minimum spanning tree of $SPG(\overrightarrow{G})$ and a minimum weight perfect matching in the complete graph over the nodes of odd degree in the minimum spanning tree. The graph that consists of these two edge sets is connected and all its nodes are of even degree, therefore, it contains an Eulerian cycle. Due to the triangle inequality, the Eulerian cycle can be relaxed into a Hamiltonian cycle (by "shortcutting" whenever a node is revisited) without increasing its weight. It has been shown that the approximation ratio of this algorithm is 3/2 [1].

Given a strongly connected symmetric disk graph $\overrightarrow{G}= (V, \overrightarrow{E})$, in Algorithm 1, we describe how to compute a set $R \subseteq \overrightarrow{E}$, such that $G_R = (V, R)$ is strongly connected. Then, in Section 2.1 we bound the weight of R with respect to OPT.

Algorithm 1

1: construct $SPG(\overrightarrow{G})$
2: compute a solution T for the TSP in $SPG(\overrightarrow{G})$ using Christofides' algorithm
3: direct T arbitrarily and denote this directed tour by $\overrightarrow{T}$
4: $R \leftarrow \emptyset$
5: **for** each edge $(u, v) \in \overrightarrow{T}$ **do**
6: $\quad R \leftarrow R \cup \delta_{\overrightarrow{G}}(u, v)$
7: **return** R

It is not hard to see that the running time of Algorithm 1 is polynomial ($O(n^3)$), and the resulting graph $G_R = (V, R)$ is strongly connected.

2.1 Approximation ratio

Let R^* be an optimal solution for the $SCSS$ problem in $\overrightarrow{G}= (V, \overrightarrow{E})$, let OPT denote the weight of R^*, and let R be the set obtained by Algorithm 1. In this section, we prove that the weight of R (i.e., $wt(R)$) is at most $\frac{3}{2} \cdot OPT$. Let $\overline{G}_{R^*}$ be the undirected graph of $G_{R^*} = (V, R^*)$, that is, $\overline{G}_{R^*}$ contains an undirected edge between nodes u and v if either $(u, v) \in R^*$ or $(v, u) \in R^*$.

Lemma 1 *If all the nodes in $\overline{G}_{R^*}$ are of even degree, then $wt(R) \leq \frac{3}{2} \cdot OPT$.*

Proof. Each edge (u, v) in $\overrightarrow{T}$ (the directed tour that is constructed during Algorithm 1) contributes to R a set $\delta_{\overrightarrow{G}}(u, v)$ of edges that compose a minimum weight path from u to v in $\overrightarrow{G}$. The weight of $\delta_{\overrightarrow{G}}(u, v)$ is equal to the weight of the edge (u, v) in $\overrightarrow{T}$. Notice that we might add to R edges that are already in R. As a result, $wt(R) \leq wt(\overrightarrow{T})$. Let T^* denote an optimal solution for the TSP in $SPG(\overrightarrow{G})$. Then, by the bound of Christofides' algorithm, $wt(\overrightarrow{T}) \leq \frac{3}{2} \cdot wt(T^*)$. Finally, $\overline{G}_{R^*}$ contains an Eulerian cycle C (since all nodes are of even degree) that yields a solution for the TSP in $SPG(\overrightarrow{G})$. Therefore, OPT is an upper bound on the weight of the edge set of T^*, i.e., $wt(T^*) \leq OPT$. Therefore, we have

$$wt(R) \leq wt(\overrightarrow{T}) \leq \tfrac{3}{2} \cdot wt(T^*) \leq \tfrac{3}{2} \cdot OPT\,. \qquad \square$$

In general, the inequality $wt(T^*) \leq OPT$ does not hold without the restriction of even degree on the nodes in $\overline{G}_{R^*}$. To see this, consider the example in Figure 2. The weight of any optimal solution T^* for the TSP in $SPG(\overrightarrow{G})$ is of weight $(4 \cdot OPT - 10)/3$. Thus, $wt(T^*) \geq (4/3 - \epsilon)OPT$, for any $\epsilon > 0$.

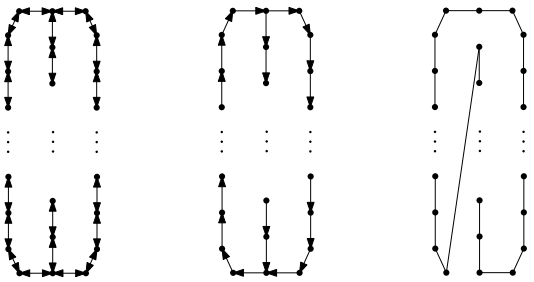


Figure 2: Left, a symmetric disk graph $\overrightarrow{G}$ on n nodes, in which the weight of each edge is 1. Middle, an optimal solution for the $SCSS$ problem in $\overrightarrow{G}$ of weight $n + 1$. Right, an optimal solution T^* for the TSP in $SPG(\overrightarrow{G})$ of weight $\frac{4}{3}n - 2$.

Lemma 2 *Let $G_{\Delta\leq 3} = (V, E)$ be a 2-edge-connected undirected graph whose maximum degree is 3. Then, $G_{\Delta\leq 3}$ contains a path composed of edges $E_p = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)\}$, such that $v_1 \neq v_k$, v_1 and v_k are of degree 3, each node in $V_p = \{v_2, \ldots, v_{k-1}\}$ is of degree 2, and $(V \setminus V_p, E \setminus E_p)$ is 2-edge-connected. We call such a path a chord.*

Proof. We show the existence of such a chord using a constructive method. In each iteration i, we maintain a 2-edge-connected component C_i and extend C_i via an unexplored node $v^* \in C_i$ of degree 3. Initially, $i = 0$, C_i is a cycle, and $v^* \in C_i$ is a node of degree 3. Let P_i be a path connecting v^* to a node $u \in C_i$ that is edge disjoint from C_i (such a path exists since otherwise $G_{\Delta\leq 3}$ is not 2-edge-connected). If the inner nodes of P_i are of degree 2 then P_i is a chord, and we are done. Otherwise, P_i contains a node w of degree 3. Let $C_{i+1} = C_i \cup P_i$ and set v^* to be w. Repeat this procedure until a chord is

found. This procedure halts, since in each iteration a new node v^* of degree 3 is explored. □

Lemma 3 *Let P be a simple path composed of vertices $V_p = \{v_1, v_2, \ldots, v_k\}$ and edges $E_p = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)\}$. There exists a perfect matching M_p in P of the nodes in V_p except for at most the two end-vertices v_1 and v_k, i.e., $V_p \setminus W$, where $W \subseteq \{v_1, v_k\}$, such that the weight of M_p is at most half of the weight of P.*

Proof. The correctness follows from the pigeonhole principle for both cases of the parity of k.

- If k is odd, then one of the two matchings $\{(v_1, v_2), (v_3, v_4), \ldots, (v_{k-2}, v_{k-1})\}$ or $\{(v_2, v_3), (v_4, v_5), \ldots, (v_{k-1}, v_k)\}$ is at most half of the weight of P.
- If k is even, then one of the two matchings $\{(v_1, v_2), (v_3, v_4), \ldots, (v_{k-1}, v_k)\}$ or $\{(v_2, v_3), (v_4, v_5), \ldots, (v_{k-2}, v_{k-1})\}$ is at most half of the weight of P. □

Lemma 3 yields the following corollary.

Corollary 4 *Let P, E_p, and V_p be as in Lemma 3, and let $V'_p \subseteq V_p$. There exists a perfect matching M'_p of the nodes in $V'_p \cup \{v_1, v_k\} \setminus W$, where $W \subseteq \{v_1, v_k\}$, such that the weight of M'_p is at most half of the weight of P, where the weight of an edge $\{v_i, v_j\}$ in M'_p is the weight of the subpath between v_i and v_j in P.*

Let $\mathcal{T}$ be the minimum spanning tree of $SPG(\vec{G})$ that is found during Christofides' algorithm. Let V_{odd} be the set of nodes of odd degree in $\mathcal{T}$, let $G_{odd} = (V_{odd}, E_{odd})$ be the (complete) subgraph of $SPG(\vec{G})$ induced by V_{odd} (E_{odd} is the set of all edges of $SPG(\vec{G})$ having both end-vertices in V_{odd}), and let $\mathcal{M}$ denote a minimum weight perfect matching of G_{odd}. Recall that R^* is an optimal solution for the $SCSS$ problem in $\vec{G}$ of weight OPT. In the following, we bound the weights of $\mathcal{T}$ and $\mathcal{M}$ with respect to OPT.

Lemma 5 $wt(\mathcal{T}) \leq OPT$.

Proof. Since the graph $G_{R^*} = (V, R^*)$ is a spanning subgraph of $\vec{G}$ that is strongly connected, the undirected graph $\overline{G}_{R^*}$ of G_{R^*} contains a spanning tree T of weight at most OPT. Let $\{u, v\}$ be an edge in T such that, w.l.o.g., it is the undirected edge of (u, v) in G_{R^*}. Since $SPG(\vec{G})$ is a complete graph over V, it also contains T, and the weight of the edge $\{u, v\}$ in $SPG(\vec{G})$ is equal to the weight of a minimum weight path from u to v in $\vec{G}$. Thus, the weight of $\{u, v\}$ in $SPG(\vec{G})$ is equal to the weight of (u, v) in G_{R^*}. Therefore,

$$wt(\mathcal{T}) \leq wt(T) \leq OPT.$$ □

Lemma 6 $wt(\mathcal{M}) \leq OPT/2$.

Proof. Let $G' = (V', R')$, where $V' \subseteq V$ and $R' \subseteq R^*$, be the minimum weight subgraph of $G_{R^*} = (V, R^*)$, in which the nodes of V_{odd} are strongly connected. Clearly, $wt(R') \leq OPT$. We first show that G' can be converted to a graph $G'_{\Delta\leq 3}$ (i.e., a 2-edge connected undirected graph with degree at most 3) whose weight is equal to $wt(G')$, and thus, $wt(G'_{\Delta\leq 3}) \leq OPT$. Then, we show that there exists a perfect matching M' of V_{odd} in $G'_{\Delta\leq 3}$, such that $wt(M') \leq \frac{1}{2} \cdot wt(G'_{\Delta\leq 3})$.

Let $\overline{G'}$ be the undirected graph of G', such that $\overline{G'}$ contains an undirected edge $\{u, v\}$ between nodes u and v if either $(u, v) \in R'$ or $(v, u) \in R'$, and $wt(\{u, v\}) = wt(u, v)$. If both (u, v) and (v, u) are in R', then $\overline{G'}$ contains two undirected edges $\{u, v\}$ and $\{u, v\}'$ between u and v, each of weight $wt(u, v)$. Notice that $\overline{G'}$ is a 2-edge connected undirected graph with the same weight as G', and the minimum degree of each node in $\overline{G'}$ is 2. Moreover, if $\overline{G'}$ contains two edges $\{u, v\}$ and $\{u, v\}'$, then the nodes u and v are a cut pair in G'. We show how to convert $\overline{G'}$ to $G'_{\Delta\leq 3}$. First, while there exists a node u of degree greater than 3 in $\overline{G'}$ that is incident to two edges $\{u, v\}, \{u, v\}'$, select an adjacent node $w \neq v$ of u in $\overline{G'}$. Add the edge $\{v, w\}$ of weight $wt(\{u, v\}) + wt(\{u, w\})$ to $\overline{G'}$, and remove the edge $\{u, w\}$ and $\{u, v\}'$ from $\overline{G'}$. At this stage, $\overline{G'}$ does not contain any cut pairs, i.e., if the number of vertices in $\overline{G'}$ is greater than two, then there is no node u in $\overline{G'}$ that is incident to two edges $\{u, v\}, \{u, v\}'$.

Next, while there exists a node u of degree greater than 3 in $\overline{G'}$, select an adjacent node w of u in $\overline{G'}$. Since $\overline{G'}$ is 2-edge connected undirected graph, there is a path $P_{wu} = (w, \ldots, w', u)$ in $\overline{G'}$ from w to u which is different from the edge $\{w, u\}$. Notice that w' is the last node before u in this path P_{wu}, and let $v \notin \{w', w\}$ be a node that is adjacent to u in $\overline{G'}$. Add the edge $\{v, w\}$ of weight $wt(\{u, v\}) + wt(\{u, w\})$ to $\overline{G'}$, and remove the edges $\{u, v\}$, $\{u, w\}$ from $\overline{G'}$.

The obtained graph is 2-edge connected undirected graph. Since, in each iteration, the degree of one node is reduced (by two), and the degree of the other nodes remains the same, this routine ends. Moreover, for each edge that is added to the graph, two edges with equal total weight are removed and, thus, the weight of the graph $\overline{G'}$ is preserved. At the end of this routine, set $G'_{\Delta\leq 3}$ to be $\overline{G'}$.

We now show that there exists a perfect matching M' of V_{odd} in $G'_{\Delta\leq 3}$, such that (i) each edge in M' corresponds to a path in $G'_{\Delta\leq 3}$; (ii) the weight of each edge $e \in M'$ is equal to the weight of the corresponding path of e in $G'_{\Delta\leq 3}$; and (iii) $wt(M') \leq \frac{1}{2} \cdot wt(G'_{\Delta\leq 3})$.

The existence of such a matching M' is shown in Procedure 2. In each iteration (Lines 2–19), the number of nodes of degree 3 in G_{temp} is reduced by 2, thus, this while loop ends, and at Line 20 the resulting graph is

Procedure 2 Constructing a matching M'

1: $M' \leftarrow \emptyset$, $G_{temp} \leftarrow G'_{\Delta\leq 3}$
2: **while** there is a node v in G_{temp} of degree 3 **do**
3: let $P = (V_P, E_P)$ be a chord in G_{temp}
/* *Such P exists by Lemma 2* */
4: let U be the set of the two endvertices of P
5: let $V'_P \leftarrow V_{odd} \cap V_P$
6: let M_{chord} be a perfect matching in P of the nodes in $V'_P \cup U \setminus W$, where $W \subseteq U$, such that $wt(M_{chord}) \leq \frac{1}{2}wt(P)$
/* *Such M_{chord} exists by Corollary 4* */
7: $M' \leftarrow M' \cup \{\{v_i, v_j\} | v_i, v_j \in V'_P\}$
8: $G_{temp} \leftarrow G_{temp} \setminus (P \setminus U)$
/* *Remove all inner nodes of P and their incident edges from G_{temp}* */
9: **for** each $v \in U$ such that $\{v_i, v\} \in M_{chord}$ **do**
10: let p and q be the two nodes adjacent to v that are not in P
11: **if** $v \in V'_P$ **then**
12: add the edge $\{p, q\}$ to G_{temp}
13: set $wt(\{p, q\})$ to be $wt(\{p, v\}) + wt(\{v, q\})$
14: remove v and its incident edges from G_{temp}
15: **else**
16: let $P_{v_i,v}$ be the path from v_i to v in P that corresponds to the edge $\{v_i, v\}$
17: replace v by v_i in G_{temp},
18: set $wt(\{p, v_i\})$ to be $wt(\{p, v\})$
19: set $wt(\{q, v_i\})$ to be $wt(\{q, v\})$
20: let M_c be a perfect matching in G_{temp} of the nodes in V'_P, such that $wt(M_c) \leq \frac{1}{2}wt(G_{temp})$
/* *At this stage, G_{temp} is a cycle* */
21: $M' \leftarrow M' \cup M_c$
22: **return** M'

a 2-edge-connected graph with nodes of degree 2, i.e., a cycle C. The number of nodes in V_{odd} is even, and while removing a chord from G_{temp}, an even number of nodes from V_{odd} are removed. Therefore, C contains an even number of nodes from V_{odd}.

In the following we bound the weight of M' obtained by Procedure 2. The weight of the matching found at Line 6 is at most $\frac{1}{2} \cdot wt(P)$. Thus, at Line 7, we add to M' at most half of the weight of the path P. Then, at Line 8, the edges of P are removed from G_{temp}, and these edges are not charged again. Clearly, the same bound holds for the matching that is found at Line 20. Thus, the weight of M' is bounded by half of the weight of the edge set of $G'_{\Delta\leq 3}$, i.e., $wt(M') \leq \frac{1}{2} \cdot wt(G'_{\Delta\leq 3})$.

Consider a node $v \notin V_{odd}$ such that $\{v_i, v\} \in M_{chord}$ in some iteration j of the while loop. Notice that the weight $wt(\{v_i, v\})$ is charged in this iteration, even though the edge $\{v_i, v\}$ is not added to M'. This is done to compensate that later, in some iteration $j' > j$, the node v_i is matched to some node $v_l \in V_{odd}$, and the weight $wt(\{v_i, v_l\})$ corresponds to the weight $wt(\{v, v_l\})$ (see Lines 18 or 19). Therefore, the weight $wt(\{v_i, v_l\})$ might not include the weight $wt(\{v_i, v\})$. However, as mentioned, this does not affect the bound on the weight of the matching M', since the weight $wt(\{v_i, v\})$ has already been charged in the iteration j.

In order to prove the lemma, we generate a perfect matching M^* in G_{odd} based on M'. For each edge $\{v_i, v_j\} \in M'$, we add to M^* the edge $\{v_i, v_j\}$ of G_{odd}. Each edge $\{v_i, v_j\}$ has a corresponding path from v_i to v_j in $G_{\Delta\leq 3}$, i.e., an equivalent (in weight) path from v_i to v_j in $\overrightarrow{G}$, and, therefore, the weight of the edge $\{v_i, v_j\}$ in M' is an upper bound on the weight of the edge $\{v_i, v_j\}$ in G_{odd}, so, $wt(M^*) \leq wt(M')$. Recall that $wt(G'_{\Delta\leq 3}) \leq OPT$. To sum up, we found a perfect matching in G_{odd} of weight at most half of the weight of R^*. Clearly, the weight of the perfect matching found is an upper bound on the weight of a minimum one, $\mathcal{M}$. Thus, we have $wt(\mathcal{M}) \leq wt(M^*) \leq wt(M') \leq \frac{1}{2} \cdot wt(G'_{\Delta\leq 3}) \leq \frac{1}{2} \cdot OPT$. □

Theorem 7 *Algorithm 1 is a $\frac{3}{2}$-approximation algorithm for the SCSS problem in symmetric disk graphs.*

Proof. $wt(R) \leq wt(\overrightarrow{T}) \leq wt(\mathcal{T}) + wt(\mathcal{M}) \leq \frac{3}{2} \cdot OPT$, where the first inequality is already noted in the proof of Lemma 1, the second inequality follows immediately from the description of Christofides' algorithm, and the last inequality holds due to Lemma 5 and Lemma 6. □

3 The $SCSS$ problem in t-spanners

Given a set V of points in the plane and a constant $t \geq 1$, a directed graph $\overrightarrow{G}$ is a *t-spanner* of V if, for every two points u and v in V, there exists a directed path from u to v in $\overrightarrow{G}$ of length at most $t \cdot |uv|$. In this section, we generalize Theorem 7 for t-spanners. The Shortest Paths Graph of a t-spanner $\overrightarrow{G}$ of V (denoted by $SPG(\overrightarrow{G})$), is an undirected complete graph over V, in which the weight of an edge $\{u, v\}$ equals to $\min\{wt(\delta_{\overrightarrow{G}}(u, v)), wt(\delta_{\overrightarrow{G}}(v, u))\}$, where $\delta_{\overrightarrow{G}}(u, v)$ is a minimum weight path from u to v in $\overrightarrow{G}$.

Theorem 8 *Algorithm 3 is a $\frac{3}{4} \cdot (t+1)$-approximation algorithm for the SCSS problem in t-spanners.*

Proof. Let E_t be the tour computed during Algorithm 3. Consider an edge $\{u, v\} \in E_t$ of weight $\min\{wt(\delta_{\overrightarrow{G}}(u, v)), wt(\delta_{\overrightarrow{G}}(v, u))\}$, and assume, w.l.o.g., that $wt(\{u, v\}) = \delta_{\overrightarrow{G}}(u, v)$. Since the graph $\overrightarrow{G}$ is t-spanner,

$$\begin{aligned} wt(\delta_{\overrightarrow{G}}(u, v)) + wt(\delta_{\overrightarrow{G}}(v, u)) &\leq wt(\delta_{\overrightarrow{G}}(u, v)) + t \cdot |uv| \\ &\leq wt(\delta_{\overrightarrow{G}}(u, v)) + t \cdot wt(\delta_{\overrightarrow{G}}(u, v)) \\ &= (t+1) \cdot wt(\delta_{\overrightarrow{G}}(u, v)) \\ &= (t+1) \cdot wt(\{u, v\}). \end{aligned}$$

Algorithm 3

1: construct $SPG(\overrightarrow{G})$ of $\overrightarrow{G}$
2: compute an Eulerian tour E_t using Christofides' algorithm (the tour before the shortcuts)
3: let $\overrightarrow{E_t}$ be a directed tour obtained by traversing the Eulerian tour E_t arbitrary
4: let $\overleftarrow{E_t}$ denote the opposite directed tour of $\overrightarrow{E_t}$
5: $\overrightarrow{R} \leftarrow \emptyset$, $\overleftarrow{R} \leftarrow \emptyset$
6: traverse the edges of $\overrightarrow{E_t}$, (resp. $\overleftarrow{E_t}$) and, for each edge (u, v) visited during the traversal, add the set of directed edges $\delta_{\overrightarrow{G}}(u, v)$ to $\overrightarrow{R}$ (resp. $\overleftarrow{R}$)
7: **if** $wt(\overrightarrow{R}) \leq wt(\overleftarrow{R})$ **then**
8: **return** $\overrightarrow{R}$
9: **else**
10: **return** $\overleftarrow{R}$

We now bound the the output of Algorithm 3.

$$\begin{aligned}
&\min\{wt(\overrightarrow{R}), wt(\overleftarrow{R})\} \\
&\quad \leq \frac{1}{2} \cdot \Big(wt(\overrightarrow{R}) + wt(\overleftarrow{R}) \Big) \\
&\quad \leq \frac{1}{2} \cdot \sum_{\{u,v\} \in E_t} \Big(wt(\delta_{\overrightarrow{G}}(u, v)) + wt(\delta_{\overrightarrow{G}}(v, u)) \Big) \\
&\quad \leq \frac{1}{2} \cdot \sum_{\{u,v\} \in E_t} (t+1) \cdot wt(\{u, v\}) \\
&\quad = \frac{1}{2} \cdot (t+1) \cdot wt(E_t) \\
&\quad \leq \frac{3}{4} \cdot (t+1) \cdot OPT,
\end{aligned}$$

where the later inequality follows from Theorem 7. □

Corollary 9 *Algorithm 3 is a $\frac{3}{4} \cdot (t+1)$-approximation algorithm for the SCSS problem in any graph $\overrightarrow{G}$, where the weight of $\delta_{\overrightarrow{G}}(u, v)$ is at most t times the weight of $\delta_{\overrightarrow{G}}(v, u)$, for each pair of nodes u and v in $\overrightarrow{G}$.*

References

[1] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. *Technical Report 388, Graduate School of Industrial Administration, Carnegie-Mellon University*, 1976.

[2] B. Csaba, M. Karpinski, and P. Krysta. Approximability of dense and sparse instances of minimum 2-connectivity, TSP and path problems. In *SODA*, pages 74–83, 2002.

[3] A. Czumaj and A. Lingas. On approximability of the minimum-cost k-connected spanning subgraph problem. In *SODA*, pages 281–290, 1999.

[4] K. P. Eswaran and R. E. Tarjan. Augmentation problems. *SIAM J. Comput.*, 5(4):653–665, 1976.

[5] C. G. Fernandes. A better approximation ratio for the minimum size k-edge-connected spanning subgraph problem. *J. Algorithms*, 28(1):105–124, 1998.

[6] G. N. Frederickson and J. JáJá. Approximation algorithms for several graph augmentation problems. *SIAM J. Comput.*, 10(2):270–283, 1981.

[7] G. N. Frederickson and J. JáJá. On the relationship between the biconnectivity augmentation and travelling salesman problems. *Theoretical Computer Science*, 19(2):189 – 201, 1982.

[8] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[9] R. Jothi, B. Raghavachari, and S. Varadarajan. A 5/4-approximation algorithm for minimum 2-edge-connectivity. In *SODA*, pages 725–734, 2003.

[10] S. Khuller, B. Raghavachari, and N. Young. Approximating the minimum equivalent directed graph. *SIAM J. Comput.*, 24(4):859–872, 1995.

[11] S. Khuller, B. Raghavachari, and N. E. Young. On strongly connected directed graphs with bounded cycle length. *Discrete Applied Mathematics*, 69(3):281–289, 1996.

[12] S. Khuller and U. Vishkin. Biconnectivity approximations and graph carvings. *J. ACM*, 41(2):214–235, 1994.

[13] D. J. Rosenkrantz, R. E. Stearns, and P. M. Lewis II. An analysis of several heuristics for the traveling salesman problem. *SIAM J. Comput.*, 6(3):563–581, 1977.

[14] S. Sahni and T. Gonzalez. P-complete approximation problems. *J. ACM*, 23(3):555–565, 1976.

[15] S. Vempala and A. Vetta. Factor 4/3 approximations for minimum 2-connected subgraphs. In *APPROX*, pages 262–273, 2000.

[16] A. Vetta. Approximating the minimum strongly connected subgraph via a matching lower bound. In *SODA*, pages 417–426, 2001.

[17] L. Zhao, H. Nagamochi, and T. Ibaraki. A linear time 5/3-approximation for the minimum strongly-connected spanning subgraph problem. *Inf. Process. Lett.*, 86(2):63–70, 2003.

A Faster 4-Approximation Algorithm for the Unit Disk Cover Problem

Ahmad Biniaz* Paul Liu† Anil Maheshwari* Michiel Smid*

Abstract

Given a set P of n points in the plane, we consider the problem of covering P with a minimum number of unit disks. This problem is known to be NP-hard. We present a simple 4-approximation algorithm for this problem which runs in $O(n \log n)$-time and uses the plane-sweep technique. Previous algorithms that achieve the same approximation ratio have a higher time complexity. We also show how to extend this algorithm to other metrics, and to three dimensions.

1 Introduction

In this paper we consider the *unit disk cover* (UDC) problem. Given a set P of n points in the plane, the UDC problem asks for the minimum number of disks of prescribed radius r (or simply unit disks of radius 1), which cover all points of P. Unless otherwise specified, we assume that the disks are in the L_2-norm. This problem is motivated by VLSI design, facility location, and motion planning.

The UDC problem is known to be NP-hard in the L_1, L_2, and L_∞ norms [7]. For points in $\mathbb{R}^d$ and any integer $l \geq 1$, it is possible to approximate the UDC problem in the L_2-norm within a factor of $\left(1+\frac{1}{l}\right)^d$ with running time $(dl)^{O(d)} n^{O((dl)^d)}$ [11] and within a factor of $2\left(1+\frac{1}{l}\right)^{d-1}$ with running time $(dl)^{O(d)} n^{O(d^d)}$ [10]. For points under the L_1 and L_∞ norms, similar ideas lead to a $\left(1+\frac{1}{l}\right)^d$ approximation algorithm with running time $l^d n^{2l^d+1}$ [11] and a $\left(1+\frac{1}{l}\right)^{d-1}$ approximation algorithm with running time $dl^{O(d-1)} n^{O(dl^{d-1})}$ [10]. However, these algorithms are mainly of theoretical interest, and are impractical for large data sets.

Gonzalez [10] presented a 2-approximation algorithm for the UDC problem in the L_1 and L_∞ norms and an 8-approximation in the L_2-norm. These algorithms run in $O(n \log S)$-time, where $S \leq n$ is the number of disks in an optimal solution. A constant approximation algorithm running in $O(n^3 \log n)$-time is also presented in [4]. The algorithm uses the fact that the UDC problem is equivalent to a set cover in a range space of finite VC dimension. However, no efforts were made to optimize or determine the exact value of the approximation factor. By constraining the disk centers to lie on a grid, Franceschetti et al. [8] developed, for any $l \geq 1$, an $O(Kn)$ time algorithm with approximation factor $3(1+\frac{1}{l})^2$, where K is a function of l and the size of the approximation grid. A 2.8334-approximation algorithm which runs in $O(n(\log n \log\log n)^2)$-time is presented in [9]. We note that this algorithm is quite difficult to implement, and has a high constant factor in the running time. Using a different approach of dividing the input into vertical strips, Liu and Lu [12] presented a $\frac{25}{6}$-approximation algorithm for this problem running in $O(n \log n)$ time. A listing of all the algorithms as well as their approximation factors is given in Table 1.

Reference	Approximation	Running Time
[10]	$2\left(1+\frac{1}{l}\right)$	$O(l^2 n^7)$
[10]	8	$O(n \log S)$
[4]	$O(1)$	$O(n^3 \log n)$
[8]	$3\left(1+\frac{1}{l}\right)^2$	$O(Kn)$
[9]	2.8334	$O(n(\log n \log\log n)^2)$
[12]	25/6	$O(n \log n)$
This paper	4	$O(n \log n)$

Table 1: A history of approximation algorithms for the unit disk cover problem in L_2.

There are numerous variants of the UDC problem. If the disk centers are constrained to an arbitrary point set Q, the UDC problem becomes the discrete unit disk cover problem (DUDC), which is also NP-hard. Many approximation algorithms are proposed for the DUDC problem, where the best known approximation factor is $9+\epsilon$ for any $0 < \epsilon \leq 6$ [2]. An instance of the UDC problem can be reduced to an instance of the DUDC problem as follows. Any solution for the UDC problem can be transformed so that each unit disk D has at least 2 input points on its boundary or an input point on its center; in the former case the center of D can be computed easily. Since each disk has unit radius, any pair of input points defines at most two possible centers for disks in our cover. Hence by choosing Q to be the union of P and these $O(n^2)$ centers, an instance of the DUDC problem is obtained. Thus, any approximation algorithm for the DUDC problem gives a solution for the UDC problem with the same approximation factor.

In the L_∞-norm, the UDC problem further reduces to

*School of Computer Science, Carleton University, Ottawa, Canada. Research supported by NSERC.

†Department of Computer Science, University of British Columbia.

the minimum clique cover problem [6]. The reduction uses the L_t *unit disk graph* on P. Each point in P corresponds to a vertex in the graph, and every edge (u, v) in the graph corresponds to intersecting L_t unit discs centered at u and v. Any family F of unit squares (L_∞ unit disks) satisfies Helly's property: if each pair of squares in F has a non-empty intersection, then the intersection of all squares in F is non-empty. Hence any clique in the L_∞ unit disc graph can be covered by a single L_∞ unit disc. Unfortunately, this reduction does not hold in the L_2-norm. The minimum clique problem on both the L_∞ and L_2 unit disk graphs has a large body of work, see [6] and the references contained therein.

We present an $O(n \log n)$-time constant-ratio approximation algorithm for the UDC problem in L_t-norms. In Section 2, we present a 4-approximation algorithm for this problem in the Euclidean norm (L_2-norm). By using the plane sweep technique, we show in Section 3 that this algorithm can be implemented to run in $O(n \log n)$ time. We emphasize that this algorithm is usable in practical settings and simple to implement. The most costly step is sorting of the points with respect to some dimension. In Section 4, we extend this algorithm to other L_t-norms. It is a 2-approximation for $t \in \{1, \infty\}$, a 6-approximation for $t > 2$, and a 5-approximation for $1 < t < 2$. Concluding remarks and extension to three dimensions are presented in Section 5.

2 A 4-Approximation Algorithm in L_2

In this section we consider the UDC problem in the Euclidean norm. Given a point set P in the plane, let C_{opt} be an optimal unit disk cover for P. Recall that the unit disks have radius 1. The *unit disk intersection graph*, UDIG(P), is defined to have the points of P as its vertices and has a straight-line edge between two points $p, q \in P$ if and only if $|pq| \leq 2$, where $|pq|$ is the Euclidean distance between p and q. We begin with the following observation:

Observation 1 *For two points $p, q \in P$, if $(p, q) \notin$ UDIG(P), then p and q cannot be covered by a unit disk.*

An *independent set* in UDIG(P) is a subset I of P such that there is no edge between any pair of points in I. I is said to be a *maximal independent set* if for all $p \in P \setminus I$, $I \cup \{p\}$ is not an independent set in UDIG(P). A maximal independent set in UDIG(P) can easily be found by a greedy algorithm.

Assume I is a maximal independent set in UDIG(P). By Observation 1, the size of any independent set in UDIG(P) is a lower bound for the number of disks needed to cover P. Therefore,

$$|I| \leq |C_{opt}|. \quad (1)$$

It is known that to cover a disk of radius 2, seven unit disks of radius 1 are necessary and sufficient; see Figure 1. Moreover, to cover a ball of radius 2 in three dimension, 21 unit balls are necessary and sufficient [1]. Based on that, a 7-approximation algorithm for the UDC problem is obtained as follows. Let I be any maximal independent set in UDIG(P). For a point $p \in I$, let $D(p, 2)$ be the disk of radius 2 which is centered at p. Let $d(p)$ be a disk in any unit disk cover which covers p. By Observation 1, none of the points of P which are at distance greater than 2 from p can be covered by $d(p)$. Therefore, all points of P which are not in $D(p, 2)$ must be covered by disks different from $d(p)$. Moreover, all points of P which are covered by $d(p)$ are in $D(p, 2)$. Therefore, by covering $D(p, 2)$ with seven unit disks (Figure 1), for all $p \in I$, a 7-approximation algorithm is obtained. Note that UDIG(P) may have up to $O(n^2)$ edges, and hence the time complexity of computing UDIG(P) is quadratic in the worst case.

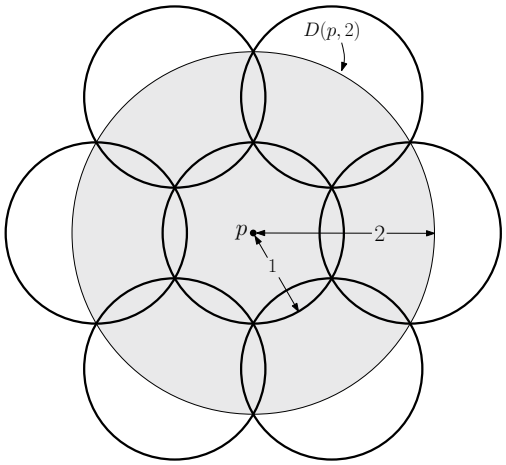


Figure 1: $D(p, 2)$ can be covered by 7 unit disks.

Now we show how to reduce the approximation ratio to 4. Let p be the leftmost point in P. In case of degeneracy, we consider the leftmost point with the smallest y-coordinate. Let ℓ be the vertical line passing through p. Let $R(p)$ be the intersection of $D(p, 2)$ with the halfplane to the right of ℓ, i.e., $R(p)$ is the right half-disk of $D(p, 2)$ (see Figure 2(a)). As discussed earlier, all points of P which are covered by $d(p)$ are in $D(p, 2)$ and consequently in $R(p)$. As shown in Figure 2(a), $R(p)$ can be covered by 4 unit disks. Figure 2(b) shows a configuration of seven points in $R(p)$ such that at least four unit disks are needed to cover all these seven points: in any unit disk cover, the disk which covers p can cover at most one of the points on the boundary. The remaining five points need at least three unit disks to be covered.

For a point p and a given point set I, the *distance*, $\mathrm{d}(p, I)$, between p and I is defined as the minimum Euclidean distance between p and any point in I, i.e.,

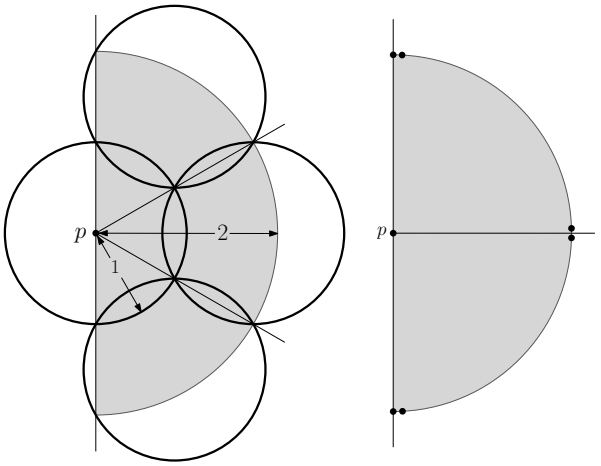


Figure 2: (a) Any half-disk of radius 2 can be covered by four unit disks. (b) Seven points in a half-disk of radius 2 which cannot be covered by less than four unit disks.

$\mathrm{d}(p, I) = \min\{|pq| : q \in I\}$. If $I = \emptyset$, then $\mathrm{d}(p, I) = \infty$. Our 4-approximation algorithm is given in Algorithm 1. The output of this algorithm is a set C of unit disks that cover P. The algorithm starts by creating a sorted list of points from left to right. Then it repeatedly selects and deletes the first element in the list, say p. If $\mathrm{d}(p, I) \leq 2$, then p is already covered by some disk in C. Otherwise, i.e., if $\mathrm{d}(p, I) > 2$, the algorithm covers $R(p)$ by four unit disks, and adds them to C. Finally it returns the set C of unit disks.

Algorithm 1 UnitDiskCover(P)

Input: A point set P in the plane.
Output: A set C of unit disks that cover P.

```
1: C ← ∅
2: I ← ∅
3: L ← list of points in P sorted from left to right
4: while L is not empty do
5:     p ← first element of L
6:     if d(p, I) > 2 then
7:         Cover R(p) by four unit disks c1, c2, c3, c4
8:         C ← C ∪ {c1, c2, c3, c4}
9:         I ← I ∪ {p}
10:    L ← L − {p}
11: return C
```

In each iteration, Algorithm 1, adds p to I if and only if $\mathrm{d}(p, I) > 2$. Thus, in UDIG(P), p is not connected to any point in I. Therefore, I is an independent set in UDIG(P). In addition, the while loop iterates over all points. Thus, after Algorithm 1 terminates, I is a maximal independent set in UDIG(P).

Theorem 1 *Algorithm 1 is a 4-approximation for the unit disk cover problem.*

Proof. Consider the set I of points and the set C of unit disks after the termination of Algorithm 1. Since I is a maximal independent set in UDIG(P), by Inequality (1) we have $|I| \leq |C_{opt}|$. Each point $q \in P$ is in a half-disk $R(p)$, for some $p \in I$ (possibly $q = p$). Since for each $p \in I$, we cover $R(p)$ with four unit disks, C covers P. Moreover, $|C| \leq 4|I| \leq 4|C_{opt}|$. This proves the statement of the theorem. □

The running time of Algorithm 1, can be expressed as $O(n \log n + n \cdot t(\mathrm{d}))$, where $t(\mathrm{d})$ is the time for computing $\mathrm{d}(p, I)$. Any nearest-neighbor data structure is sufficient here, and only insertions and queries are needed. As the nearest-neighbor problem is a decomposable search problem, the general techniques of Bentley and Saxe [3] gives an $O(\log^2 n)$-amortized time bound for both insertions and queries, and uses only $O(n)$-space. Using this data structure, $\mathrm{d}(p, I)$ can be computed in $O(\log^2 n)$-amortized time, and hence Algorithm 1 can be implemented to run in $O(n \log^2 n)$-time.

3 Improving the Time Complexity

Instead of computing $\mathrm{d}(p, I)$ dynamically, we can speed up Algorithm 1 by taking advantage of the fact that we only need to check if $\mathrm{d}(p, I)$ is greater than 2. Every time we add a new point p to I in Algorithm 1, we are essentially removing every point in P lying in $R(p)$. We can do this in $O(n \log n)$-time with a simple sweep-line algorithm.

We sweep a vertical line from left to right and maintain a binary search tree (BST) storing the centers of all the half-disks intersecting the sweep line. The points in BST are sorted in non-decreasing order of their y-coordinates. In case of ties, we sort them in increasing order of their x-coordinates. Since all half-disks have radius 2, they are uniquely defined by their centers which are stored in BST. Initially BST is empty.

We also keep an event queue that stores two types of events: *site events* and *deletion events*. A site event is a point of P. Each deletion event is associated with a site event; for each point $p \in P$ its deletion event is the rightmost point of $R(p)$. Thus, for every point $p = (p_x, p_y)$ in P, we have a deletion event $p' = (p_x + 2, p_y)$. The event queue is kept as a priority queue sorted by the x-coordinates of the events. Initially we add to the event queue each point $p \in P$ as a site event and p' as a deletion event. At each step of the sweep algorithm, we pop the event with the smallest x-coordinate from the queue, and "move" the sweep-line to that point.

Deletion events are straight-forward to handle, as we remove the center of the half-disk—which corresponds to this event—from BST.

Now we describe how to handle site events. Let p be the current site event which is encountered by the sweep-line SL. If p is covered by a half-disk in BST, then we proceed to the next event. If p is not covered by any half-disk in BST, then we insert a new half-disk (its center) into BST. Since the half-disks in BST have radius 2, we have the following observation:

Observation 2 *The distance between any two points in BST is more than 2.*

Note that the half-disks corresponding to the points to the left of SL which are not in BST do not intersect SL. Therefore, these points have distance bigger than 2 from SL, and p cannot be covered by their half-disks.

In order to check if p is covered by any half-disk intersecting the sweep-line we do the following. We search for p in BST by its y-coordinate. Let p^- and p^+ be the predecessor and the successor of p in BST, respectively. In other words, p^- is the point in BST with the largest y-coordinate and p^+ is the point in BST with the smallest y-coordinate such that $p_y^- < p_y < p_y^+$. If $|pp^-| \leq 2$ (or $|pp^+| \leq 2$), then p is covered by $R(p^-)$ (or $R(p^+)$). However, this may not be the only case to decide if p is covered by a half-disk in BST. As shown in Figure 3(a), p is covered by a half-disk which is neither $R(p^-)$ nor $R(p^+)$.

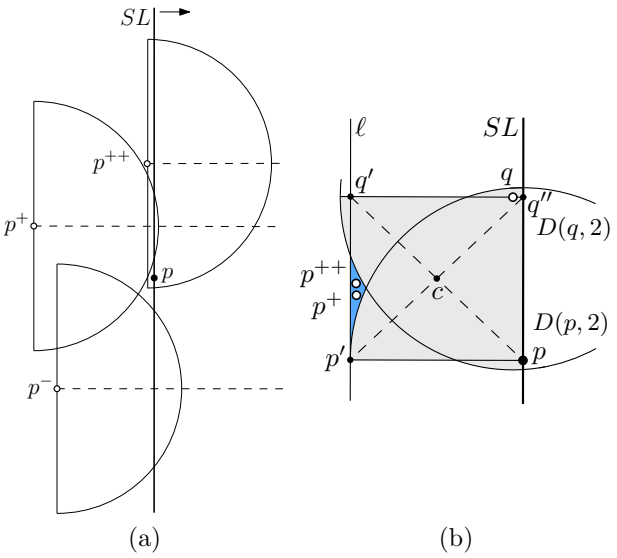


Figure 3: (a) p is covered by a half-disk other than $R(p^-)$ and $R(p^+)$. (b) Proof of Lemma 2

Let p^{--} be the predecessor of p^- and p^{++} be the successor of p^+ in BST.

Lemma 2 *If p is covered by any half-disk intersecting the sweep line, then $p \in R(p^{--}) \cup R(p^-) \cup R(p^+) \cup R(p^{++})$.*

Proof. The proof is by contradiction. Assume p is covered by a half-disk $R(q)$ which is centered at a point q in BST while $p \notin R(p^{--}) \cup R(p^-) \cup R(p^+) \cup R(p^{++})$. Without loss of generality assume $q_y \geq p_y$. Since p^+ is the successor of p and p^{++} is the successor of p^+ in BST, we have $q_y \geq p_y^{++}$. Let l be the vertical line which is at distance 2 from p and to the left of the sweep line SL; see Figure 3(b). All points in BST (including p^+, p^{++}, and q) lie between (or on) l and SL.

Let p' be the intersection point of l and the horizontal line passing through p. Let q' (resp. q'') be the intersection point of l (resp. SL) and the horizontal line passing through q. See Figure 3(b). Let R be the rectangle having its corners on p, p', q' and q''. Observe that the maximum side length for R is 2.

Since $p_y \leq p_y^+ \leq p_y^{++} \leq q_y$, p^+ and p^{++} lie in R. Consider $D(p,2)$ and $D(q,2)$. Since $p \in R(q)$, $|pq| \leq 2$; this implies that $p, q \in D(p,2) \cap D(q,2)$. By Observation 2, both p^+ and p^{++} are outside $D(q,2)$. In addition, p is to the right of p^+ and to the right of p^{++} and $p \notin R(p^+) \cup R(p^{++})$, which implies that both p^+ and p^{++} are outside $D(p,2)$. Therefore p^+ and p^{++} lie in region $Q = R - (D(p,2) \cup D(q,2))$; the blue region in Figure 3(b). Let c be the intersection point of the two diagonals of R. The triangle $\triangle pq'q''$ is a subset of $D(q,2)$ and the triangle $\triangle pp'q''$ is a subset of $D(p,2)$. Thus, Q is a subset of the triangle $\triangle cp'q'$. $\triangle cp'q'$ has diameter at most 2. Thus, the distance between any two points in Q is at most 2. Therefore, $|p^+p^{++}| \leq 2$; which contradicts Observation 2. □

Given a site event p, in $O(\log n)$-time we can find p^{--}, p^-, p^+, and p^{++} in BST. In order to check if p is in the coverage of any point in BST, by Lemma 2, it is enough to check if the distance of p to p^{--}, p^-, p^+, or p^{++} is at most 2. Therefore, each site event can be handled in $O(\log n)$-time; each deletion event can be handled in $O(\log n)$-time as well. Since we have $2n$ events, we conclude that Algorithm 1 can be implemented to run in $O(n \log n)$-time and $O(n)$-space.

4 Extensions to Other Metrics

In this section we consider the unit disk cover problem for a point set P in the L_t-norm, for $t \geq 1$. We show how to extend Algorithm 1 to a constant-approximation algorithm. In the L_t-norm, a unit circle which is centered at the origin is expressed by the equation

$$|x|^t + |y|^t = 1.$$

Figure 4 shows the unit circles in different L_t-norms. We refer to the union of a unit circle in the L_t-norm and its interior as an L_t-unit disk.

Observation 3 *For any t and t', with $1 \leq t < t' \leq \infty$, the L_t-unit disk which is centered at the origin is*

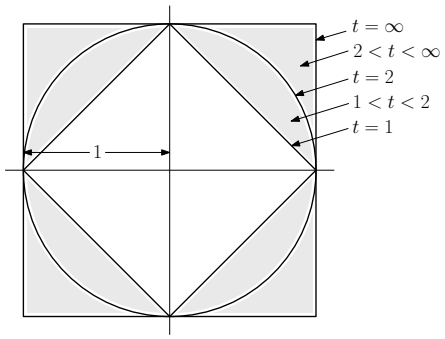


Figure 4: Illustration of unit circles in different L_t-norms.

contained in the $L_{t'}$-unit disk which is centered at the origin.

Let $D_t(p, 2)$ be the L_t-unit disk which is centered at point p and scaled by a factor of 2. Observe that any L_t-unit disk which covers p, does not cover any point outside $D_t(p, 2)$. Let $R_t(p)$ be the right half-disk of $D_t(p, 2)$. By Observation 3, $R_t(p)$ is contained in $R_\infty(p)$.

4.1 L_t for $t \geq 2$

Assume $t \geq 2$. As shown in Figure 5(a), $R_\infty(p)$ can be covered by six L_2-unit disks. Since $R_t(p) \subseteq R_\infty(p)$, $R_t(p)$ can also be covered by six L_2-unit disks. By Observation 3, any L_2-unit disk is contained in an L_t-unit disk. Thus, $R_t(p)$ also can be covered by six L_t-unit disks. Therefore, a modified version of Algorithm 1 gives an L_t-unit disk cover C for P such that $|C| \leq 6|C_{opt}|$.

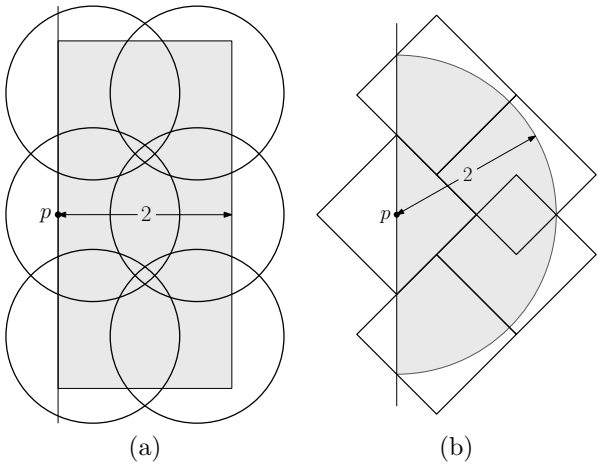


Figure 5: (a) $R_\infty(p)$ which is covered by six L_2-unit disks. (b) $R_2(p)$ which is covered by five L_1-unit-disks.

Since an L_t-unit disk contains an L_2-unit disk, Lemma 2 can be extended to the L_t-norm:

Lemma 3 *If p is covered by any L_t-half disk intersecting the sweep line, then $p \in R_t(p^{--}) \cup R_t(p^{-}) \cup R_t(p^{+}) \cup R_t(p^{++})$.*

Therefore, an $O(n \log n)$-time 6-approximation algorithm for the UDC problem in the L_t-norm is obtained.

4.2 L_t for $1 \leq t \leq 2$

Assume $1 \leq t \leq 2$. As shown in Figure 5(b), $R_2(p)$ can be covered by five L_1-unit disks. By Observation 3, $R_t(p)$ is contained in $R_2(p)$. In addition, an L_1-unit disk is contained in an L_t-unit disk. Thus, $R_t(p)$ can also be covered by five L_t-unit disks. Therefore, a modified version of Algorithm 1 gives an L_t-unit disk cover C for P such that $|C| \leq 5|C_{opt}|$. Lemma 2 can be extended to the L_1-norm as follows.

Lemma 4 *In L_1-norm, if p is covered by any half-disk intersecting the sweep line, then $p \in R_1(p^{--}) \cup R_1(p^{-}) \cup R_1(p^{+}) \cup R_1(p^{++})$.*

Proof. The proof is by contradiction; and similar to the proof of Lemma 2. We skip the details. Consider $D_1(p, 2)$ and $D_1(q, 2)$. Note that both p^+ and p^{++} are outside $D_1(p, 2) \cup D_1(q, 2)$. See Figure 6(a). Therefore p^+ and p^{++} lie in region $Q = R - (D_1(p, 2) \cup D_1(q, 2))$, where R is a unit square which has its bottom-right corner on p. As shown in Figure 6(a), Q (the blue region) can be covered by the L_1-unit disk S. Therefore, the L_1-distance between p^+ and p^{++} is at most 2; which contradicts Observation 2. □

Since an L_t-unit disk contains an L_1-unit disk, Lemma 4 can be extended to the L_t-norm. Therefore, an $O(n \log n)$-time 5-approximation algorithm for the UDC problem in the L_t-norm is obtained.

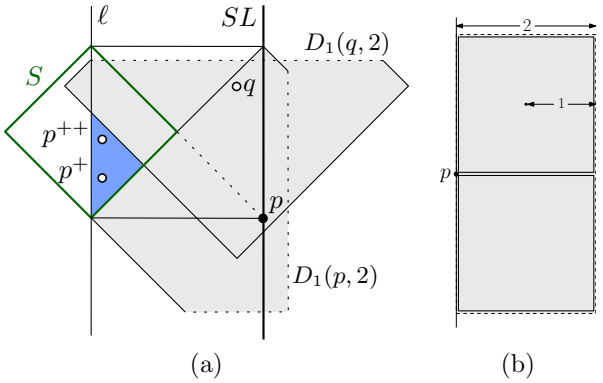


Figure 6: (a) Illustration of Lemma 4. (b) $R_\infty(p)$ which is covered by two L_∞-unit-disks.

4.3 L_∞ and L_1

Assume $t = \infty$. An L_∞-unit disk is an axis-aligned square of side length 2. As shown in Figure 6(b), $R_\infty(p)$ can be covered by two L_∞-unit disks. Therefore, a modified version of Algorithm 1 gives an L_∞-unit disk cover C for points in P such that $|C| \leq 2|C_{opt}|$. In addition, we have the following Lemma, which is stronger than Lemma 2.

Lemma 5 *If p is covered by any L_∞-half disk intersecting the sweep line, then $p \in R_\infty(p^-) \cup R_\infty(p^+)$.*

Therefore, a simple $O(n \log n)$-time 2-approximation algorithm for the UDC problem in the L_∞-norm is obtained. Gonzalez [10] presented a faster $O(n \log S)$-time 2-approximation algorithm for this problem, where S is the size of an optimal solution.

The UDC problem in the L_1-norm can easily be reduced to a UDC problem in the L_∞-norm by simply rotating the x and y axes by 45° around the origin, followed by scaling with $\sqrt{2}/2$. Therefore, a simple $O(n \log n)$-time 2-approximation algorithm for the UDC problem in L_1 is obtained.

5 Conclusion

We considered the NP-hard problem of covering n given points in the plane with the minimum number of unit disks. We presented an easily implementable 4-approximation algorithm which runs in $O(n \log n)$-time and $O(n)$-space. The presented algorithm is faster than previous algorithms having a similar approximation ratio. It is interesting that the most time consuming step of the algorithm is sorting and maintaining a BST.

We extended the algorithm to other L_t-norms. As a result we obtained $O(n \log n)$-time algorithms; a 2-approximation for $t \in \{1, \infty\}$, a 6-approximation for $t > 2$, and a 5-approximation for $1 < t < 2$.

The natural problem is to reduce the approximation ratio, while not increasing the running time.

Another open problem is to extend this algorithm to higher dimensions. In three dimensions, a ball of radius 2 can be covered by 21 unit-balls [1]. Therefore, Algorithm 1 is a 21-approximation for the UDC problem in $\mathbb{R}^3$. In order to check if $\mathrm{d}(p, I) > 2$, it is sufficient to check if the ball of radius 2 which is centered at p does not contain any point of I. A ball emptiness query in $\mathbb{R}^3$ can be transformed to a half-space emptiness query in $\mathbb{R}^4$ by projecting the points of P to the paraboloid $x_4 = x_1^2 + x_2^2 + x_3^2$. Chan [5] presented a linear-size data structure which can be constructed in $O(n \log n)$-time that answers half-space emptiness queries in $\mathbb{R}^4$ in $O(\sqrt{n})$-time. Based on the techniques of Bentley and Saxe [3], this gives an insertion-only dynamic data structure which supports insertions and half-space emptiness queries in $\mathbb{R}^4$ in $O(\sqrt{n} \log n)$-amortized time. Therefore, an $O(n\sqrt{n} \log n)$-time 21-approximation algorithm for the UDC problem in $\mathbb{R}^3$ is obtained.

However, we believe that a half-ball of radius 2 can be covered by 14 unit-balls; which would imply an approximation ratio of 14.

References

[1] Covering a unit ball with balls half the radius. http://www.mathoverflow.net/questions/98007/covering-a-unit-ball-with-balls-half-the-radius.

[2] R. Acharyya, M. Basappa, and G. K. Das. Unit disk cover problem in 2D. In *Proceedings of 13th Int. Conf. in Comput. Sci. and its App.-ICCSA*, pages 73–85, 2013.

[3] J. L. Bentley and J. B. Saxe. Decomposable searching problems. I. Static-to-dynamic transformation. *J. Algorithms*, 1(4):301–358, 1980.

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

[5] T. M. Chan. Optimal partition trees. *Discrete & Computational Geometry*, 47(4):661–690, 2012.

[6] M. De, G. K. Das, and S. C. Nandy. Approximation algorithms for the discrete piercing set problem for unit disks. In *Proceedings of the 23rd Annual Canadian Conf. on Comput. Geom.*, 2011.

[7] R. J. Fowler, M. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inf. Process. Lett.*, 12(3), 1981.

[8] M. Franceschetti, M. Cook, and J. Bruck. A geometric theorem for approximate disk covering algorithms. Technical report, 2001.

[9] B. Fu, Z. Chen, and M. Abdelguerfi. An almost linear time 2.8334-approximation algorithm for the disc covering problem. In *Proceedings of 3rd International Conference of Algorithmic Aspects in Information and Management*, pages 317–326, 2007.

[10] T. F. Gonzalez. Covering a set of points in multidimensional space. *Inf. Process. Lett.*, 40(4):181–188, 1991.

[11] 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.

[12] P. Liu and D. Lu. A fast 25/6-approximation for the minimum unit disk cover problem. *CoRR*, abs/1406.3838, 2014.

Bounds on Mutual Visibility Algorithms*

Gokarna Sharma† Costas Busch† Supratik Mukhopadhyay†

Abstract

We consider the fundamental MUTUAL VISIBILITY problem for a set of n identical autonomous point robots (n is not known to the robots) that operate following Look-Compute-Move cycles starting from arbitrary distinct positions in the Euclidean plane under obstructed visibility – a robot r_i can see robot $r_j, r_j \neq r_i$, if and only if there is no other robot in the line segment joining their positions. The objective is to determine a schedule to reposition these robots without collisions such that they reach in finite time a configuration where they all see each other. In the recently proposed so-called robots with lights model, Di Luna *et al.* [15] gave two deterministic algorithms Contain and Shrink for this problem; however, no runtime bounds were given except the proof that they terminate in finite time. In this paper, we first study the runtime bounds of these algorithms in the fully synchronous setting showing that Contain is tight ($\Theta(n)$ rounds) and Shrink needs $\Omega(n^2)$ rounds in the worst-case. We then present a new deterministic algorithm, called Modified_Shrink, for fully synchronous setting that solves this problem in $\mathcal{O}(n \log n)$ rounds, improving significantly over Shrink. We also show that Modified_Shrink has the lower bound of $\Omega(n)$ rounds.

1 Introduction

Consider a set of n autonomous point robots (n is not known to the robots) in the distinct positions in the Euclidean plane $\mathbb{R}^2$ which are anonymous, indistinguishable, and without any direct means of communication. Each robot is equipped with a local coordinate system and sensor capabilities (i.e., vision) to determine the positions of other robots. The local coordinate system of a robot may be different with that of other robots. The robots execute the same algorithm. They operate in *Look-Compute-Move* cycles, i.e., when a robot becomes active, it uses its vision to get a snapshot of its surroundings (*Look*), computes a destination point based on the snapshot (*Compute*), and finally moves towards the destination (*Move*), if any. Most of the literature assumes that the robots are *oblivious* - each robot has no memory of its past Look-Compute-Move actions - and visibility is *unobstructed* - three collinear robots are assumed to be mutually visible to each other [2, 8, 9, 12, 19, 22].

In this paper, we consider *obstructed visibility* [4, 3, 10, 1, 5, 6] under which a robot r_i can see robot $r_j, r_i \neq r_j$, if and only if there is no other robot in the line segment joining their positions. We study the following fundamental MUTUAL VISIBILITY problem: Starting from the arbitrary distinct positions in the Euclidean plane $\mathbb{R}^2$, determine a schedule to reposition the robots without collisions such that they reach within finite time a configuration where they all see each other. Note that robots moves do not follow grid coordinates of the plane $\mathbb{R}^2$, i.e., we do not assume the existence of some underlying universal grid in $\mathbb{R}^2$. Although obstructed visibility is considered before in the classical oblivious robots model for the SPREADING problem [4] and in the so-called *fat robots* model [1, 6, 12, 17], the technique of [4] cannot be generalized for MUTUAL VISIBILITY, since it works only in the one-dimensional space $\mathbb{R}^1$, and the techniques of [1, 6, 12, 17] are also not suitable, since collisions are allowed and used as an explicit communication tool.

Di Luna *et al.* [16] were the first to study MUTUAL VISIBILITY problem. They studied MUTUAL VISIBILITY in the *robots with lights* model initially suggested by Peleg [18], where each robot has an externally visible persistent light that can assume colors from a fixed set of colors and the color set is identical to all the robots. The robots communicate with other robots using these colored lights [12, 7, 11, 13, 21, 18]; the reason for considering robots with lights model is that there is no MUTUAL VISIBILITY algorithm in the classical oblivious robots model when n is not known. The lights are not erased at the end of each cycle in this model and the robots are oblivious, except the direct communication capability provided by lights. Moreover, this model corresponds to the classical oblivious robots model when the number of colors $c = 1$ in the color set, since a light with only one possible color acts as no light. Di Luna *et al.* [16] gave a deterministic algorithm that solves MUTUAL VISIBILITY with $c = 6$ colors in the semi-synchronous setting and with $c = 10$ colors in the asynchronous setting. Later, Di Luna *et al.* [15] gave two deterministic algorithms Contain and Shrink with $c = 3$ and $c = 2$

*The project is supported by Army Research Office (ARO) under Grant #W911-NF1010495. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the ARO or the United States Government.

†School of Electrical Engineering and Computer Science, Louisiana State University, {gokarna, busch, supratik}@csc.lsu.edu.

colors, respectively, in the semi-synchronous setting.

Di Luna *et al.* [15] proved the correctness of both algorithms Contain and Shrink. However, no runtime bounds were given except the proof that they terminate in finite time. Recently, Vaidyanathan *et al.* [20] gave an algorithm similar to Contain for MUTUAL VISIBILITY in the fully synchronous setting and proved that it has running time of $\mathcal{O}(\log n)$ rounds. However, their algorithm assumes *chirality* [1, 6] and does not avoid robot collisions due to the crossing of paths during robot movements.

In this paper, we consider the fully synchronous setting (where all robots are activated in a round and robots perform their cycles in a perfectly synchronous setting) and study the runtime bounds of MUTUAL VISIBILITY algorithms. In particular, we have made following three contributions.

- We show that Contain [15] has the tight bound of $\Theta(n)$ rounds on running time.
- We show that there exists an initial configuration of n robots in which Shrink [15] needs $\Omega(n^2)$ rounds.
- We present a new deterministic algorithm, called Modified_Shrink, for fully synchronous setting that uses $c = 3$ colors and needs only $\mathcal{O}(n \log n)$ rounds to solve MUTUAL VISIBILITY starting from any initial configuration of n robots. This is a significant improvement over the runtime bound of Shrink which is at least $\Omega(n^2)$ rounds. We also prove that Modified_Shrink has a lower bound of $\Omega(n)$ rounds.

Paper Organization: We proceed as follows. We present model in Section 2. We prove bounds for Contain in Section 3. We then prove a lower bound for Shrink in Section 4. In Section 5, we present and analyze Modified_Shrink. Many proofs are omitted due to space constraints.

2 Model

We consider a set of n anonymous robots $\mathcal{R} = \{r_1, r_2, \ldots, r_n\}$ operating in the Euclidean plane $\mathbb{R}^2$; n is not assumed to be known. We denote by $p_i(k) \in \mathbb{R}^2$ the position occupied by robot $r_i \in \mathcal{R}$ at time k. A robot r_i sees robot $r_j, r_j \neq r_i$, at time k if and only if the line segment $\overline{p_i(k)p_j(k)}$ does not contain any other robot at time k. Two robots r_i and r_j are said to *collide* at time k if $p_i(k) = p_j(k)$. If no ambiguity arises, we omit k from $r_i(k)$ and $p_i(k)$, and use r_i to denote both the robot r_i and its position p_i. Each robot r_i has its own coordinate system centered in itself and it knows its position with respect to its coordinate system. Moreover, robots have their own unit of distance which may not agree on the unit of measure of other robots. The robots do not agree on the orientation of their coordinate system, i.e., there is no common notion of the clockwise direction.

Each robot r_i is equipped with an externally visible persistent light which can assume any color from a fixed finite set of colors $\mathcal{C}$. The colors in $\mathcal{C}$ are the same for all robots in $\mathcal{R}$. We use variable $r_i.light$ to denote the light of a robot r_i. The color of the light of a robot r at time k can be seen by all robots that are visible to r at time k. Robots are oblivious − do not remember decisions performed in previous cycle − and a robot's decision at any cycle is only based on the positions of the robots visible to it at that cycle. Robots are *autonomous* (i.e., without any external control), *indistinguishable* (i.e., do not have external markings), and do not have any direct means of communication (except the lights). Moreover, they are *anonymous* (i.e., do not have internal identifiers). Each robot executes the same algorithm locally every time it is activated.

A *configuration* $\mathbb{C}$ is a set of n tuples in $\mathbb{C} \times \mathbb{R}^2$ which defines the position and color of a robot. Let $\mathbb{C}_k$ denotes the configuration at time k. Let $\mathbb{C}_k(r_i)$ denotes the configuration $\mathbb{C}_k$ for robot r_i. A configuration $\mathbb{C}$ is *obstruction-free* if $\forall r_i \in \mathcal{R}$, we have that $|\mathbb{C}(r_i)| = n$ (i.e., all robots can see each other). Let $\mathbb{H}_k$ denotes the convex hull formed by $\mathbb{C}_k$ which can be easily computed using Graham's convex hull algorithm [14]. Let $\partial\mathbb{H}_k = \mathcal{V}_k \cup \mathcal{E}_k$ denotes the robots in the boundary of $\mathbb{H}_k$, where $\mathcal{V}_k \subseteq \mathcal{R}$ are the set of robots lying at the vertices of $\mathbb{H}_k$ and $\mathcal{E}_k \subseteq \mathcal{R}$ are the set of robots lying at the sides (or edges) of $\mathbb{H}_k$. The robots in the set $\mathcal{V}_k$ are called *vertex robots* and in the set $\mathcal{E}_k$ are called *edge robots*. The robots in $\mathcal{V}_k \cup \mathcal{E}_k$ are called *boundary robots*. The robots in the set $\mathbb{H}_k \backslash \partial\mathbb{H}_k$ are called *internal robots*. Given a robot $r_i \in \mathcal{R}$, we denote by $\mathbb{H}_k(r_i)$ the convex hull of $\mathbb{C}_k(r_i)$. Given two points $a, b \in \mathbb{R}^2$, we denote by $\overleftrightarrow{ab}$ the line that contains them.

We assume that the execution starts at time 0. Therefore, at time $t = 0$, the robots start in an arbitrary configuration $\mathbb{C}_0$ occupying distinct positions in $\mathbb{R}^2$ and the color of the light of each robot is set to *Off*. The MUTUAL VISIBILITY problem is defined as follows: Given any $\mathbb{C}_0$, reach in finite time an obstruction-free configuration without collisions. An algorithm is said to solve MUTUAL VISIBILITY if it always achieves an obstruction-free configuration regardless of the choices of the adversary and from any arbitrary $\mathbb{C}_0$.

We assume that, when active, each robot $r_i \in \mathcal{R}$ performs a sequence of *Look-Compute-Move* (LCM) operations: a robot takes the snapshot of the positions of the robots visible to it in its own coordinate system (*Look*); executes its algorithm using the snapshot which returns a destination point $x \in \mathbb{R}^2$ and a color $c \in \mathcal{C}$ (*Compute*); and sets its own light to color c and moves towards the computed destination $x \in \mathbb{R}^2$ (if x is differ-

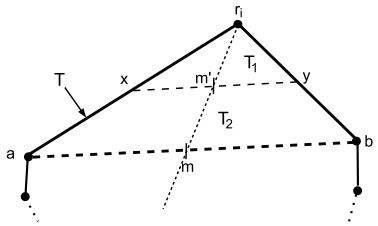


Figure 1: An illustration of T, T_1, and T_2 of a vertex robot r_i in $\mathbb{H}(r_i)$, where a and b, respectively, are the neighbor robots of r_i in $\mathbb{H}(r_i)$ in its counterclockwise and clockwise direction, x, y, and m are the midpoints of line segments $\overline{r_i a}$, $\overline{r_i b}$, and $\overline{ab}$, respectively, and m' is the intersection point of $\overline{r_i m}$ and $\overline{xy}$.

ent than its current position), if any (*Move*). We assume the rigid moves throughout the paper in the sense that the movement of robots are not controlled by an adversary and every robot reaches its destination at all times when it moves from a current position to its computed destination. Moreover, we consider a fully synchronous scheduler for the activation of the robots in $\mathcal{R}$. In the fully synchronous scheduler, the time is discrete and at each time instant k all the robots of $\mathcal{R}$ are activated and perform their LCM operations instantaneously, ending at time $k+1$. Therefore, we use round k instead of time k from now on. Finally, we measure the quality of the algorithm by counting the number of rounds until the robots have reached the mutual visibility configuration (all robots are the vertices of $\mathbb{H}$).

As shown in Fig. 1, let r_i be a vertex of $\mathbb{H}$ and a and b are its counterclockwise and clockwise neighbors in $\mathbb{H}$. Moreover, let x and y be the midpoints of line segments $\overline{r_i a}$ and $\overline{r_i b}$, respectively, and m be the midpoint of line segment $\overline{ab}$. We have that, according to construction, $\overline{xy}$ is parallel to $\overline{ab}$. For each vertex robot r_i, we denote by T the triangular area $r_i ab$, by T_1 the triangular area $r_i xy$, and by T_2 the trapezoidal area $xyba$ (i.e., $T_2 := T \backslash T_1$). When we say that a robot w is *closest* to r_i then we mean that there is no other robot in the area of $\mathbb{H}(r_i)$ between r_i and a line parallel to $\overline{ab}$ (or $\overline{xy}$) that passes through w's position.

3 Tight Bounds for Contain Algorithm

Contain [15] has two phases: an *interior depletion* phase and a *vertex adjustment* phase. The second phase is executed only after the first phase is finished. In the interior depletion phase, the robots in the interior of $\mathbb{H}$ move towards the boundary of $\mathbb{H}$ and in the vertex adjustment phase the robots in the vertices of $\mathbb{H}$ move towards the interior of $\mathbb{H}$ to reach a strictly convex configuration with all the robots being in the vertices of $\mathbb{H}$. Three colors are used, namely $\mathcal{C} = \{Off, External, Adjusting\}$.

We prove the following lemma for the lower bound.

Lemma 1 *There is an initial configuration $\mathbb{C}_0$ of the robots in which* Contain *takes $\Omega(n)$ rounds to solve* Mutual Visibility *in the fully synchronous setting.*

We prove the following lemma for the upper bound.

Lemma 2 *Starting from any initial configuration of a set of n robots,* Contain *needs $\mathcal{O}(n)$ rounds to solve* Mutual Visibility *in the fully synchronous setting.*

Combining the lower and upper bounds of Lemmas 1 and 2, we obtain the following theorem.

Theorem 3 *The round complexity of* Contain *for* Mutual Visibility *is $\Theta(n)$ in the worst-case in the fully synchronous setting.*

4 Lower Bound for Shrink Algorithm

Shrink works as follows. The vertex robots set their light to *Vertex*. Let r_i be a vertex robot of $\mathbb{H}(r_i)$ and a and b be r_i's counterclockwise and clockwise neighbors both in the boundary of $\mathbb{H}(r_i)$. Let x and y be the midpoints of line segments $\overline{r_i a}$ and $\overline{r_i b}$, respectively, and m be the midpoint of $\overline{ab}$. If there is an interior robot, say r', in T_1, then r_i moves to some point in T_1 in the line segment that is parallel to $\overline{xy}$ and passes through r. If there are more than one robot in T_1, then some point in the line segment parallel to $\overline{xy}$ passing through the closest robot is chosen. However, if there is no robot inside T_1 (i.e. all interior robots are outside T_1), then r_i moves to the point m' in $\overline{xy}$, irrespective of the positions of the interior robots, where m' is the point in which the line segment $\overline{r_i m}$ intersects $\overline{xy}$ (see Fig. 1). If there is only one robot in the interior of $\mathbb{H}(r_i)$, then that interior robot moves to the midpoint of some edge in the boundary of $\mathbb{H}(r_i)$. Two colors are used, namely $\mathcal{C} = \{Off, Vertex\}$. We prove the following theorem for the lower bound.

Theorem 4 *There is an initial configuration $\mathbb{C}_0$ of the n robots in which* Shrink *takes $\Omega(n^2)$ rounds to solve* Mutual Visibility *in the fully synchronous setting.*

5 The Modified_Shrink Algorithm

We present Modified_Shrink which improves significantly over the runtime of Shrink in the fully synchronous setting. The pseudocode of Modified_Shrink is given in Algorithms 1. Algorithm 1 uses Algorithms 2–4 as subroutines to accomplish the mutual visibility of robots starting from any arbitrary initial configuration $\mathbb{C}_0$.

Modified_Shrink uses three colors in the set $\mathcal{C}$ of colors, namely $\mathcal{C} = \{Off, Vertex, Edge\}$. The colors in the set $\mathcal{C}$ are used by robots to detect whether Mutual

Algorithm 1: Modified_Shrink algorithm for any round $k > 0$

1 // Look-Compute-Move cycle for each robot $r_i \in \mathcal{R}$
2 $\mathbb{C}_k(r_i) \leftarrow$ configuration $\mathbb{C}_k$ for robot r_i (including r_i);
3 $\mathbb{H}_k(r_i) \leftarrow$ convex hull of the positions of the robots in $\mathbb{C}_k(r_i)$;
4 **if** $|\mathbb{C}_k(r_i)| = 3 \wedge \mathbb{H}_k(r_i)$ *is a line segment* **then**
5 Move orthogonal to (the line segment) $\mathbb{H}_k(r_i)$ by any non-zero distance;
6 **else**
7 **if** r_i *is in vertex of* $\mathbb{H}_k(r_i)$ **then** $Corner(r_i, \mathbb{C}_k(r_i), \mathbb{H}_k(r_i))$;
8 **else if** r_i *is in edge of* $\mathbb{H}_k(r_i)$ **then** $Side(r_i, \mathbb{C}_k(r_i), \mathbb{H}_k(r_i))$;
9 **else if** $\forall r \in \mathbb{C}_k(r_i)\backslash\{r_i\}$, $r.light \in \{Vertex, Edge\} \wedge r_i$ *is in interior of* $\mathbb{H}_k(r_i)$ **then** $Interior(r_i, \mathbb{H}_k(r_i))$;

Algorithm 2: $Corner(r_i, \mathbb{C}_k(r_i), \mathbb{H}_k(r_i))$

1 **if** $r_i.light = Off$ **then** $r_i.light \leftarrow Vertex$;
2 **if** $\forall r \in \mathbb{C}_k(r_i)$, $r.light = Vertex$ **then** Terminate;
3 **else if** $|\mathbb{C}_k(r_i)| > 2$ **then**
4 $a \leftarrow$ counterclockwise neighbor on the boundary of $\mathbb{H}_k(r_i)$;
5 $b \leftarrow$ clockwise neighbor on the boundary of $\mathbb{H}_k(r_i)$;
6 $x \leftarrow$ midpoint of the line segment $\overline{r_i a}$;
7 $y \leftarrow$ midpoint of the line segment $\overline{r_i b}$;
8 **if** *there exists at least a robot in* $\mathbb{C}_k(r_i)\backslash\{r_i\}$ *with light Off* **then**
9 $r' \leftarrow$ robot in $\mathbb{C}_k(r_i)\backslash\{r_i\}$ with light *Off* that is closest to r_i w.r.t. the line parallel to the line segment $\overline{ab}$ (if more than one robot satisfies this criteria, choose as r' the robot that is closer to b);
10 **if** r' *is not in the triangular area* r_iab **then**
11 Move to the midpoint of the line segment $\overline{r_i r'}$;
12 **else if** r' *is in the triangular area* $r_iab \wedge r'$ *is not in the triangular area* r_ixy **then**
13 Move to the intersection point of the line segments $\overline{r_i r'}$ and $\overline{xy}$;
14 **else if** r' *is in the triangular area* r_ixy **then**
15 $L \leftarrow$ line parallel to $\overline{ab}$ that passes through r';
16 $z \leftarrow$ intersection point of L and $\overline{r_i b}$;
17 Move to the midpoint of the line segment $\overline{r'z}$;
18 **else if** *there exists at least a robot in* $\mathbb{C}_k(r_i)\backslash\{r_i\}$ *with light Edge* **then**
19 Move to the midpoint of the line segment $\overline{xy}$;

Visibility is solved and correctly terminate their computation. The lights of all robots in $\mathcal{R}$ are set to *Off* in the initial configuration $\mathbb{C}_0$. When a robot after activation in some round $k > 0$ realizes that it is a vertex of $\mathbb{H}$, it sets its light to *Vertex* (Line 7 of Algorithm 1, Line 1 of Algorithm 2). This task is easy since, if a robot r_i with light *Off* after activation in some round $k > 0$ sees that $\mathbb{C}_k(r_i)$ contains a region of plane that is free of robots and wider than 180°, then r_i knows it is a vertex of $\mathbb{H}$. If a robot realizes after activation in some round $k > 0$ that it is on an edge of $\mathbb{H}$, it sets its light to *Edge* (Line 8 of Algorithm 1, Line 1 of Algorithm 3). Similar to the realization of vertex robots, if a robot r_i with light *Off* after activation in some round k sees that $\mathbb{C}_k(r_i)$ contains a region of plane that is free of robots and wide exactly 180°, then r_i knows it is on an edge of $\mathbb{H}$. When the lights of all the robots that are seen by a robot are set to *Vertex*, the robot knows that it can see all the robots in $\mathcal{R}$ and hence it terminates (without the knowledge of n, the total number of robots).

Since our algorithm uses the convex hull shrinking process, the vertex and edge robots move inside. We now describe how vertex robots move and give the details on how edge robots move in the next paragraph. If there is an interior robot (i.e., a robot with light *Off*), say r', in T_1, then the vertex robot r_i moves somewhere in the line parallel to the line segment $\overline{ab}$ that passes through r', where a and b are the neighbor robots of r_i in $\mathbb{H}_k(r_i)$ in its counterclockwise and clockwise direction, respectively. If there is more than one robot in T_1, the closest one from r_i (w.r.t. a line parallel to $\overline{ab}$) is chosen as the robot r' (Lines 14-17 of Algorithm 2). In case all the interior robots are outside T_1 then if there are robots in T_2, r_i moves to the point in the line segment $\overline{xy}$ that intersects the line segment $\overline{r_i r'}$, where r' is the robot in T_2 that is closest to r_i again w.r.t. a line parallel to $\overline{ab}$, and x and y, respectively, are the midpoints of the line segments $\overline{r_i a}$ and $\overline{r_i b}$ (Lines 12,13 of Algorithm

Algorithm 3: $Side(r_i, \mathbb{C}_k(r_i), \mathbb{H}_k(r_i))$

1 **if** $r_i.light = Off$ **then** $r_i.light \leftarrow Edge$;
2 **if** *there exists at least a robot in* $\mathbb{C}_k(r_i)\backslash\{r_i\}$ *with light Off* **then**
3 $a \leftarrow$ counterclockwise neighbor on the boundary of $\mathbb{H}_k(r_i)$;
4 $b \leftarrow$ clockwise neighbor on the boundary of $\mathbb{H}_k(r_i)$;
5 $r_j \leftarrow$ closest robot to r_i in $\mathbb{C}_k(r_i)\backslash\{r_i\}$ with light *Off* w.r.t. a line parallel to line segment $\overline{ab}$;
6 Move to the midpoint of the line segment $\overline{r_i r_j}$;

Algorithm 4: $Interior(r_i, \mathbb{H}_k(r_i))$

1 **if** r_i *is not in the triangular area formed by any three consecutive robots of* $\mathbb{H}_k(r_i)$ **then**
2 $x \leftarrow$ midpoint of any edge between two consecutive robots of $\mathbb{H}_k(r_i)$;
3 Move to the midpoint of the line segment $\overline{r_i x}$;
4 **else**
5 $e \leftarrow$ the closest among two edges of the triangular area that are in $\mathbb{H}_k(r_i)$;
6 Move to the midpoint of the edge e;

2). When all the interior robots are outside T, r_i moves halfway to the line segment $overliner_i r'$ connecting r_i with the robot r' that is closest to it (Lines 10,11 of Algorithm 2). Note that if more than one robot is closest to the vertex robot r_i according to our criteria (w.r.t. a line parallel to $\overline{ab}$), then the robot that is closer to b among the closest robots is chosen as r' (Line 9 of Algorithm 2).

On the other hand, a robot, r_j, on the edge of $\mathbb{H}$ (which is not a vertex of $\mathbb{H}$) moves halfway to an internal robot that is closest to r_j w.r.t. a line parallel to $\overline{ab}$ ($\overline{ab}$ in this case is in fact a straight line segment that passes through r_j) (Lines 2–5 of Algorithm 3).

Moreover, there can be a situation in our algorithm that there is no robot in the interior of $\mathbb{H}$ but there are still some robots on the edges of $\mathbb{H}$. The robots in the edges do not move since there should not be any robot with light *Off* in the system for this situation to happen. On the other hand, the vertex robots recognize this situation and start moving to the midpoint of the line segment $\overline{xy}$ (Lines 18,19 of Algorithm 2). These moves of vertex robots are sufficient since we can show that eventually all edge robots become vertices even under these moves.

We now have two special cases in our algorithm. The first special case is when there is only one internal robot. In this case, we can show that if the internal robot does not move, Mutual Visibility cannot be achieved without collisions since, all the robots in $\mathcal{R}$ converge to the position of the only internal robot. However, our algorithm resolves this situation as follows. When there is only one robot, say w, in the interior of $\mathbb{H}_k(w)$, then the robot w recognizes this situation and moves towards the boundary of $\mathbb{H}_k(w)$. This recognition is easy as all the robots w sees have lights either *Vertex* or *Edge* (Line 9 of Algorithm 1). If w is inside the triangular area $r_i ab$ of some vertex robot r_i, it chooses the closest edge between $r_i a$ and $r_i b$ and moves to the midpoint of that edge (Lines 1, 5, 6 of Algorithm 4). Otherwise, it moves halfway from its location to the line segment connecting it with the midpoint of any edge between two consecutive robots of $\mathbb{H}_k(w)$ (Lines 2,3 of Algorithm 4). The second special case is when a robot $r_i \in \mathcal{R}$ sees only two other robots (Lines 4,5 of Algorithm 1). In this case, $H_k(r_i)$ must be a line segment. The robot r_i then moves orthogonal to $H_k(r_i)$ by any non-zero distance. These movements translate line segment $H_k(r_i)$ into a polygonal $H_k(r_i)$ which remains as polygonal $H_k(r_i)$ in future rounds.

5.1 Analysis of the Modified_Shrink Algorithm

We here analyze Modified_Shrink for both correctness and running time. We first show that the paths that robots follow when they move inside do not cross which is essential to show that Modified_Shrink avoids collisions due to hitting each other while relocating. We have the following lemma.

Lemma 5 *The paths of robots do not cross during the execution of* Modified_Shrink.

We now show that two robots do not land up to the same position during the execution of Modified_Shrink which is essential to prove that there is no collision due to position sharing. Note that in $\mathbb{C}_0$, robots do not share their positions since it is assumed that they start from the distinct positions (otherwise no collision requirement can not be achieved).

Lemma 6 *Two robots do not land up to the same position during the execution of* Modified_Shrink.

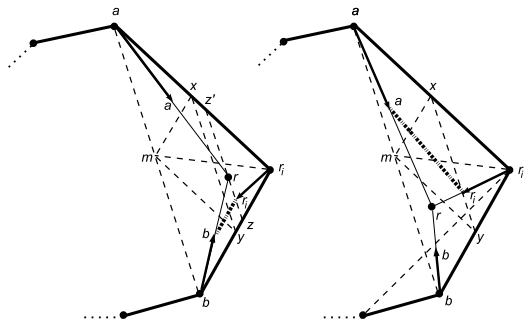


Figure 2: An illustration of robot movements in Modified_Shrink: (left) the closest robot to r_i, r, is in T_1; (right) the closest robot to r_i, r, is in T_2.

Combining Lemmas 5 and 6, we obtain the following theorem on collision avoidance property of Modified_Shrink.

Theorem 7 *Starting from any configuration of n robots,* Modified_Shrink *avoids robot collisions.*

We now show that vertex robots remain vertex which is essential to prove the convergence property of our algorithm and guarantee progress towards a mutual visibility configuration.

Lemma 8 *No vertex robot of $\mathbb{H}$ becomes internal or edge robot during the execution of* Modified_Shrink.

Proof. When there is no interior robot in the triangular area T of the vertex robots, the external (vertex and edge) robots converge to the same limit, i.e., in every round, all the robots in the boundary of $\mathbb{H}$ move exactly half distance to their closest robots in the line segment connecting them to their closest internal robots in $\mathbb{H}$. It can be easily seen that this process guarantees that the vertices of $\mathbb{H}$ remain as vertices of $\mathbb{H}$ in every future round. Therefore, we focus on the scenario where some vertex robots have internal robots in their triangular areas T and others do not have internal robots in their triangular areas.

We will show that the moves of some vertex robots to the positions inside T_1 and the moves of other vertex robots to the positions outside T_1 still guarantee that vertex robots remain as vertices of $\mathbb{H}$. One example configuration for such scenario is given in Fig. 2. Let r_i be a vertex robot in $\mathbb{H}$ and a and b are its neighbors in $\mathbb{H}$ with a being in its counterclockwise direction and b being in its clockwise direction. Let r be the robot that is inside the triangular area T of r_i. Assume that r is also the closest robot in the interior of $\mathbb{H}$ from the vertex robots a and b and it is not inside the triangular area T of both a and b. According to Algorithm 1, if r is inside the triangular area T_1 of r_i, r_i moves as shown in the left of Fig. 2 in the line, say L, that is parallel to the line segment $\overline{xy}$ that passes through the position of r. The exact point where r_i moves is the midpoint of the line segment $\overline{rz}$ of the line L, where x is the intersection point of the line L and the line segment $\overline{r_ib}$. The robots a and b move halfway to r since r is outside the triangular area T of both a and b.

We now show that the convexity of $\mathbb{H}$ is maintained in this situation. We first consider robot b and then the argue for robot a. We have that line segments $\overline{br}$ and $\overline{r_ib}$ intersect at b before b and r_i move inside which is also the vertex of $\mathbb{H}$. After they move inside, the line segment $\overline{r_ib}$ (bold dotted line in the left of Fig. 2) connecting the new positions of r_i and b is parallel to the line segment $\overline{r_ib}$ connecting their old positions because their new positions are the midpoints of two sides rb and rz of the triangle rbz. Now since r is also the closest robot to a the move of a makes the line segment $\overline{ar}$ (from a's new position to r) parallel to the line segment $\overline{ar_i}$ (from a's old position to r_i) if r_i moves to the midpoint of the line segment $\overline{rz'}$ (the argument here is similar to the one used for robot b), otherwise $\overline{ar}$ remains as the segment of line $\overleftrightarrow{ar}$ that intersects $\overleftrightarrow{r_ia}$ at a. Therefore, under any movements of r_i, a, and b, r_i does not become internal or edge robot because neither a nor b crosses the line segment $\overline{zz'}$ to reach some point in the triangular area r_izz' of r_i. Note that r actually becomes vertex after the moves of r_i, a, and b. Since this process is applied by all the vertex robots of $\mathbb{H}$, it is clear that the convexity of $\mathbb{H}$ is maintained and no vertex robot becomes an internal or edge robot.

Consider now the scenario where r is inside the trapezoidal area T_2 of r_i (the right of Fig. 2). In this case, r_i moves to the point where line segments $\overline{xy}$ and $\overline{r_ir}$ intersect. As the distance from r_i to its new position (after move) is at least half of $\overline{r_ir}$, the line segments connecting the new positions of a, r_i, and r_i, b become parallel to the current edges of $\mathbb{H}$, ap and pb, when r is at some position in the line segment $\overline{ab}$. Therefore, under any movements of r_i, a, and b, r_i does not become internal or edge robot because neither a nor b crosses the line segment $\overline{xy}$ to reach some point in the triangular area T_1 of r_i. Hence, combining the above claims, the lemma follows. □

We are now ready to analyze the runtime bound of Modified_Shrink.

Theorem 9 Modified_Shrink *solves* Mutual Visibility *in $O(n \log n)$ rounds using lights with 3 colors in the fully synchronous setting.*

Proof. When $\mathbb{H}$ is a line in $\mathbb{C}_0$ it becomes a polygonal $\mathbb{H}$ in one round due to the fully synchronous setting and it is easy to see that once line $\mathbb{H}$ is transitioned to polygonal $\mathbb{H}$, it does not become line $\mathbb{H}$ again in future. Starting from polygonal $\mathbb{H}$, according to Algorithm 1,

when there exists a robot in T_1, then it becomes vertex in one round. When there is a robot in T_2, then the robot reaches at least halfway close to it in next round. Therefore, the worst-case number of rounds of Algorithm 1 is when all the interior robots are not inside any T of vertex robots. However, we have from Algorithm 1 that external robots reach halfway to those interior robots (even if they are not inside T of any vertex robots) in every round. As vertex robots remain vertex (Lemma 8) and external robots move halfway to the interior robots in each round, after at most $\mathcal{O}(\log n)$ rounds, at least one internal robot becomes an external robot (vertex or edge). This is because the distance between a vertex robot and its closest internal robot decreases by half in every round and the closest internal robot for a vertex robot remains as closest until it eventually becomes an external robot. Therefore, as there are n robots in $\mathcal{R}$, they become external in at most $\mathcal{O}(n \log n)$ rounds. Moreover, we need at most $\mathcal{O}(n)$ rounds to make robots in edges the vertex robots after all internal robots reach the boundary of $\mathbb{H}$ (Lines 18,19 of Algorithm 2). Therefore, Algorithm 1 needs at most $\mathcal{O}(n \log n) + \mathcal{O}(n) = \mathcal{O}(n \log n)$ rounds. □

We have the following theorem for the lower bound of Modified_Shrink which shows the inherent difficulty in obtaining faster algorithms for the MUTUAL VISIBILITY problem.

Theorem 10 *There exists an initial configuration $\mathbb{C}_0$ of the robots in which* Modified_Shrink *takes $\Omega(n)$ rounds to solve* MUTUAL VISIBILITY *in the fully synchronous setting.*

References

[1] C. Agathangelou, C. Georgiou, and M. Mavronicolas. A distributed algorithm for gathering many fat mobile robots in the plane. In *PODC*, pages 250–259, 2013.

[2] N. Agmon and D. Peleg. Fault-tolerant gathering algorithms for autonomous mobile robots. In *SODA*, pages 1070–1078, 2004.

[3] K. Bolla, T. Kovacs, and G. Fazekas. Gathering of fat robots with limited visibility and without global navigation. In *SIDE*, pages 30–38, 2012.

[4] R. Cohen and D. Peleg. Local spreading algorithms for autonomous robot systems. *Theor. Comput. Sci.*, 399(1-2):71–82, June 2008.

[5] A. Cord-Landwehr, B. Degener, M. Fischer, M. Hüllmann, B. Kempkes, A. Klaas, P. Kling, S. Kurras, M. Märtens, F. Meyer auf der Heide, C. Raupach, K. Swierkot, D. Warner, C. Weddemann, and D. Wonisch. Collisionless gathering of robots with an extent. In *SOFSEM*, pages 178–189, 2011.

[6] J. Czyzowicz, L. Gasieniec, and A. Pelc. Gathering few fat mobile robots in the plane. *Theor. Comput. Sci.*, 410(6-7):481–499, 2009.

[7] S. Das, P. Flocchini, G. Prencipe, N. Santoro, and M. Yamashita. The power of lights: Synchronizing asynchronous robots using visible bits. In *ICDCS*, pages 506–515, 2012.

[8] X. Défago and S. Souissi. Non-uniform circle formation algorithm for oblivious mobile robots with convergence toward uniformity. *Theor. Comput. Sci.*, 396(1-3):97–112, 2008.

[9] Y. Dieudonné, O. Labbani-Igbida, and F. Petit. Circle formation of weak mobile robots. *TAAS*, 3(4), 2008.

[10] A. Dutta, S. G. Chaudhuri, S. Datta, and K. Mukhopadhyaya. Circle formation by asynchronous fat robots with limited visibility. In *ICDCIT*, pages 83–93, 2012.

[11] A. Efrima and D. Peleg. Distributed models and algorithms for mobile robot systems. In *SOFSEM*, pages 70–87, 2007.

[12] P. Flocchini, G. Prencipe, and N. Santoro. Distributed computing by oblivious mobile robots. *Synthesis Lectures on Distributed Computing Theory*, 3(2):1–185, 2012.

[13] P. Flocchini, N. Santoro, G. Viglietta, and M. Yamashita. Rendezvous of two robots with constant memory. In *SIROCCO*, pages 189–200, 2013.

[14] R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett.*, 1(4):132–133, 1972.

[15] G. A. D. Luna, P. Flocchini, S. G. Chaudhuri, F. Poloni, N. Santoro, and G. Viglietta. Mutual visibility by luminous robots without collisions. To appear in Information and Computation, *Available at* `http://arxiv.org/abs/1503.04347`, 2015.

[16] G. A. D. Luna, P. Flocchini, S. G. Chaudhuri, N. Santoro, and G. Viglietta. Robots with lights: Overcoming obstructed visibility without colliding. In *SSS*, pages 150–164, 2014.

[17] G. A. D. Luna, P. Flocchini, F. Poloni, N. Santoro, and G. Viglietta. The mutual visibility problem for oblivious robots. In *CCCG*, 2014.

[18] D. Peleg. Distributed coordination algorithms for mobile robot swarms: New directions and challenges. In *IWDC*, pages 1–12, 2005.

[19] I. Suzuki and M. Yamashita. Distributed anonymous mobile robots: Formation of geometric patterns. *SIAM J. Comput.*, 28(4):1347–1363, 1999.

[20] R. Vaidyanathan, C. Busch, J. L. Trahan, G. Sharma, and S. Rai. Logarithmic-time complete visibility for robots with lights. In *IPDPS*, pages 375–384, 2015.

[21] G. Viglietta. Rendezvous of two robots with visible bits. In *ALGOSENSORS*, pages 291–306, 2013.

[22] M. Yamashita and I. Suzuki. Characterizing geometric patterns formable by oblivious anonymous mobile robots. *Theor. Comput. Sci.*, 411(26-28):2433–2453, 2010.

Buttons & Scissors is NP-Complete

Harrison Gregg * Jody Leonard † Aaron Santiago‡ §Aaron Williams

Abstract

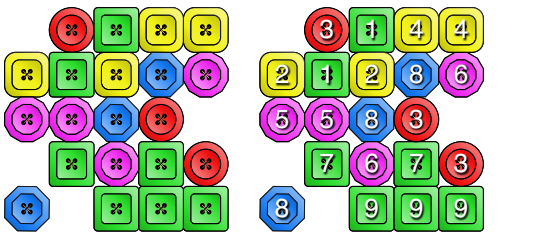
Buttons & Scissors is a popular single-player puzzle. A level is played on an n-by-n grid, where each position is empty or has a single coloured button sewn onto it. The player's goal is to remove all of the buttons using a sequence of horizontal, vertical, and diagonal scissor cuts. Each cut removes all buttons between two distinct buttons of the same colour, and is not valid if there is an intermediate button of a different colour. We prove that deciding whether a given level can be completed is NP-complete. In fact, NP-completeness holds when only horizontal and vertical cuts are allowed, and each colour is used by at most 7 buttons. Our framework was also used in an NP-completeness proof when each colour is used by at most 4 buttons, which is best possible.
Keywords: *NP-completeness, pencil-and-paper puzzle.*

1 Introduction

Buttons & Scissors is a single-player puzzle by KyWorks that is available as a free iOS and Android app. The goal is to remove all buttons from an $n \times n$ grid using a series of *scissor cuts* that have the following properties:

- a cut is a straight-line segment whose endpoints are centers of distinct buttons and which is horizontal, vertical, or diagonal at a 45° or −45° angle;
- a cut's line segment touches at least two buttons of the same color, and no buttons of another color;
- a cut removes all buttons on its line segment.

Figure 1 illustrates a sample level and solution.



(a) (b)

Figure 1: (a) Level 7 in Buttons & Scissors, and (b) a solution using nine cuts.

*hgregg11@simons-rock.edu
†jleonard11@simons-rock.edu
‡asantiago11@simons-rock.edu
§haron@uvic.ca

Buttons & Scissors is reminiscent of many grid-based pencil-and-paper puzzles that have been popularized by the Japanese company Nikoli. To analyze the computational complexity of an individual puzzle, it is necessary to generalize certain aspects of the puzzle, including its grid size. For example, the generalized version of Sudoku involves an n^2-by-n^2 grid with blocks of size n and integers from 1 to n^2 (with the standard version having $n = 3$). The book *Games, Puzzles, and Computation* by Hearn and Demaine [2] provides hardness results for many generalized grid games. We consider the following decision problem.

Decision Problem 1 B&S(B)
Input*: An n-by-n board B.*
Output*: True if B has a solution, and False otherwise.*

We also consider a cut-constrained version of Buttons & Scissors in which diagonal cuts are not allowed.

Decision Problem 2 B&S+(B)
Input*: An n-by-n board B.*
Output*: True if B has a solution using only horizontal and vertical cuts, and False otherwise.*

Notice $B\&S(B)$ is True and $B\&S+(B)$ is False for the board in Figure 1. In general, $B\&S+(B) \Rightarrow B\&S(B)$. However, *a priori*, there is no relationship between the difficulty of deciding $B\&S$ and $B\&S+$. In this article we prove that both problems are NP-complete. In fact, we achieve slightly stronger results that also constrain the number of times each distinct colour can be used.

Theorem 1 *$B\&S$ and $B\&S+$ are both NP-complete. Furthermore, both problems are NP-complete when each colour is used by at most $F = 7$ buttons.*

Section 2 describes our Buttons & Scissors gadgets, and Section 3 formalizes the version of 3-SAT that we use for Theorem 1. Section 4 provides our reduction and Section 5 proves that it is correct. Open problems and further results appear in Section 6, including an improvement of Theorem 1 to $F = 4$ for the $B\&S$ puzzle.

2 Linear Gadgets

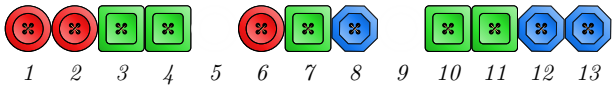
Our strategy is to prove the hardness of $B\&S$ and $B\&S+$ simultaneously using a single 3-SAT reduction. We do this by mapping instances of 3-SAT to Buttons

& Scissor boards in which no diagonal cuts are possible. More specifically, if two buttons have the same colour, then they will not lie on any common diagonal line. Towards this goal we construct gadgets whose buttons can fit on a single row or column. Linear gadgets for OR and AND are given in Sections 2.1 and 2.2, respectively.

2.1 OR Gadget

The following gadget has three Boolean inputs. Each input determines whether a given button has been removed from the board.

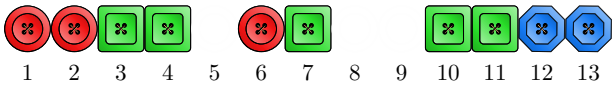
Definition 1 *Let* $\mathsf{OR}(\mathsf{X}_1, \mathsf{X}_2, \mathsf{X}_3)$ *be the following Buttons & Scissors board,*



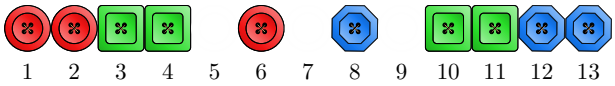
where buttons in positions $1, 2, 6$ *are one colour, those in positions* $3, 4, 7, 10, 11$ *are a second distinct colour, and those in positions* $8, 12, 13$ *are a third distinct colour. The buttons in positions* 6, 7, 8 *are absent if* $X_1 = \mathsf{T}$, $X_2 = \mathsf{T}$, $X_3 = \mathsf{T}$, *respectively.*

For example, when $X_1 = X_2 = X_3 = \mathsf{F}$, the buttons in positions $6, 7, 8$ are present, as shown below.

Notice that it is impossible to remove the button in position 7 from this board using any sequence of cuts. As another example, consider $X_1 = X_2 = \mathsf{F}$ and $X_3 = \mathsf{T}$, in which the button in position 8 is not present.



This board can be solved by successively cutting positions 3-4, then 1-6, then 7-11, then finally 12-13. For a third example, consider $X_1 = \mathsf{F}$, $X_2 = \mathsf{T}$, $X_3 = \mathsf{F}$, in which only the button in position 7 is removed.



This board can be solved by successively cutting positions 3-4, then 1-6, then 10-11, then finally 8-13.

More generally, this board can be solved if and only if at least one input is true.

Lemma 2 $B\&S(\mathsf{OR}(\mathsf{X}_1, \mathsf{X}_2, \mathsf{X}_3)) \iff X_1 \vee X_2 \vee X_3$.

Proof. The following table provides a sequence of cuts to solve the board whenever $X_1 \vee X_2 \vee X_3 = \mathsf{T}$.

X_1	X_2	X_3	Cut Sequence
T	T	T	1-2, 3-11, 12-13
T	T	F	1-2, 3-4, 10-11, 8-13
T	F	T	1-2, 3-11, 12-13
T	F	F	1-2, 3-7, 10-11, 8-13
F	T	T	3-4, 1-6, 10-11, 12-13
F	T	F	3-4, 1-6, 10-11, 8-13
F	F	T	3-4, 1-6, 7-11, 12-13

When $X_1 = X_2 = X_3 = \mathsf{F}$ it is impossible to remove the button in position 6. □

2.2 AND Gadget

The following gadget has k Boolean inputs.

Definition 2 *Let* $\mathsf{AND}(\mathsf{X}_1, \mathsf{X}_2, \ldots, \mathsf{X}_k)$ *be the following Buttons & Scissors board,*

where buttons in positions $1, 2, \ldots, k$ *have unique colours, and these colours are distinct from the common colour used by the button in position* 0 *and* $k{+}1$. *The button in position* i *is absent if* $X_i = \mathsf{T}$ *for each* $i = 1, 2, \ldots, k$.

Lemma 3 $B\&S(\mathsf{AND}(\mathsf{X}_1, \mathsf{X}_2, \ldots, \mathsf{X}_k)) \iff X_1 \wedge X_2 \wedge \cdots \wedge X_k$.

Proof. If $X_1 \wedge X_2 \wedge \ldots \wedge X_k$ is true, then the only buttons in $\mathsf{AND}(\mathsf{X}_1, \mathsf{X}_2, \ldots, \mathsf{X}_k)$ are those in positions 0 and $k{+}1$. These buttons have the same colour and can be removed with one cut, 0-$k{+}1$. If $X_1 \wedge X_2 \wedge \ldots \wedge X_k$ is false, then $\mathsf{AND}(\mathsf{X}_1, \mathsf{X}_2, \ldots, \mathsf{X}_k)$ contains at least one button with a unique colour that cannot be cut. □

3 Conventions

This section describes the version of 3-SAT that we utilize in our reduction, and our conventions for describing Buttons & Scissors boards.

3.1 3-SAT

Let $\mathcal{S}$ denote the set of 3-SAT instances that have distinct clauses and exactly three literals of distinct variables in each clause. It is easy to see that this version of 3-SAT is NP-complete, even though it varies from Karp's original formulation [3].

An arbitrary 3-SAT instance $S \in \mathcal{S}$ has variables $V_1, V_2, \ldots, V_n$ and clauses $C_1, C_2, \ldots, C_m$. We write an arbitrary clause C_x and its literals as

$$L_{i,x} \vee L_{j,x} \vee L_{k,x} \text{ where } i < j < k \text{ and } L_{i,x} \in \{V_i, \neg V_i\}$$
$$L_{j,x} \in \{V_j, \neg V_j\}$$
$$L_{k,x} \in \{V_k, \neg V_k\}.$$

In other words, $L_{a,b}$ is the literal of variable V_a that appears in clause C_b. We refer to each $L_{a,b}$ as a *literal instance*. For example, if S is the following,

$$(\mathsf{V}_1 \vee \mathsf{V}_2 \vee \mathsf{V}_3) \wedge (\mathsf{V}_1 \vee \neg\mathsf{V}_2 \vee \neg\mathsf{V}_3) \wedge (\neg\mathsf{V}_1 \vee \neg\mathsf{V}_3 \vee \neg\mathsf{V}_4),$$

then $C_1 = (\mathsf{V}_1 \vee \mathsf{V}_2 \vee \mathsf{V}_3)$ and so $L_{2,1} = V_2$. Similarly, $C_3 = (\neg\mathsf{V}_1 \vee \neg\mathsf{V}_3 \vee \neg\mathsf{V}_4)$ and so $L_{4,3} = \neg V_4$.

3.2 Buttons & Scissors Boards

Let $\mathcal{B}$ denote the set of Buttons & Scissors boards. We will use different shapes for buttons that play different roles in each $B \in \mathcal{B}$ that we construct:

- *Clause buttons* are circular ◯ (Section 4.1);
- *AND buttons* are octagonal ⯃ (Section 4.1).
- *Variable buttons* are trapezial ⏢ (Section 4.2);
- *Literal instance buttons* are rectangular □ (Section 4.3);

Each button will be uniquely identifiable by (a) its shape, (b) its label in the shape, and (c) its subscript. Our convention is that buttons have the same colour if and only if they have (a) the same shape, and (b) the same label in the shape. For example, ③$_L$ and ③$_R$ are the same colour. Similarly, [1,2]$_L$ and [1,2]$_R$ are the same colour. However, [3/]$_L$ and [4/]$_L$ are different colours.

4 Reduction

This section describes our reduction $r : \mathcal{S} \to \mathcal{B}$ that maps an instance of 3-SAT $S \in \mathcal{S}$ to a Buttons & Scissors board $B = r(S) \in \mathcal{B}$. Sections 4.1–4.3 describe buttons in B resulting from clauses, variables, and literal instances, respectively. The layout of the entire board is then discussed in Section 4.4, and various properties of the constructed board are given in Section 4.5.

Figure 2 contains both a high-level view of our reduction, as well as a specific example.

4.1 Clauses

Each clause in the 3-SAT instance is translated into its own row of buttons in the board B (see Figure 2). In particular, $C_x = L_{i,x} \vee L_{j,x} \vee L_{k,x}$ with $i < j < k$ contains the following OR gadget

[i]$_{L_1}$ [i]$_{L_2}$ [j]$_{L_1}$ [j]$_{L_2}$ [i]$_M$ [j]$_M$ [k]$_M$ [j]$_{R_1}$ [j]$_{R_2}$ [k]$_{R_1}$ [k]$_{R_2}$

Each of these rectangular literal buttons has the additional label [,x] which is not shown for space reasons. Also, buttons corresponding to negative literals have an overline which is not indicated above (see Figure 2). Notice that there are three distinct button colours above and they follow the OR gadget pattern. The subscripts for each literal instance colour are Left (twice for i and j), Middle (once each), and Right (twice for j and k). The horizontal positioning of the middle buttons is below its respective variable positive/negative column as discussed in Section 4.2. An additional pair of Cleanup buttons for each colour are placed above each middle button.

The row of buttons for clause C_x contains one additional pair of clause buttons as bookends.

ⓧ$_L$ [i]$_{L_1}$ [i]$_{L_2}$ $\cdots$ [k]$_{R_1}$ [k]$_{R_2}$ ⓧ$_R$

The subscripts for these clause buttons are Left and Right. These are the only two buttons of this colour on the entire board B. Therefore, by Lemma 2 we have the following remark.

Remark 1 *The clause buttons ⓧ$_L$ and ⓧ$_R$ can only be removed after all of the buttons in the OR gadget for clause C_x are removed.*

We place all of the Left clause buttons within an AND gadget as follows

◯$_1$ ①$_L$ ②$_L$ ③$_L$ $\cdots$ ⓜ$_L$ ◯$_2$

which appears as a single vertical column in B (see Figure 2). The AND buttons ◯$_1$ and ◯$_2$ are the only such pair on B, and they share a unique colour. Therefore, the previous remark and Lemma 3 imply the following.

Remark 2 *The AND buttons ◯$_1$ and ◯$_2$ can only be removed after all of the clause buttons are removed.*

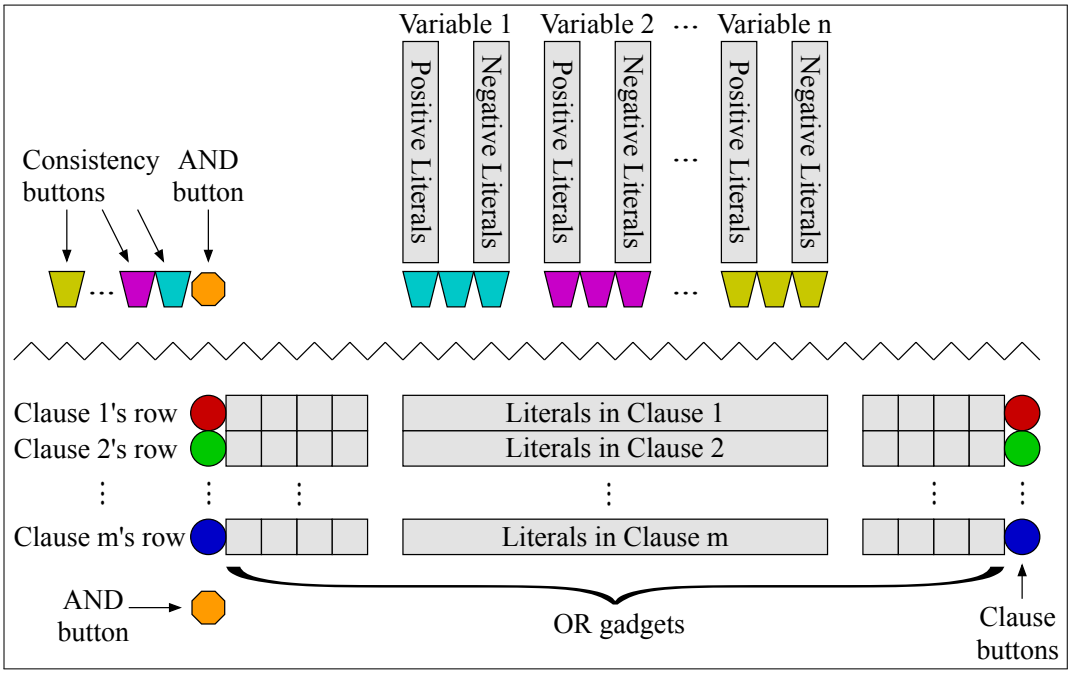
4.2 Variables

The following row of buttons appears in B

[n/]$_C$ $\cdots$ [2/]$_C$ [1/]$_C$ ◯$_1$ [1/]$_P$ [1/]$_D$ [1/]$_N$ [2/]$_P$ [2/]$_D$ [2/]$_N$ $\cdots$ [n/]$_P$ [n/]$_D$ [n/]$_N$.

There are four buttons for each variable, with subscripts for Positive, Decision, Negative, and Consistency. Each quartet has its own distinct colour that is not used by any other buttons on B, and the consistency button is separated from the others by the first AND button.

As we will see in Section 5, this configuration ensures that a variable's decision button will either be cut with its positive button, or its negative button, and this choice will correspond to setting the variable equal to T or F, respectively. (If all four buttons are cut simultaneously, then the variable is 'free' and can be assigned either way in a satisfying assignment.)



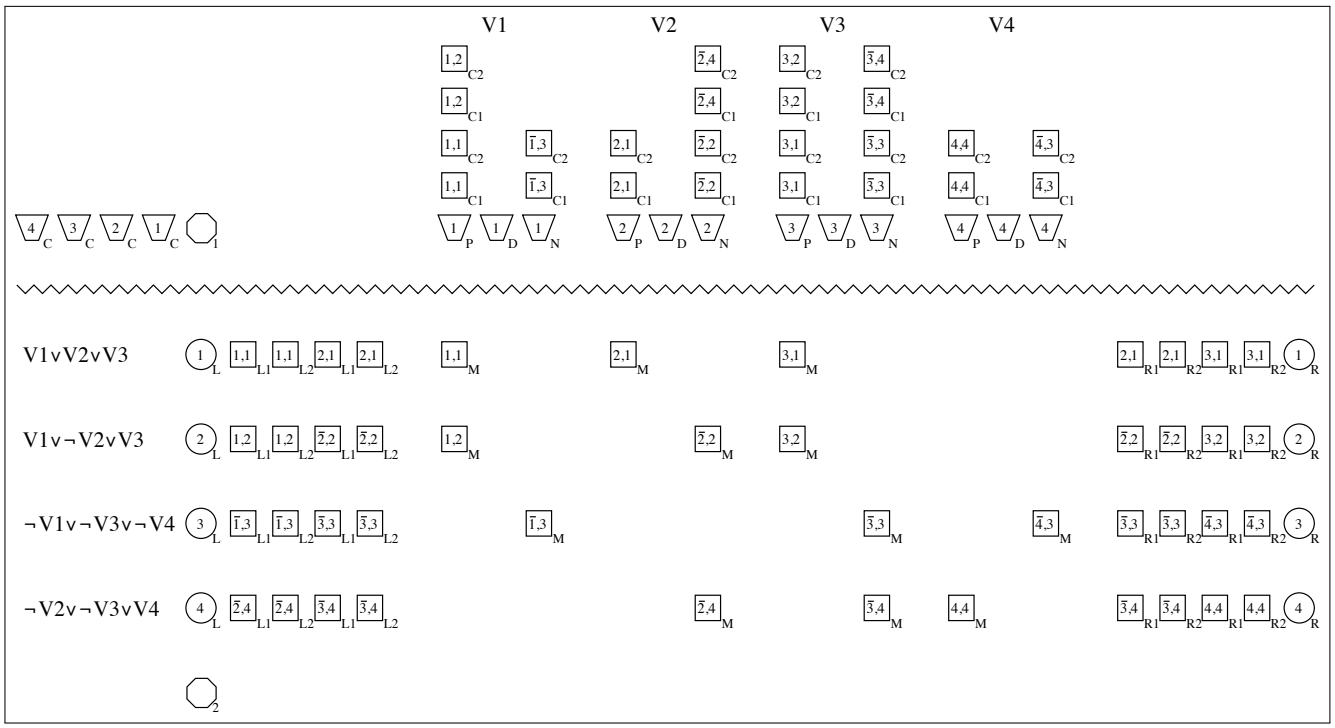
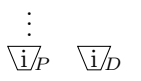


Figure 2: Top: An overview of our reduction. The gray squares contain buttons whose colours are not specified in the figure, and the gray rectangles contain blank spaces and buttons whose colours are not specified. Bottom: $(V_1 \vee V_2 \vee V_3) \wedge (V_1 \vee \neg V_2 \vee \neg V_3) \wedge (\neg V_1 \vee \neg V_3 \vee \neg V_4) \wedge (\neg V_2 \vee \neg V_3 \vee V_4)$. The zig-zags denote a series of empty rows.

4.3 Literal Instances

For each variable V_i, pairs of Cleanup buttons for each of its positive and negative literal instances are stacked vertically above its positive button $\boxed{\text{i}}_P$ and negative button $\boxed{\text{i}}_N$, respectively, as follows



More precisely, if V_i appears as a positive literal in clauses $C_{p_1}, C_{p_2}, \ldots, C_{p_x}$ for $p_1 < p_2 < \cdots < p_x$, then the buttons above the variable's positive button appear as follows (with left-to-right being bottom-to-top)

$$\boxed{\text{i}}_P \; \boxed{\text{i},p_1}_{C_1} \; \boxed{\text{i},p_1}_{C_2} \; \boxed{\text{i},p_2}_{C_1} \; \boxed{\text{i},p_2}_{C_2} \; \cdots \; \boxed{\text{i},p_x}_{C_1} \; \boxed{\text{i},p_x}_{C_2}$$

where $\boxed{\text{i}}_P$ is at the bottom of this vertical stack. Similarly, if V_i appears as a negative literal in clauses $C_{n_1}, C_{n_2}, \ldots, C_{n_x}$ for $n_1 < n_2 < \cdots < n_x$, then the buttons above the variable's negative button appear from left-to-right as follows

$$\boxed{\text{i}}_N \; \boxed{\text{i},n_1}_{C_1} \; \boxed{\text{i},n_1}_{C_2} \; \boxed{\text{i},n_2}_{C_1} \; \boxed{\text{i},n_2}_{C_2} \; \cdots \; \boxed{\text{i},n_x}_{C_1} \; \boxed{\text{i},n_x}_{C_2}$$

where $\boxed{\text{i}}_N$ is at the bottom of this vertical stack.

These cleanup pairs appear in the same column and in opposite order as the literal instance buttons in their respective clauses (see Figure 2). Therefore, we have the following remark.

Remark 3 *If variable V_i's positive button $\boxed{i}_P$ is removed from B, then all buttons in V_i's positive column can be removed. Similarly, if $\boxed{i}_N$ is removed, then all buttons in V_i's negative column can be removed.*

4.4 Layout

In our construction, the only button colours that appear on more than one single row or column are the literal instance buttons $\boxed{\text{i,j}}$. More specifically, a button of colour $\boxed{\text{i,j}}$ appears both on clause C_j's row and in variable v_i's positive or negative literal column. To avoid possible diagonal cuts between these buttons, we include a series of blank rows between the clause rows and the variable row. By our construction, a total of $4n + 6$ blank rows ensures this property since the leftmost literal instance button in clause C_1's row is strictly below the bottom literal instance button in variable V_n's negative literal column, with respect to a 45° diagonal line.

4.5 Properties

We conclude this section by summarizing a number of simple properties of our reduction.

Remark 4 *If two buttons in B have the same colour, then they do not lie on the same diagonal line.*

Remark 5 *If $S \in \mathcal{S}$ has n variables and m clauses, then the board $B = r(S)$ is an $O(n+m)$-by-$O(n)$ grid.*

Remark 6 *Each colour is used by at most 7 buttons. In particular, variable buttons $\boxed{i}$ appear 4 times, clause buttons (j) appear twice, AND buttons ◯ appear twice, and instance literals buttons $\boxed{i,j}$ appear 5 or 7 times.*

5 NP-Completeness

In this section we prove Theorem 1. That is, $B\&S$ and $B\&S+$ are both NP-complete. First we demonstrate that the problems are in NP.

Lemma 4 *$B\&S$ and $B\&S+$ are in NP.*

Proof. Suppose $B \in \mathcal{B}$ is an n-by-n board. In both versions of the puzzle, a cut can be specified by two grid co-ordinates, and at most $O(n^2)$ cuts are necessary to clear B. Therefore, a sequence of cuts that solves B can be specified in polynomial-size with respect to B. It is also clear that we can verify if a sequence of cuts clears a board in polynomial-time. Therefore, a sequence of cuts provides a polynomial-size certificate that can be verified in polynomial-time when $B\&S(B) = \mathsf{T}$ or $B\&S+(B) = \mathsf{T}$. □

Our reduction creates a polynomial-size board by Remark 5. Therefore, to complete the proof of Theorem 1 we need to prove that if $S \in \mathcal{S}$ and $B = r(S)$ then

$$S \text{ is satisfiable} \iff B\&S(B) \text{ and}$$
$$B\&S(B) \iff B\&S+(B).$$

By Remark 4 there are no diagonal cuts in B, so $B\&S(B) \iff B\&S+(B)$ has been established. We prove the first $\iff$ in the following two subsections.

5.1 Clearing the board

In this subsection we provide a solution for the Buttons & Scissors board given a satisfying assignment to the 3-SAT problem.

Lemma 5 *Suppose $S \in \mathcal{S}$ and $B = r(S)$. If S is satisfiable, then $B\&S(B) = \mathsf{T}$.*

Proof. Consider a fixed satisfying assignment for S. We now provide a sequence of cuts that clears B.

1. Perform the following for each $i = 1, 2, \ldots, n$:
   - If $V_i = \mathsf{T}$ in the satisfying assignment, then cut $\boxed{\text{i}}_P$ and $\boxed{\text{i}}_D$. Then cut all buttons in variable V_i's positive literal column by Remark 3.
   - Otherwise, cut $\boxed{\text{i}}_N$ and $\boxed{\text{i}}_D$. Then cut all buttons in variable V_i's negative column.
2. Perform the following for each $j = 1, 2, \ldots, m$:

- Remove every button in the OR gadget on clause C_j's row.
- Remove the clause buttons (j)_L and (j)_R.

The first step is possible by Lemma 2 and the fact that we stared with a satisfying assignment. The second step is possible by Remark 1 .

3. Cut the AND buttons $\bigcirc_1$ and $\bigcirc_2$ by Remark 2.
4. Perform the following for $i = 1, 2, \ldots, n$ in order:
   - If $V_i = \mathsf{T}$ in the assignment, then cut $\overline{\text{i}}_C$ and $\overline{\text{i}}_N$. Then cut all buttons in variable V_i's negative literal column by Remark 3.
   - Otherwise, cut $\overline{\text{i}}_C$ and $\overline{\text{i}}_P$. Then cut all buttons in variable V_i's positive literal column by Remark 3.

The cuts remove all buttons, so $B\&S(B) = \mathsf{T}$. □

5.2 Satisfying the formula

Now we provide a satisfying assignment to the 3-SAT problem given that its equivalant board is solvable. Lemma will allow us to map a solution to the Buttons & Scissors board to a 3-SAT variable assignment.

Lemma 6 *Consider a sequence of cuts that clears board $B = r(S)$. For each variable V_i, the variable buttons for V_i are cut in one of three ways:*

1. *First $\overline{i}_P$ and $\overline{i}_D$ are cut. Then $\overline{i}_C$ and $\overline{i}_N$ are cut after the two AND buttons are cut.*
2. *First $\overline{i}_D$ and $\overline{i}_N$ are cut. Then $\overline{i}_C$ and $\overline{i}_P$ are cut after the two AND buttons are cut.*
3. *All four buttons — $\overline{i}_C$, $\overline{i}_P$, $\overline{i}_D$ and $\overline{i}_N$ — are cut together after the two AND buttons are cut.*

Proof. The relative order of these buttons is

$$\overline{\text{i}}_C \quad \bigcirc_1 \quad \overline{\text{i}}_P \quad \overline{\text{i}}_D \quad \overline{\text{i}}_N.$$

Thus, the three cases follow immediately. □

We now complete our proof of Theorem 1.

Lemma 7 *Suppose $S \in \mathcal{S}$ and $B = r(S)$. If $B\&S(B) = \mathsf{T}$, then S is satisfiable.*

Proof. Let $c_1, c_2, \ldots, c_k$ be a sequence of cuts that clears B. By Lemma 6, we can create a variable assignment for S as follows:

- If the first case occurs, then set $V_i = \mathsf{T}$.
- If the second case occurs, then set $V_i = \mathsf{F}$.
- Otherwise, the choice is arbitrary and set $V_i = \mathsf{T}$.

We will prove that this assignment is satisfying.

Consider the cut c_a that removes the AND buttons $\bigcirc_1$ and $\bigcirc_2$. Prior to c_a, all clause buttons must have been removed by Remark 2. Therefore, by Remark 1 all of the OR gadget buttons must have been removed prior to c_a. Therefore, by Lemma 2 and the construction of B, at least one of the central buttons in each OR gadget must have been removed prior to c_a by some vertical cut. Therefore, we have the following prior to c_a for each clause C_j: There exists a variable V_i such that $V_i = \mathsf{T}$ and the literal V_i is in C_j and its button $\boxed{\text{i,j}}_M$ was removed, or $V_i = \mathsf{F}$ and the literal $\neg V_i$ is in C_j and its button $\boxed{\text{i,j}}_M$ was removed. Therefore, the variable assignment satisfies S. □

6 Additional Results and Open Problems

Buttons & Scissors has a number of natural variations including the following colour-constrained versions:

1. There are at most C distinct colours of buttons.
2. Each colour can be used by at most F buttons.

We proved $B\&S$ and $B\&S+$ are NP-complete for $F = 7$ in Theorem 1. We also conjectured NP-completeness for $F = 4$ in our initial submission. This was recently verified by a second research group for the $B\&S$ puzzle.

Theorem 8 ([1]) *$B\&S$ is NP-complete when each colour is used by at most $F = 4$ buttons.*

The proof of Theorem 8 uses our reduction with a new OR gadget (see Figure 2 in [1]). The new gadget uses colours less frequently but requires all four cut directions. Thus, the hardness of $B\&S+$ with $F = 4$ is still open. The $F = 4$ cases are particularly interesting because $B\&S$ and $B\&S+$ are polytime solvable when $F = 3$. To see this, notice that all buttons of a given colour must be removed by a single cut when $F = 3$. Furthermore, removing all buttons of a given colour cannot turn a solvable board into an unsolvable board. Thus, a simple greedy algorithm suffices.

Our initial submission also conjectured hardness when $C = 2$, and this was also recently verified in [1]. A full journal article with the authors of [1] is also planned.

An implementation of our reduction is available: http://jabdownsmash.com/button3sat/index.html.

References

[1] K. Burke, E. Demaine, R. Hearn, A. Hesterberg, M. Hoffman, H. Ito, I. Kostitsyna, M. L offler, Y. Uno, C. Schmidt, R. Uehara, and A Williams. Single-player and two-player buttons & scissors games. In *Proceedings of the 18th Japan Conference on Discrete and Computational Geometry and Graphs*, page 2 pages, 2015.

[2] R. A. Hearn and E. D. Demaine. *Games, Puzzles, & Computation*. A K Peters, 1st edition, 2009.

[3] Richard M. Karp. Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103, 1972.

Computational complexity of numberless Shakashaka

Aviv Adler* Michael Biro† Erik Demaine* Mikhail Rudoy ‡ Christiane Schmidt§

Abstract

Shakashaka, like Sudoku, is a pencil-and-paper puzzle. In this paper we show that Shakashaka is NP-complete in the case of numberless black squares.

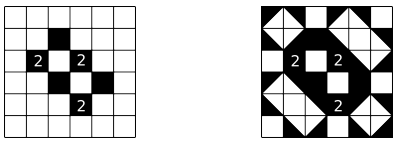
1 Introduction

Shakashaka is a pencil-and-paper puzzle, proposed by Guten in 2008 and popularized by the Japanese publisher Nikoli [1].

An instance of Shakashaka consists of an $m \times n$ rectangle of unit squares. Initially, each square is colored either black or white, and black squares may also contain an integer between 0 and 4, inclusive. The solver proceeds by filling in the initially white squares with squares consisting of a black and a white triangle in one of four orientations: ◢ ◣ ◤ ◥. We denote these collectively as *b/w squares*. The white squares may also be left blank. In addition, the numbers written in black squares constrain the solver by specifying the number of b/w squares that must neighbor the given square (in its four vertically and horizontally neighboring squares).

An instance is considered *solved* if every maximal connected white region on the board is a rectangle (axis-aligned or rotated by 45°) and each numbered black square has exactly as many b/w square neighbors as is specified by its number. For an example and its (unique) solution, refer to Figure 1.

Demaine et al. [3] proved that Shakashaka is NP-complete. They used a reduction from planar 3-SAT, and the black squares in the reduction either contained the number 1 or remained blank. In addition, they showed that Shakashaka can be formulated as a 0-1-integer program and gave experiments using IP-solvers (namely SCIP 3.0.0) to solve instances of sizes up to 20×36. Two questions remained in the concluding remarks of this paper, one concerned settling the computational complexity of Shakashaka without numbers in the black squares.



(a) (b)

Figure 1: (a) An instance of Shakashaka and (b) its unique solution.

In this paper we answer this question and show that Shakashaka without numbers in the black squares is NP-complete by a reduction from POSITIVE PLANAR 1-IN-3 SAT, a variant of the well known PLANAR 3-SAT problem, shown to be NP-complete by Mulzer and Rote [5]. The reduction is parsimonious, and, hence, also shows #P-completeness. We also include an easier, but non-parsimonious, reduction from PLANAR 3-SAT, which is well-known to be NP-complete.

2 NP-completeness of numberless Shakashaka

Definition 1 *An instance F of the POSITIVE PLANAR 1-IN-3 SAT problem is a Boolean formula in 3-CNF: it consists of a set $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ of m clauses over n variables $\mathcal{V} = \{x_1, x_2, \ldots, x_n\}$, where each clause C_i consists of three variables ("POSITIVE" indicates that no negated variables appear in the C_i's). Moreover, the variable-clause incidence graph $G = (\mathcal{C} \cup \mathcal{V}, E)$ is planar, where $\{C_i, x_j\} \in E \Leftrightarrow x_j$ is in C_i. It is sufficient to consider formulae where G has a rectilinear embedding, see Knuth and Raghunathan [4]. The POSITIVE PLANAR 1-IN-3 SAT problem is to decide whether there exists a truth assignment to the variables such that exactly one variable in each clause is true.*

Theorem 1 *Shakashaka without numbers in the black squares is NP-complete.*

Proof. [via Positive Planar 1-in-3 SAT]

The proof is by reduction from POSITIVE PLANAR 1-IN-3 SAT, which was shown to be NP-complete by Mulzer and Rote [5]. Let F be an instance of the POSITIVE PLANAR 1-IN-3 SAT problem. We turn the rectilinear embedding of G into a Shakashaka board: we present the variables, clauses and edges by pieces of

*Computer Science and Artificial Intelligence Laboratory, MIT, Massachusetts, USA. Email: adlera@mit.edu, edemaine@mit.edu.

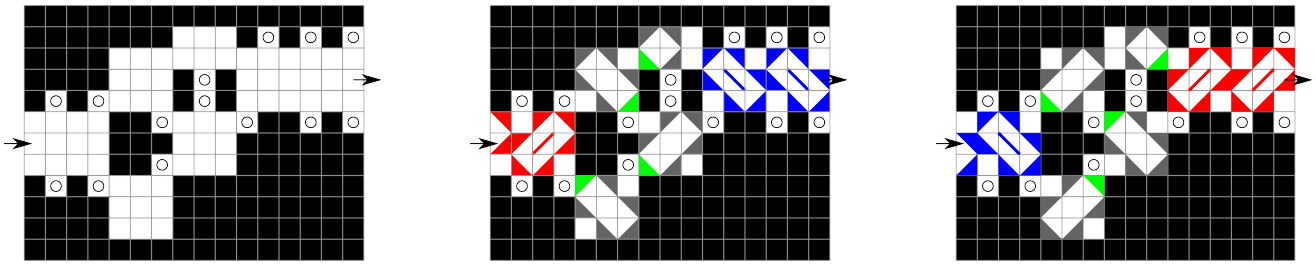
†Department of Mathematics and Statistics, Swarthmore College, USA. Email: mbiro1@swarthmore.edu.

‡Electrical Engineering and Computer Science department and Mathematics department, MIT, Massachusetts, USA. Email: mrudoy@mit.edu.

§The Rachel and Selim Benin School of Computer Science and Engineering, The Hebrew University of Jerusalem, Israel. Email: cschmidt@cs.huji.ac.il. Supported by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11).

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

Figure 2: (a) The variable gadget, (b) with enforced white pixels and (c),(d) the two possible feasible solutions. We associate the "kite" in blue with a truth setting of "false" and the "kite" in red with a truth setting of "true".



(a) (b) (c)

Figure 3: (a) The NOT gadget. (b),(c) The wires connected by the NOT gadget always satisfy opposite truth assignments. In (b) the gadget is entered with a truth assignment corresponding to "true" and left with a truth assignment corresponding to "false". Those roles are reversed in (c). Some enforced triangles are shown in green to facilitate understanding.

the board that need to be filled in. We will give the details of the gadgets in the following.

The **variable gadget** is shown in Figure 2(a). The empty circles in Figure 2(b) indicate (by the construction) enforced white pixels. There exist exactly two feasible solutions for the variable gadget, shown in blue and red in Figures 2(c) and (d) and corresponding to a truth setting of "false" and "true", respectively. In both cases we use a "kite", a sloped structure, occupying 7 out of the pixels of a 3×3-square. For the blue solution, the kite is oriented from top left to bottom right, for the red solution it is oriented from top right to bottom left (both indicated by a line in the rectangle's center).

The initial truth value is propagated by a **wire gadget** as indicated in Figure 2.

Parity. Note that, by construction, the kites propagating through the wires (and all other gadgets below) do so at regular intervals of three squares in both the up/down and left/right directions; i.e., the kites that propagate the truth assignments each fit inside a 3×3-square. In fact, the gadgets of our construction force these kites to be placed inside the tiles of a single 3×3-tiling of the plane. This ensures that the gadgets can be constructed and the kites will align without any shifting.

The **NOT gadget**, shown in Figure 3, enables us to reverse the truth assignment in a variable wire. The NOT gadget will be used in the bend gadget and split gadget as a black box, and is only used there.

The **bend gadget**, shown in Figure 4, enables us to bend a wire to match bends in the rectilinear embedding of G while enforcing that the same truth assignments continue to propagate along the wire.

The **split gadget**, shown in Figure 5, enables us to increase the number of wires propagating the truth assignment of a variable gadget.

The **at-most gadget**, shown in Figure 6, enables us to enforce that at most one of a pair of truth assignments is true. It admits a feasible Shakashaka board in all cases except for two true inputs, in which case an infeasible Shakashaka board is obtained.

The related **at-least gadget**, shown in Figure 7, enables us to enforce that at least one of a pair of truth assignments is true. It admits a feasible Shakashaka board in all cases except for two false inputs, in which case an infeasible Shakashaka board is obtained.

The **"XOR" gadget**, shown in Figure 8, takes two

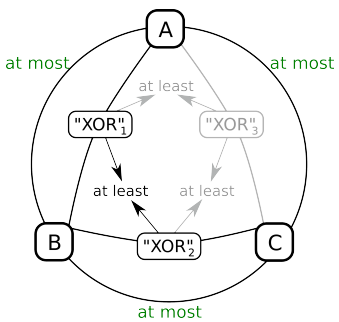
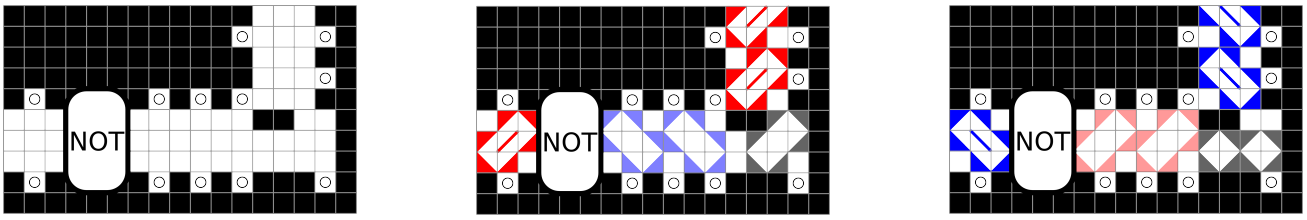
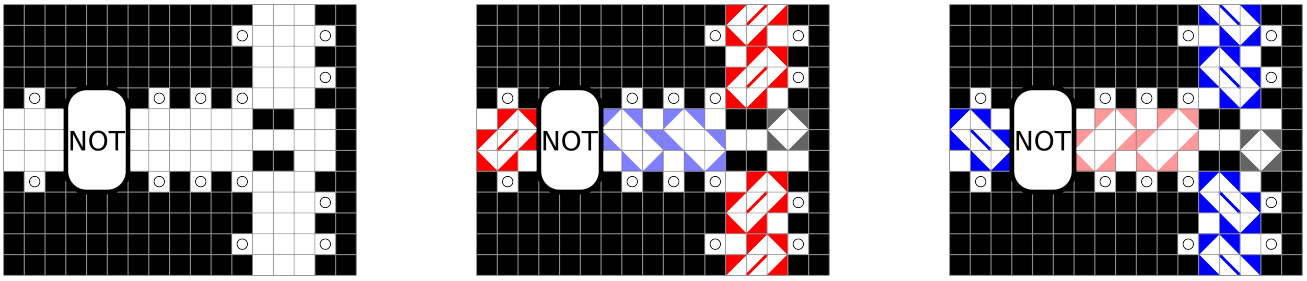


Figure 9: The clause gadget. The gray components ensure that the reduction is parsimonious.



(a) (b) (c)

Figure 4: (a) The bend gadget. (b),(c) The wires connected by the bend always satisfy the same truth assignment.



(a) (b) (c)

Figure 5: (a) A split of the corridor. (b),(c) The wires connected by the split always satisfy the same truth assignment.

wires as input and outputs:

false/false	→	false
false/true	→	true or false possible
true/false	→	true or false possible
true/true	→	infeasible.

Finally, the **clause gadget**, shown in Figure 9, enforces that exactly one of the three variables included in the clause is set to true. Three variables, represented by A, B and C in Figure 9, are pairwise combined by the at-most gadget. This combination can only be solved if there is at most one true variable among A, B, and C (i.e. the possibilities are false/false/false, true/false/false, false/true/false, and false/false/true). Consequently, we only need to exclude the false/false/false case. We combine each of two pairs of variables with an "XOR" gadget ("XOR"$_1$ and "XOR"$_2$) and combine the results in the at-least gadget. Note that the "XOR" gadgets would yield an infeasible Shakashaka board for two true inputs, but this case has already been excluded.

If all variables are set to false, both "XOR" gadgets must output false. The subsequent combination of the two "XOR" outputs with an at-least gadget results in an infeasible Shakashaka board. If one variable is set to true, at least one "XOR" gadget can output true. Therefore, the subsequent combination of the two "XOR" outputs with an at-least gadget is possible and does not render the board infeasible.

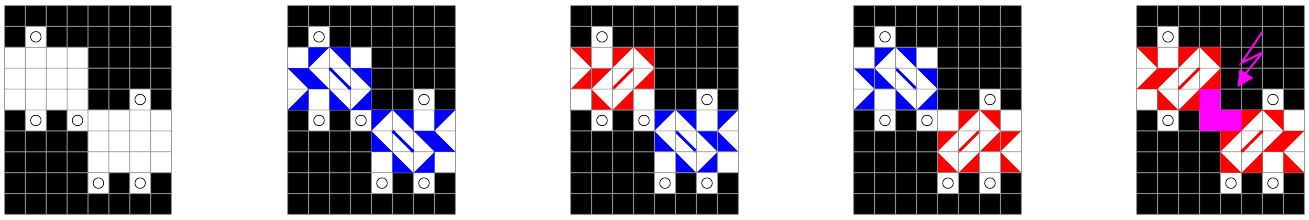
Thus, the resulting Shakashaka has a solution if and only if exactly one variable per clause is set to true, that is, if and only if the original POSITIVE PLANAR 1-IN-3 SAT formula F is satisfiable. It is easy to see, that this reduction is possible in polynomial time. Moreover, given a filled in board it is easy to check whether it constitutes a feasible Shakashaka solution, hence, Shakashaka is in the class NP. Consequently, Shakashaka without numbers in the black squares is NP-complete. So, this gadget, i.e., everything except the gray part in Figure 9, yields the Theorem's statement. But, we want to obtain a parsimonious reduction: once we fix an assignment of F, the filling pattern of the resulting Shakashaka instance is uniquely determined.

Given that the only possible combinations are the possibilities are false/false/false, true/false/false, false/true/false, and false/false/true, one of the "XOR"$_i$ has input false/false, hence, it has to output false. The other two "XOR"$_j$, "XOR"$_k$ ($i \neq j \neq k, i, j, k \in \{1, 2, 3\}$) have input true/false or false/true, i.e., they output either true or false. But then we combine all pairs of "XOR" outputs with an at-least gadget: "XOR"$_i$ outputs false, thus, each of "XOR"$_j$ and "XOR"$_k$ must output true to obtain a feasible board. Hence, we have a one-to-one correspondence between solutions to F and the resulting Shakashaka instance, i.e., the reduction is parsimonious. □

Because the counting version of POSITIVE PLANAR 1-IN-3 SAT is #P-complete [2], we have:

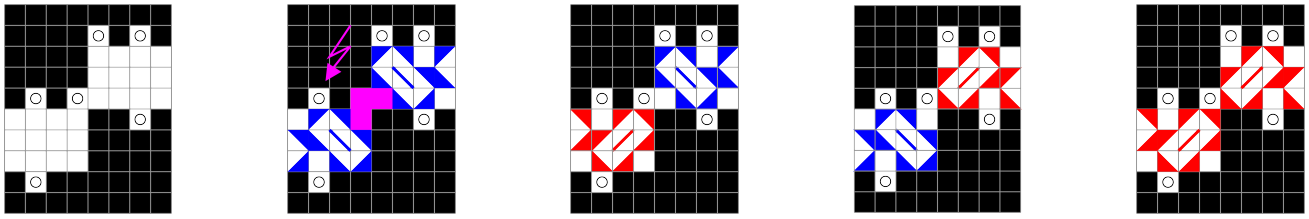
Corollary 1 *The counting version of Shakashaka is #P-complete.*

Definition 2 *An instance F of the PLANAR 3-SAT problem is a Boolean formula in 3-CNF consisting of a set $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ of m clauses over n variables $\mathcal{V} = \{x_1, x_2, \ldots, x_n\}$. Clauses in F contain variables*



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

Figure 6: (a) The "at most" gadget. (b) with two false inputs, (c)/(d) with one true and one false input, (e) with two true inputs the board cannot be completed.



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

Figure 7: (a) The "at least" gadget: (b) with two false inputs the board cannot be completed, (c)/(d) with one true and one false input, (e) with two true inputs.

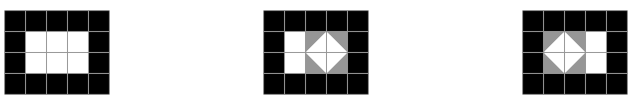
and negated variables, denoted as literals *(e.g. 'x_1' or '$\neg x_7$'). A clause is satisfied if and only if it contains at least one* `true` *literal, and the formula F is true if and only if all its clauses are satisfied. The variable-clause incidence graph G is planar and it is sufficient to consider formulae where G has a rectilinear embedding.*

We now present the alternate reduction.

Proof. [via Planar 3-SAT]

In this proof, we give a reduction from PLANAR 3-SAT. We turn the rectilinear embedding of G into a Shakashaka board, much along the lines of the previous proof. We will first discuss the basic representation of variables in this reduction, and then show how this can be used to build wires (which are very easy to split and negate) and clauses.

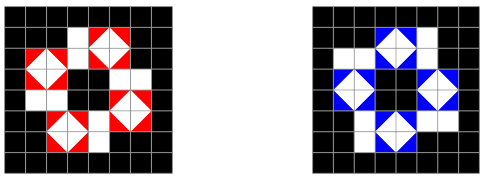
The basic variable gadget is a 2×3 rectangle, which can be filled in 3 ways, as shown in Figure 10; by looping it as shown in Figure 11, we can eliminate the trivial solution of Fig. 10(a), giving the two solutions shown in Fig. 10(b),(c) (the orientations of the 2×2 groups in each 2×3 block must match because otherwise there will be an 'L'-shaped white tetromino). We can set a given loop of this kind to represent each variable x_i; the variable is set to `true` if the 2×2 half-filled group is in the clockwise direction, and `false` if the 2×2 half-filled group is in the counterclockwise direction. There cannot be any other solutions because the edge shared between any pair of 2×3 blocks is only 1 wide, and hence no diagonally-oriented white rectangles can fit through (otherwise they would need to have a width of less than 1 which is impossible in Shakashaka). As long as our attachments are only 1 tile wide, this will prevent white rectangles from bridging two adjacent blocks and ensure that the only solutions possible are those described here.



(a) (b) (c)

Figure 10: Basic variable gadget with possible solutions.

There are 3 different patterns of loops, as shown in Figure 12, so as to allow maximum flexibility in our wire construction (shown in Figure 13); these all satisfy the property of having exactly two solutions, which differ in the orientation of the 2×2 groups. Wires are built by attaching loops together to form chains; each chain consists of a series of loops, each adjacent pair of which shares a 2×3 block; in each chain is a special 'variable' loop, which is where the variable setting is read. We note that these loops alternate between having the 2×2 group in the clockwise direction and in the counterclockwise direction; we refer to a loop as being 'synced'



(a) (b)

Figure 11: Variable loop with a truth setting of (a) true (red) and (b) false (blue).

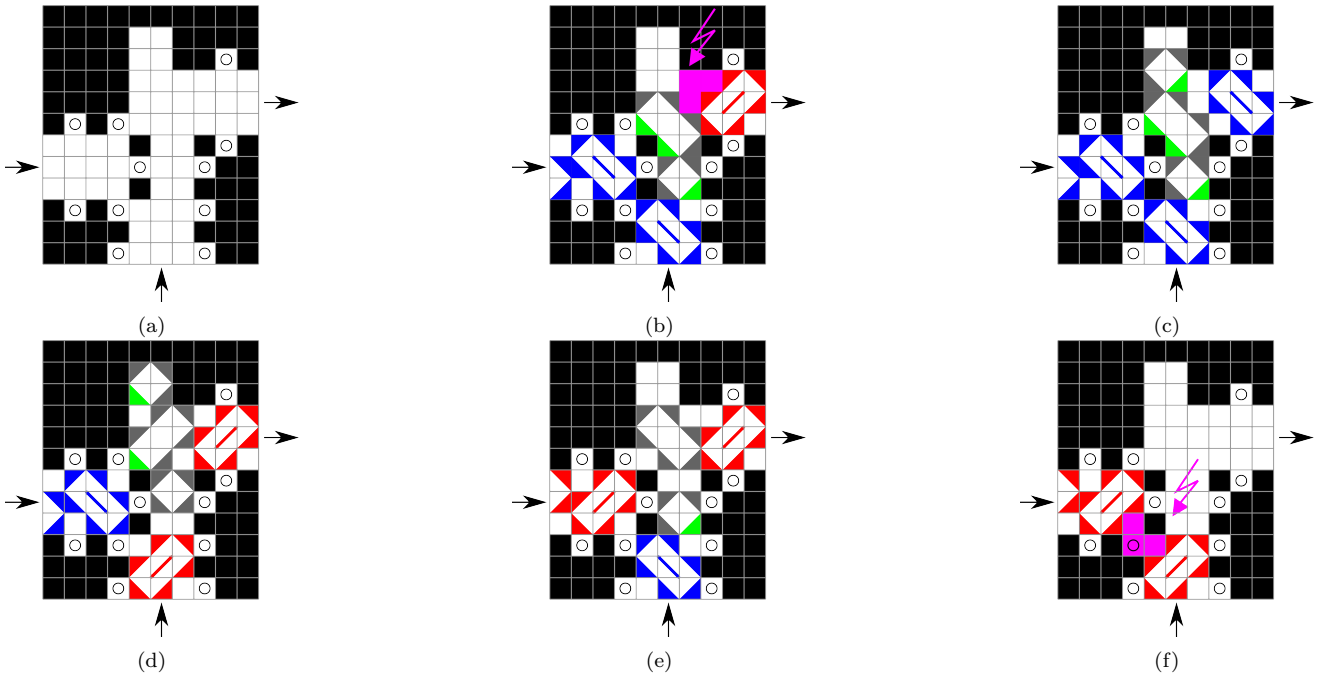


Figure 8: (a) The "XOR" gadget. (b)/(c) two false inputs cannot be completed for a true output (infeasible Shakashaka board indicated in purple), but may be completed for a false output. (d)/(e) both true/false false/true combinations allow a true output, (f) two inputs of true result in an infeasible Shakashaka board. Enforced triangles are shown in green.

if it shares its orientation with the variable loop (i.e. if it is an even distance away in the chain); otherwise it is 'de-synced' and has the opposite orientation. Thus, once the 'variable' loop contained in the chain is set, every other loop within the chain is forced into a particular orientation depending on whether it is synced with the variable loop, as sketched in Figure 13.

We also note that it is very easy to bend a chain (by attaching the next loop to one of the 2×3 blocks to the side rather than to the one opposite to the previously-shared 2×3 block) and to split a chain (by simply attaching two loops to one in a T-junction). As before, the parity of the distance of each loop in the chain from the variable loop determines whether it is synced or de-synced.

The clause gadget is as shown in Figure 14(a); each of the three 2×3 "input" blocks (denoted a, b, and c) is attached to a corresponding variable chain. The attachment occurs at a synced loop if the literal represented is not negated, and at a de-synced loop if the literal represented is negated (i.e. to represent 'x_i' we attach to a synced loop, and to represent '$\neg x_i$' we attach at a de-synced loop). Because of the multiple loop patterns, by increasing the scale of the board by a constant factor we can allow space for the chains to correct the offset and allow the correct parity loop to be in the given spot. If the literal is attached to any of the three 'input' blocks is `false`, the attached loop will be in the `false` state, thus forcing the 2×2 group in the 2×3 'input' block to be placed away from the main body of the clause gadget; if the literal is `true` then there is a choice of where

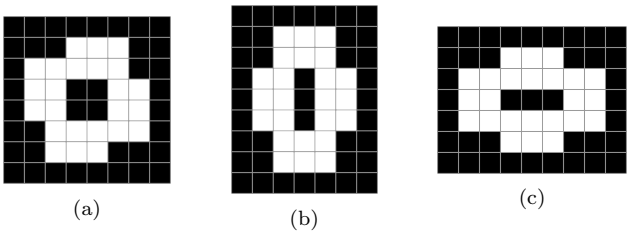


Figure 12: The different patterns for variable loops.

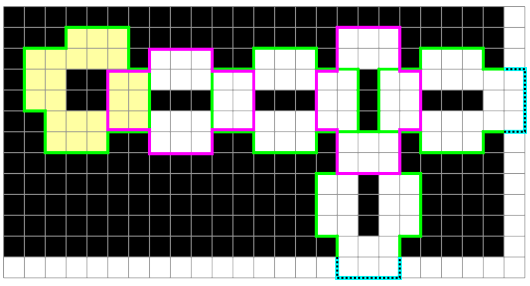
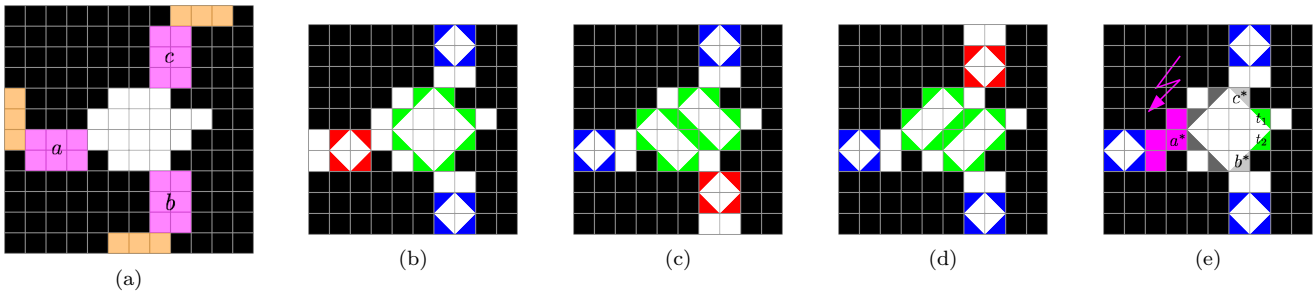


Figure 13: The variable loop is shaded yellow. Synced loops are indicated in green, de-synced loops in pink. Chain continuations are shown in turquoise; note that by using the different attachment points to a loop, bends and splits can be achieved (as shown).



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

Figure 14: (a) The clause gadget. The input blocks (a, b, c) are indicated in pink, the chains feeding those blocks in orange. Note that the clause still works if input block a is rotated to be vertical (so in panel (e), the disallowed pink region would be a rotated 'L'); this allows extra flexibility in connecting the clauses up to the chains. (b)-(d) The board for one true (red) and two false (blue) literals. (e) For three false literals (blue) the board cannot be completed.

the 2×2 group can be (either towards or away).

To get the clause gadgets to conform to the format given by Knuth and Raghunathan [4], we can wind the chain connecting at the top of the gadget over the right side of the clause's main body (with two 90° bends), such that the three chains come in from the bottom.

We now need to show the following:

1. If at least one of the three literals is `true`, the clause can be satisfied (there is a solution in the gadget).
2. If all of the three literals are `false` then there is no solution within the clause (thus preventing the whole Shakashaka instance from being solved).

This would mean that the Shakashaka board generated has a solution if and only if the formula F is satisfiable.

We note that since a `true` literal can be made to mimic a `false` literal (by allowing placement of the 2×2 group away from the body of the clause), we only need to show that the clause is satisfiable if exactly one of the three literals is `true`; this is because if more than one is `true`, we can have one of the satisfied literals act as `true` and the others mimic `false`. This allows us to handle (1) by Figure 14(b)-(d).

The second result is then shown by the following, as depicted in Figure 14(e). Since each literal is `false` in this case, we are forced to put all three 2×2 groups away from the main body of the clause. The tiles highlighted in green (t_1 and t_2) are forced to be filled in the given way. This is because if t_2 is left blank, then t_1 cannot be left blank as it would create a non-rectangular white shape; but t_1 in this case cannot be filled either, as each of the four orientations of fill result in a non-right angle, again violating the rectangular shape constraint. We then consider the tiles adjacent to the three 'input' blocks; we refer to them as a^*, b^*, and c^* (which are next to inputs a, b, and c respectively). None of these can be left blank (or else there is 270° angle, which is not allowed). However, b^* and c^* can only be filled in the manner shown, as any other orientation of fill will also result in a non-right angle. This forces the white rectangle formed (in part) by tiles t_1, t_2, b^*, and c^* to be closed as shown. But this means that a^* cannot be filled, thus proving that no solution is possible.

Hence, a clause gadget can be solved in Shakashaka if and only if the clause in F that it represents is satisfiable. Thus, we can conclude that the Shakashaka board generated from F via this reduction (which can easily be seen to be polynomial-time) is solvable if and only if all clauses in F can be simultaneously satisfied, i.e. if and only if F is satisfiable. This completes the reduction, showing that Shakashaka is NP-hard (and by virtue of polynomial-time verification, as discussed in the previous proof, it is therefore NP-complete). □

3 Conclusion

In this paper we showed that Shakashaka without numbers in the black squares is NP-complete.

In the future, we like to address the second question from the paper by Demaine et al. [3]: given an $m \times n$ board, what is the minimum number k of black squares that is necessary to obtain a board with a unique solution? Another natural question asks for this number if the black squares contain numbers.

References

[1] http://www.nikoli.co.jp/en/puzzles/shakashaka.html, NIKOLI Co., Ltd. Accessed January 5, 2014.

[2] E. D. Demaine. Lecture 15, 6.890. http://courses.csail.mit.edu/6.890/fall14/lectures/L15.html.

[3] E. D. Demaine, Y. Okamoto, R. Uehara, and Y. Uno. Computational complexity and an integer programming model of Shakashaka. In *Proc. 25th Canad. Conf. Comput. Geom., CCCG 2013*. Carleton University, Ottawa, Canada, 2013.

[4] D. E. Knuth and A. Raghunathan. The problem of compatible representatives. *SIAM Journal of Discrete Math.*, 5(3):422–427, 1992.

[5] W. Mulzer and G. Rote. Minimum-weight triangulation is NP-hard. *Journal of the ACM*, 55(2), 2008.

The Inapproximability of Illuminating Polygons by α-Floodlights

Ahmed Abdelkader* Ahmed Saeed † Khaled Harras‡ Amr Mohamed§

Abstract

We consider variants of the art gallery problem where guard visibility is limited to a certain angular aperture α. We show that the problem is NP-hard even when guards can be located in the interior of the polygon. We then proceed to prove that both this problem and its vertex variant, where guard placement is restricted to the vertices of the polygon, are APX-hard.

We observe that earlier constructions for such results in art gallery problems with 360° guards, usually required them to cover few specific elements. We exploit this by carefully updating the construction to replace 360° guards with α-floodlights. Similar transformations may be applicable to other constructions in traditional art gallery theorems, which is of independent interest.

1 Introduction

The study of art gallery problems is a rich area in geometry with a variety of combinatorial bounds, algorithms and hardness results [20, 21, 24]. While we are only concerned with floodlight illumination, we build upon the construction of Lee and Lin [18] through the work of Eidenbenz, Stamm and Widmayer [11]. This construction was used in [18] to show that deciding the minimum number of guards in a polygon without holes is NP-hard. The construction was refined in [11] to further show that there exists a constant $\epsilon > 0$, such that no polynomial time algorithm can guarantee an approximation ratio of $1 + \epsilon$ unless $P = NP$. In other words, the problem is APX-hard, as was obtained independently in [5]. Exact [10], approximate [17, 4] and heuristic [1] solutions have been developed.

Most of the aforementioned work focused on omnidirectional guards, i.e., guards with 360° range of vision. However, many recent applications in sensor networks and smart surveillance are more concerned with sensors that have limited sensing ranges. This leads us to study the α-floodlight illumination variant of the art gallery problem.

*Department of Computer Science, University of Maryland, College Park, akader@cs.umd.edu

†School of Computer Science, Georgia Institute of Technology, ahmed.saeed@gatech.edu

‡School of Computer Science, Carnegie Mellon University, kharras@cs.cmu.edu

§Department of Computer Science and Engineering, Qatar University, amrm@ieee.org

The first documented floodlight illumination problem is perhaps the stage illumination problem (SIP), presented in 1992 by Urrutia [7, 3]. Given a line segment, i.e., the stage, together with a set of floodlights, of known origins and angles, decide whether the floodlights may be rotated to illuminate the stage. The original SIP remained unsolved for more than ten years [7] and was later shown to be NP-complete [16], even under two different restrictions. Variants of the SIP and other problems related to floodlights include [23, 22, 6, 13, 9].

Estivill-Castro and Urrutia [14] asked whether computing the minimum set of covering α-floodlights is NP-hard. Indeed, Bagga, Gewali and Glasser [2] showed that the vertex Floodlight Illumination Problem (FIP) is NP-hard, for $0 < \alpha \leq 360°$. The status of the point variant, where floodlights can be placed anywhere inside the polygon, remains open.

The renewed interest in this classical problem is motivated by several coverage problems in visual and directional sensor networks. α-floodlights, which restrict visibility to a certain angular aperture α, are particularly appealing as a better model for sensors with a limited sensing range, e.g., cameras.

We define α-floodlights and the two polygon illumination problems at hand. We also define distinguished arrows [11], which will be used in some of our arguments.

Definition 1 *An α-floodlight at point p, with orientation θ, is the infinite wedge $W(p, \alpha, \theta)$ bounded between the two rays $\overrightarrow{v_l}$ and $\overrightarrow{v_r}$ starting at p with angles $\theta \pm \frac{\alpha}{2}$. In a polygon P, a point q belongs to the α-floodlight if $\overline{pq}$ lies entirely in both P and $W(p, \alpha, \theta)$.*

Definition 2 *A distinguished arrow (DA) is an infinitesimal ray along an edge of the polygon such that any α-floodlight that covers it must be placed in a prespecified region, i.e., the interior of a gadget or a cone.*

Definition 3 *The Vertex Floodlight Illumination Problem (FIP) [2] Given a simple polygon P with n sides, a positive integer m and angular aperture α, determine if P can be illuminated by at most m α-floodlights placed only on the vertices of P with at most one α-floodlight per vertex.*

Definition 4 *The Point Floodlight Illumination Problem (PFIP) Given a simple polygon P with n sides, a positive integer m and angular aperture α, determine if P can be illuminated by at most m α-floodlights placed in the interior of P.*

In both FIP and PFIP, floodlights can be oriented in any direction as long as P is illuminated. However, to verify a given solution in polynomial time, we cannot deal with arbitrary orientations. To remedy this, [2] introduced a *flushing* restriction which brings FIP into NP. As our main result uses a gap-preserving reduction from 5-OCCURRENCE-MAX-3-SAT (FOM-3SAT), which we define below, a similar restriction will be necessary. When the restriction is in effect, we prefix the problem name with the letter R. We define flushing as follows:

Definition 5 *An α-floodlight is flush with the vertices of the polygon P if at least one of $\overrightarrow{v_l}$ or $\overrightarrow{v_r}$ passes through some vertex of P, different from p, such that θ is determined implicitly.*

Definition 6 *(FOM-3SAT) Given a boolean formula Φ in conjunctive normal form, with m clauses and n variables, 3 literals at most per clause, and 5 literals at most per variable, find an assignment of the variables that satisfies as many clauses as possible.*

We develop a construction for point α-floodlights and outline how to adapt it for vertex α-floodlights. This allows us to obtain the following.

Theorem 7 *PFIP is NP-hard.*

Theorem 8 *RPFIP is NP-complete.*

Theorem 9 *RFIP is APX-hard.*

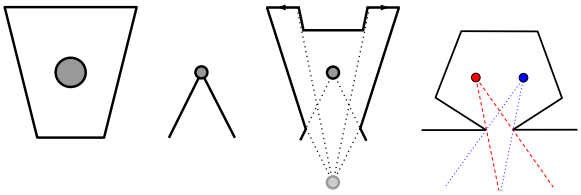
Theorem 10 *RPFIP is APX-hard.*

The construction in [2] utilizes beam machine gadgets [8] to control the visibility of the α-floodlight guards in FIP. In Section 2, we develop beam machines for point α-floodlights in addition to the *Point α-Floodlight Gadget (PFG)* to have corresponding tools in PFIP. This immediately yields Theorems 7 and 8 by plugging the new gadgets in the construction from [2].

In Section 3, we start by examining the construction of [11] and describe how 360° guards can be replaced with α-floodlights without changing the essence of the construction. The main observation is that while guards can see in all directions, the construction only requires them to guard few specific elements or regions. We exploit this to carry over the construction of [11] from the 360° guard setting to the α-floodlight setting, and carry along the result obtained in the former to get Theorems 9 and 10.

2 Point α-floodlights

We develop the *Point α-Floodlight Gadget (PFG)* and use it to create a Point α-Floodlight Beam Machine (BM). Then, we discuss the extension of [2] using the new BM to obtain the first proof of Theorem 7.



(a) Trapezoid, *vertex*-PFG, PFG. (b) ABM.

Figure 1: PFGs and Abstract Beam Machine (ABM).

2.1 PFG

The building block of our construction is the Point Floodlight Gadget (PFG) in Figure 1a. The PFG is attached to the polygon through its mouth and extrudes outside forming a cavity. The cavity is the union of two overlapping wedges. Both wedges share the same axis with one outward wedge looking into the cavity and one inward wedge extending into the interior of the polygon. The extrusion includes two ears which require an α-floodlight guard at the apex of the outward wedge to cover their pockets. Depending on how the PFG is used in a larger gadget, the PFG can be configured such that a second α-floodlight at the apex of the inward wedge is either optional or obligatory. Note that both α-floodlights would satisfy the flushing condition.

When using vertex α-floodlights, a PFG equivalent is just an ear vertex. We refer to both as PFGs and use a trapezoidal symbol in our schematic diagrams as a placeholder for the appropriate PFG. Figure 1a demonstrates the correspondence.

2.2 Beam Machines

Beam machine gadgets were introduced in [8] which showed the hardness of finding a minimum convex cover for a given polygon. The beam machine (BM) is a butterfly shaped extrusion that attaches to the polygon through a mouth. The internal design of the BM requires 4 convex polygons to cover the BM itself and allows one of two slim polygons, i.e., *beams*, to *shoot* into the interior of the polygon in two different directions. The construction needed such shooting beams to cover other parts of the polygon, i.e., *dents*, which corresponds to the satisfaction of boolean clauses by the assignment of their literals. This enabled a reduction from 3SAT to show the problem is NP-hard.

BMs were reused in [2] to force the inclusion of one of two vertex α-floodlights in a construction similar to the one in [8]. The BMs in [2] required 3 vertex α-floodlights to cover their interior and could shoot *light beams* to illuminate their dents. Again, this enabled a reduction from 3SAT to show that FIP is NP-hard.

A BM can be stretched and skewed to control the beams, which need not be symmetric. We abstract BMs as an extrusion with two potential points for α-floodlight placement, as in Figure 1b. We identify `True` and `False` with the red and blue colors, respectively.

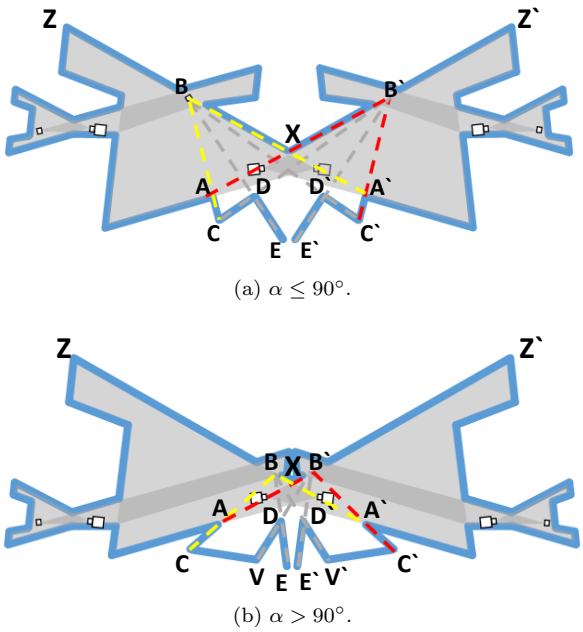
We can now develop a BM for point α-floodlights. The BM is basically one big PFG to create the two wings of the butterfly plus one PFG on each side to extrude two cavities on the upper sides of the wings. All 3 PFGs require 2 floodlights each, e.g. the big PFG needs one guard for the edge denoted Z and another for Z' as in Figure 2. The mouth is designed to require one α-floodlight at one of the two cavities denoted B and B', which results in the two BM configurations. We identify the red and blue points of the ABM with B and B', respectively. The BM requires 7 α-floodlights which all satisfy the flushing condition by construction.

2.3 Updating the reduction by Bagga et al. [2]

Using the point α-floodlight BM and PFG, it is straightforward exercise to update the construction in [2]. The Background of Variable Generator requires 4 PFGs at vertices $\{v_4, v_{11}, v_{13}, v_{20}\}$ where the inward wedge of the PFGs at either v_4 or v_{20} is used to specify an assignment for the variable, for a total of 7 point α-floodlights. Each literal is represented by a BM and the final polygon requires a single PFG contributing 2 additional α-floodlights. Given a `3SAT` instance with m clauses and n variables, the `PFIP` instance output by the reduction can be covered using $21m+7n+2$ point α-floodlights iff the `3SAT` instance is satisfiable. This yields Theorem 7. As all our gadgets satisfy the flushing condition, Theorem 8 follows as well. These two theorems also follow from the construction presented in the next section.

3 Reusing the construction of Eidenbenz et al. [11]

[18] showed that determining the minimum number of guards to cover an art gallery is NP-hard. They presented a construction for vertex guards and showed how it can be modified to yield similar results for the edge and point variants. [11] followed the lines of the reduction in [18] to describe a gap-preserving reduction from the MAXSNP-complete `FOM-3SAT`, which shows these problems are APX-hard. In doing so, [11] gives a detailed construction for all gadgets to guarantee certain properties necessary for the gap-preserving reduction. A similar approach was applied to the construction in [8] for the problem of finding a minimum convex cover to show it is APX-hard as well [12]. Later on, [15] assigned weights to the edges of the construction of [11] to show that maximizing the guarded boundary of an art gallery is APX-hard. For that problem, a constant-factor approximation was developed earlier [19], so the problem is actually APX-complete.



(a) $\alpha \leq 90°$.

(b) $\alpha > 90°$.

Figure 2: BMs for different values of α.

We briefly recall the construction of [11] before we list our observations and the modifications we apply.

3.1 Recalling the gadgets of [11]

Literal pattern for literal l is a triangular extrusion with a spike that requires one *literal guard* at one of two locations called $T^{lit}(l)$ and $F^{lit}(l)$.

Clause pattern for clause c_i uses 3 literal patterns $l_j(c_i)$ such that it can be covered iff at least one literal is assigned a guard at its $T^{lit}(l_j(c_i))$.

Variable pattern for variable x_k has two quadrilateral extrusions called *legs* and a *tail* that requires one *variable guard* at one of two locations called $T^{var}(x_k)$ and $F^{var}(x_k)$.

Ear pattern is a cavity at the top-left corner of the final polygon which hosts one *ear guard* w that covers the ear itself plus the background quadrilateral supporting the gadgets which define the polygon and all left and right legs of the variable patterns.

Spike pattern for a literal is a tiny extrusion in the legs of its variable pattern to ensure consistent truth assignments. The spike pattern is a cone that, in a *canonical solution*, must be covered by either the variable guard of the leg containing it or the literal guard tied to it. Positive and negative literal guards are tied to their variable by a spike in the appropriate leg.

3.2 Observations and modifications

Spike patterns are only a subset of the guard's visibility polygon. A guard can typically see a much larger area containing the spike pattern. When using α-floodlights, located in a BM, it is only necessary that the spike extrusion is covered by the beam the floodlight shoots through the BM's mouth.

T^{lit} and F^{lit}. The only functions these two locations may serve are: (1) Cover the interior of the literal gadget. (2) Cover the corresponding spikes in the variable gadget. (3) Satisfy the clause. When using BMs, (1) will be taken care of by the design of the BM. (2) and (3) turn out to be difficult to achieve using a single α-floodlight. To remedy this, we use two *coupled* BMs per literal to collectively support two configurations corresponding to the assignment of the literal's variable. Figure 3 illustrates the coupling technique. Basically, we copy the `TRUE` signal communicated through the spike in the variable pattern by introducing a dent. A literal can satisfy the clause iff the BM at the top is allowed to shoot its left beam. This would only work if the dent is covered by the BM to the right which only happens iff this BM is allowed to shoot its `TRUE` beam. In addition, we ensure that no single floodlight can cover two such dents.

This allows us to redesign the clause pattern as in Figure 4. Satisfying a clause corresponds to illuminating the dent containing the DA denoted by 2. This dent is adjusted such that it may not be illuminated by a floodlight in any of the spike patterns of the 3 literals of that clause. A single PFG at the top left corner covers the background quadrilateral of the clause pattern and DA-1, which only leaves uncovered the interiors of the BMs, their dents and DA-2.

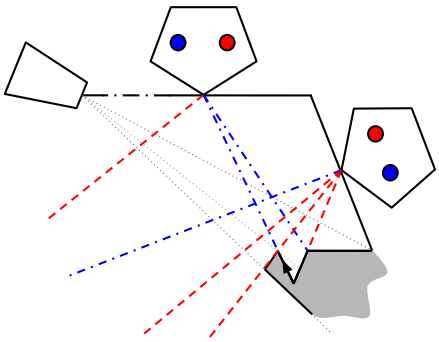


Figure 3: Coupled BMs. Dent must be covered.

Locating T^{lit} and F^{lit}. These two vertices of the literal pattern are at a distance controlled by two arbitrary constants [11]: (1) Distance between T^{lit} and s_6. (2) Distance between s_6 and the vertical line v'. They can be made arbitrarily close as required by the BM to enable shooting the beams to illuminate the corresponding spikes. Finally, we move these locations along the lines defined by the spike patterns to place the BMs on an oblique edge in the clause pattern to give it more flexibility to adjust all BMs and beams to cover their assigned targets. Note that we only generate a restricted class of the spike patterns constructed by the algorithm in [11], but otherwise we do not move them. This preserves the property that no 3 spike patterns of 3 different legs intersect in a common point per Lemma 1 in [11].

Switching T^{lit} and F^{lit}. The roles played by either of these two locations is determined by the spikes they are tied to, which depends on the literal being positive or negative. In addition, T^{lit} can satisfy the clause while F^{lit} cannot. To avoid changing the construction in [11] by much, we effectively exchange the roles of the guards at T^{lit} and F^{lit} such that F^{lit} is the location that can satisfy the clause. While this would not work for the literal pattern in [11], we will be replacing it anyway with a BM.

Moving F^{lit}. Due to the modifications we apply to the variable pattern, we identify F^{lit} with s_4 instead of s_5. We then move it along to find its location in the BM attached to the oblique edge. Again, while this does not make sense in the construction of [11], we are only interested in the coordinates produced for these vertices. In particular, we only need to make sure the spike patterns in the construction of [11] include the locations of the α-floodlights inside their literal BMs.

Limiting the required aperture T^{var} and F^{var}. Each of these two vertex guards is required to cover the variable pattern's tail in addition to the literal spikes in its leg. This implies the effective range of vision is bounded by the variable tail and the *lowest spike* in the leg. To make sure a single α-floodlight can cover both the variable tail and all the spikes in its leg, we require that literal patterns are far enough to the right from all variable patterns such that the lowest spike in any leg does not require an aperture larger than α. Adjusting the variable tail accordingly can be achieved by stretching the variable pattern as shown in Figure 5.

T^{var} and F^{var} for vertex α-floodlights. As we only assign one guard to either of these two locations, the cavity of the unassigned vertex-PFG, as in Figure 5, will need to be covered. This can be achieved by cutting off the left supporting edge of the vertex-PFG such that the cavity is covered by the ear guard. Note that the right supporting edge is still sufficient for the variable guards to satisfy the flushing condition.

The ear pattern and the final polygon. We replace the ear pattern with a PFG and stretch the polygon to include the background quadrilateral and the legs of all variable patterns in the PFG's inward wedge.

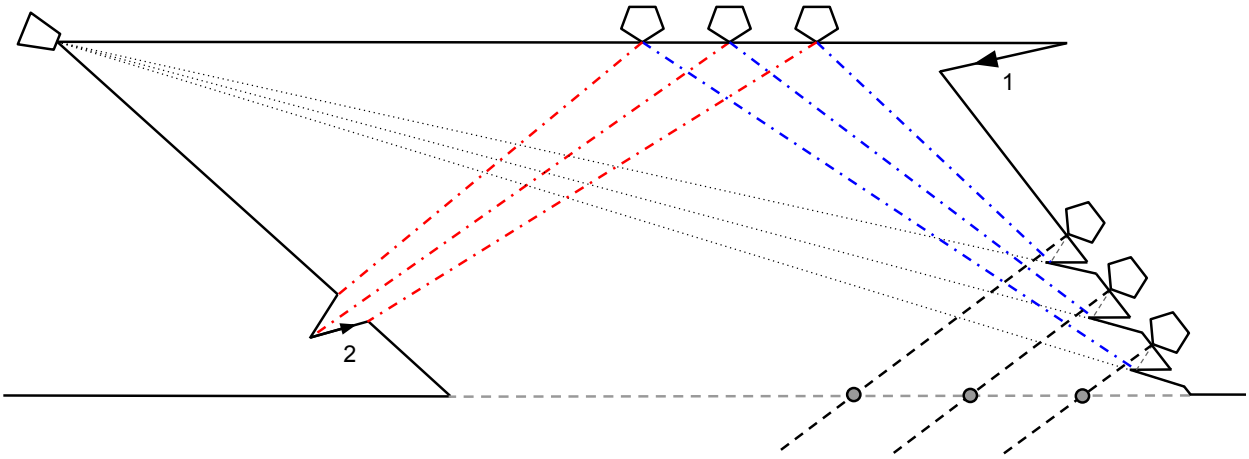


Figure 4: Clause Gadget. Circles highlight the neighborhoods of $T^{lit}(l_j(c_i))$ computed in [11].

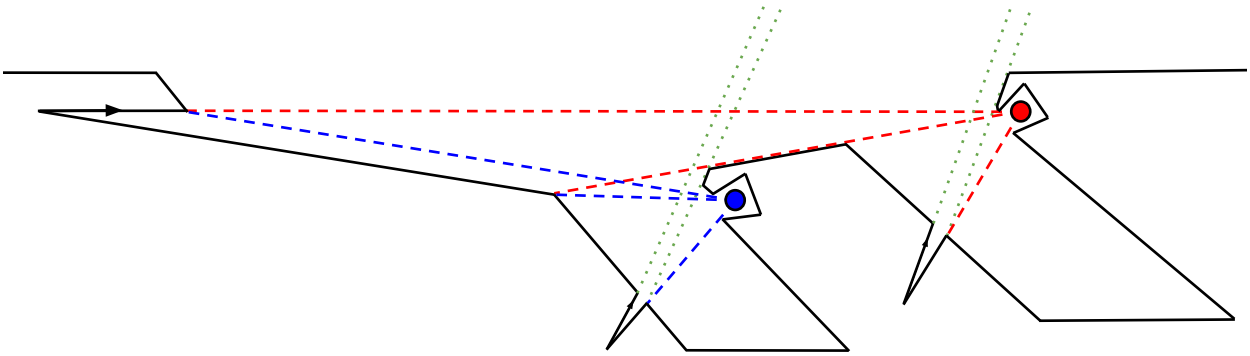


Figure 5: Variable Gadget. The spike to the left and the lowest spike in each leg must fit in the wedges of the PFGs.

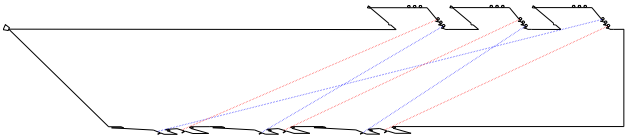


Figure 6: Rough sketch of the final construction.

4 Inapproximability results

Using this construction for PFIP, we get that the PFIP instance can be covered by $44m + 3n + 2$ point α-floodlights iff the FOM-3SAT instance is satisfiable. For FIP, the number is $19m + n + 1$. This provides an alternative proof that both problems are NP-hard. The *if* part is a straightforward mapping from Lemma 2 in [11], observing the number of α-floodlights required for each gadget. The *only if* part is obtained by observing that all variable patterns will have exactly one α-floodlight in such solutions, which yields a satisfying assignment.

Updating the construction of [11], per 3.2, preserves all its relevant properties. In particular, at most two spike patterns belonging to two different legs intersect.

Now, we may find an ϵ-approximate solution S to a given FOM-3SAT instance I by reducing it to an RFIP instance I', computing an ϵ'-approximate solution S' of I' and then transforming S' into S. We develop a transformation process similar to the one described in [11], which we could not fit here due to space constraints. This amounts to a gap-preserving reduction from FOM-3SAT to RFIP. Since the former is MAXSNP-complete, this shows RFIP is APX-hard.

As we managed to stay close to the construction in [11], we carry over a close equivalent of their Lemma 3 and Theorem 1. With that, unless $P = NP$, no polynomial time approximation algorithm for RPFIP can achieve an approximation ratio of

$$\frac{44m + 3n + 2 + \epsilon m}{44m + 3n + 2} = 1 + \frac{\epsilon m}{44m + 3n + 2} \geq 1 + \frac{\epsilon}{54}.$$

This yields Theorem 10. As pointed out in [11], since there will be no floodlights added in the transformation of a given solution of RFIP, we would get a slightly bigger constant for the inapproximability of RFIP than the constant of RPFIP and Theorem 9 follows.

5 Conclusion

In this paper, we resolved the hardness and inapproximability of two classical α-floodlight illumination problems for both vertex and point floodlights. We observed that many earlier constructions for 360° guards, only required guards to cover specific regions in the construction. We exploit this to present a structured update of such constructs to work for guards with limited angle of view. We gave two examples of this process by presenting APX-hardness proofs for vertex and point α-floodlight polygon illumination problems for simple polygons. A flushing restriction is introduced to avoid dealing with arbitrary orientations of floodlights and allow polynomial-time verification and gap-preserving reduction. We believe that similar approaches can be used to carry over more results for 360° guards to α-floodlights which can greatly help the ongoing work in sensor networks and smart surveillance.

Acknowledgement

This work was made possible by NPRP grant # 4-463-2-172 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

References

[1] Y. Amit, J. S. Mitchell, and E. Packer. Locating guards for visibility coverage of polygons. *International Journal of Computational Geometry & Applications*, 20(05):601–630, 2010.

[2] J. Bagga, L. Gewali, and D. Glasser. The complexity of illuminating polygons by alpha-flood-lights. In *CCCG*, pages 337–342. Carleton University Press, 1996.

[3] P. Bose, L. Guibas, A. Lubiw, M. Overmars, D. Souvaine, and J. Urrutia. The floodlight problem. *International Journal of Computational Geometry & Applications*, 7(01n02):153–163, 1997.

[4] A. Bottino and A. Laurentini. A nearly optimal algorithm for covering the interior of an art gallery. volume 44, pages 1048–1056. Elsevier, 2011.

[5] B. Brodén, M. Hammar, and B. J. Nilsson. Guarding Lines and 2-Link Polygons is APX-hard. In *CCCG*, 2001.

[6] M. Cary, A. Rudra, A. Sabharwal, and E. Vee. Floodlight illumination of infinite wedges. *Computational Geometry*, 43(1):23–34, 2010.

[7] F. Contreras, J. Czyzowicz, E. Rivera-Campo, and J. Urrutia. Optimal floodlight illumination of stages. In *Proceedings of the Fourteenth Annual Symposium on Computational Geometry*, SCG '98, pages 409–410, New York, NY, USA, 1998. ACM.

[8] J. C. Culberson and R. A. Reckhow. Covering polygons is hard. In *FOCS*, pages 601–611, 1988.

[9] J. Czyzowicz, S. Dobrev, B. Joeris, E. Kranakis, D. Krizanc, J. Maňuch, O. Morales-Ponce, J. Opatrny, L. Stacho, and J. Urrutia. Monitoring the plane with rotating radars. *Graphs and Combinatorics*, 31(2):393–405, 2015.

[10] A. Efrat and S. Har-Peled. Guarding galleries and terrains. *Information processing letters*, 100(6):238–245, 2006.

[11] S. Eidenbenz, C. Stamm, and P. Widmayer. Inapproximability results for guarding polygons and terrains. *Algorithmica*, 31(1):79–113, 2001.

[12] S. J. Eidenbenz and P. Widmayer. An approximation algorithm for minimum convex cover with logarithmic performance guarantee. *SIAM Journal on Computing*, 32(3):654–670, 2003.

[13] D. Eppstein, M. T. Goodrich, and N. Sitchinava. Guard placement for efficient point-in-polygon proofs. In *Proceedings of the twenty-third annual Symposium on Computational geometry*, SCG '07, pages 27–36. ACM, 2007.

[14] V. Estivill-Castro and J. Urrutia. Optimal floodlight illumination of orthogonal art galleries. In *CCCG*, pages 81–86, 1994.

[15] C. Fragoudakis, E. Markou, and S. Zachos. Maximizing the guarded boundary of an art gallery is APX-complete. *Computational Geometry*, 38(3):170–180, 2007.

[16] H. Ito, H. Uehara, and M. Yokoyama. Np-completeness of stage illumination problems. In *Discrete and Computational Geometry*, pages 158–165. Springer, 2000.

[17] J. King. Fast vertex guarding for polygons with and without holes. *Computational Geometry*, 46(3):219 – 231, 2013.

[18] D.-T. Lee and A. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32(2):276–282, 1986.

[19] E. Markou, C. Fragoudakis, and S. Zachos. Approximating visibility problems within a constant. In *3rd Workshop on Approximation and Randomization Algorithms in Communication Networks, Rome*, pages 91–103, 2002.

[20] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, Inc., New York, NY, USA, 1987.

[21] T. Shermer. Recent results in art galleries [geometry]. *Proceedings of the IEEE*, 80(9):1384–1399, Sep 1992.

[22] A. Spillner and H.-D. Hecker. Minimizing the size of vertexlights in simple polygons. *Mathematical Logic Quarterly*, 48(3):447–458, 2002.

[23] W. Steiger and I. Streinu. Illumination by floodlights. *Computational Geometry*, 10(1):57–70, 1998.

[24] J. Urrutia et al. Art gallery and illumination problems. *Handbook of computational geometry*, pages 973–1027, 2000.

Appendix A: Additional Figures

We include additional figures to aid our description of the gadgets we created.

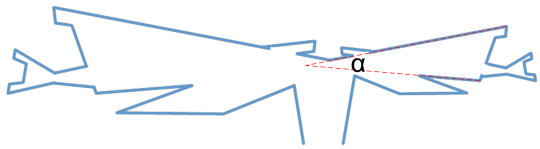


Figure 7: Demonstration of the flexibility of the BM.

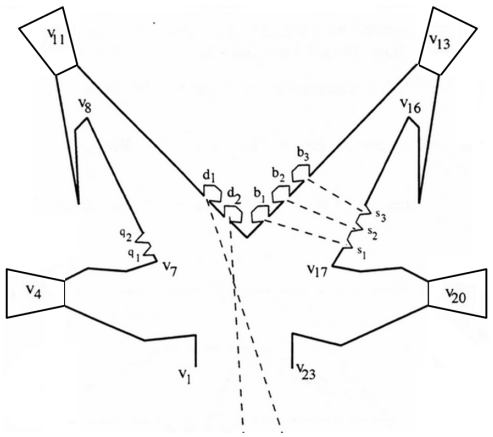


Figure 8: The BVG of [2] updated with PFGs.

Appendix B: α-Floodlights with $\alpha \geq \pi$

(Subsection 3.2) To motivate the intuition for the modifications we applied to the construction of [11], we informally discuss the case when $\alpha \geq \pi$.

Define $\texttt{RFIP}^{\geq\pi}$ and $\texttt{RPFIP}^{\geq\pi}$ to be the restriction of these α-floodlight illumination problems with $\alpha \geq \pi$. We can easily extend the construction in [11] to show similar inapproximability results for these two problems. The key idea is to ensure that all elements required to be covered by a given guard location lies in a half-plane defined by a line passing through this location.

We can ensure the ear guard only looks *down* by attaching its ear to the left edge, instead of the top one. For T^{lit}, we smooth out the right pockets of the clause pattern and introduce a second ear at the bottom side of the polygon, right below the first ear, that looks *up* so it can cover the right side of all clause patterns. F^{lit} can be moved to the same edge of the literal pattern as T^{lit} by introducing a little bend to it to create a new *convex* vertex, such that F^{lit} can still cover the entire literal pattern and its spike, but not satisfy the clause. Finally, T^{var} and F^{var} need only cover the half-plane below the line connecting them to the point w, which requires no change.

Appendix C: How we computed the numbers

The construction we created in Section 3, by modifying the one given in [11] uses the following gadgets:

1. Ear gadget: 1 PFG.
2. Literal gadget: 2 BMs.
3. Clause: 3 literal gadgets + 1 PFG = 6 BMs + 1 PFG.
4. Variable gadget: 2 PFGs.

Note that the PFGs in the variable gadget need not be *activated*, i.e., receive a floodlight at their inward wedge is optional. All other PFGs must be activated. For `FIP`, we use 3 vertex α-floodlights per BM and 1 α-floodlight for PFGs. The number of vertex α-floodlights required to *operate* the gadgets is

$$1 + (6 \times 3 + 1)m + 1 \times n = 19m + n + 1. \quad (1)$$

For `PFIP`, we use 7 α-floodlights per BM, 2 α-floodlights per *active* PFG and 1 α-floodlight per *inactive* PFG. The number of α-floodlights required to *operate* the gadgets is

$$2 + (6 \times 7 + 2)m + 3 \times n = 44m + 3n + 2. \quad (2)$$

Appendix D: Transformation of a feasible solution

Following the lines of the transformation process in [11] we move α-floodlights in such a way that the set of DAs that a floodlight sees changes in only one of two ways: either more arrows are included or it is ensured that another floodlight, possibly added to the solution, covers any arrows removed from this set.

With that, the α-floodlights in a given solution S' computed for the `RFIP` instance I' are moved as follows:

1. Determine the, at least, 2 floodlights that cover the ear PFG and move them to the standard PFG configuration. In addition to the PFG itself, this also ensures that the legs of all variable patterns are covered.
2. For each clause pattern, determine the, at least, 2 floodlights that cover its PFG and move them to the standard PFG configuration. In addition to the PFG itself, this also ensures that the clause pattern, except for the dent denoted by arrow 2 in Figure 4, is entirely covered.
3. For each BM, there will be at least 7 floodlights inside it. We start with the, at least 6, floodlights that do not illuminate any part of the mouth. We move these 6 to illuminate the interior of the BM, except for the mouth, by the configurations shown in Figure 2. Any remaining floodlights that do not illuminate any part of the mouth are moved to the red configuration, i.e., such that they illuminate the entire mouth, the associated dent and spike, if one is associated with the BM at hand, corresponding to setting the literal to `TRUE`. For floodlights that illuminate parts of the mouth, there will be three cases:
   (a) If the floodlight also illuminates the DA associated with a `FALSE` assignment, we move it to the blue configuration, i.e., such that it illuminates the entire mouth and spike corresponding to setting the literal to `FALSE`.

(b) If the floodlight also illuminates the DA associated with a `TRUE` assignment, we move it to the red configuration.

(c) If the floodlight does not illuminate any DAs, we move it to the red configuration.

4. If a BM has more than one floodlight in either the red or blue configurations, we leave only one and move the extra floodlights to the configuration of the same color in the variable pattern of its variable.
5. If a BM has floodlights in both the red and blue configurations, switch all BMs of its variable to the red configuration and move the, at least one, extra floodlights to the red configuration in its variable pattern. If there is already a floodlight there, move the extra floodlights to the blue configuration instead. This ensures that all dents and spikes associated with this variable are illuminated. In addition, any clause dents that were illuminated in the input solution by floodlights in any gadget of this variable are still illuminated.
6. For floodlights inside a clause pattern but outside any BM or PFG, we have a number of cases. As shown in Figure 9, we need to consider floodlight configurations within the intersection of cones belonging to different dents. Observe the following: (a) The literal dents are set up such that no two can be illuminated by a single floodlight. (b) The design of the BM and the steps thus far outlined in the transformation process ensure that all BMs will have a floodlight in at least one of the red or blue configurations. (c) The PFG illuminates the entire clause pattern except for the dents and DA-2. .
   - Cases 1 and 4: Even if the floodlight can illuminate the DAs of both the literal dent and clause dent, the BM above associated with the dent in question must also be able to illuminate at least one of the two DAs. As such, it suffices to move the floodlight to the unoccupied configuration in this BM, if any.
   - Cases 2 and 3: Similar to the previous case, even if the floodlight can illuminate the DAs of both the literal dent and the spike, the BM must be able to illuminate at least one. Likewise, the floodlight is moved to the unoccupied configuration of the BM, if any.
   - Cases 5 and 6: Since the floodlight may be able to illuminate the DAs of both a literal dent and a spike belonging to a different literal, we will need to add a floodlight and move one to illuminate the spike from its corresponding configuration in the BM and move the other to illuminate the literal dent from the red configuration of its BM.
   - Cases 7, 8 and 9: Such floodlights may illuminate the DA of the clause dent or the spikes of some literal. By moving these floodlights to the corresponding configuration in the BM associated with the DA, we can still illuminate it.
7. For each variable pattern, move the floodlight that sees the DA of the variable pattern to the red configuration, if it also lies in a spike pattern containing the red point. Otherwise, move it to the blue configuration.
8. Move all floodlights that cover a single spike to the red or blue point of the spike pattern of that spike.
9. If a floodlight illuminates DAs of two spike pattens that connect literals to two different legs of variable patterns, add a floodlight and move one floodlight each to the two red or blue points of the variable patterns of these two spikes.

   This is the only case where we *add* an α-floodlight and increase the cost of the solution. Note that because of Lemma 1 [11], no floodlight can see the DAs of three spike patterns that belong to three different legs.
10. Any floodlights that can be removed without leaving any DA uncovered are moved, and fixed, to any red or blue point of any variable pattern, if there is no floodlight there already.

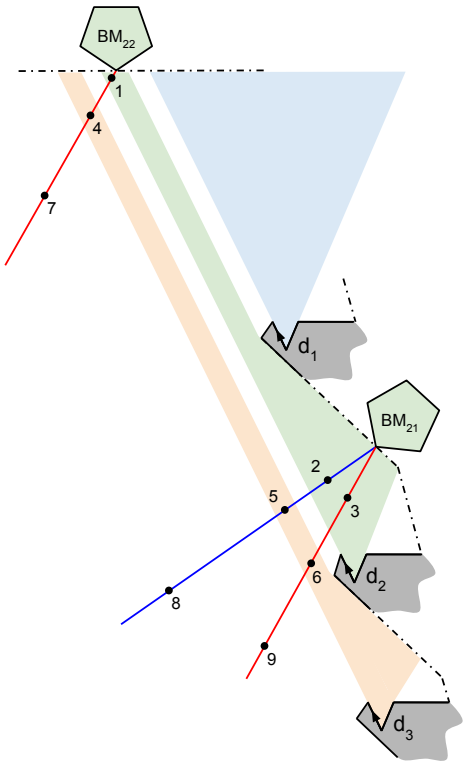


Figure 9: Additional cases for floodlight placement.

We iterate this process until the locations of all floodlights are fixed. One can verify that the transformed solution of S'' is still a feasible solution of I' as any element which was covered in S' remains covered in S''. To obtain the solution S of the `FOM-3SAT` instance I using S'', we set the truth values of the variables as follows. For variable x_k, if the corresponding variable pattern only has floodlights at the blue point, we set it to `False`. If it has only floodlights at the red point, we set it to `True`. If it has floodlights at both points, we assign x_k in such a way that makes the majority of its, at most 5, literals `True`.

Appendix E: Omitted theorems and proofs

We fill in the omitted steps following the analysis in [11].

5.1 $Satisfiable(I) \implies MinFloodlightCost(I')$

The proofs below also provide the *if* part needed for the NP-hardness results we obtained in Section 4. Note that for the *only if* part, we assign `True` to all variables having the α-floodlight at the red points of their variable patterns and `False` otherwise. In particular, these two proofs work for `FIP` and `PFIP` as well and show both the two problems and their restricted variants are NP-hard.

Lemma 11 *If an instance of* `FOM-3SAT`*, with n variables and $m \leq \frac{5}{3}n$ clauses, is satisfiable, then there exists a feasible solution of the corresponding instance of* `RFIP` *with $19m + n + 1$ α-floodlights.*

Proof. Given a satisfying assignment of the n variables in the `FOM-3SAT` instance, we add α-floodlights to a solution of `RFIP` as follows. We start by placing 1 at the ear PFG, 1 for the PFG in all m clauses, and 2 in each of the 2 BMs for all $3m$ literal couples. Next, for each variable x_k, we do the following:

1. If x_k is true, place 1 α-floodlight at the red point in its variable pattern, 1 α-floodlight in the red point of each of its positive literals and 1 α-floodlight in blue point of its negative literals.
2. If x_k is true, place 1 α-floodlight at the blue point in its variable pattern, 1 α-floodlight in the blue point of each of its positive literals and 1 α-floodlight in red point of its negative literals.

For each positive literal, we place 1 α-floodlight in the red point of its coupled BM. For each false literal, we place 1 α-floodlight in the blue point of its coupled BM. This solution is feasible and costs $1 + m + (2 \times 2)3m + n + (1 + 1)3m = 19m + n + 1$. □

Lemma 12 *If an instance of* `FOM-3SAT`*, with n variables and $m \leq \frac{5}{3}n$ clauses, is satisfiable, then there exists a feasible solution of the corresponding instance of* `RPFIP` *with $44m + 3n + 2$ α-floodlights.*

Proof. Given a satisfying assignment of the n variables in the `FOM-3SAT` instance, we add α-floodlights to a solution of `RPFIP` as follows. We start by placing 2 at the ear PFG, 2 for the outward wedges of the 2 PFGs in all n variable patterns, 2 for the PFG in all m clauses, and 6 in each of the 2 BMs for all $3m$ literal couples. We proceed as in the proof of Lemma 11. This solution is feasible and costs $2+2n+2m+(6\times 2)3m+n+(1+1)3m = 44m+3n+2$. □

5.2 ϵ'-$APPROX(I') \implies \epsilon$-$APPROX(I)$

Given a feasible ϵ-approximate solution S' to I' of the α-floodlight illumination problem, we apply the transformation process described in Appendix D. The transformed solution S'' is still feasible, i.e., illuminates the entire polygon. However, due to the possibility of having α-floodlights at the intersection of two spike patterns, we resolved to adding α-floodlights and ended up having variable patterns with α-floodlights at both the red and blue points. Such variables were then assigned in a manner that satisfies the majority of their clauses, but we will not be able to guarantee satisfying all clauses.

Lemma 13 *If there exists an $\epsilon > 0$ and a feasible solution of the* `RPFIP` *instance I' with at most $44m + 3n + 2 + \epsilon m$ α-floodlights, then there exists an assignment of the variables of the corresponding* `FOM-3SAT` *instance I that satisfies at least $m(1 - 4\epsilon)$ clauses.*

Proof. Any feasible solution S' can be transformed into a *canonical* solution S'' that only illuminates the polygon using the gadgets the way we designed them. In such a canonical solution, we know the minimum number of α-floodlights required by the gadgets themselves. Clearly, the algorithm for the illumination problem could not illuminate the entire polygon using that minimum number, possibly because it was created using an unsatisfiable boolean formula. In both cases, we know the algorithm incurred *at most* an additional ϵm cost to ensure the entire polygon is covered. In the worst case, all these additional ϵm α-floodlights were placed in the intersections of two spike patterns. This means that when the transformation process terminates, *at most* $2\epsilon m$ variable patterns will have received an additional α-floodlight that results, in the worst case, in all the $2\epsilon m$ variable patterns having two α-floodlights at both their red and blue points. This leaves at least $n - 2\epsilon m$ variable patterns with only one α-floodlight. For all variables in the second group, they can be assigned a truth value unambiguously. For the variables in the first group, however, we assign truth values to satisfy the majority of their clauses. In the worst case, each such variable will satisfy only 3 out of its 5 clauses. In the worst case, all the 2 clauses left out by *each* of the variables in the second group will not be satisfied by any other literal. This means we may not be able to satisfy at most $2 \times 2\epsilon m = 4\epsilon m$ clauses. The number of satisfied clauses can then be lower bounded by $m - 4\epsilon m = m(1 - 4\epsilon)$. □

5.3 Don't make a promise that is *hard* to keep

Using Lemma 12 and the contraposition of Lemma 13, we obtain the following.

Theorem 14 *Let I be an instance of the promise problem of* `FOM-3SAT`*, with n variables in I, $m \leq \frac{5}{3}n$ clauses. Let $OPT(I)$ denote the maximum number of clauses that can be satisfied using any assignment of the n variables. Furthermore, let I' be the corresponding instance of* `RPFIP` *and let $OPT(I')$ denote the minimum number of α-floodlights needed to illuminate the polygon in I'. Then the following hold:*

- *If $OPT(I) = m$, then $OPT(I') \leq 44m + 3n + 2$.*
- *If $OPT(I) \leq m(1 - 4\epsilon)$, then $OPT(I') \geq 44m + 3n + 2 + \epsilon m$.*

Theorem 14 shows that the reduction is indeed gap-preserving and that the promise problem of `RPFIP` with parameters $44m + 3n + 2$ and $44m + 3n + 2 + \epsilon m$ is NP-hard.

Tradeoffs between Bends and Displacement in Anchored Graph Drawing

Martin Fink* † Subhash Suri*

Abstract

Many graph drawing applications entail geographical constraints on positions of vertices; these constraints can be at odds with aesthetic requirements such as the use of straight-line edges or the number of crossings. Without positional constraints on vertices, of course, every planar graph can be drawn crossing-free with straight-line edges. On the other hand, inflexible and precise specification of all vertex positions essentially leaves no room for presenting the graph in an aesthetically pleasing drawing. However, small deviations from precise vertex positions can often be tolerated, and so a natural middle ground is to impose *soft positional constraints* on vertices and then optimize for an appropriate aesthetic criterion.

We explore one such trade-off: the amount of vertex position displacement vs. the number of bends in planar polyline drawings. In particular, let $G = (V, E)$ be a planar graph, where each vertex v has a specified (target) position $\alpha(v)$. We wish to draw G so that no vertex is placed at distance more than δ from its target position and no edge has more than b bends. Given a bound on b, what is the smallest value of δ achievable for all n-vertex planar graphs? Our main result establishes that $\delta = \Theta(n)$ is both necessary and sufficient if b is constant. We also derive trade-offs between δ and b.

1 Introduction

Visual representations of graphs face multiple, often conflicting, constraints. This paper explores one such trade-off: the tension between aesthetic aspects of a graph drawing and its informational distortion. Specifically, we have a planar graph $G = (V, E)$ on $n = |V|$ vertices, where each vertex v has a specified (target) position $\alpha(v)$ in the plane. Such positional constraints naturally arise in many geo-spatial datasets, such as positions of cities or municipalities in country maps. Positional constraints also arise when the graph is visualized in a larger context: for instance, if the graph is to be overlaid on another graph with a common or overlapping set of entities, then a close correspondence of vertices is highly desirable; the same holds if both graphs are shown next to each other or one after the other. In all these scenarios, placing vertices far away from their intended position can create loss of information and readability since it distorts the user's knowledge of positions (the mental map).

Unfortunately, such positional constraints on vertices can be at odds with other aspects of the drawing, such as aesthetics and readability. For instance, every planar graph can be drawn with straight-line edges and no edge crossings [4], but doing so requires the freedom to move vertices in the drawing space. On the other hand, fixing each vertex's position precisely does not leave much room for an informative drawing: indeed, the vertex positions fix the straight-line drawing and also the number of edge crossings. A natural middle ground, therefore, is to treat the vertex position constraints as *soft constraints*, allowing the flexibility to place the vertices close to their ideal position while improving the quality of the drawing.

Our paper is an exploration of one such trade-off. We ask how much better can the drawing be made if each vertex v is allowed to be displaced by some distance δ from its target position $\alpha(v)$. All the edges must be drawn as polylines with no crossings, and no polyline has more than b *bends*, which we call the *curve complexity* of the drawing. (The curve complexity of a straight-line drawing is $b = 0$.) Our trade-off explores how much benefit in terms of the curve complexity one can expect by increasing the displacement as a function $\delta(n)$; more precisely, given a maximum displacement δ, what is the smallest curve complexity $b(\delta, n)$ achievable for all n-vertex planar graphs? Similarly, for a given curve complexity b, we want to know the smallest displacement $\delta(b, n)$ that is sufficient for all n-vertex graphs. We call our problem the *anchored graph drawing* problem because each vertex has an ideal (anchor) position.

Previous Work. Our research touches two important topics in graph drawing: positional constraints on vertices and bend-minimization in polyline drawings. If each vertex has a disk-shaped region within which it must be placed, then it is NP-hard to decide if a straight-line planar drawing exists, as Godau showed [5]. In a followup work, Angelini et al. showed that the problem remains NP-hard even if the disk regions of all vertices have the same radius [1]. Extending these hardness results, Löffler [9] showed that it is NP-hard to decide if a straight-line embedding exists for regions that are vertical line segments, even if the graph is only a cycle.

When the displacement shrinks to $\delta = 0$ vertex move-

*Department of Computer Science, University of California, Santa Barbara, {fink|suri}@cs.ucsb.edu

†M. Fink was supported by a fellowship within the Postdoc-Program of the German Academic Exchange Service (DAAD).

ments are disallowed; this yields the classic point-set embeddability problem with fixed vertex-point mapping. In contrast to the version without given mapping, the straight-line problem is trivial since there is only one straight-line drawing which either is or is not planar. However, there are several results for polyline edges.

Pach and Wenger [10] showed that, in polynomial time, every planar graph can be embedded with fixed vertex positions and $O(n)$ bends per edge. Furthermore, they proved that if the points are in convex position, for every planar graph, the probability is high that a linear number of edges will need a linear number of bends. For non-convex position, Badent et al. [2] constructed a family of instances in which a linear number of edges must have a linear number of bends. In a recent arxiv submission, Gordon [6] shows that for every set of vertex positions, a planar graph that is sampled uniformly at random (with fixed vertex-point mapping) will require $\Omega(n^2)$ bends in total with high probability.

Point-set embeddability has also been considered without prescribed vertex-point mapping. In this setting, a set of n points is prescribed and for each vertex one of the input points must be chosen as the vertex position. Gritzmann et al. [7] introduced this problem class and showed that a planar straight-line embedding can be constructed for every outerplanar graph. However, for general planar graphs, it is NP-hard to decide whether a planar straight-line embedding exists, as Cabello proved [3]. Kaufmann and Wiese [8] showed that for every planar graph a vertex-point mapping can be found such that the graph can be embedded with only two bends per edge (one bend for four-connected graphs).

Our Results. Our work differs from these earlier lines of research in that we explore the trade-off between positional displacement δ and maximum number b of bends per edge (the *curve complexity*) for which a feasible planar drawing exists. More specifically, given a curve complexity b, we ask for the smallest displacement δ such that a feasible anchored drawing exists. (b and δ may be constants or functions of n.) When $\delta = 0$, we get the classic point-set embeddability problem. In our problem, however, vertices can be displaced within the distance bound δ, and we wish to explore the effect of δ on the curve complexity b that is necessary for a plane drawing. Since we must relate the value of δ to the area of the drawing and the distance between vertices, we usually assume that the target vertex positions are points of the integer $n \times n$ grid, so that the smallest distance between vertices is 1. However, we do not demand that in the final drawing vertices and bends have integer coordinates.

We will see that even with a positive (but small) value of δ, there are graphs and target positions for which a linear number of edges require a linear number of bends in every feasible drawing. If we allow displacement $\delta = O(n)$, then an easy construction can achieve $b = 0$, namely, a straight-line embedding, for any n-vertex planar graph. With some more effort and care, we can show that curve complexity $b = 2$ is always possible for $\delta > (n-1)/2$.

Our main result is to show that, surprisingly, this linear displacement is necessary for any constant number of bends. More specifically, we show that for any constant b, there are planar graphs that require a minimum vertex displacement of $\Omega(n)$ to realize a polyline drawing with at most b bends. In fact, if the vertex displacement is $o(n)$, then at least $\Omega(n)$ edges require more than b bends. We also show that for curve complexity $b = \Theta(\sqrt[3]{n})$, a displacement of $\Omega(\sqrt[3]{n})$ is necessary, and that for any constant displacement, there are instances that force a curve complexity of $\Omega(\sqrt{n})$.

2 Preliminaries

We call our problem the Anchored Graph Drawing Problem (AGD), following Angelini et al. [1]. In addition to planar input graph and target vertex positions, the problem takes two parameters: the maximum displacement δ of vertices from their target position and the curve complexity b. Since we are interested in the relation between δ and b, we call the problem δ-b-AGD.

Problem (δ-b-AGD) *Given a planar graph $G = (V, E)$ with $n = |V|$, a function $\alpha\colon V \to \mathbb{N} \times \mathbb{N}$ that assigns to each vertex v a position $\alpha(v)$ on the $n \times n$ grid, a number $\delta \in \mathbb{R}^+$, and a number $b \in \mathbb{N}$ find a planar polyline drawing $\mathcal{E}$ of G such that every vertex v is placed within distance δ of $\alpha(v)$ and no edge has more than b bends.*

We call a feasible embedding for such an instance a δ-b-AGD embedding. We will sometimes speak of *moving* a vertex v to mean that the vertex is placed within distance δ of its target position $\alpha(v)$. Similarly, a δ-*movement* of the vertices allows to place each vertex at a position up to δ from its target position.

Depending on instance and parameters, it is not clear whether a δ-b-AGD embedding exists. Hence, the question is how the parameters relate to each other and to n. For given b and n, we would like to know how big δ must be so that every instance of n vertices has a δ-b-AGD embedding. To this end, we define two values.

Definition 1 *Let $G = (V, E)$ be a planar graph with $n = |V|$ and let $\alpha\colon V \to \mathbb{N} \times \mathbb{N}$ define target positions for the vertices on the $n \times n$ grid. Let $b \geq 0$ be the curve complexity. We define $\delta(b, G, \alpha)$ to be the minimum value δ such that a δ-b-AGD embedding of G exists.*

Now, we consider the relation between b, n, and δ.

Definition 2 ($\delta(b, n)$) *Let $b, n \geq 0$ be integer values. We define $\delta(b, n)$ to be the maximum value $\delta(b, G, \alpha)$ over all instances of a planar graph*

G with n vertices and target positions α, i.e., $\delta(b,n) = \max\{\delta(b,G,\alpha) \mid G=(V,E)$ planar, $|V|=n$, $\alpha\colon V \to \mathbb{N}\times\mathbb{N}\}$.

In the following sections, we will find upper and lower bounds for $\delta(b,n)$. Since it turns out that for every constant b the lower bound for $\delta(b,n)$ is linear in n, it makes sense to also consider values for b that depend on the size of the graph; then, b can be a function of n.

Analogously, we can define values corresponding to the minimum number of bends for which a δ-b-AGD embedding with given δ exists.

Definition 3 *Let $G=(V,E)$ be a planar graph on n vertices and let a function $\alpha\colon V\to\mathbb{N}\times\mathbb{N}$ define target positions for the vertices of G on the $n\times n$ grid. Let $\delta\geq 0$ be the maximum vertex displacement. We define $b(\delta,G,\alpha)$ to be the minimum curve complexity $b\geq 0$ such that a δ-b-AGD embedding of G exists.*

Definition 4 ($b(\delta,n)$) *Let $n\geq 0$ be an integer value and let $\delta\geq 0$. We define $b(\delta,n)$ to be the maximum value $b(\delta,G,\alpha)$ over all instances of a planar graph G with n vertices and target positions α, i.e., $b(\delta,n) = \max\{b(\delta,G,\alpha) \mid G=(V,E)$ planar, $|V|=n$, $\alpha\colon V\to\mathbb{N}\times\mathbb{N}\}$.*

3 Upper Bounds

We recall that even without displacement of the vertices (i.e. with fixed vertex positions), every planar graph can be embedded with curve complexity $O(n)$ using the algorithm of Pach and Wenger [10]. Thus, there is always a δ-$O(n)$-AGD embedding, no matter how small δ is. Our lower bound result will later (cf. Theorem 8) establish that a linear curve complexity is necessary even if we allow positive displacement of vertices. We begin with our upper bounds for $\delta(b,n)$.

By choosing $\delta=\sqrt{2}(n-1)$, any vertex can be placed freely in the area spanned by the $n\times n$ grid, thus effectively removing the restriction of the δ-movement and allowing to use any algorithm for creating a planar straight-line embedding. Since the final vertex positions after the movement do not have to lie on the grid, any value $\delta > \sqrt{2}(n-1)/2 = (n-1)/\sqrt{2}$ is sufficient; such a value allows all vertices to be moved to and within a small area around the center of the $n\times n$ grid.

Observation 1 *For any $\varepsilon>0$, $\delta(0,n)\leq (n-1)/\sqrt{2}+\varepsilon$, that is, for $\delta=(n-1)/\sqrt{2}+\varepsilon$, there is a δ-0-AGD embedding for every planar graph whose target positions lie on the $n\times n$ grid.*

If we allow two bends per edge, a smaller bound on δ can be shown, using the following result of Kaufmann and Wiese [8]. (This is not explicitly stated in their paper, but follows from their point-set embeddability construction without prescribed vertex-point mapping.)

Lemma 1 ([8]) *For every planar graph $G=(V,E)$ there is an ordering $V=\{v_1,\dots,v_n\}$ of the vertices, such that for any assignment of coordinates that follows the left-to-right order $x(v_1)<x(v_2)<\ldots<x(v_n)$ a planar 2-bend embedding can be found.*

Using this lemma, we can prove the following result.

Theorem 2 *For any $\varepsilon>0$, $\delta(2,n)\leq (n-1)/2+\varepsilon$, that is, for every planar graph $G=(V,E)$ with n vertices whose target positions $\alpha\colon V\to\mathbb{N}\times\mathbb{N}$ lie on the $n\times n$ grid there is a δ-2-AGD embedding with $\delta=(n-1)/2+\varepsilon$.*

Proof. Let $V=\{v_1,\dots,v_n\}$ be the order of vertices achieved by Lemma 1. If we can find a δ-movement that orders $v_1,\dots,v_n$ from left to right, then a 2-bend embedding follows. We achieve the ordering as follows.

By moving all vertices horizontally by at most $(n-1)/2$ we can put them on a vertical line through the middle of the $n\times n$ grid. With the remaining movement of ε, we create the correct left-to-right order. With these vertex positions, the 2-bend embedding can be created. □

Since b bends are also feasible if higher curve complexity would be allowed, we also get bounds for other complexities. Summarizing, we get the following result.

Theorem 3 *For any $\varepsilon>0$, it holds $\delta(1,n)\leq\delta(0,n)\leq (n-1)/\sqrt{2}+\varepsilon$ and $\delta(b,n)\leq (n-1)/2+\varepsilon$ for $b\geq 2$.*

Our upper bound for δ does not improve with larger values of b. Finding a construction for general b is an interesting open problem.

We now present the main result of our paper, which is a family of lower bounds for $\delta(b,n)$. Somewhat surprisingly it turns out that the $O(n)$ vertex displacement, so easily achieved by our upper bound above, cannot be improved, namely, $\delta(b,n)=\Omega(n)$ for any constant value of b.

4 Lower bounds

We will construct negative examples that show that even for relatively large values of δ, depending on b, no feasible δ-b-AGD embeddings exist. More precisely, for every number b (from constant to $\Theta(n)$), we find a family of planar graphs and point sets with according δ, such that in every feasible embedding with a δ-movement of the vertices, a *linear number* of edges will have more than b bends. Our proof is constructive and draws inspiration from the bad instances of point-set embeddability used by Badent et al. [2].

Theorem 4 *Let $b\geq 0$. Then, for $n\geq 4\sqrt{2}(4b+5)/\pi^2+1$ it holds that $\delta(b,n)\geq (n-1)\pi^2/(16(4b+5)^2)$, that is, $\delta(b,n)=\Omega(n/b^2)$. More precisely, for any such n there is an example in which a linear number of edges must have more than b bends if the vertices are moved by at most $(n-1)\pi^2/(16(4b+5)^2)$.*

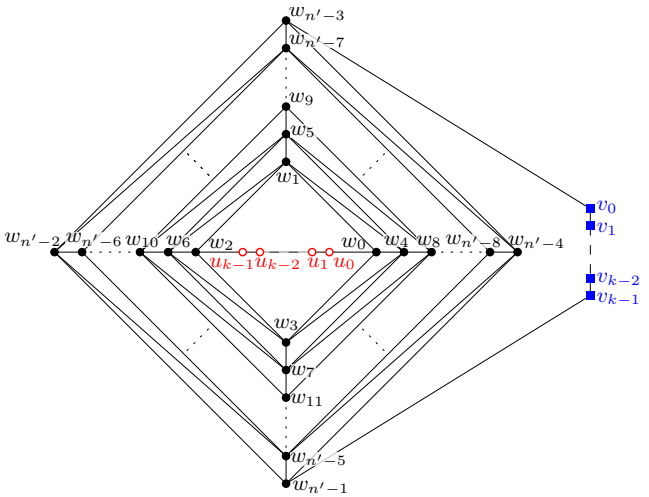


Figure 1: Schematic visualization of the graph used in our constructions for lower bounds. Except for the choice of the outer face, this is the only possible embedding.

Proof. The main idea is to have many 4-cycles such that, in every planar embedding, every 4-cycle has to separate two sets of *red* and *blue vertices* of equal size into vertices inside and outside of the cycle. By carefully placing the target positions of these red and blue vertices and choosing the right value δ, we achieve that even after a δ-movement of the vertices every 4-cycle separating the red and blue vertices must be realized as a complex polygon, having at least one edge with $1/4$ of the necessary bends.

Our graph is constructed as follows; see Fig. 1. Let n' be a multiple of 4, and let $k \geq 1$. The graph consists of a set of n' black vertices $V_0 = \{w_0, w_1, \ldots, w_{n'-1}\}$, a set of k *red* vertices $V_1 = \{u_0, u_1, \ldots, u_{k-1}\}$, and a set of k *blue* vertices $V_2 = v_0, v_1, \ldots, v_{k-1}\}$. The set of edges consists of five subsets, i.e., $E = E_0 \cup E_1 \cup E_2 \cup E_3 \cup E_4$, which are defined as follows.

$$E_0 = \{(w_0, u_0), (u_0, u_1), \ldots, (u_{k-2}, u_{k-1}), (u_{k-1}, w_2)\}$$
$$E_1 = \{(w_{n'-3}, v_0), (v_0, v_1), \ldots, (v_{k-1}, w_{n'-1})\}$$
$$E_2 = \{(w_i, w_{i+1}), (w_{i+1}, w_{i+2}), (w_{i+2}, w_{i+3}), (w_{i+3}, w_i) \mid 0 \leq i < n', i \bmod 4 = 0\}$$
$$E_3 = \{(w_{i+4}, w_{i+1}), (w_{i+1}, w_{i+6}), (w_{i+6}, w_{i+3}), (w_{i+3}, w_{i+4}) \mid 0 \leq i < n' - 4, i \bmod 4 = 0\}$$
$$E_4 = \{(w_i, w_{i+4}) \mid 0 \leq i < n' - 4\}$$

The edges in E_0 form a path from w_0 to w_2 containing all red vertices and the edges in E_1 form a path from $w_{n'-3}$ to $w_{n'-1}$ containing all blue vertices. The edges of E_2 and E_3 form $n'/4$ and $n'/4 - 1$ independent 4-cycles, respectively. If we replace the paths $(w_0, u_0, \ldots, u_{k-1}, w_2)$ and $(w_{n'-3}, v_0, \ldots, v_{k-1}, w_{n'-1})$ both by a single edge, the graph is triangulated; hence, up to the choice of the outer face, there is only one combinatorial embedding, the one in Fig. 1. Therefore, each of the $2n'/4 - 1$ edge-disjoint 4-cycles defined by E_2 and E_3 must separate the red vertices from the blue vertices outside in any planar embedding.

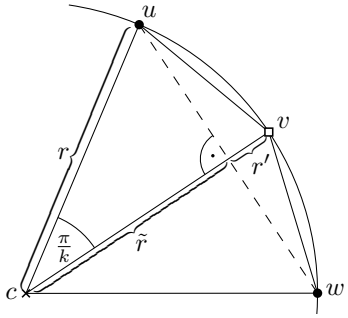


Figure 2: Angles and distances in the regular $2k$-gon.

In our target positions, the red and blue vertices will form a *bi-colored sequence* $(u_0, v_0, u_1, v_1, \ldots, u_{k-1}, v_{k-1})$. Since every pair of consecutive vertices in the sequence has different colors, each 4-cycle must cross the straight-line segment connecting the vertices in order to separate them. Badent et al. [2] arranged the bi-colored sequence as consecutive points on a straight-line. However, if $\delta' > 0$, a δ'-movement that moves all red vertices down and all blue vertices up, easily allows the points to be separated by a single straight-line segment. Hence, we need a different construction in order to ensure that even after a δ-movement the vertices are hard to separate.

We do so by putting the points of the bi-colored sequence at the corners of a regular $2k$-gon with circumradius $r = (n-1)/2$ centered in the center c of the $n \times n$ grid. We want that after a δ'-movement of the vertices the $2k$-gon must still be convex, no matter how the vertices are moved. To this end, consider the relative positions of three consecutive vertices u, v, and w in the bi-colored sequence; see Fig. 2. As long as v stays on the same side of the line $\overline{uw}$, the angle at v remains convex. Since we can move both this line (by moving u and w) and v, we therefore require that $\delta' \leq r'/2$. We have $\cos(\pi/k) = \tilde{r}/r$. Hence,

$$\delta' \leq \frac{r'}{2} = \frac{r - \tilde{r}}{2} = \frac{r - r\cos(\pi/k)}{2} = \frac{r \cdot (1 - \cos(\pi/k)}{2}.$$

The Taylor series definition of the cosine function yields $\cos(\pi/k) \geq 1 - (\pi/k)^2/2 = 1 - \pi^2/(2k^2)$. Hence, $\delta' \leq (r\pi^2)/(4k^2)$. For any such δ' any δ'-movement of the vertices of the bi-colored sequence results in a convex polygon defined by the sequence in the input order.

We now modify the construction for the bi-colored sequence so that all target positions lie on points of the $n \times n$ grid. We assume that n is sufficiently large so that we can feasibly set $r\pi^2/4k^2 \geq \delta' \geq \sqrt{2}/2$. Let $\delta = \delta'/2$. We move each vertex of the $2k$-gon to the nearest grid point, which is at most $\sqrt{2}/2$ away. In any feasible solution, the distance of a vertex to its position on the

curve complexity b	$\Theta(1)$	$\Theta(\sqrt[3]{n})$	$\Theta(\sqrt{n})$
$\delta(b,n)$	$\Omega(n)$	$\Omega(\sqrt[3]{n})$	$\Omega(1)$

Table 1: Lower bounds for the displacement δ depending on the curve complexity b from Theorem 4.

$2k$-gon—resulting both from moving the vertex and from placing it on a grid point—is at most $\delta + \sqrt{2}/2 \leq \delta'$. Hence, the bi-colored sequence forms a convex polygon.

After placing the target positions for the bi-colored sequence, we place the target positions of the remaining vertices on unused points of the $n \times n$ grid.

The intersection of a straight line with a convex polygon is a straight-line segment, and, hence, each edge segment crosses the polygon's boundary at most twice.

Property A polyline that crosses the boundary of a convex polygon b times has at least $\lceil b/2 \rceil - 1$ bends.

On the other hand, each 4-cycle must separate the two sets of the bi-colored sequence and especially every pair of consecutive vertices; hence, every 4-cycle must cross each of the $2k$ edges of the corresponding convex polygon. Therefore, at least one of the edges of such a 4-cycle must have at least $\lceil 2k/4 \rceil = \lceil k/2 \rceil$ crossings with the boundary of the convex polygon. This edge must, hence, have at least $\lceil (\lceil k/2 \rceil)/2 \rceil - 1 = \lceil k/4 \rceil - 1$ bends.

Recall that $\delta \leq r\pi^2/(8k^2) = (n-1)\pi^2/(16k^2)$. Since we want to create an instance where each 4-cycle has an edge with at least $b+1$ bends, we must have $k \geq 4b+5$; we choose $k = 4b+5$. Hence, we can set $\delta = (n-1)\pi^2/(16 \cdot (4b+5)^2) = \Theta(n/b^2)$. Recall that we required that $r\pi^2/4k^2 \geq \sqrt{2}/2$. This is the case if $n \geq 4\sqrt{2}(4b+5)^2/\pi^2 + 1$. Furthermore, since there is a total of $2k = 8b+10$ red and blue vertices, there is a linear number $n' = n - 2k$ of remaining vertices forming the 4-cycles. Hence, in any planar drawing with just a δ-movement of the vertices, there will be a linear number of edges with more than b bends. □

The general form of the theorem allows to choose the curve complexity b, as long as $n \geq 4\sqrt{2}(4b+5)^2/\pi^2 + 1$. This yields some interesting bounds; see also Table 1.

We first consider constant curve complexity. It is not surprising, that there are examples in which no constant curve complexity is sufficient. However, this is even the case with $\delta = \Theta(n)$, i.e., a linear size of δ may still be not enough freedom for constant curve complexity.

Corollary 5 *For every constant number $b \geq 0$ of bends and every number $n \geq 4\sqrt{2}(4b+5)^2/\pi^2 + 1$ it holds that $\delta(b,n) = \Omega(n)$. Furthermore, for every such n, there is an instance with $\delta = \Theta(n)$ such that in every feasible embedding $\Theta(n)$ edges must have more than b bends.*

Our construction also works for a curve complexity of $b = \Theta(\sqrt{n})$, and yields a constant δ-value.

Corollary 6 *Let $b = \Theta(\sqrt{n})$. For $n \geq 4\sqrt{2}(4b(n)+5)^2/\pi^2 + 1$ it holds that $\delta(b,n) = \Omega(1)$.*

Finally, both δ and b can be of $\Theta(\sqrt[3]{n})$; especially, both values are sublinear but not constant.

Corollary 7 *Let $b = \Theta(\sqrt[3]{n})$. For $n \geq 4\sqrt{2}(4b(n)+5)^2/\pi^2 + 1$ it holds that $\delta(b,n) = \Omega(\sqrt[3]{n})$.*

Note that Theorem 4 does not yield a bound for linear b. However, this restriction stems only from requiring that the target positions must lie on the grid. If we drop this requirement, we can place the bi-colored sequence on the corners of the regular $2k$-gon and get an example with a small—but positive—δ, for which a linear number of edges needs a linear number of bends. Note that, although the target positions do not lie on grid points, we still have the property that between every pair of vertices there is a larger distance, i.e., points do not come too close; in this case, the distance is at least constant.

Corollary 8 *Let b be a function linear in n. For every n with $n \geq 4b(n)+9$, there is a planar graph with target positions (not lying on the $n \times n$ grid) and a value $\delta > 0$ such that every AGD embedding will have an edge with at least $b(n)$ bends.*

If $n - 4b = \Theta(n)$, these instances will even have a linear number of edges with a linear number of bends.

5 Bounds for $b(\delta, n)$

We now assume that δ is prescribed and derive bounds for the minimum $b(\delta, n)$ that is sufficient for all instances on n vertices. We reuse the constructions for $\delta(b,n)$. However, we must be careful with the modified analysis.

Upper Bounds. The constructions in Section 3 can be used directly for obtaining upper bounds on $b(\delta, n)$.

Theorem 9 *For $\delta > (n-1)/2$ it holds that $b(\delta,n) \leq 2$ and for $\delta > (n-1)/\sqrt{2}$ it holds that $b(\delta,n) = 0$.*

Lower Bounds. Using the examples of Section 4, we can also derive lower bounds on $b(\delta, n)$.

Theorem 10 *Let $\delta \geq \sqrt{2}/4$. Then, for every $n \geq 1$ it holds that $b(\delta,n) \geq \sqrt{(n-1)/\delta} \cdot \pi/16 - 1$, that is, $b(\delta,n) = \Omega(\sqrt{n/\delta})$.*

More precisely, there is always an example in which a linear number of edges must have at least $\sqrt{(n-1)/\delta} \cdot \pi/16 - 1$ bends if the vertices are moved by at most δ.

Proof. We can use the same construction as in Theorem 4. However, now the displacement δ is prescribed and we want to maximize k such that the resulting $2k$-gon will stay convex after any δ-movement of the vertices. We require $\delta \geq \sqrt{2}/4$ since otherwise moving the vertices

maximum displacement δ	$\Theta(1)$	$\Theta(\sqrt[3]{n})$	$\Theta(n)$
$b(\delta, n)$	$\Omega(\sqrt{n})$	$\Omega(\sqrt[3]{n})$	$\Omega(1)$

Table 2: Lower bounds for the curve complexity b depending on the displacement δ from Theorem 10.

from the corners of the regular $2k$-gon to the closest grid points can have more influence than the δ-movement.

Recall that we required $\delta \leq (n-1)\pi^2/(16k^2)$. Hence, we can set $k = \lceil\sqrt{(n-1)/\delta} \cdot \pi/4\rceil$. Since every 4-cycle must have an edge with at least $\lceil k/4\rceil - 1$ bends, we know that every feasible planar embedding with only a δ-movement of the vertices must have curve complexity at least $\lceil\sqrt{(n-1)/\delta} \cdot \pi/16\rceil - 1$. Therefore, $b(\delta, n) \geq \lceil\sqrt{(n-1)/\delta} \cdot \pi/16\rceil - 1$.

Since in the example we have $2k = O(\sqrt{n})$ vertices in the bi-colored sequence, there will still be a linear number of 4-cycles and, hence, a linear number of edges that need at least $\sqrt{(n-1)/\delta} \cdot \pi/16 - 1$ bends. □

As a consequence, for every constant δ we get $b(\delta, n) = \Omega(\sqrt{n})$. Especially, no constant curve complexity can be guaranteed with a constant displacement.

Corollary 11 *For every constant displacement $\delta \geq 0$ and $n \geq 1$ it holds that $b(\delta, n) = \Omega(\sqrt{n})$.*

Furthermore, for every n, there is an instance with $b = \Theta(\sqrt{n})$ such that in every feasible δ-b-AGD embedding $\Theta(n)$ edges must have at least b bends.

Again, we can also make δ depend on n. We get essentially the same relation between δ and $b(\delta, n)$ as we did for b and $\delta(b, n)$; see Table 2.

Corollary 12 *For $\delta = \Theta(n)$, $b(\delta, n) = \Omega(1)$.*

Again, there are examples in which both δ and b are of $\Theta(\sqrt[3]{n})$, that is, both are sublinear but not constant.

Corollary 13 *For $\delta = \Theta(\sqrt[3]{n})$, $b(\delta, n) = \Omega(\sqrt[3]{n})$.*

6 Conclusion and Open Problems

We explored the interplay between flexibility in moving vertices away from their target position with the number of bends in planar drawings. We proved upper and lower bounds for the value $\delta(b, n)$ that describes the displacement that has to be allowed in order to be able to draw all planar instances with only b bends per edge. Most importantly, we have seen that for every constant curve complexity b, $\delta(b, n)$ is still linear. Furthermore, even $\Theta(\sqrt[3]{n})$ curve complexity is not achievable with constant displacement, but requires $\Omega(\sqrt[3]{n})$ displacement. On the other hand, we have also shown that any constant maximum displacement δ still requires $b(\delta, n) = \Omega(\sqrt{n})$.

There are still several interesting open questions. For instance, for higher constant values of b, the gap in terms of the constants for the upper and lower bounds is quite large. Is there an algorithm that finds a drawing with constant curve complexity b and relatively small linear displacement? Furthermore, we know that for curve complexity $\Theta(\sqrt{n})$ a constant displacement is necessary. Is such a displacement also sufficient, i.e., is there a constant δ such that we can draw every planar graph with displacement just δ and curve complexity $O(\sqrt{n})$?

References

[1] P. Angelini, G. Da Lozzo, M. Di Bartolomeo, G. Di Battista, S.-H. Hong, M. Patrignani, and V. Roselli. Anchored drawings of planar graphs. In C. A. Duncan and A. Symvonis, editors, *Proc. 22nd Int. Symp. Graph Drawing (GD'14)*, volume 8871 of *LNCS*, pages 404–415. Springer-Verlag, 2014.

[2] M. Badent, E. Di Giacomo, and G. Liotta. Drawing colored graphs on colored points. *Theor. Comput. Sci.*, 408(2-3):129–142, 2008.

[3] S. Cabello. Planar embeddability of the vertices of a graph using a fixed point set is NP-hard. *Journal Graph Alg. Appl.*, 10(2):353–366, 2006.

[4] I. Fáry. On straight-line representation of planar graphs. *Acta Sci. Math. (Szeged)*, 11:229–233, 1948.

[5] M. Godau. On the difficulty of embedding planar graphs with inaccuracies. In R. Tamassia and I. G. Tollis, editors, *DIMACS Int. Workshop Graph Drawing (GD'94)*, volume 894 of *LNCS*, pages 254–261. Springer-Verlag, 1995.

[6] T. Gordon. The minimum bends in a polyline drawing with fixed vertex locations. *CoRR*, abs/1406.3860, 2014.

[7] P. Gritzmann, B. Mohar, J. Pach, and R. M. Pollack. Embedding a planar triangulation with vertices at specified positions. *The American Mathematical Monthly*, 98:165–166, 1991.

[8] M. Kaufmann and R. Wiese. Embedding vertices at points: Few bends suffice for planar graphs. *Journal Graph Alg. Appl.*, 6(1):115–129, 2002.

[9] M. Löffler. Existence and computation of tours through imprecise points. *Int. J. Comput. Geometry Appl.*, 21(1):1–24, 2011.

[10] J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. *Graphs and Combinatorics*, 17(4):717–728, 2001.

Index