First Iranian Conference on

Computational Geometry

(ICCG 2018)

Proceedings

Amirkabir University of Technology
Tehran, Iran, February 27, 2018

Foreword

The first Iranian Conference on Computational Geometry was held on February 27, 2018 at the Department of Mathematics and Computer Science of the Amirkabir University of Technology, in Tehran. The goal of this annual, international conference is to bring together students and researchers from academia and industry, in order to promote research in the fields of combinatorial and computational geometry.

This volume of proceedings contains a selection of fifteen refereed papers and an invited talk that were presented during the conference, in three sections. I would like to thank my pc co-chair Mohammad Ali Abam, all the pc members, and members of the local organizing committee (Mohammad Javad Hekmat Nasab, Mohammad Reza Kazemi, and Ali Mohades Khorasani).

I also want to thank the sponsors: Amirkabir University of Technology for financial supports, Islamic World Science Citation Center (ISC) for indexing the conference (#ISC 96171-11103), and Institute for Research in Fundamental Sciences (IPM) and Laboratory of Algorithms and Computational Geometry for other supports they provided.

Zahed Rahmati

Invited Speaker

Maarten Löffler Utrecht University, Netherlands

General Chair

Zahed Rahmati Amirkabir University of Technology

Program Committee

Mohammad Ali Abam	Sharif University of Technology (co-chair)
Mansoor Davoodi Monfared	Institute for Advanced Studies in Basic Sciences
Mohammad Farshi	Yazd University
Mohammad Ghodsi	Sharif University of Technology
Shahin Kamali	University of Manitoba
Ali Mohades Khorasani	Amirkabir University of Technology
Debajyoti Mondal	University of Saskatchewan
Zahed Rahmati	Amirkabir University of Technology (co-chair)
Mohammadreza Razzazi	Amirkabir University of Technology
Alireza Zarei	Sharif University of Technology
Hamid Zarrabi-Zadeh	Sharif University of Technology

Local Organizers

Mohammad Javad Hekmat Nasab	Amirkabir University of Technology
Mohammad Reza Kazemi	Amirkabir University of Technology
Ali Mohades Khorasani	Amirkabir University of Technology
Zahed Rahmati	Amirkabir University of Technology

Conference Program

Accepted (not presented)

Geometry and Topology in Trajectory Analysis

Maarten Löffler*
Utrecht University, Netherlands

Abstract

Trajectories, sequences of time-stamped points in the plane or higher dimensions, are collected in massive quantities across a range of application domains. Consequently, the computational complexity of analysis tasks on trajectories is increasingly studied. One fundamental analysis task for a given set of trajectories is to *cluster* it; that is, to subdivide it into groups of *similar* trajectories.

Clustering of points is a highly studied topic in computational geometry. However, the clustering of trajectories adds significant complexity compared to traditional clustering. We identify three key steps in the design of a clustering algorithm: the choice of a *distance* measure, the choice of a notion of a *center*, and the choice of what we consider a *cluster*. Traditional clustering focuses on the third step, as the first two steps are much less intricate.

We discuss the modeling aspects of each step, focusing on the geometric and topological structure and the resulting algorithmic questions that arise: even evaluating distance or calculating the center of a fixed set of trajectories is not trivial, depending on the modeling choices made. We discuss in detail several recent algorithmic results that can be related to different steps in the clustering paradigm.

*Department of Computing and Information Sciences, Utrecht University, the Netherlands. m.loffler@uu.nl

Connected Guards in A Simple Polygon

Arash Ahadi* Alireza Zarei†

Abstract

Sadhu *et al.* considered cluster connecting problem inside a polygon [6]: For a given set of initial guards inside a given simple polygon $\mathcal{P}$, the goal is to obtain a minimum set of new guards, such that new guards along with the initial ones make a connected visibility graph. The visibility graph of a set of points inside $\mathcal{P}$ is a graph whose vertices correspond to the point set and each edge represents the visibility between its endpoints inside $\mathcal{P}$. They showed that if the new guards are restricted to lie on the vertices of $\mathcal{P}$, the problem is NP-hard, and proposed an approximation solution with logarithmic approximation factor in term of the number of vertices of $\mathcal{P}$. We show that this problem is NP-hard in all cases, where the initial guards and the new ones are restricted to vertices, boundary or all points of $\mathcal{P}$. We propose constant factor approximation algorithms for all cases.

1 Introduction

The *art gallery problem* in a simple polygon, introduced by Klee in 1973 [4], is to determine the minimum number of guards that see all points of the polygon. Afterward, many results have been obtained for different versions of this problem. We refer the reader to [2] for a survey on art gallery problems.

In some versions of the art gallery problem, other than covering the polygon, the inter-visibility of the guards has been considered as well. For example, in some applications we need direct communication between guards or we may prohibit such links. These problems are respectively called *cooperative guard set* and *hidden guard set*. These versions of the problem are well described by *visibility graph*. The visibility graph of a set of points inside a simple polygon is a graph whose vertices are this point set, and there is an edge between corresponding vertices of a pair of points if and only if they are visible from each other. Two points of a polygon are visible from each other (or for simplicity, visible) when their connecting segment completely lies inside the polygon. In this definition, the boundary of the polygon is considered to be inside the polygon. According to this definition, the visibility graph of a *connected guard set* is connected, and it has no edge for a *hidden guard set*.

*Department of Mathematics Science, Sharif University of Technology, arash(underline)ahadi5@yahoo.com

†zarei@sharif.ir

Sadhu *et al.* in [6] introduced another version of the art gallery problem, named *cluster connected*, in which we have an initial set of guards in a given simple polygon and the goal is to obtain minimum number of new guards such that the visibility graph of the whole guards (initial and new ones) is connected. This problem has applications in sensor network, where a set of sensors are distributed to monitor a polygonal region. It may happen that the placed sensors cannot form a connected network either due to the obstructions of the polygonal boundary or due to the malfunction of some sensors. The goal is to locate the minimum number of new sensors to have a connected network [6]. In this paper, we call the initial guards as *terminal guards* and the new guards as *Steiner guards*. To be convenient with the well-known *minimum Steiner tree* problem, we call this version of the art gallery problem *minimum connected Steiner guard set*.

This problem is formally defined as follows:

Problem Γ (minimum connected Steiner guard set):

Instance: A simple polygon $\mathcal{P}$ and a finite set of points Q as *terminal guards* inside (or maybe on the boundary of) $\mathcal{P}$.

Question: Find a minimum set X of points such that $Q \subseteq X$ and the visibility graph of points X, with respect to $\mathcal{P}$, is connected.

For a set X that its visibility graph is connected, we call the points of $X - Q$ by *Steiner guards*. If locations of Steiner guards are restricted to a set of points A, we denote the problem by Γ_A. Specifically, we consider $\Gamma_{V(\mathcal{P})}$ and $\Gamma_{\rho(\mathcal{P})}$, where $V(\mathcal{P})$ is the set of vertices of $\mathcal{P}$ and $\rho(\mathcal{P})$ is the boundary of $\mathcal{P}$. Based on the restriction of locations of terminal or Steiner guards, this problem can be considered in some versions: when locations of (i) terminal guards, (ii) Steiner guards, (iii) both, are restricted to (i′) the vertices of $\mathcal{P}$, (ii′) the boundary of $\mathcal{P}$ or (iii′) any point inside $\mathcal{P}$.

In [6] the NP-hardness of this problem has been shown, when locations of both terminal and Steiner guards are restricted to the vertices of $\mathcal{P}$. They also presented a $\log n$ factor approximation algorithm for the number of new guards in this restricted version of the problem where n is the number of vertices of $\mathcal{P}$.

In this paper, we consider all versions of Γ. We show NP-hardness of all cases and present $\mathcal{O}(1)$ factor approximation algorithms. Our NP-hardness proof is simpler and shorter than the proof in [6]. In the approximation algorithms we build an enhanced graph over a

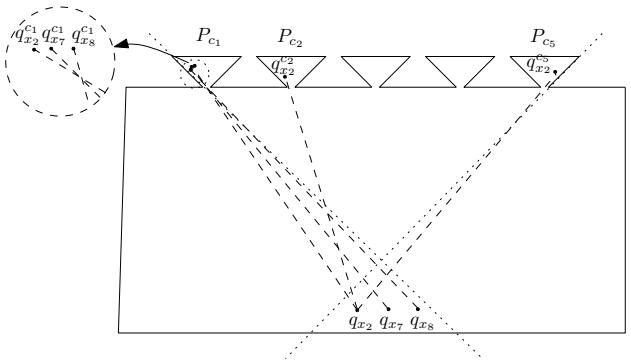


Figure 1: The instance of Problem Γ for $\psi = (X; C)$, where $c_1 = x_2 \wedge x_7 \wedge x_8$ and x_8 is a literal of c_1, c_2 and c_5.

finite subset of points of $\mathcal{P}$ (compared to the graph used in [6]) and solve an instance of *Steiner tree problem*. We prove that approximation factors of our algorithms are 2.79 for the general and $\Gamma_{\rho(\mathcal{P})}$ versions of the problem and 1.39 for $\Gamma_{V(\mathcal{P})}$.

In the rest of the paper, we first prove the NP-hardness in Section 2 and present approximation algorithms in Section 3.

2 Complexity

We reduce a known NP-complete satisfiability problem named *cubic monotone 1-in-3 SAT* to Γ. A satisfiability problem is monotone if there is no negative literal in any clause and is cubic if each literal appears in exactly three cluases. In an instance of a *cubic monotone 1-in-3 SAT* problem, the question is to determine the existence of a $\{true, false\}$ assignment to the variables such that each clause contains exactly one *true* literal. It has been shown that *cubic monotone 1-in-3 SAT* is NP-complete [5].

Theorem 1 *Problem Γ is NP-hard.*

Proof. Let ψ be an instance of *cubic monotone 1-in-3 SAT* problem with literal set X and clause set C. Consider the polygon $\mathcal{P}$ shown in Figure 1 that has $|C|$ petals, one for each clause $c_i \in C$, with sufficiently small aperture. For each literal $x_j \in X$ there is a point q_{x_j} in the bottom part of P. Finally, for each clause $c_i = \alpha \vee \beta \vee \gamma$ there are three distinct points $q_\alpha^{c_i}, q_\beta^{c_i}$ and $q_\gamma^{c_i}$ in petal P_{c_i} such that $q_x^{c_i}$ will be visible from q_x, for every $x \in \{\alpha, \beta, \gamma\}$.

Now consider the input $(\mathcal{P}, \{q_{x_j}^{c_i} : x_j \in c_i\})$ for problem Γ. According to the structure of $\mathcal{P}$, each point $q_{x_j}^{c_i}$ only sees the points $q_\alpha^{c_i}$ of its petal P_{c_i}. Let L be the set of segments each of which connects a literal point $q_{x_j}^{c_i}$ to its clause point q_{c_i} where x_j is a literal of clause c_i. Positions of points q_{c_i}'s have this property that no three segments of L have common intersection point except at their endpoints. This property can be easily achieved by properly positioning points q_{c_i}'s. Moreover, we can decrease the length of aperture of each petal such that if three points q_x^c, $q_{x'}^{c'}$ and $q_{x''}^{c''}$ have common visible region, then either $c = c' = c''$ or $x = x' = x''$.

Each new guard sees at most 3 initial guards. So, every Steiner guard set has at least $\frac{|C|}{3}$ points. There is a set of $\frac{|C|}{3}$ Steiner guards if and only if ψ has some 1-in-3 assignment. The reduction is polynomial which completes the proof. □

By keeping the locations of guards and constricting the boundary of $\mathcal{P}$, we can draw the polygon $\mathcal{P}$ in such a way that points $q_{x_j}^{c_i}$'s or q_{c_i}'s lie on $V(\mathcal{P})$ or $\rho(\mathcal{P})$. Therefore, all nine versions of Problem Γ ((terminal guards, Steiner guards, both types of guards) are restricted to (vertices, boundary, all points) of $\mathcal{P}$) are NP-hard.

3 Approximation Algorithms

Our approximation algorithms for different versions of Γ are based on one idea. Before presenting these algorithms, we need some definitions.

Inside a simple polygon $\mathcal{P}$, a point p is visible from another point q if and only if the line segment $\overline{pq}$ lies entirely inside $\mathcal{P}$. For a point q inside or on the boundary of $\mathcal{P}$, the visibility polygon of q, denoted by $V(q)$ is the set of all points which are visible from q (see Figure 2-A). For a finite set of points A inside $\mathcal{P}$, their *visibility graph*, denoted by $VG(A)$, is a graph with vertex set A, and there is an edge between two vertices if and only if their corresponding points in $\mathcal{P}$ are visible from each other (see Figure 2-B).

If we extend the corresponding segment of each edge of a visibility graph $VG(A)$ in both sides until intersecting the boundary of $\mathcal{P}$, a new subdivision on $\mathcal{P}$ is obtained which is called the *extended visibility graph* of A and is denoted by $EVG(A)$. More precisely, $EVG(A)$ is a graph with vertex set $A \cup B$, which B is the intersection points of the obtained subdivision and each edge corresponds to a segment in this subdivision (see Figure 2-C). The corresponding subdivision of the extended visibility graph $EVG(A)$ has this property that all points of a region see the same subset of points in A and the visible subsets of two adjacent regions differ only in one vertex.

The *second order visibility graph* of a point set A, denoted by $VG^2(A)$, is the visibility graph built on the vertex set of $EVG(A)$. Therefore, the vertex set of $VG^2(A)$ is the same as the vertices of $EVG(A)$. But, have more edges than $EVG(A)$. For example, the edge px exists in $VG^2(\{a, b, c, d, p\})$ in Figure 2-C.

In a graph whose vertices are partitioned into two sets *terminal* vertices T and *Steiner* vertices S, a *Steiner*

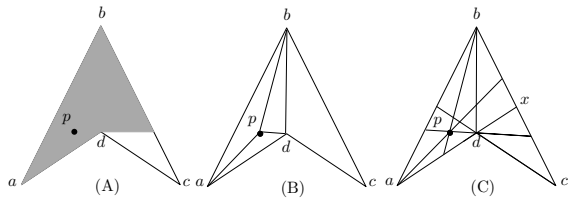


Figure 2: (A) The visibility polygon $V(p)$. (B) The visibility graph $VG(\{a, b, c, d, p\})$. (C) The extended visibility graph $EVG(\{a, b, c, d, p\})$. All points of a region in the subdivision of $EVG(\{a, b, c, d, p\})$ see the same vertices of $\{a, b, c, d, p\}$.

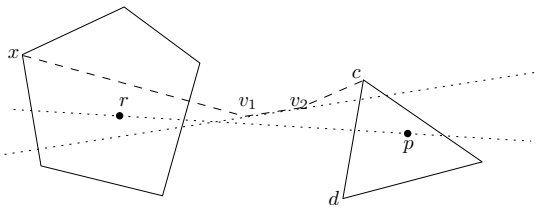


Figure 3: The shortest path between x and c can have at most one internal vertex.

tree is a tree containing all terminal vertices and a *minimum Steiner tree* is a *Steiner tree* of minimum possible number of Steiner vertices.

Now, we consider the relations between Γ, second order visibility graph and a minimum Steiner tree. Let OPT be an optimal solution of Γ for polygon $\mathcal{P}$ and terminal guards Q. In the following we denote the set of vertices of $\mathcal{P}$ by V.

Lemma 2 *There is a Steiner tree of at most* $2|OPT|$ *Steiner vertices for the Steiner tree problem on graph* $VG^2(Q \cup V)$ *and terminal vertices* Q.

Proof. We use a constructive argument to prove. Let T be a spanning tree for $VG(Q \cup OPT)$. We build a Steiner tree S on $VG^2(Q \cup V)$ for Steiner vertices Q as follows. Initialize S as an spanning forest of induced subgraph of T on vertices Q. Trivially, S is a subgraph (not necessarily connected) of $VG^2(Q \cup V)$. To make S connected, for each vertex $v \in OPT$, we select at most two vertices from the vertex set of $VG^2(Q \cup V)$, and add them along with some new edges to S.

This is done by iteratively considering vertices in OPT. We start from vertices which are connected to some vertices of Q in T. Let e_{qp} be an edge connecting the vertex $q \in Q$ to the vertex $p \in OPT$ in T. The point p lies inside a region of the subdivision of $EVG(Q \cup V)$ and all points of this region (including vertices of the boundary of this region) see q as well. Remember that all points of a region of $EVG(Q \cup V)$ see the same subset of points in $Q \cup V$, and q belongs to Q. On the other hand, all boundary vertices of this region belong to the vertices of $VG^2(Q \cup V)$ and are connected to q in this graph. Therefore, we can add to S anyone of these boundary vertices along with all edges connecting this vertex to all vertices of Q that are connected to p in T. Note that in this step, any one of the vertices of OPT is handled by adding one vertex to S.

The remaining vertices are handled by processing edges of T that connect pairs of vertices of OPT. For this purpose, we iteratively choose an edge which connects a *currently handled vertex* of OPT to another vertex of OPT. Currently handled vertex means that we have already added to S one of the boundary vertices of its containing region. Let e_{rp} be an edge which connects the currently handled vertex $r \in OPT$ to a vertex $p \in OPT$. As shown in Figure 3, assume that x is the vertex from the boundary region of r that has been added to S, and cd is the boundary edge of the containing region of p in $EVG(Q \cup V)$ intersected by segment rp. As shown in Figure 3, c is the vertex that lies on the same side of the line through r and p as x. Consider the shortest path between vertices x and c in $\mathcal{P}$. This geodesic path is a concave polygonal chain with end points x and c whose internal breakpoints are vertices of $\mathcal{P}$ (this is because r and p are visible from each other).

We prove that this chain has at most one internal vertex. For the sake of a contradiction, assume that there are two internal vertices v_1 and v_2 on this path. These vertices are visible from each other and therefore, extensions (in both sides) of their connecting segment exist in $EVG(Q \cup V)$. These extensions intersect at least one of the containing regions of r and p which is in contradiction with the definition of a region in the extended visibility graph for points p and r. For more details, in that case at least one of the points x or c cannot lie on the boundary region of respectively r and p.

If the chain has no internal vertex, we add to S the vertex c and the edge xc, which both exist in $VG^2(Q \cup V)$. Otherwise, we add the only internal vertex v_1 along with the edge xv_1, which both exist in $VG^2(Q \cup V)$. Moreover (for the latter case only), if we have not yet added a vertex on the boundary of the containing region of p to S, the vertex c and the edge v_1c are also added to S, which again both exist in $VG^2(Q \cup V)$, and otherwise, if we have already added a vertex x' on the boundary region of p to S, we only add the edge $x'v_1$ to S. The edge $x'v_1$ exists in $VG^2(Q \cup V)$ because c and v_1 are visible from each other, $v_1 \in V$, and x' and c belong to the same region of the extended visibility graph $EVG(Q \cup V)$.

After processing all edges between vertices of OPT in T, we obtain a connected Steiner tree in S. This tree has at most $|OPT|$ Steiner vertices added in the first and second step which are vertices of the boundary of the regions that contain vertices OPT. Moreover, it

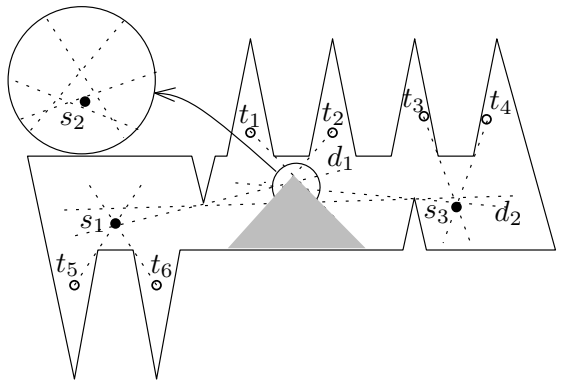


Figure 4: In any optimal answer for terminal guards $\{t_1, \ldots, t_6\}$, the middle Steiner guard, s_2, must lie inside (not on the boundary) of a region of the extended visibility graph (above line d_2 and under d_1).

has at most $|OPT| - 1$ vertices added in the second step (one for each edge). □

Figure 4 shows an example in which Steiner guards in any optimal solution are not a subset of the vertices of $VG^2(Q \cup V)$ graph. In this figure, $t_1, \ldots, t_6$ are the initial guards and we need at least three Steiner guards to have a connected guard set. If we put s_1 and s_3 Steiner guards on some vertices of $VG^2(Q \cup V)$ (as put in this figure), the third Steiner guard s_2 must lie in the gray region (to see both t_1 and t_2) and above the line d_2 (to see s_3) and bellow d_1 (to see s_1). As the figure shows, there is no vertex of $VG^2(Q \cup V)$ in the region constrained by these three conditions.

The above lemma gives an approximation algorithm for the general version of Problem Γ. Note that the process used for building S is only for the proof purpose; and is not a part of our approximation algorithm.

Theorem 3 *There is a polynomial time approximation algorithm which for any instance of Γ, obtains at most $(4 \ln 2 + \epsilon)|OPT| \simeq 2.79|OPT|$ Steiner guards, where $|OPT|$ is the number of Steiner guards in an optimal solution.*

Proof. For solving Γ on an instance polygon $\mathcal{P}$ and terminal guards Q, we compute the second order visibility graph $VG^2(V \cup Q)$. This is possible in polynomial time. Then, we solve an instance of the minimum Steiner tree problem on graph $VG^2(V \cup Q)$ with terminal vertices Q and output the obtained Steiner vertices as the Steiner guards (precisely, their corresponding points in $\mathcal{P}$). There is an $2 \ln 2 + \epsilon \simeq 1.39$ approximation algorithm for the minimum Steiner tree problem ([1] and the first theorem of [3]). Combining this algorithm with the result of Lemma 2, the approximation factor of our algorithm is obtained. □

Let B be the vertices of $EVG(Q \cup V)$ which lie on the boundary of $\mathcal{P}$, and OPT be an optimal solution of $\Gamma_{\rho(\mathcal{P})}$ for polygon $\mathcal{P}$ and terminal guards Q. By the same argument as Lemma 2, it can be proved that there is a Steiner tree of at most $2|OPT|$ Steiner vertices for the Steiner tree problem on graph $VG(B)$ and terminal vertices Q. This means that $\Gamma_{\rho(\mathcal{P})}$ can be solved in polynomial time by an approximation algorithm which obtains at most $(4 \ln 2 + \epsilon)|OPT| \simeq 2.79|OPT|$ Steiner guards.

The following theorem shows that we can halve the approximation factor for $\Gamma_{V(\mathcal{P})}$ version of the problem in which the Steiner guards are restricted to the vertices of $\mathcal{P}$.

Theorem 4 *There is a* 1.39*-approximation algorithms for $\Gamma_{V(\mathcal{P})}$ problem.*

Proof. Similar to the proof of Theorem 3, we build the visibility graph $VG(V \cup Q)$ on terminal guards Q and vertices of $\mathcal{P}$ and solve an instance of the minimum Steiner tree problem on this graph with terminal vertices Q using the $2 \ln 2 + \epsilon \simeq 1.39$ approximation algorithm [1]. This gives us a 1.39 approximation factor algorithm for Problem $\Gamma_{V(\mathcal{P})}$. □

References

[1] J. Byrka, F. Grandoni, T. Rothvo and L. Sanita An improved LP-based approximation for Steiner tree *Proc. of the 42nd ACM symposium on theorey of computing* 583-592, 2010.

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

[3] M. Hauptmann and M. Karpińsky A compendium on Steiner tree problems *für Informatik*, 2013.

[4] D. T. Lee and A. K. Lin Computational complexity of art gallery problems. *IEEE Transactions on Information Theory* 32(2):276-282, 1986.

[5] C. Moore and J. M. Robson Hard tiling problems with simple tiles. *Discrete Computational Geometry* 26(4):573-590, 2001.

[6] S. Sadhu, A. Bishnu, S. C. Nandy and P. P. Goswami Cluster connecting problem inside a polygon. *Proc. of the 22-th Canadian Conference on Computational Geometry*, 2010.

Fault Tolerancy of Continuous Yao Graph

Davood Bakhshesh* Mohammad Farshi†

Abstract

Let S be a point set in the plane and $0 < \theta \leq 2\pi$ be a real number. The continuous Yao graph $cY(\theta)$ for S is constructed as follows. For each $p, q \in S$, we add the edge (p, q) to $cY(\theta)$, if there exists a cone C with apex at p and aperture θ such that q is the closest point to p inside C. In this paper, we show that for every $\pi/3 \leq \theta < 2\pi/5$, the continuous Yao graph $cY(\theta)$ is a region-fault tolerant t-spanner in which t only depends on θ.

1 Introduction

Let $S \subset \mathbb{R}^2$ be a set of points. A graph G with vertex set S is called *geometric graph*, if each edge $e = (p, q)$ in G is a straight-line between p and q, and the weight of e is the Euclidean distance between p and q, denoted by $|pq|$. A geometric graph G with vertex set S is called a *t-spanner* for $t \geq 1$, if for each pair $p, q \in S$, there exists a path in G between p and q of length at most $t|pq|$, where the length of a path in G is the sum of the weights of all edges on the path. We call a such path a *t-path* between p and q. The smallest value of $t \geq 1$ that makes a geometric graph G a t-spanner is called *dilation* (*stretch factor*) of G. For a geometric graph $G = (S, E)$, a geometric graph $G' = (S, E')$ with $E' \subseteq E$ is called a *t-spanner for G*, if for each two points $p, q \in S$, $\delta_{G'}(p, q) \leq t \times \delta_G(p, q)$, where $\delta_G(p, q)$ is the length of the shortest path between p and q in G. The reader can see the book by Narasimhan and Smid [4] to find some algorithms for efficient construction of t-spanners and their applications.

In 2014, Barba et al. [3] introduced the *continuous Yao graphs* to construct a t-spanner for a given point set. Let S be a set of points in the plane and let $0 < \theta \leq 2\pi$ be a real number. The *continuous Yao graph* $cY(\theta)$ for S is constructed as follows. For each point $p \in S$, $cY(\theta)$ contains an edge (p, q) if and only if there is a cone with apex at p and aperture θ such that q is the closest point to p inside the cone. Although the continuous Yao graphs unlike the regular Yao graphs may have a quadratic number of edges, they have some advantages [2]. For example the continuous Yao graphs unlike the regular Yao graphs are invariant under the rotation of the point set [2].

An important feature of a network is *fault tolerancy* in the sense that if we remove some of its nodes and edges, then the remaining network is still a suitable network. Abam et al. [1] introduced the concept of *region-fault tolerant t-spanner*. Let F be a region in the plane. Assume that we delete all edges and vertices of G that have intersection with the region F. We denote the remaining graph by $G \ominus F$. Let $\mathcal{F}$ be a set of regions in the plane. We call G is an *$\mathcal{F}$-fault tolerant* t-spanner if, for any region $F \in \mathcal{F}$, the graph $G \ominus F$ is a t-spanner for $K_S \ominus F$, where K_S is the complete geometric graph on S.

In [2], Bakhshesh et al. presented some results on the dilation of the continuous Yao graphs and their fault tolerancy. They proved that the dilation of $cY(\theta)$ is at most $1/(1-2\sin(\theta/4))$ when $\theta < 2\pi/3$ and 6.0411 when $\theta = 2\pi/3$. For larger angles, they showed that $cY(\theta)$ may be disconnected when $\theta > \pi$ and it is connected when $\theta \leq \pi$. They also showed that the dilation of $cY(\pi)$ is unbounded. Moreover, they showed that if $0 < \theta < \pi/3$, then the continuous Yao graph $cY(\theta)$, is a $\mathcal{C}$-fault tolerant geometric t-spanner for $t \geq \frac{1}{1-2\sin(\theta/2)}$, where $\mathcal{C}$ is the family of all convex regions in the plane. They left the study of fault tolerancy of $cY(\theta)$ for $\theta \geq \pi/3$ as an open problem.

In this paper, we show that the continuous Yao graph $cY(\theta)$ for $\pi/3 \leq \theta < 2\pi/5$, is a $\mathcal{C}$-fault tolerant geometric t-spanner for a constant number t that only depends on θ.

2 Preliminaries

Let h be an arbitrary half-plane, and a and b be an arbitrary pair of points among the points of $cY(\theta) \ominus h$. We define the cone C^*_{ab} as follows. Let ℓ_{ab} be the line supported by a and b and α be the obtuse or right angle outside h between ℓ_{ab} and the boundary of h as depicted in Figure 1. Also, let A be the region obtained by ℓ and boundary of h as depicted in Figure 1.

Let C^*_{ab} be the cone with apex a such that one of its boundaries passes through b and its other boundary is in the region A. Analogously, we can define C^*_{ba}. Since $\alpha \geq \pi/2$ and a and b are outside h, according to the definition of the cones C^*_{ab} and C^*_{ba}, it is not hard to see

*Department of Computer Science, University of Bojnord, Bojnord, Iran. dbakhshesh@gmail.com

†Combinatorial and Geometric Algorithms Lab., Department of Computer Science, Yazd University, Yazd, Iran. mfarshi@yazd.ac.ir

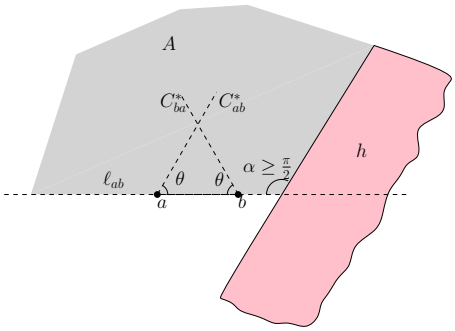


Figure 1: Region A and angle α.

that the closest point to a in C^*_{ab} and the closest point to b in C^*_{ba} are outside h too.

Let $a = (0,0)$ and $b = (1,0)$ be two points in the plane. Here, we use the notations of Bakhshesh et al [2]. They defined *inductive set of a with respect to b*, denoted by I_{ab}, as follows.

$$I_{ab} = \{p \in \mathbb{R}^2 : |ap| + t|pb| \leq t|ab|\}$$

Let $c \neq b$ be the point on the boundary of C^*_{ab} such that $|ac| = |ab|$ (see Figure 2). Symmetrically, let $c' \neq a$ be the point on the boundary of C^*_{ba} such that $|bc'| = |ab|$. Let u be the intersection points of the boundary of I_{ab} with the segment ac. Symmetrically, let w be the intersection points of boundary of I_{ba} with the segment bc'. Also, let v be the intersection point between I_{ab} and the circular arc of $\mathcal{C}(a,b,c)$, where $\mathcal{C}(a,b,c)$ is the the circular sector between two radii ab and ac of a circle with apex at a. Analogously, let v' be the intersection point between I_{ba} and $\mathcal{C}(b,c',a)$.

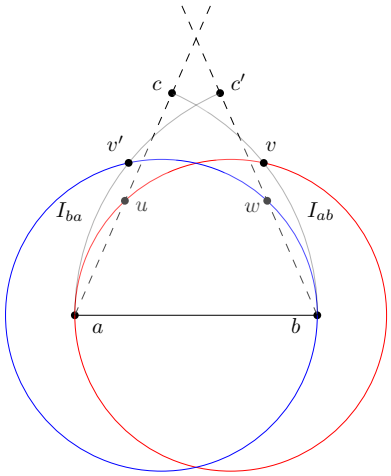


Figure 2: Inductive sets I_{ab} and I_{ba} and the intersection points.

Since $a = (0,0)$ and $b = (1,0)$, if $p = (x,y) \in I_{ab}$, we have

$$((-2+x)x + y^2)^2\, t^4 + (x^2+y^2)^2 - 2(2 + (-2+x)x + y^2)(x^2+y^2)\, t^2 = 0. \quad (1)$$

Now, using Equation (1), we have

$$u = \left(\frac{2t\left(t - \sqrt{\tan^2(\theta)+1}\right)}{\left(\tan^2(\theta)+1\right)(t^2-1)}, \frac{2t\tan(\theta)\left(t - \sqrt{\tan^2(\theta)+1}\right)}{\left(\tan^2(\theta)+1\right)(t^2-1)} \right),$$

$$w = \left(1 - \frac{2t\left(t - \sqrt{\tan^2(\theta)+1}\right)}{\left(\tan^2(\theta)+1\right)(t^2-1)}, \frac{2t\tan(\theta)\left(t - \sqrt{\tan^2(\theta)+1}\right)}{\left(\tan^2(\theta)+1\right)(t^2-1)} \right),$$

$$v = \left(\frac{t^2+2t-1}{2t^2}, \frac{(t-1)\sqrt{3t^2+2t-1}}{2t^2} \right),$$

$$v' = \left(\frac{(t-1)^2}{2t^2}, \frac{(t-1)\sqrt{3t^2+2t-1}}{2t^2} \right),$$

$$c = (\cos(\theta), \sin(\theta)),$$

$$c' = (1 - \cos(\theta), \sin(\theta)).$$

Now, we have the following claim.

Claim 1 *If* $t \geq \frac{1}{2\cos(\theta)+1-\sqrt{2\cos(\theta)+2}}$, *then* $\max\{|vv'|, |uc'|\} \leq |uw|$.

Proof. To prove the lemma, we show that $|uc'| \leq |uw|$ and $|vv'| \leq |uw|$. First, we show that $|uc'| \leq |uw|$. As we know the coordinates of points u, w and c', we have

$$|uc'| = \sqrt{\left(\frac{2\left(t-\sqrt{\tan^2(\theta)+1}\right)t}{(\tan^2(\theta)+1)(t^2-1)} - 1 + \cos(\theta)\right)^2 + \left(\frac{2\tan(\theta)\left(t-\sqrt{\tan^2(\theta)+1}\right)t}{(\tan^2(\theta)+1)(t^2-1)} - \sin(\theta)\right)^2} \quad (2)$$

and

$$|uw| = 1 - \frac{4\left(t - \sqrt{\tan^2(\theta)+1}\right)t}{\left(\tan^2(\theta)+1\right)(t^2-1)}. \quad (3)$$

Now, by substituting equations (2) and (3) in the inequality $|uc'| \leq |uw|$ and simplifying the inequality, we have

$$\Big((t^2 + 2t - 1 - 2t^2\cos(\theta))(2\cos(\theta) - 1) \Big/ (4\cos^2(\theta)t^2 + +2t^2\cos(\theta) - 4t\cos(\theta) - t^2 - 2t + 1)/(t^4 - 2t^2 + 1) \Big) \leq 0. \quad (4)$$

Since $\theta \geq \pi/3$ and $t > 1$, obviously we have $2\cos(\theta) - 1 \leq 0$ and $t^2 + 2t - 1 - 2t^2\cos(\theta) = (1 - 2\cos(\theta))t^2 + 2t - 1 \geq 0$. Hence, the inequality (4) is true if we have

$$\left(4\cos^2(\theta)t^2 + 2t^2\cos(\theta) - 4t\cos(\theta) - t^2 - 2t + 1\right) \geq 0,$$

which is equivalent to

$$1+\left(4\cos^2(\theta)+2\cos(\theta)-1\right)t^2+(-4\cos(\theta)-2)\,t\geq 0. \quad (5)$$

Consider the equation

$$1+\left(4\cos^2(\theta)+2\cos(\theta)-1\right)x^2+(-4\cos(\theta)-2)\,x=0.$$

This equation has two solutions, as

$$x=\frac{1}{2\cos(\theta)+1\pm\sqrt{2\cos(\theta)+2}}.$$

Clearly, $\frac{1}{2\cos(\theta)+1+\sqrt{2\cos(\theta)+2}} < 1$. On the other hand, for $\pi/3 \leq \theta < 2\pi/5$, we have $4\cos^2(\theta)+2\cos(\theta)-1>0$. Hence, since $t>1$, the inequality (5) is true if

$$t\geq \frac{1}{2\cos(\theta)+1-\sqrt{2\cos(\theta)+2}}.$$

Hence, if $|uc'|\leq|uw|$, then $t\geq\frac{1}{2\cos(\theta)+1-\sqrt{2\cos(\theta)+2}}$. It is not hard to see that the above conclusions are reversible. Hence, if $t\geq\frac{1}{2\cos(\theta)+1-\sqrt{2\cos(\theta)+2}}$, then $|uc'|\leq|uw|$.

Now, we show that $|vv'|\leq|uw|$. Using the coordinates of the points v and v', we have $|vv'|=\frac{2t-1}{t^2}$. Now, consider the function

$$f(\theta)=|uw|=1-\frac{4\left(t-\sqrt{\tan^2(\theta)+1}\right)t}{\left(\tan^2(\theta)+1\right)(t^2-1)}.$$

Clearly $f(\pi/3)=\frac{2t-1}{t^2-1}\geq\frac{2t-1}{t^2}=|vv'|$. To prove that $|vv'|\leq|uw|$, it is sufficient to show that the function f is an increasing function for $\pi/3\leq\theta<2\pi/5$. Now, the derivative f' of f with respect to θ is

$$f'(\theta)=\frac{4t\sin(\theta)\left(2t\cos(\theta)-1\right)}{t^2-1}.$$

Since $t\geq\frac{1}{2\cos(\theta)+1-\sqrt{2\cos(\theta)+2}}$ and $\cos(\theta)\leq 1/2$, clearly $t\geq\frac{1}{2\cos(\theta)+1-\sqrt{3}}\geq\frac{1}{2\cos(\theta)}$, and therefore $2t\cos(\theta)-1\geq 0$. Hence, obviously, we have $f'(\theta)\geq 0$, and therefore f is an increasing function. Hence, if $t\geq\frac{1}{2\cos(\theta)+1-\sqrt{2\cos(\theta)+2}}$, then $|vv'|\leq|uw|$. This proves the claim. □

Let n_a be the closest point to a in C^*_{ab} and n_b be the closest point to b in C^*_{ba}. Now, we have

Claim 2 *If $n_a\notin I_{ab}$ and $n_b\notin I_{ba}$, then $|n_an_b|\leq\max\{|uc'|,|uw|\}$.*

Proof. Proof is similar to the proof of Lemma 3 in [2]. The only difference is that we define N_a and N_b to be the convex hull of $\mathcal{C}(a,b,c)\backslash I_{ab}$ and the convex hull of $\mathcal{C}(b,c,a)\backslash I_{ba}$, respectively as depicted in Figure 3. Notably, one of the two points realizing the maximum $|n_an_b|$ must be on the boundary of N_a and another one on the boundary of N_b that follows from [2]. The proof is follows as the proof of Lemma 3 in [2]. □

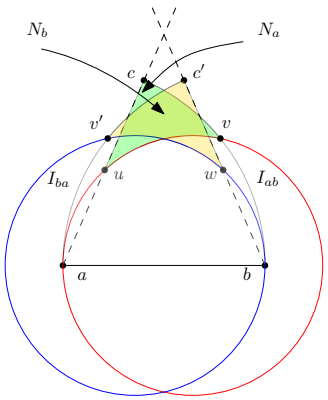


Figure 3: The regions in Claim 2.

3 Main Result

In this section, we show that for any $\pi/3\leq\theta<2\pi/5$, the continuous Yao graph $cY(\theta)$ is a $\mathcal{C}$-fault tolerant t-spanner. We will need the following lemma.

Lemma 1 ([1]) *A geometric graph G on point set S is a $\mathcal{C}$-fault tolerant t-spanner if and only if it is an $\mathcal{H}$-fault-tolerant t-spanner, where $\mathcal{H}$ is the family of all half-planes.*

Theorem 2 *For any $\pi/3\leq\theta<2\pi/5$, $cY(\theta)$ is a $\mathcal{C}$-fault-tolerant geometric t-spanner, where t is the smallest real number such that $t\geq\frac{1}{2\cos(\theta)+1-\sqrt{2\cos(\theta)+2}}$ and $4\cos^2(\theta)t^3-(2+4\cos(\theta))t^2+2\geq 0$.*

Proof. By Lemma 1, it is sufficient to prove that $cY(\theta)$ is an $\mathcal{H}$-fault-tolerant geometric t-spanner. Hence, we show that for every half-plane h and for each pair of points a and b in $cY(\theta)\ominus h$, there is a t-path between a and b in $cY(\theta)\ominus h$.

Let h be an arbitrary half-plane, and a and b be an arbitrary pair of points among the points of $cY(\theta)\ominus h$. We make the proof by induction on the rank of distance $|ab|$. For the base step, suppose that (a,b) is the closest pair of points among the points of $cY(\theta)\ominus h$. Consider the cone C^*_{ab}. Now, rotate it a bit around the point a clockwise such that the point b completely lies inside the cone. We denote the resulting cone by C^+. Since (a,b) is the closest pair of points, the edge ab is added by C^+. It is obvious that if (a,b) is not a unique closest point, using similar reason, it is added to $cY(\theta)$. Assume now that for each two points x and y among the points of $cY(\theta)\ominus h$ with $|xy|<|ab|$, there exists a t-path between

x and y in $cY(\theta) \ominus h$. In the following, we prove that there is a t-path between a and b in $cY(\theta) \ominus h$.

Without loss of generality, assume that $a = (0, 0)$ and $b = (1, 0)$. Recall the points u, w, v and v' defined in the previous section. According to the definition of points u and w, clearly $|uw| < 1$. Hence, by Claim 1, we have $\max\{|vv'|, |uc'|\} < 1$.

In the following, we assume that $t \geq \frac{1}{2\cos(\theta)+1-\sqrt{2\cos(\theta)+2}}$. Let n_a be the closest point to a in C^*_{ab} and n_b be the closest point to b in C^*_{ba}. Since $n_a \neq a$, $|n_a a| > 0$. Hence, $t|n_a b| < |n_a a| + t|n_a b|$. Now, if $n_a \in I_{ab}$, then by the definition of I_{ab}, $|n_a a| + t|n_a b| \leq t|ab|$. Hence, by the combination of above equations, we have $|n_a b| < |ab|$. Therefore, we can apply the induction hypothesis on the pair (n_a, b). Hence, we obtain a path Q from n_a to b in $cY(\theta) \ominus h$ of length at most $t|n_a b|$. Now, consider the path $P := (a, n_a) \cup Q$ that is a path in $cY(\theta) \ominus h$ between a and b. By the definition of the set I_{ab}, the length of P, denoted by $|P|$, is equal to

$$|an_a| + |Q| \leq |an_a| + t|n_a b| \leq t|ab|. \text{ (Since } n_a \in I_{ab})$$

Hence, P is a path in $cY(\theta) \ominus h$ between a and b of length at most $t|ab|$ as desired.

Analogously, we can find a t-path between a and b in $cY(\theta) \ominus h$ when $n_b \in I_{ba}$.

Now, assume that $n_a \notin I_{ab}$ and $n_b \notin I_{ba}$. Since $|uc'|$ and $|uw|$ are less than one, by Claim 2 we have $|n_a n_b| < 1 = |ab|$. Therefore, we can apply induction on (n_a, n_b). Hence, we obtain a path Q in $cY(\theta) \ominus h$ from n_a to n_b of length at most $t|n_a n_b|$. Consider the path $P = (a, n_a) \cup Q \cup (n_b, b)$ from a to b in $cY(\theta)$. Obviously, P is a path in $cY(\theta) \ominus h$ from a to b. We show that the length of P is at most $t|ab| = t$.

By Claim 2, we have $|n_a n_b| \leq \max\{|uc'|, |uw|\}$, and by Claim 1, we have $\max\{|uc'|, |vv'|\} \leq |uw|$, and therefore $|n_a n_b| \leq |uw|$. Since $|an_a|$ and $|bn_b|$ are both at most 1, by the combination of above inequalities, we have

$$|P| = 2 + |Q| \leq 2 + t|n_a n_b| \leq 2 + t|uw|.$$

We now prove that $2+t|uw| \leq t|ab|$. By Equation (3), the inequality $2 + t|uw| \leq t|ab|$, after simplifying, is equivalent to

$$\frac{4\cos^2(\theta)\, t^3 - (2 + 4\cos(\theta))\, t^2 + 2}{t^2 - 1} \geq 0$$

which is true, provided that $4\cos^2(\theta)\, t^3 - (2 + 4\cos(\theta))\, t^2 + 2 \geq 0$ and $t \geq \frac{1}{2\cos(\theta)+1-\sqrt{2\cos(\theta)+2}}$. The proves Theorem 2. □

Since we could not find an explicit formula for t, in Table 1, we presented the dilation t of $cY(\theta)$ for some angles θ with $\pi/3 \leq \theta < 2\pi/5$ in which $cY(\theta)$ is a region-fault tolerant t-spanner. In the table, the angles θ are expressed in degree instead of radian.

Table 1: The dilation t of $cY(\theta)$ for some angles θ with $\pi/3 \leq \theta < 2\pi/5$ in which $cY(\theta)$ is a region-fault tolerant t-spanner

θ	t
60°	3.8662
61°	4.061
62°	4.4522
63°	4.9334
64°	5.5357
65°	6.311
66°	7.3458
67°	8.7958
68°	10.9724
69°	14.6021
70°	21.8645
71°	43.6581

4 Conclusion

In this paper, we showed that for any $\pi/3 \leq \theta < 2\pi/5$, the continuous Yao graph $cY(\theta)$ is a region-fault tolerant t-spanner with a constant value t. We close the paper with the following open problem.

Is $cY(\theta)$ for any $\theta \geq 2\pi/5$, a $\mathcal{C}$-fault tolerant t-spanner for a constant value t?

References

[1] M. A. Abam, M. de Berg, M. Farshi, and J. Gudmundsson. Region-fault tolerant geometric spanners. *Discrete and Computational Geometry*, 41(4):556–582, 2009.

[2] D. Bakhshesh, L. Barba, P. Bose, J.-L. D. Carufel, M. Damian, R. Fagerberg, M. Farshi, A. van Renssen, P. Taslakian, and S. Verdonschot. Continuous Yao graphs. *Computational Geometry*, 67(Supplement C):42 – 52, 2018.

[3] L. Barba, P. Bose, J.-L. de Carufel, M. Damian, R. Fagerberg, A. van Renssen, P. Taslakian, and S. Verdonschot. Continuous Yao graphs. In *Proceedings of the 26th Canadian Conference on Computational Geometry*, CCCG'14, pages 100–106, August 2014.

[4] G. Narasimhan and M. Smid. *Geometric spanner networks*. Cambridge University Press, 2007.

A Theoretical Proof of Angular Random Walk

Sepideh Aghamolaei* Mohammad Ghodsi†

Abstract

Covering a polygonal room with a memoryless robot without distance measurement capability, i.e. a minimal sensing robot, is important in the field of robotics. A common algorithm for the aforementioned task is for the robot to start moving in the direction of one of its sensors randomly and to choose another direction when it hits a wall. Current algorithms lack theoretical models and time complexity analyses. In this paper, we model the environment from the robot's point of view with a directed graph and prove that this algorithm is equivalent to a random walk on this graph. Therefore, the angular random walk with 8 directions eventually covers an orthogonal polygon if the polygon can be covered using such moves.

1 Introduction

Coverage (path) planning [6, 2] are problems that optimize the coverage of the area of a polygon with holes, with a continuous movement of a shape. The objective functions are minimizing the number of passes over an area, the number of stops and rotations and etc. Most of these problems are NP-hard by reduction from covering salesman problem [1].

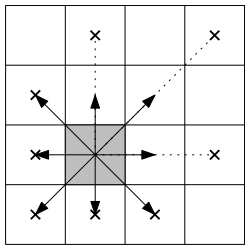


Figure 1: Possible moves of a robot with 8 sensors based on algorithm 1.

The **robot** is a disk with 8 sensors, each $\pi/4$ apart from the next sensor. The robot can only move in the direction of its sensors (see Figure 1). The robot has minimal sensing [7] capabilities, i.e. it can only sense non-geometric properties of the environment. In other words, only the closeness of points relative to each other can be measured. This is usually modeled using the concept of **gaps**. A gap is any discontinuity in the depth of the environment that can be detected by a simple robot. The notion of gaps is mostly used in gap navigation trees [8], but here we only keep the set of visible gaps. Also, assume the robot can rotate until one of its sensors faces the wall perfectly. Random walk with angle-changing after bumping into an object or wall has already been used in practice [9, 3]. Those algorithms choose an angle from $[0, 2\pi]$ randomly [3], which based on the rotation precision of their motor is in fact a set of discrete angles. Here we discuss 4 and 8 directions, aligned with the walls.

Random walks, unlike path planning methods and simultaneous localization and mapping (SLAM) approaches, do not require a map of the environment and perform simple operations. Therefore, performing random walks needs simpler hardware, and less memory and processing time per decision. One of the advantages of using simple robots is their lower cost and power usage.

We define two types of rooms (polygon) for our analysis:

- **Rectangular Room Environment:** Assume a room (B) is an axis-aligned $a \times b$ rectangle, divided into square grid cells of width d, where d is the diameter of the disk robot. The initial position of the center of the robot must be on the center of a cell. Assume it is possible to cover the rectangle with $d \times d$ tiles, i.e. d divides a and b.

- **Orthogonal Polygon Environment:** A polygon whose edges are vertical or horizontal. This polygon is divided into cells of width d, where d is the diameter of the disk robot. The initial position of the center of the robot must be on the center of a cell. Also, assume every opening between two rectangular rooms is marked, i.e. the robot knows when it leaves a room. Assume it is possible to cover the polygon with $d \times d$ tiles.

The contribution of this paper is providing a model that preserves the complications of a real environment, while providing theoretical guarantees that the algorithm will terminate and the amount of time it requires is linearly dependent on the area of the polygon.

*Department of Computer Engineering, Sharif University of Technology, aghamolaei@ce.sharif.ir

†Department of Computer Engineering, Sharif University of Technology,ghodsi@sharif.ir

2 Preliminaries

We review some definitions in this section.

Abstract Sensor: A combination of a set of physical sensors that provides a mathematically accurate measurement capability. For example, a gap sensor is an inaccurate distance measurement sensor.

Markov Model: (Also known as Markov Chain or Markov Process) is a directed graph which is specified along with a set of probabilities on its edges and a start vertex. A Markov process starts from the start vertex and chooses the next vertex with the probability specified on each edge. The hitting time of a Markov process is the number of steps (edge transitions) required to visit each vertex at least once, with high probability. The state of a Markov model is a vector containing the probability of being in each vertex. If a Markov model reaches a state and never leaves it, we say it has converged.

3 Angular Random Walk

In this section we introduce an algorithm for covering orthogonal polygons and provide a theoretical proof based on Markov chain.

3.1 Algorithm 1

We call the following algorithm **angular random walk**:

1. Follow a wall to find a corner.
2. Rotate until one of your sensors faces the wall
3. Repeat:
4. Choose one of the 8 directions or staying at the same cell uniformly at random and move in that direction,
5. Until you hit a wall

Algorithm 1 is similar to real-world implementations such as [4, 9], but the idea behind those algorithms comes from artificial intelligence algorithms.

Warm-up: Choosing The Number of Directions

A robot with orthogonal movements (4 directions) cannot cover a room with algorithm 1. Assume the robot starts from a corner of the room. Regardless of the direction it chooses next, the robot can only travel the boundary of the polygon, since it only changes direction in corners of the polygon.

The robot cannot measure distances, so it is impossible for this robot to stop anywhere between the two corners. Therefore, there is no algorithm that can solve this problem using this robot.

3.2 Covering The Area of A Rectangular Room

According to the definition of the robot and its movements, there are 8 possible directions for entering or exiting each cell (see Figure 1). The only limitation is

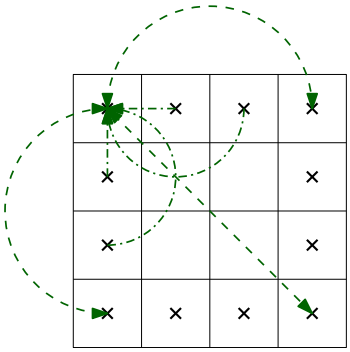


Figure 2: The movement graph for the topmost and leftmost cell.

that if a direction is chosen, the robot must follow it until reaching the boundary.

Construct the directed graph $G = (V, E)$ by assigning a vertex to each pair (c_i, d_j), where c_i is a boundary cell and d_j is a direction. Then add an edge from pair (c_i, d_i) to (c_j, d_j) iff $d_i = d_j$ and if is possible to go from c_i to c_j by moving only in direction d_i. The set of possible movements starting from a corner cell is shown in Figure 2.

Theorem 1 *Algorithm 1 is equivalent to a random walk on graph G.*

Proof. Every time the robot chooses a direction, since only vertices assigned to that direction are used and because the edges are directed, with probability 1 the next vertex is the next grid cell in that direction, if possible. If there are no more vertices in that direction, we have reached a boundary vertex, which is connected to all directions; So one of those directions can be chosen randomly. □

We use a Markov chain to model this random walk, and use its convergence to prove the termination of the algorithm.

Lemma 2 *The Markov model of graph G converges, if uniform probabilities are used in the algorithm and* $GCD(a, b) = 1$.

Proof. It is enough to prove that G is strongly connected and the greatest common divisor (GCD) of all cycles in G is 1. Since we can go from any cell in any direction to any other cell and any other direction, the graph is strongly connected. The robot starts at one corner and goes to another corner and then comes back.

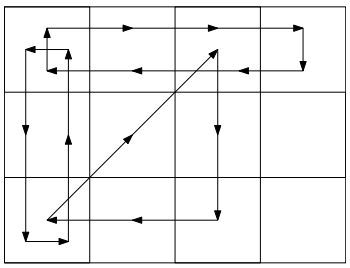


Figure 3: Three loops of the movement graph for a $a \times b$ rectangle of lengths $2a$, $2b$ and $3b$.

If the direction the robot chooses faces a wall, it is considered a self-loop in the graph. As shown in Figure 3, considering these three loop lengths gives, the GCD of loops is at most:

$$GCD(2a, 2b, 3b) = GCD(a, b)$$

Therefore, if $GCD(a, b) = 1$, then the GCD of all cycle lengths is 1. □

Theorem 3 *The time complexity of the algorithm is upper bounded by the cover time of the Markov model of Lemma 2 times the time required to traverse the diameter of a cell.*

Proof. Each cell is repeated 8 times, once for each direction. So, if the Markov model visits every vertex, it is equivalent to covering the whole room. Since each transition in this model is equivalent to entering a new cell in the algorithm, this is also the time complexity of the algorithm. □

3.3 Covering An Orthogonal Polygon

We assume the polygon is already partitioned into rectangular regions, called rooms. Assume we have an upper bound M on the time required to cover a room. After spending M units of time in a room, the robot decides which room it should go to next.

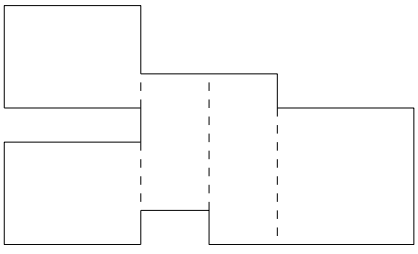


Figure 4: An orthogonal polygon. The dashed lines show possible divisions into rooms.

Build a graph G of the Markov chain with a vertex for each room and connect two vertices if their rooms (rectangles) are adjacent. Each time the robot sees a room, if it is passed time M , it chooses to stay in the room (equivalent to traversing a self-loop in the Markov chain) with probability q or leave it with probability p. Since the sum of probabilities of each vertex in the Markov chain should be 1, if a polygon has f rooms, the probability is $fp + q = 1$, so $p = \frac{1-q}{f}$. If the actual number of rooms is less, then $q > 1 - fp$. Note that staying in the room just means choosing another direction randomly.

This Markov chain converges, since

- G is connected, because the rooms are connected, and
- it has self-loops (as a result of staying in the same room), so the GCD of loops is 1, it eventually converges.

To implement this, assume the number of rooms is upper-bounded ($f \leq 20$). Let $p = \frac{1}{f} = \frac{1}{20}$, and assume there is at least one room connected to less than f rooms, (so there is always a self-loop in G).

Since f is the number of rooms (vertices) of the random walk on G and the degree of each vertex is at most f, then the hitting time is $2|V|.|E|$ [5], which here is $2f^2$.

Algorithm 2: Covering An Orthogonal Polygon

1. timer$\leftarrow$ 0.
2. Repeat $2f^2$ times:
3. Run algorithm 1.
4. If you see another room and timer $> M$, then
5. Stay with probability p or
6. leave with probability q and update timer$\leftarrow$ 0.

3.4 Initialization

In order to run the aforementioned algorithms, we need to find upper bounds. Assume there is a fixed point on the boundary. The robot can use $O(1)$ memory to store the values of f and M.

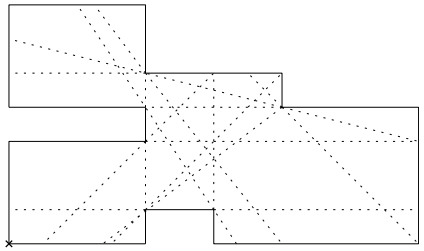


Figure 5: The visibility regions of Figure 4.

3.4.1 Algorithm 3

We use the number of regions in the visibility regions as an upper bound for the number of rooms. Since rooms are convex, each of them is at most one visible region, so the number returned by the algorithm is at least as much as the number of rooms.

A **timer** is required to measure M. We compute the time required to cover the boundary and use it to upper bound the area.

Algorithm for Counting the Rooms

Input: an orthogonal polygon, a fixed point on the boundary

Output: a number f, a time upperbound M

1. $f = 0$.
2. Start timer ($t \leftarrow 0$).
3. Traverse the boundary from the fixed point in a ccw order.
4. If the visible region changes, $f \leftarrow f + 1$.
5. Stop timer and store its value t.
6. $M \leftarrow \frac{t^2}{4}$.

Return f, M.

3.5 Turn Complexity

The number of stops and turns in an algorithm might make it slower in practice, compared to straight line movements. This depends on the speed of the robot; for slow robots the time complexity works better because turning and moving take almost the same amount of time, but for fast robots, the turn complexity is a better measure since changing the moving direction is more time consuming.

Modify the Markov model of Section 3.2 to include only boundary cells, by shortcutting other vertices. Since the robot only turns when it hits a wall, this model includes all the turns the algorithm makes.

Corollary 4 *The cover time discussed in Lemma 2 is the turn complexity of the algorithm.*

4 Conclusion

We used the memoryless property of Markov chains to prove an algorithm for covering an orthogonal polygon using a robot with $O(1)$ memory. Our analysis shows the performance of this algorithm depends on the convergence rate of the Markov chain, and therefore, the dimensions of the room(s). But finding formulas, or proving the possibility of covering a room with such a robot remains open, for example, the room in Figure 2 cannot be covered using a minimal sensing robot with 8 sensors. Note that the robot cannot stop at a non-boundary cell because it requires exact distance measurement.

Related open problems include providing models for more complex polygons and polygons with holes. Using Markov chain to analyze other robot's state diagram and creating randomized memoryless algorithms is also important.

References

[1] E. M. Arkin and R. Hassin. Approximation algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics*, 55(3):197–218, 1994.

[2] H. Choset. Coverage for robotics–a survey of recent results. *Annals of mathematics and artificial intelligence*, 31(1):113–126, 2001.

[3] K. M. Hasan, K. J. Reza, et al. Path planning algorithm development for autonomous vacuum cleaner robots. In *Informatics, Electronics & Vision (ICIEV), 2014 International Conference on*, pages 1–6. IEEE, 2014.

[4] iRobot Corporation. irobot roomba 400 series manual. `http://www.irobotweb.com/~/media/Files/Support/Home/Roomba/400/iRobot-Roomba-400-Manual.pdf?sc_lang=en`. Accessed: 31-August-2016.

[5] H. J. Karloff, R. Paturi, and J. Simon. Universal traversal sequences of length no (log n) for cliques. *Information Processing Letters*, 28(5):241–243, 1988.

[6] S. M. LaValle. *Planning algorithms*. Cambridge university press, 2006.

[7] S. Suri, E. Vicari, and P. Widmayer. Simple robots with minimal sensing: From local visibility to global geometry. *The International Journal of Robotics Research*, 27(9):1055–1067, 2008.

[8] B. Tovar, L. Guilamo, and S. M. LaValle. Gap navigation trees: Minimal representation for visibility-based tasks. In *Algorithmic Foundations of Robotics VI*, pages 425–440. Springer, 2004.

[9] Wikipedia. Roomba. `https://en.wikipedia.org/wiki/Roomba`. Accessed: 12-August-2016.

Answering Time-Windowed Queries of Contiguous Hotspots

Ali Gholami Rudi*

Abstract

A hotspot of a moving entity is a region in which it spends a significant amount of time. Given the location of a moving object through a certain time interval, i.e. its trajectory, our goal is to find its hotspots. We consider axis-parallel square hotspots of fixed side length, which contain the longest contiguous portion of the trajectory. Gudmundsson, van Kreveld, and Staals (2013) presented an algorithm to find a hotspot of a trajectory in $O(n \log n)$, in which n is the number of vertices of the trajectory. We present an algorithm for answering *time-windowed* hotspot queries, to find a hotspot in any given time interval. The algorithm has an approximation factor of 1/2 and answers each query with the time complexity $O(\log^2 n)$. The time complexity of the preprocessing step of the algorithm is $O(n)$. When the query contains the whole trajectory, it implies an $O(n)$ algorithm for finding approximate contiguous hotspots.

Keywords: Trajectory, Hotspot, Geometric algorithms, Time-windowed queries

1 Introduction

The identification of popular places or hotspots is an important preprocessing step for analyzing trajectories (Zheng mentions several interesting applications [11]) and the recent growth of trajectory data sets is increasing the emphasis on efficient trajectory analysis algorithms, hotspot identification algorithms not excepted.

Several heuristics have been proposed for identifying hotspots (e.g. [8, 9, 10]). To pave the way for devising accurate geometric algorithms for identifying hotspots, some authors have formalized the definition of hotspots and posed several questions about their identification [3, 7]. One of the problems introduced by Gudmundsson et al. is that of finding a hotspot, i.e. a placement of an axis-aligned square of fixed side length, which maximizes the time the entity spends inside it without leaving it [7]. In other words, the square should contain a contiguous sub-trajectory with the maximum duration. They also present an algorithm for this problem with the time complexity $O(n \log n)$ (this time complexity is best possible, as shown in the extended version of this paper[1]). Their algorithm relies on the fact that there exists a hotspot with at least one trajectory vertex on its boundary. It finds the hotspot by testing the squares, on one of whose boundaries lies a vertex of the sub-trajectory.

We study the problem of answering *time-windowed* hotspot queries. Each query specifies a time interval and the goal is finding a hotspot for the sub-trajectory of this interval (e.g. for finding the stay point of a bird in August or finding the hotspot of a mobile device per day, week, and month). Time-windowed geometric queries have recently attracted much attention. Bannister et al. introduce a framework for answering time-windowed queries and present algorithms for answering such queries for three problems including convex hull, for which they present an algorithm that answers each query in poly-logarithmic time and performs the preprocessing in $O(n \log n)$ time [1]. For the time-windowed closest pair of points problem, Chan and Pratt present a quadtree-based algorithm in the word-RAM model of computation [5]. For decision problems, Bokal et al. present time-windowed algorithms for deciding hereditary properties (i.e., if a sequence has the property, all of its subsequences do as well) [4]. Chan and Prat improve Bokal et al.'s results by reducing time-windowed decision problems to range successor problem and using dynamic data structures [6]. None of these results, however, can be used directly for answering hotspot queries. Since its goal is finding a hotspot, the problem cannot be stated as a decision problem and the techniques used for time-windowed problems like convex hull do not seem applicable to hotspot identification.

In this paper we present an approximation algorithm for answering time-windowed hotspot queries. It answers each query with the time complexity $O(\log^2 n)$ and with an approximation ratio of 1/2. The space complexity of the algorithm and the time complexity of its preprocessing step is $O(n)$. This also implies a 1/2-approximation algorithm for finding contiguous hotspots of whole trajectories in $O(n)$ time. The low complexity of this algorithm, its practicable data structures, and small approximation factor makes this algorithm suitable for large real world trajectory data sets.

*Department of Electrical and Computer Engineering, Bobol Noshirvani University of Technology, Babol, Iran. Email: gholamirudi@nit.ac.ir.

[1] https://arxiv.org/abs/1711.03795

The rest of this paper is organized as follows. In Section 2 we describe the notation used in this paper and in Section 3 we present our algorithm. Finally, in Section 4 we conclude this paper and mention possible directions for further studies.

2 Preliminaries and Notation

Let T be a polygonal trajectory, which describes the location of a moving entity through a specific time interval. We use $T(t)$ to denote the entity's location at time t in trajectory T. In polygonal trajectories, the location of the entity is recorded as different points in time; these we call the vertices of the trajectory. These vertices are linearly interpolated to decide the location of the entity between two contiguous vertices. The sub-trajectory that connects any two contiguous vertices of the trajectory are its edges.

For each vertex v of trajectory T, let $\text{loc}(v)$ denote its location and $\text{tstamp}(v)$ denote its time-stamp. With slight abuse of notation, we use vertices and time-stamps interchangeably. Thus, $u < v$ for vertices u and v means $\text{tstamp}(u) < \text{tstamp}(v)$. For two vertices or time-stamps u and v of trajectory T, T_{uv} denotes the sub-trajectory from u to v.

For a square r of fixed side length s, the weight of r with respect to trajectory T is the maximum duration of a contiguous sub-trajectory of T contained in r. A hotspot of trajectory T is an axis-parallel square of fixed side length s with the maximum weight. Every square discussed in this paper has fixed side length s and is axis-parallel. We may not mention these constraints explicitly hereafter.

3 An Algorithm for Answering Time-Windowed Queries

To improve the readability, we first present the algorithm with the assumption that queries start and end with the time-stamp of a trajectory vertex and prove its approximation factor. We then discuss how to construct the data structures required in the algorithm. Finally, we handle general queries that may start or end at a time different from the time-stamps of all trajectory vertices and show that this preserves the approximation factor.

3.1 The Main Algorithm

For any trajectory vertex v, we define $\text{hotend}_-(v)$ to denote the start of a sub-trajectory of T ending at v with the maximum duration that can be contained in a square, $\text{hotsquare}_-(v)$ to denote one such square, and $\text{hotdur}_-(v)$ to denote its duration. Similarly, $\text{hotend}_+(v)$ denotes the end of a sub-trajectory of T starting at v with the maximum duration that can be contained in a square, $\text{hotsquare}_+(v)$ denotes one such square, and $\text{hotdur}_+(v)$ denotes its duration. The implementation of these functions is described in Section 3.2.

We also define $\text{hotdur}_-(u, v)$ as the maximum value of $\text{hotdur}_-(w)$ for all vertices like w in T_{uv}, $\text{hotsquare}_-(u, v)$ as its corresponding square, and $\text{hotend}_-(u, v)$ as its corresponding starting vertex, in which u and v are two trajectory vertices. We define $\text{hotdur}_+(u, v)$, $\text{hotsquare}_+(u, v)$, and $\text{hotend}_+(u, v)$ similarly. The implementation of these functions are also explained in Section 3.2.

We now describe the algorithm for answering query (u, v), to find an approximate hotspot of sub-trajectory T_{uv}. As mentioned before, here we assume that both u and v are trajectory vertices.

1. If uv is an edge of the trajectory, it is trivial to find its hotspot by considering the largest portion of the edge uv that can fit in a square.

2. Let w be a trajectory vertex between u and v such that the number of vertices in sub-trajectories T_{uw} and T_{wv} differ by at most one. A binary search between the vertices of T_{uv} can find w. Note that the duration of sub-trajectories T_{uw} and T_{wv} may differ greatly.

3. Let u' be $\text{hotend}_-(w, v)$ and let v' be $\text{hotend}_+(u, w)$. There are three cases to consider.

   (a) If $u' < u$ and $v < v'$, $\text{hotsquare}_-(w, v)$ contains T_{uw} and $\text{hotsquare}_+(u, w)$ contains T_{wv}; let r be $\text{hotsquare}_-(w, v)$ if the duration of T_{uw} is greater than that of T_{wv} and $\text{hotsquare}_+(u, w)$, otherwise. Given that the duration of T_{uv} is the sum of the durations of T_{uw} and T_{wv}, the weight of r is at least half of the weight of the hotspot of T_{uv}.

   (b) Now suppose $u < u'$ and $v' < v$. Any hotspot of T_{uv} either starts at a vertex of T_{uw} or ends at a vertex of T_{wv}. Consider the first case: if a hotspot of T_{uv} starts at a vertex of T_{uw}, its weight should be equal to $\text{hotdur}_+(u, w)$. For the second case, we can similarly argue that the weight of a hotspot that ends at a vertex of T_{wv} is $\text{hotdur}_-(w, v)$. Therefore, we can return either $\text{hotsquare}_+(u, w)$ or $\text{hotsquare}_-(w, v)$ based on the relative values of $\text{hotdur}_+(u, w)$ and $\text{hotdur}_-(w, v)$.

   (c) The remaining case is when $u' < u$ and $v' < v$ (the case when $u < u'$ and $v' < v$ is similar and is omitted for brevity). Again, any hotspot of T_{uv} either starts at a vertex of T_{uw} or ends at a vertex of T_{wv}. If a hotspot of T_{uv} starts at a

vertex of T_{uw}, its weight equals $\text{hotdur}_+(u, w)$. Otherwise, the hotspot should start and end at two vertices of T_{wv}. We perform this algorithm recursively for the interval (w, v) to find the approximate hotspot r for this interval. We can return either $\text{hotend}_+(u, w)$ or r based on their weight.

The time complexity and approximation factor of this algorithm is shown in Theorem 1.

Theorem 1 *Given a trajectory T and after some preprocessing for computing the functions* hotdur*,* hotsquare*, and* hotend*, for each query (u, v) we can find a square, whose weight is at least half of the weight of the hotspot of T_{uv}, with the time complexity $O(\log^2 n)$.*

Proof. The correctness of the algorithm is explained in its description. For the approximation ratio, consider the tree formed by the recursive invocations of the algorithm for a query. The only step of the algorithm that returns an approximate result is step 3.a, which appears only once and as a leaf in this tree. Therefore the weight of the square returned by the algorithm is at least half of the weight of a hotspot.

For the time complexity, note that the algorithm is invoked recursively only once in step 3.c and for a query containing half as many points as the original query. In each invocation, hotend and hotdur functions, the time complexity of both of which is $O(1)$, are called a constant number of times. Therefore, the time complexity of answering a query containing m vertices is $T(m) \leq T(m/2) + O(\log m)$ (the $O(\log m)$ term is for finding w in step 2). Solving this recurrence yields $T(n) = O(\log^2 n)$ as required. □

3.2 Preprocessing

We next show how to implement functions hotend, hotdur, and hotsquare for single vertices. The multi-vertex version of these functions can be implemented using Range Minimum Query (RMQ) data structures. RMQ data structures support finding the minimum (or the maximum) of any contiguous interval of a sequence. Using Cartesian trees, RMQ can be implemented with linear space and $O(1)$ query complexity (for details, consult [2]).

In the following algorithm, we use MinQueue data structure, supporting the following operations: Insert for inserting an item, Remove for removing the oldest item, and Min for finding the item with the minimum value in the queue (MaxQueue data structure is similar with a Max operation instead). There exists a clever implementation of MinQueue (and MaxQueue) using two stacks, in which the time complexity of all three operations is $O(1)$.

We now describe how to compute $\text{hotdur}_-(v)$. $\text{hotend}_-(v)$ and $\text{hotsquare}_-(v)$ can be computed in parallel but are omitted for brevity. Also note that $\text{hotdur}_+(v)$, $\text{hotend}_+(v)$, and $\text{hotsquare}_+(v)$ can be implemented similarly by negating the time-stamps of the vertices. Suppose *minx* and *miny* are instances of MinQueue and *maxx* and *maxy* are instances MaxQueue and are initially empty. The following steps are performed for every vertex of T ordered by their time-stamps.

1. Define (x, y) as $\text{loc}(v)$. Insert v with the value x into *minx* and *maxx*. Insert v with the value y into *miny* and *maxy*.
2. Repeatedly remove the oldest items from each of the four queues, until both $\text{Max}(maxx) - \text{Min}(minx)$ and $\text{Max}(maxy) - \text{Min}(miny)$ are at most s. Defining u as the oldest item in any of the queues, the sub-trajectory T_{uv} is the longest that ends at vertex v, starts at a vertex of T, and can be contained in a square.
3. Let u' be the vertex before u in T (the last vertex removed from the queues). Based on the condition in the previous step, $T_{u'v}$ cannot be contained in a square but T_{uv} can be. To find $\text{hotend}_-(v)$, we need to find p, the earliest point on the edge $u'u$ such that T_{pv} can be contained in a square. To do so, consider the four ways of aligning a corner of a square with a corresponding corner of the bounding box of T_{uv}, and choose the one that contains the longest portion of $u'u$.

Theorem 2 *The time and space complexity of the preprocessing step for functions* hotdur*,* hotsquare*, and* hotend *is $O(n)$.*

Proof. All steps of the preprocessing perform $O(1)$ computation for each vertex except step 2, which may extract many items from the queues. However, since only n items are inserted into each queue and each item can be removed at most once, $O(n)$ items are removed from the queues during the whole algorithm. Therefore, the time complexity of the algorithm is $O(n)$. Since the Cartesian tree-based RMQ implementation of the multi-vertex version of the functions has linear space and time complexity, the time and space complexity of the preprocessing is thus likewise linear. □

3.3 Handling General Queries

Theorem 3 shows how to handle queries whose start or end does not coincide with the time-stamp of a trajectory vertex.

Theorem 3 *After $O(n)$ preprocessing for a trajectory T with n vertices, time-windowed hotspot queries can be answered approximately with the time complexity*

$O(\log^2 n)$. *Each query is specified as two time-stamps x and y ($x < y$), which may not coincide with the time-stamp of any trajectory vertex. The algorithm returns a square whose weight is at least half of the weight of the hotspots of T_{xy}.*

Proof. For the query (x, y), let u be the first vertex on the trajectory at or after x and v be the first vertex at or before y (they can be found using binary search on the vertices of T in $O(\log n)$).

If the query is totally contained in a trajectory edge, a hotspot can be found trivially by finding a square that contains the longest portion of the edge. Otherwise, suppose square r is a hotspot of T_{xy} and let x' and y' denote the start and end of a sub-trajectory of T_{xy} contained in r with the maximum duration. There are two cases to consider. If $u \leq x'$ and $y' \leq v$, based on Theorem 1 we can find an approximate hotspot with an approximation factor of $1/2$. Otherwise, suppose $x' < u$ (handling the case $v < y'$ is similar and is omitted). Suppose $T_{x'y'}$ contains u (otherwise we can move r towards u without changing its weight, since the r is on a single edge). Clearly, the weight of r equals the sum of the durations of $T_{x'u}$ and $T_{uy'}$; the former is at most $\text{hotdur}_-(u)$ and the latter is at most $\text{hotdur}_+(u)$. Therefore, the weight of either $\text{hotsquare}_-(u)$ or $\text{hotsquare}_+(u)$ in respect to T_{xy} is at least half of the weight of r. □

4 Concluding Remarks

The algorithm presented in this paper is very fast, even for finding an approximate hotspot of the whole trajectory. Querying the whole trajectory after preprocessing yields Corollary 4.

Corollary 4 *An approximate contiguous hotspot of a trajectory can be found with the time complexity $O(n)$ and an approximation ratio of $1/2$.*

Several related problems seem interesting for further investigation. It may be possible to include the side length of the hotspot s in the query by returning a sequence from the functions introduced in Section 3.1; an algorithm to answer these extended queries would be very interesting. The approximation ratio may be improved; this looks very important, especially from a practical point of view, for querying large trajectory data sets. Also, it seem interesting to answer time-windowed queries of non-contiguous hotspots (see [7] for details).

Acknowledgements

The author wishes to thank Dal for discussing this problem in Challenging Thursdays 28.

References

[1] 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 *The Canadian Conference on Computational Geometry*, pages 11–19, 2014.

[2] M. A. Bender and M. Farach-Colton. The LCA problem revisited. In *LATIN*, pages 88–94, 2000.

[3] M. Benkert, B. Djordjevic, J. Gudmundsson, and T. Wolle. Finding popular places. *International Journal of Computational Geometry and Applications*, 20(1):19–42, 2010.

[4] D. Bokal, S. Cabello, and D. Eppstein. Finding all maximal subsequences with hereditary properties. In *Symposium on Computational Geometry*, pages 240–254, 2015.

[5] T. M. Chan and S. Pratt. Time-windowed closest pair. In *The Canadian Conference on Computational Geometry*, pages 141–144, 2015.

[6] T. M. Chan and S. Pratt. Two approaches to building time-windowed geometric data structures. In *Symposium on Computational Geometry*, pages 28:1–28:15, 2016.

[7] J. Gudmundsson, M. J. van Kreveld, and F. Staals. Algorithms for hotspot computation on trajectory data. In *SIGSPATIAL/GIS*, pages 134–143, 2013.

[8] Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W.-Y. Ma. Mining user similarity based on location history. In *GIS*, page 34, 2008.

[9] J. Yuan, Y. Zheng, L. Zhang, X. Xie, and G. Sun. Where to find my next passenger. In *UbiComp*, pages 109–118, 2011.

[10] N. J. Yuan, Y. Zheng, X. Xie, Y. Wang, K. Zheng, and H. Xiong. Discovering urban functional zones using latent activity trajectories. *IEEE Transactions on Knowledge and Data Engineering*, 27(3):712–725, 2015.

[11] Y. Zheng. Trajectory data mining - an overview. *ACM Transactions on Intelligent Systems and Technology*, 6(3):29:1–29:41, 2015.

On the Generalized Minimum Spanning Tree in the Euclidean Plane

Homa Ataei Kachooei* Mansoor Davoodi† Dena Tayebi ‡

Abstract

We aim at finding a minimum spanning tree consisting of exactly one point per cluster, for a set of n points in the plane partitioned into $k < n$ clusters. We show that this problem is NP-complete even if every cluster contains two points with equal y coordinates. Further, we show that this problem does not have an FPTAS unless $P = NP$.

Keywords: Minimum Spanning Tree, NP-completeness, Approximation Algorithm

1 Introduction

In this paper, we study Generalized Minimum Spanning Tree (GMST) problem in the plane. First we review the GMST problem on graphs. Given a connected undirected graph $G = (V, E)$ in which nodes are partitioned into k clusters:

$$V = V_1 \cup V_2 \cup .... \cup V_k \qquad \forall i \neq j,\ V_i \cap V_j = \emptyset, \quad (1)$$

where V_i is a subset of nodes. The GMST problem is to find a Minimum Spanning Tree (MST) which consists of exactly one point from each cluster.

The GMST problem on graphs was proved NP-hard by a reduction from the Vertex Cover problem [9]. It was also proved that the GMST problem on graphs cannot be approximated within any constant factor [10]. Furthermore when $G = (V, E)$ is a tree, the problem is NP-hard [10]. However, there is an approximation algorithm for the GMST problem when the cluster size is bounded by a constant ρ. In this case, the GMST problem can be approximated to within 2ρ [11].

A geometric version of the GMST problem was studied with grid clustering. In this version, the graph was considered complete and the nodes are placed in a $(r \times l)$-grid and weight of each edge is the distance between two nodes. The nodes belonging to a cell of the grid make a cluster. A PTAS for the GMST problem with grid clustering was proposed in [3]. Moreover, a $(1 + 4\sqrt{2} + \epsilon)$-approximation algorithm was presented for the GMST problem with the grid clustering [1]. The existence of a PTAS shows that the GMST problem with grid clustering is easier than the GMST problem. An alternative version of the GMST problem focuses on finding the MST which consists of at least one point per cluster [7]. The NP-completeness of this version was shown in [7] including the case where each cluster contains three points. Additionally, it was shown that this version cannot be approximated within any constant factor [12]. Also it was shown that this version of the GMST problem with grid clustering is strongly NP-hard even if non-empty grid cells are connected and each grid cell (cluster) contains at most two points [5]. Further, a $(r \times l)$-grid $(r \leq l)$ was used in [5] and a dynamic programming algorithm was presented which solves this version in $O(l\rho^{6r}2^{34r^2}r^2)$ time. Note that if r or l are bounded, this algorithm is polynomial. Feremans et al. [5] using the dynamic programming algorithm, presented a PTAS when all non-empty grid cells are connected and the number of non-empty grid cells is superlinear in r and l.

The Class Steiner Tree (CST) problem (known also as Group Steiner Tree (GST) problem) is similar to the GMST problem. A short review of the CST problem is provided in the following.

Given a connected undirected graph $G = (V, E)$ in which the nodes are partitioned into disjoint sets, such that:

$$V = S \cup R_1 \cup ... \cup R_k, \quad (2)$$

where R_i is a required class for $i = 1, 2, ..., k$ and S is Steiner class, the CST problem tries to find an MST which include at least one node per required class. The CST problem was proved to be NP-hard even if there is no Steiner node, the weight of all edges are unit and the nodes degree are less than or equal to three [6]. Finally, the CST problem cannot be approximated within any constant factor even for trees without Steiner node and unit edges [6].

As pointed out before, the focus of this paper is on studying the GMST problem in the plane (not graph). Hence, for a given set P containing n points in the plane which is partitioned into k clusters:

$$P = P_1 \cup P_2 \cup .... \cup P_k \qquad \forall i \neq j,\ P_i \cap P_j = \emptyset, \quad (3)$$

the GMST problem in the plane tries to find an MST which consists of exactly one point from each cluster.

We prove that the GMST problem in the plane is NP-complete even if each cluster contains two points with equal y coordinates. We further prove that this problem

*ataei.homa@gmail.com

†Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan, Iran, mdmonfared@iasbs.ac.ir

‡denatayebi@yahoo.com

does not have an FPTAS unless $P = NP$. This version is a simple case of the GMST problem.

2 NP-completeness of the GMST Problem in the Euclidean Plane

By a reduction from the planar 3SAT problem we prove that the GMST problem in the Euclidean plane is NP-complete.

Consider the 3SAT problem where $C = \{c_1, c_2, ..., c_m\}$ is the set of clauses and $V = \{v_1, v_2, ..., v_n\}$ is the set of variables. Create a graph $G = (U, E)$ for every instance of the 3SAT problem such that:

$$U = C \cup V, \tag{4}$$

and

$$E = E_1 \cup E_2, \tag{5}$$

where E_1 and E_2 are:

$$E_1 = \{(c_i, v_j) \mid v_j \in c_i \ \ or \ \ \overline{v}_j \in c_i\} \tag{6}$$

and

$$E_2 = \{(v_j, v_{j+1}) \mid 1 \leq j < n\} \cup \{(v_n, v_1)\}. \tag{7}$$

The set of all edges in E_2 is called *spinal path* [4]. There is a node for each variable and for each clause in the graph G, resulting in $|U| = |C| + |V|$. Draw one edge between a variable node and a clause node in G, if and only if the clause contains a literal of the variable. The planar 3SAT problem includes all instances of the 3SAT problem with similar planar graphs. The planar 3SAT is proved to be NP-complete by a reduction from the 3SAT problem [8].

Theorem 1 *The GMST problem in the plane is NP-complete. This claim is true even under the constraint that every cluster contains two points with equal y coordinates. Also the GMST problem does not have an FPTAS unless $P = NP$.*

Proof. We utilize a similar approach to that of Theorem 7 in [4] for demonstrating the correctness of the theorem. As such, We perform the proof by using a reduction from the planar 3SAT problem for the GMST problem.

Every instance of the planar 3SAT problem should be converted to an instance of the GMST problem in the plane. First, we design two gadgets for the variables and clauses called *variable gadget* and *clause gadget*, respectively. The design of these gadgets is based on some clusters of points where every cluster consists of a pair of points. The designed gadgets in the graph of ϕ are replaced as follows: if a node in the graph of ϕ is corresponding to a variable in ϕ, it is replaced by a variable gadget and if it is corresponding to a clause in ϕ, it is replaced by a clause gadget. Consequently, we replace all the nodes in the graph of ϕ with the gadgets. One should ensure that the number of clusters (pairs of pionts) in this reduction is polynomially bounded in the size of ϕ. Therefor, we use a special drawing graph called *orthogonally drawing* [2]. In the orthogonally drawing each node is shown with a rectangle and each edge is denoted by a sequence of vertical and horizontal segments. The orthogonally drawing provide a practical mean to draw the graph in the defined space. To continue, we need the Theorem 2 from [2]. The theorem is provided below.

Theorem 2 *[12, Theorem 4] Let H be a simple graph without nodes of degree ≤ 1, where n is the number of nodes and m is the number of edges. Then H has an orthogonally drawing in an $(\frac{m+n}{2} \times \frac{m+n}{2})$-grid with one bend per edge. The box size of each node v is at most $\frac{deg(v)}{2} \times \frac{deg(v)}{2}$. It can be found in $O(m)$ time.*

At this stage, we want to convert the planar 3SAT instance to a GMST instance. Therefor we first draw the orthogonally drawing and then, replace the variable gadgets with the variables and the clause gadgets with the clauses.

2.1 Variable Gadget

For each variable in the planar 3SAT instance, we design a gadget which consists of k cluster, such that k is an even number and $4 \leq k \leq 2c + 4$, where c is the number of clauses that include this variable. Consider two variables i and j such that $1 \leq i \leq k$. Variable i is the number of the clusters and j is an index of i which is equivalent to the number of points in the cluster. Because there are only two points in every cluster, we set $j = 0$ or $j = 1$. The $j = 0$ and $j = 1$ cases correspond to the first and second points in the cluster, respectively. If the coordinates of the first point in the first cluster is denoted by (x, y), then for every i and j where $1 \leq i \leq k - 1$ and $j = 0, 1$, coordinates of point i_j are

$$(x + 2(i - 1) + j, y + ((i + 1) \ mod \ 2)).$$

This way, $k-1$ clusters are being embedded in the plane and the coordinates of the points in the cluster k are $(x-\sqrt{5}, y)$ and $(x+2(k-1)-1+\sqrt{5}, y)$. Consequently, the placement of these points in the plane leads to a structure which is a part of the variable gadget. We denote this structure by A.

Structure A with $k = 8$ is depicted in Figure 1. Here, the location of the points in the plane is not important but the distance between them is an issue.

The proof of the Theorem 1 and the design for the variable gadget are not completed yet, we need Lemma 3 in that regard.

Figure 1: Structure A. The segment between two points shows that they are in a cluster. Also the right side and the left side points are in a cluster.

Lemma 3 *In the structure A with k clusters, there are two different choices from clusters which lead to the optimal solution of the GMST problem. In one of these solutions the right side points choose in all clusters and in other solution the left side points choose in all clusters. The weight of the Euclidean MST (EMST) for these solutions is* $\sqrt{5}(k-1)$*. Further, the weight of EMST in every other selection of points except the aforementioned two, is at leas 0.1 more than the weight of an optimal solution.*

Proof. See Appendix at the end of the article. □

With regard to this Lemma, there are two possible solutions for the GMST problem for the structure A. We assume where the right side points are selected from all clusters to be equivalent to the case for which the variable is True. Moreover we assume where the left side points are selected from all clusters to be equivalent to the case for which the variable is False.

These assumptions along with Lemma 3, provide the necessary requirements to complete the design of the variable gadgets. Suppose that the number of clauses containing this variable is equal to c. For each of these clauses, we posit a fixed point at a distance of 2 units to one of the right side points in the direction of y. Similarly, for each clause containing a variable negation, a fixed point is placed at a distance of 2 units of one of the left side points in the direction of y.

Because of these clauses are located above or below this variable, we locate mentioned points above or below the gadget. Additionally, we locate two points for spinal path connections such that they don't have any effect on choosing the right or the left side points. Therefore, these points should be selected in a way that they have the same distances from the nearest right and left side points in the structure A. So, the location of these points are selected as $(x+0.5, y-1)$ and $(x+2(k-2)+0.5, y-1)$. Indeed, these are the endpoints of the edges that establish the spinal path connections.

Figure 2 is an example of a variable gadget. This Figure illustrates the variable such as z attending in three clauses which $\overline{z}$ comes in one clause and z in the other two.

2.2 Clause Gadget

Consider a sequence of points which are located in unit distances along a line. A clause gadget is formed from three of the mentioned sequences that collide at a point

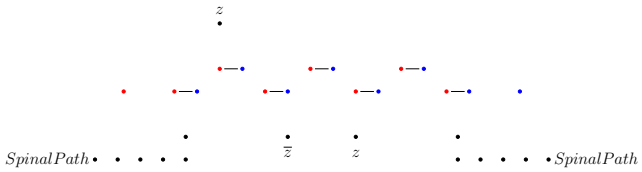


Figure 2: An example of the variable gadget

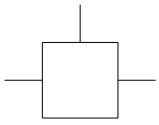
[4]. Figure 3a is related to a node which is corresponding to a clause in the orthogonally drawing. If we replace this node with a clause gadget Figure 3b is obtained.

2.3 Reduction

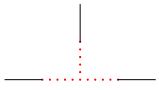
We scale up orthogonally drawing with factor of 2 and replace the graph nodes of the orthogonally drawing with the introduced gadgets. This is also the case for the graph edges which are exchanged with a sequence of points located in a unit distances along the edges.

As pointed out before, in the orthogonally drawing, each node is a box with the maximum size of $\frac{deg(v)}{2} \times \frac{deg(v)}{2}$. Designed gadget may not fit into the intended box in the orthogonally drawing, but size of this gadget is at most $(4(deg(v)-1)+2\sqrt{5}+1)\times 5$ which is bounded in the size of the box.

So far we converted each planar 3SAT instance to a GMST instance. The next step at this point is to show that every solution of the GMST problem is a solution of the planar 3SAT problem. We used the orthogonally drawing which is drawn in a $(\frac{m+n}{2} \times \frac{m+n}{2})$-grid, and the drawing is polynomially bounded in the size of the planar 3SAT instance. Hence the number of used fixed points in this reduction is polynomially bounded in the size of the planar 3SAT instance. These fixed points have a unique MST with constant weight. Therefor,the MST obtained from the gadgets themselves and their connections determine the weight of the MST. Sum of the MST weight of the connection between the spinal path and the gadgets, and the MST weight of the fixed points is denoted by W_{edges}. Also the MST weight of the



(a) A clause node



(b) A clause gadget

Figure 3: Replacing a clause node in the orthogonally drawing with a clause gadget

gadgets and gadgets connections to edges is denoted by $W_{clusters}$. So the total weight of MST is sum of W_{edges} and $W_{clusters}$:

$$W_{total} = W_{edges} + W_{clusters}. \quad (8)$$

In the optimal solution of MST, the connectivity should be reached by minimum cost. W_{edges} has a constant value, no matter which points are selected. The connections between spinal paths and variable gadgets have constant weight and these connections cause all variable gadgets to be connected to each other. Now we investigate connection of the edges to the variable gadget. In the optimal solution of the GMST problem each clause gadgets should be connected to a variable gadget and the cost of each connection is 2 units. So the total cost of these connections is twice the number of all clauses. If only the right or the left side points are selected in each gadget, the weight of the obtained MST can be calculated as :

$$W_{clusters} = (\sqrt{5}(R-n)) + 2c + 2n\sqrt{4+(0.5)^2}, \quad (9)$$

where n is the number of variables, c is the number of clauses and R is the number of all clusters. In this case, each gadget is just connected to a clause containing the variable or its negation. This means there is a True assignment for the planar 3SAT problem. If $W_{clusters}$ is more than this value, there is at least one variable gadget which is connected to a clauses containing the variable and a clause containing variable negation. This means there is no True assignment for the planar 3SAT problem.

Now we show that the GMST problem does not have an FPTAS unless $P \neq NP$. Consider the existence of an FPTAS for the GMST problem. Given a planar 3SAT instance, we build the GMST problem input as explained previously and calculate W_{total}. We determine the ϵ value such that $\epsilon < \frac{0.1}{W_{total}}$. So a $(1+\epsilon)$-approximation solution for the GMST problem can be used to verify whether there is a True assignment for the planar 3SAT problem or not. Since the planar 3SAT is NP-Hard, we can conclude there is no FPTAS for the GMST problem unless $P \neq NP$. □

3 Discussion

The maximization version of this problem has not been studied yet. In this version, given a set P consisting points in the plane which is partitioned into k clusters.

$$P = P_1 \cup P_2 \cup .... \cup P_k \qquad \forall i \neq j,\ P_i \cap P_j = \emptyset. \quad (10)$$

The GMST problem in the plane is to find maximum MST which consists of exactly one point from each cluster. Complexity of this problem and whether it has approximation algorithm with constant factor, would be studied in future.

References

[1] B. Bhattacharya, A. Ćustić, A. Rafiey, A. Rafiey, and V. Sokol. Approximation algorithms for generalized mst and tsp in grid clusters. In *Combinatorial Optimization and Applications*, pages 110–125. Springer, 2015.

[2] T. C. Biedl and M. Kaufmann. Area-efficient static and incremental graph drawings. In *European Symposium on Algorithms*, pages 37–52. Springer, 1997.

[3] F. Corinne, G. Alexander, et al. An approximation scheme for the generalized geometric minimum spanning tree problem with grid clustering. Technical report, 2004.

[4] R. Dorrigiv, R. Fraser, M. He, S. Kamali, A. Kawamura, A. López-Ortiz, and D. Seco. On minimum-and maximum-weight minimum spanning trees with neighborhoods. *Theory of Computing Systems*, 56(1):220–250, 2015.

[5] C. Feremans, A. Grigoriev, and R. Sitters. The geometric generalized minimum spanning tree problem with grid clustering. *4OR: A Quarterly Journal of Operations Research*, 4(4):319–329, 2006.

[6] E. Ihler. Minimum rectilinear steiner trees for intervals on two parallel lines. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 123–134. Springer, 1992.

[7] E. Ihler, G. Reich, and P. Widmayer. On shortest networks for classes of points in the plane. In *Computational Geometry-Methods, Algorithms and Applications*, pages 103–111. Springer, 1991.

[8] D. Lichtenstein. Planar formulae and their uses. *SIAM journal on computing*, 11(2):329–343, 1982.

[9] Y.-S. Myung, C.-H. Lee, and D.-W. Tcha. On the generalized minimum spanning tree problem. *Networks*, 26(4):231–241, 1995.

[10] P. C. Pop. *The generalized minimum spanning tree problem*. Twente University Press, 2002.

[11] P. C. Pop, W. Kern, and G. Still. An approximation algorithm for the generalized minimum spanning tree problem with bounded cluster size. 2001.

[12] S. Safra and O. Schwartz. On the complexity of approximating tsp with neighborhoods and related problems. *computational complexity*, 14(4):281–307, 2006.

Appendix

We prove lemma 3 in this part.

Lemma 3. *Proof.* In the structure A with k clusters (pairs of points), there are two different choices from every cluster which lead to optimal solution of the GMST problem.

Proof 2.1.in the structure A, every cluster regardless of choice of points will be connected to the next cluster in solution of EMST. Also cluster k will be connected to the cluster 1 or $k-1$. We consider L and R symbols, which are equivalent to selection of the left side point and the right side point in a cluster, respectively. Any selection of points in the clusters are shown with sequence of these symbols. As an example, LRR sequence shows that the left side point is chosen in the first cluster and the right side point is chosen in the second and third clusters. Now, we show the optimal solutions are $LLL....L$ sequence or $RRR....R$ sequence. In these cases because in EMST every cluster connects to the next cluster, weight of optimal solution of EMST is $\sqrt{5}(k-1)$. We prove this claim for $LLL....L$ sequence, and for $RRR....R$ sequence will be proved similarly. In order to prove this claim, we assume the right side point is chosen in one of the clusters. It means we have $LL...LRL...L$. Consequently, in the cluster in which the right side point is chosen and in the next and the former cluster, EMST is as Figure 4a or Figure 4b and in both cases weight of EMST is $\sqrt{10}+\sqrt{2}$. Therefore weight of EMST in the entire structure is $((k-3)\sqrt{5}+\sqrt{10}+\sqrt{2})$ and it is $(\sqrt{10}+\sqrt{2}-2\sqrt{5}) \approx 0.1$ more than weight of EMST when the left side points are chosen in all clusters. When the right side points are chosen in more than one consecutive cluster it means we have $LL...LRR...RL...L$, yet the weight of EMST is $((k-3)\sqrt{5}+\sqrt{10}+\sqrt{2})$ which is $(\sqrt{10}+\sqrt{2}-2\sqrt{5}) \approx 0.1$ more than Weight of EMST when the left side points are chosen in all clusters. Consequently every time when the consecutive left side points are chosen, at least 0.1 is added to the weight of EMST.

As an example in $LL...LR...RL...LR...RL...LL$ sequence, the weight of EMST is 0.2 more than when all of the right side points are chosen in all clusters. Optimal solutions of structure A with $k = 8$ is shown in Figure 5. □

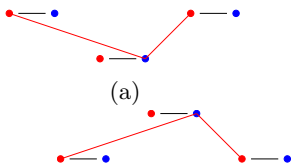


Figure 4: Part of EMST in $LL...LRL...L$ sequence

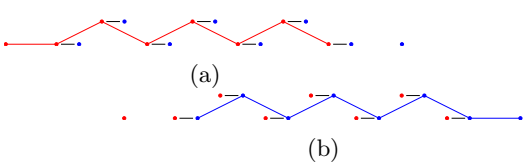


Figure 5: Optimal solutions of structure A with $k = 8$.

Routing in Well-Separated Pair Decomposition Spanners

Fatemeh Baharifard* Majid Farhadi† Hamid Zarrabi-Zadeh†

Abstract

In this paper, we present a local routing scheme for the well-separated pair decomposition (WSPD) spanners. Given a point set P in the plane, a WSPD spanner is a geometric graph whose vertex set is P, and for each pair (A, B) in the well-separated pair decomposition of P, an edge is added to the graph from an arbitrary point $a \in A$ to an arbitrary point $b \in B$. It is well-known that such a graph is a $(1+\varepsilon)$-spanner of P, where $\varepsilon > 0$ is an input parameter used for constructing the well-separated pair decomposition. Our routing scheme assigns to each point $p \in P$ a routing table of size $O(\frac{1}{\varepsilon^2} \log \alpha)$, where α is the ratio of the furthest distance to the closest distance in P. It can then locally route a message from any arbitrary point p to any point q in P along a path whose length is at most $1+\varepsilon$ times the Euclidean distance between the pair of points. The WSPD construction considered in this paper is based on compressed quadtrees. To the best of our knowledge, this is the first time that a local routing scheme with an optimal competitive routing ratio is considered for this famous class of WSPD spanners.

1 Introduction

A geometric graph G is a *t-spanner* for a point set P, if for each pair of points p and q in P, there is a path in G between p and q, whose length is at most t times the Euclidean distance between p and q. The minimum t such that G is a t-spanner of P is called the *spanning ratio* of G.

One of the most important problems in communication networks is to send/route a message from a source point to any other target point in such a way that the total distance traveled by the message is at most a constant times the shortest path or Euclidean distance between the two points. Network routing strategies such as Dijkstra's algorithm [10] require knowledge of the whole network topology in order to compute a short route. In many settings, this assumption is impractical, and the routing algorithm is supposed to work without knowing the full structure of the graph. Therefore, a *local routing* strategy is usually preferred, meaning that the algorithm can route the message to the target using only information stored in the message itself and in the current node [15]. If the information stored in the current node is of size k, we say that the local routing algorithm has a *routing table* of size k. Moreover, a local routing algorithm $\mathcal{A}$ is called *μ-memory* if it uses a memory of size μ stored with the message [4]. The algorithm $\mathcal{A}$ is *c-competitive* if the total distance traveled by the message is not more than c times the Euclidean distance between source and destination. The minimum c such that a routing algorithm $\mathcal{A}$ is c-competitive is called the *routing ratio*.

Related Work. Recently, a stream of research has explored local routing algorithms for some geometric spanners such as Delaunay triangulations and θ-graphs (for definitions, see [5, 9]). Chew [9] was the first to describe a local routing algorithm on the L_1-Delaunay triangulation with a routing ratio of $\sqrt{10}$, using only the information of the target point, the current point, and all neighborhood of the current point. Subsequently, local routing algorithms using the same set of information were presented for TD-Delaunay triangulation by a spanning ratio of $5/\sqrt{3}$ [3], and for the standard Delaunay triangulation by a spanning ratio of 5.90 [1]. In [5], a *θ-routing* algorithm is described which has a constant routing ratio on all θ_k-graphs with $k \geq 7$. Moreover, a deterministic local routing scheme with a routing ratio of 2 is presented for θ_6-graph in [3].

Very recently, Bose *et al.* [2] considered a specific type of WSPD spanners, and presented two near-optimal local routing schemes for this type of spanners. In their settings, the WSPD construction is based on fair split trees, and the WSPD spanner is constructed by selecting a well-chosen edge from each partition of WSPD (the rightmost point in each set) as its representative, rather than picking an arbitrary edge. They showed that their WSPD spanner has an improved spanning ratio of $1+4/s+4/(s-2)$ compared to the original one, which was $1+8/(s-4)$, where $s > 0$ is the separation factor. They presented a 2-local and a 1-local routing algorithm with routing ratios of $1+4/s+6/(s-2)$ and $1+6/(s-2)+6/s+4/(s^2-2s)+8/s^2$, respectively (a routing algorithm on a graph G is called *k-local*, if each vertex v of G stores information about vertices that are at hop distance at most k from v). Their routing scheme did not use a header but required routing tables of total

*School of Computer Science, Institute for Research in Fundamental Sciences (IPM), f.baharifard@ipm.ir

†Department of Computer Engineering, Sharif University of Technology, m_farhadi@ce.sharif.edu, zarrabi@sharif.edu

size $O(s^2nB)$ bits, where B is the maximum number of bits to store a bounding box.

Competitive local routing algorithms with additional memory have been recently considered for unit disk graphs as popular wireless ad-hoc networks [15, 17]. The unit disk graph connects any two nodes which are within unit distance to each other. Yan *et al.* [17] presented a routing algorithm with low hop (edge) delay, by assigning a label of size $O(\log^2 n)$ to each node, where n is the number of nodes. Subsequently, Kaplan *et al.* [15] discovered a $(1+\varepsilon)$-competitive routing algorithm for unit disk graphs, using a modifiable header (memory) of size $O(\log n \log \Delta)$, where Δ is the diameter of the points, as well as additional polylog bits for each point. Their method is based on the well-separated pair decomposition for unit disk graphs [12].

Our Contribution. In this paper, we focus on an important and well-known class of WSPD spanners whose underlying WSPD is constructed using compressed quadtree. This construction of WSPD is widely used in the literature [8, 11, 14, 16], as it avoids the complexity of fair split trees originally used by Callahan and Kosaraju [7]. We present a competitive $O(\log \alpha)$-memory routing algorithm to route on these WSPD spanners, where α is the ratio of the farthest distance to the closest distance in the input point set. We indeed consider a standard WSPD spanner which is constructed by choosing an arbitrary edge from each pair of the WSPD, and unlike the method used in [2], we do not pose any restriction on choosing the representatives of the pairs when constructing the WSPD spanner. Assuming that we can store a static information (routing table) of size $O(\frac{1}{\varepsilon^2}\log \alpha)$ at each node of the spanner, the proposed algorithm is a $(1+\varepsilon)$-competitive local routing algorithm, which is optimal.

2 Preliminaries

In this section, we briefly describe the notions used throughout the paper.

Well-Separated Pair Decomposition. Let P be a set of n points in the plane, and $s > 0$ be a real number. Two point sets $A, B \subseteq P$ are *well-separated* with respect to a separation factor s, if there are two disjoint disks D_A and D_B with the same radius r, enclosing A and B respectively, such that the distance between D_A and D_B is at least $s \cdot r$. Here, the distance of two subsets A and B is defined as $d(A, B) = \min\{\|a - b\| \mid a \in A, b \in B\}$ where $\|a - b\|$ denotes the Euclidean distance of the points a and b (see Figure 1).

Following the definition in [7], a *well-separated pair decomposition (WSPD)* for P with respect to s is a collection $\mathcal{W} = \{(A_1, B_1), \ldots, (A_m, B_m)\}$ of pairs of non-

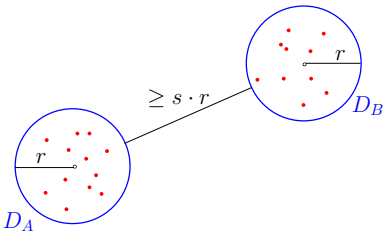


Figure 1: A well-separated pair with separation s

empty subsets of P such that each pair (A_i, B_i) for $1 \leq i \leq m$ is a well-separated pair with respect to s, and for any pair of points $p, q \in P$, there is a unique pair (A_i, B_i) in the collection, such that either $p \in A_i$ and $q \in B_i$, or $q \in A_i$ and $p \in B_i$. The number of well-separated pairs, m, is called the size of the WSPD.

WSPD Construction. A *quadtree* of P is a tree data structure $\mathcal{T}$ in which each internal node has four children, and the points of P are stored in the leaves. The root of $\mathcal{T}$ corresponds to a square bounding box of P, and each internal node $v \in \mathcal{T}$ corresponds to a cell $c(v)$ which is a square formed by splitting the parent cell into four equal-size squares by a horizontal and a vertical cut. A *compressed quadtree* is a quadtree in which any sequence of nodes with degree one are replaced by a single node. A compressed quadtree of a set of n points can be constructed in $O(n \log n)$ time [13].

Given a compressed quadtree $\mathcal{T}$ of P, one can use the following greedy algorithm to build a WSPD of P. The algorithm starts by considering any combination of two children of the root as a pair. If the current pair is not well separated, then the bigger node of the pair is replaced by its children, and the process continues until we reach a well-separated pair decomposition. For a separation factor $s > 0$, this algorithm yields a WSPD of size $O(s^2 n)$ in $O(n \log n + s^2 n)$ time [13].

WSPD Spanners. Callahan and Kosaraju [6] showed how a $(1+\varepsilon)$-spanner can be obtained from a WSPD. They first constructed a WSPD of P with separation factor $s = 4 + 8/\varepsilon$. They then chose an arbitrary point $a_i \in A_i$ and an arbitrary point $b_i \in B_i$ as the *representatives* of A_i and B_i, respectively, and showed that the resulting graph $G = (P, E)$ with $E = \{(a_i, b_i) \mid 1 \leq i \leq m\}$ is a $(1+\varepsilon)$-spanner. We refer to the resulting graph G as a *WSPD spanner* of P throughout the paper. Based on the construction described above, the WSPD spanner has size $O(n/\varepsilon^2)$ and can be computed in $O(n \log n + n/\varepsilon^2)$ time.

3 Routing in WSPD Spanners

Let P be a set of n points in the plane, and let α be the *spread* of P, namely the ratio of the farthest distance to the closest distance in P. In this section, we propose an algorithm to route a message through the WSPD spanner of P, utilizing a small additional memory (stack) along with the message, and a static data (routing tables) in the nodes of the graph.

We first prove an upper bound on the number of WSPD pairs that contain a fixed point. The following packing lemma is an ingredient of our proof.

Lemma 1 (Packing Lemma [7]) *Let D be a disk of radius r in the plane. The number of disjoint quadtree cells of side length at least ℓ overlapping D is at most $(1 + \lceil 2r/\ell \rceil)^2 = O(\max\{2, r/\ell\}^2)$.*

Lemma 2 *For each point $p \in P$, the number of WSPD pairs containing p is upper-bounded by $O(\frac{1}{\varepsilon^2} \log \alpha)$.*

Proof. Let $\mathcal{W}$ be a WSPD of P with separation s as described in Section 2. Let (A, B) be a well-separated pair in $\mathcal{W}$ containing p, and let x and y be the smallest quadtree cells of same length ℓ, enclosing A and B, respectively. Suppose that x and y are in level i of the quadtree. By the construction of WSPD, we know that if $(A, B) \in \mathcal{W}$, then $(P(A), P(B))$ is not in $\mathcal{W}$, where $P(A)$ and $P(B)$ denote the parents of A and B in quadtree, respectively. Then, $d(A, B) \leq (s+2)\sqrt{2}\ell$, because otherwise, the distance between parent cells of $c(x)$ and $c(y)$ with side length at least 2ℓ is more than $s\sqrt{2}\ell$, and hence they are well-separated, which is a contradiction (see Figure 2). Therefore, by packing lemma, at most $O(((s+2)\sqrt{2})^2) = O(s^2)$ pairs in level i can contain p. Since $s = 4 + 8/\varepsilon$, and there are at most $\log \alpha$ levels in the quadtree, the total number of pairs containing p is $O(\frac{1}{\varepsilon^2} \log \alpha)$. □

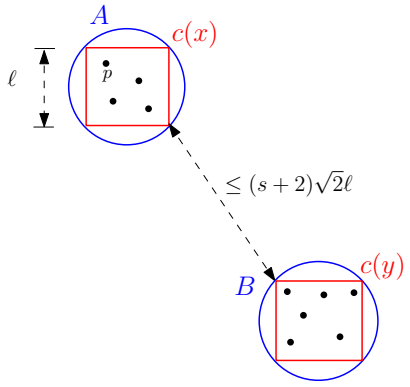


Figure 2: A well-separated pair with bounded distance.

Routing Algorithm. We are now ready to describe our routing algorithm. Let $f_p(q)$ denote a function that searches for a pair (A_i, B_i) in the WSPD such that $(p, q) \in A_i \times B_i$ or $(p, q) \in B_i \times A_i$, and returns their corresponding representatives (a_i, b_i) or (b_i, a_i) in the WSPD spanner. This function must be computable at node p. Therefore, at each node p, we store a list (table) of pairs (A_i, B_i) such that p is a member of either A_i or B_i, and for each such pair (A_i, B_i), we store in the table the boundaries of A_i and B_i (to check membership of an arbitrary point in the set), as well as the representatives of A_i and B_i in the WSPD spanner. Note that Lemma 2 bounds the size of the table stored at each node to $O(\frac{1}{\varepsilon^2} \log \alpha)$. The function is now simply computable at p by trying all pairs including p and checking membership of q in the other side of the pair, using boundaries of the squares corresponding to the sets.

Routing can be performed by simulating the following recursive algorithm, using a stack stored and transmitted along with the message. The inputs are source and destination points, (p, q), and we are at p at the beginning of the algorithm (see Figure 3).

Algorithm 1 ROUTE (p, q)

$(a, b) \leftarrow f_p(q)$
ROUTE(p, a)
traverse along edge (a, b)
ROUTE(b, q)

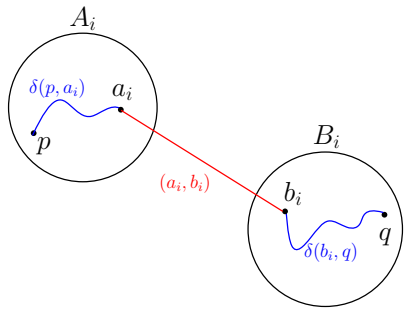


Figure 3: Routing a message from p to q.

Lemma 3 *For any pair of points $p, q \in P$, the path traversed by Algorithm 1 from p to q is at most $1 + \varepsilon$ times the Euclidean distance between p and q.*

Proof. We prove by induction on the Euclidean distance of the points. Fix a pair $p, q \in P$. Suppose by induction that for any pair $x, y \in P$ with $d(x, y) \leq d(p, q)$, the traversed path $\delta(x, y)$ has length at most $(1 + \varepsilon)d(x, y)$, where $d(x, y)$ denotes the Euclidean distance between x and y. By construction of the WSPD spanner, there is a pair (A_i, B_i) such that $p \in A_i$ and $q \in B_i$. Therefore, we have:

$$\begin{aligned}\delta(p,q) &\le \delta(p,a_i) + d(a_i,b_i) + \delta(b_i,q)\\ &\le (1+\varepsilon)d(p,a_i) + [d(p,q)+4r] + (1+\varepsilon)d(b_i,q)\\ &\le (1+\varepsilon)4r + [d(p,q)+4r]\\ &\le d(p,q) + (1/s)(8+4\varepsilon)\\ &\le (1+\varepsilon)d(p,q)\end{aligned}$$

□

Lemma 4 *The maximum depth of recursion in Algorithm 1, and thus the maximum size of the stack sent along with the message, is* $O(\log \alpha)$.

Proof. This is easy to see by noting that elements stored in the stack, corresponding to the recursion history for the current call, are monotonically deepening in the quadtree. □

Putting all these together, we get our main theorem.

Theorem 5 *Let* P *be a set of* n *points in the plane with spread* α*, and let* S *be a WSPD spanner of* P *with spanning ratio* $1+\varepsilon$*. We can locally route a message between any two nodes of* S *using a memory of size* $O(\log \alpha)$ *stored with the message, and a routing table of size* $O(\frac{1}{\varepsilon^2}\log \alpha)$ *stored at each node, such that the path traversed between the two nodes has length at most* $1+\varepsilon$ *times their Euclidean distance.*

4 Conclusion

In this paper, we considered the WSPD spanners based on compressed quadtrees, and proposed an efficient local routing algorithm on these spanners, using a memory of size $O(\log \alpha)$ stored with the message, and a routing table of size $O(\frac{1}{\varepsilon^2}\log \alpha)$ stored in the nodes of the spanner, where α is the spread of the underlying points. The path traveled between any two points by the algorithm is guaranteed to be no longer than $1+\varepsilon$ times the Euclidean distance between the two points. Although we presented our routing algorithm in the plane, the algorithm can be easily extended to any fixed dimension d, at the expense of increasing the routing table size to $O(\frac{1}{\varepsilon^d}\log \alpha)$. It is interesting to see if the size of routing table and/or memory can be improved.

References

[1] N. Bonichon, P. Bose, J. D. Carufel, L. Perković, and A. V. Renssen. Upper and lower bounds for competitive online routing on Delaunay triangulations. *arXiv:1501.01783*, 2015.

[2] P. Bose, J.-L. De Carufel, V. Dujmović, and F. Paradis. Local routing in spanners based on WSPDs. In *Proc. 15th Workshop Algorithms Data Struct.*, pages 205–216, 2017.

[3] P. Bose, R. Fagerberg, A. van Renssen, and S. Verdonschot. Competitive routing in the half-θ_6-graphs. In *Proc. 23rd ACM-SIAM Sympos. Discrete Algorithms*, pages 1319–1328, 2012.

[4] P. Bose, R. Fagerberg, A. van Renssen, and S. Verdonschot. Competitive local routing with constraints. In *Proc. 26th Annu. Internat. Sympos. Algorithms Comput.*, pages 23–34, 2015.

[5] P. Bose, A. van Renssen, and S. Verdonschot. On the spanning ratio of Theta-graphs. In *Proc. 13th Workshop Algorithms Data Struct.*, pages 182–194, 2013.

[6] P. B. Callahan and S. R. Kosaraju. Faster algorithms for some geometric graph problems in higher dimensions. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 291–300, 1993.

[7] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42(1):67–90, 1995.

[8] T. M. Chan. Well-separated pair decomposition in linear time? *Inform. Process. Lett.*, 107(5):138–141, 2008.

[9] L. P. Chew. There is a planar graph almost as good as the complete graph. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, pages 169–177, 1986.

[10] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[11] J. Fischer and S. Har-Peled. Dynamic well-separated pair decomposition made easy. In *Proc. 17th Canad. Conf. Computat. Geom.*, volume 5, pages 235–238, 2005.

[12] J. Gao and L. Zhang. Well-separated pair decomposition for the unit-disk graph metric and its applications. *SIAM J. Comput.*, 35(1):151–169, 2005.

[13] S. Har-Peled. *Geometric approximation algorithms*. American Mathematical Society, 2011.

[14] S. Har-Peled and M. Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006.

[15] H. Kaplan, W. Mulzer, L. Roditty, and P. Seiferth. Routing in unit disk graphs. In *Proc. 12th Latin American Symp. Theoret. Informatics*, pages 536–548, 2016.

[16] E. Park and D. M. Mount. Output-sensitive well-separated pair decompositions for dynamic point sets. In *Proc. 21st ACM SIGSPATIAL Internat. Conf. Adv. Geographic Info. Syst.*, pages 354–363, 2013.

[17] C. Yan, Y. Xiang, and F. F. Dragan. Compact and low delay routing labeling scheme for unit disk graphs. *Comput. Geom.*, 45(7):305–325, 2012.

Progressive Algorithm For Euclidean Minimum Spanning Tree

Amir Mesrikhani * Mohammad Farshi * Mansoor Davoodi †

Abstract

Designing efficient algorithms that process massive data is a challenging task. The progressive algorithms are methods to handle massive data efficiently. In these algorithms, partial solutions are reported to user in some middle steps that approximates the final solution. The user can decide whether to continue the running of the algorithm based on the error of the partial solutions. In this paper, we propose a progressive algorithm for computing Euclidean minimum spanning tree of a set of n points in the plane that consists of $O(\log n)$ steps. The error of the partial solution in step r is $O(1-\frac{4^r}{n-1}\alpha^{-1})$, where α is the aspect ratio of the point set.

1 Introduction

One method to process massive data efficiently is designing algorithms that solve a problem with a massive input data interactively with users. Progressive algorithms are one of these interactive methods. In these algorithms, partial solutions, whose error are measured by an error function (err), are reported to user in particular steps. The error function err takes a partial solution as an argument and returns non-negative value that represents the error value of the argument. Based on the error value of the partial solution in each step, the user can decide to stop the algorithm or continue toward a partial solution with smaller error value. The convergence speed of the partial solutions to the final solution is determined by a convergence function f_{conv}. The function f_{conv} takes step number r as an argument and returns an upper bound of the error value of the partial solution in step r.

Formally, in 2015, Alewijnse et al. [1] introduced the following defintion for progressive algorithms:

Definition 1 *A progressive algorithm is an algorithm that produces partial solution s_r in step r such that:*

$$err(s_r) \leq f_{conv}(r).$$

*Combinatorial and Geometric Algorithms Lab., Department of Mathematical Sciences, Yazd University, Yazd, Iran, mesrikhani@stu.yazd.ac.ir, mfarshi@yazd.ac.ir (corresponding author)

†Department of Computer Science and Information Technology, Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan, Iran, mdmonfared@iasbs.ac.ir

Computing Euclidean Minimum Spanning Tree (EMST) of a set of points is a classical problem that has many applications in designing low-cost road network and designing VLSI circuit. In this paper, we aim to design a progressive algorithm to compute EMST of a set of points in the plane.

1.1 Our results

For a set of n points in the plane, we propose a progressive algorithm for computing EMST with $O(\log_4 n)$ steps and the most time-consuming step takes $O(n^2)$ time. In step r, our algorithm produces a partial solution whose error is $O(1-\frac{4^r}{n-1}\alpha^{-1})$. The value α denotes the apsect ratio of input point set which is the ratio of the maximum pairwise distance and the minimum pairwise distance of the input points.

1.2 Related work

The framework of the progressive algorithm was introduced by Alewijnse et al. in 2015 [1]. They studied some fundamental problems in computational geometry like finding convex hull of a set of points and computing k-popular regions for a set of trajectories. Also, progressive algorithms are studied in other contexts and the results could be found in [9, 10]. As aforementioned, a progressive algorithm produces approximation solution in each step . Several approximation algorithms have been proposed for computing Minimum Spanning Tree (MST) [8]. Clarkson et al. [6] studied finding EMST of a set of points in $\mathbb{R}^d$. They proposed $(1+\varepsilon)$-approximation algorithm that takes $O(n(\log n + (1/\varepsilon)\log\alpha))$ time for $d = 3$. Czmuaj et al. [7] focused on approximating the weight of EMST in sublinear time. Their algorithm approximates the weight in $\mathbb{R}^d$ in $O(\sqrt{n}poly(1/\varepsilon))$ time with high probability, where *poly* denotes a polynomial function based on $1/\varepsilon$. For a given connected graph of an average degree d, Chazelle et.al [5] presented probabilistic algorithm in $O(d\omega\varepsilon^{-2}\log\frac{d\omega}{\varepsilon})$ time to approximate the weight of MST with error at most ε, where ω is the maximum weight of edges of the input graph. In Hausdorff metric, Alvarez et al. [2] proposed an algorithm in $O(\tau 1/\varepsilon^d \log(1/\varepsilon^d))$ time to approximate the weight of MST with error at most ε, where τ denotes the time needed to compute MST of points in a fixed metric L_p. Some randomized algorithms for approximating MST

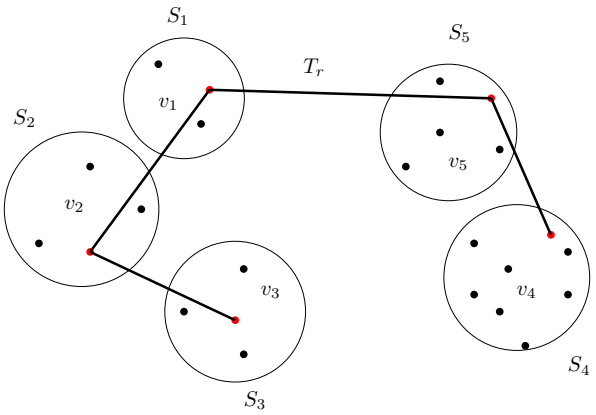


Figure 1: The red points are corresponding points of the super vertices. T_r is the partial solution in step r of the algorithm.

could be found in [4, 8]. Also, some results for approximating MST in the modern parallel models such as MapReduce could be found in [3].

2 Progressive algorithm for computing EMST

In this section, we aim to propose a progressive algorithm for computing EMST of a set P of n points in the plane. Before describing the algorithm, the partial solution and error function should be defined. Let $S = \{S_1, ...S_k\}$ be a partition of set P into the subsets $S_i \neq \emptyset$ such that $S_i \cap S_j = \emptyset$ for any $1 \leq i \neq j \leq k$. For each set $S_i \in S$, we assign a super vertex v_i. The super vertex v_i contains an arbitary point $p_i \in S_i$. Let E_r be a set of edges of a complete graph induced by $V = \{v_1, \ldots, v_k\}$. The weight of any edges $(v_i, v_j) \in E_r$ is defined by the Euclidean distance between p_i and p_j. Consider the graph $G_r = (V, E_r)$, the tree T_r in step r of the algorithm is an EMST of G_r and the weight of T_r denotes the partial solution (see Figure 1).

Let w_r be the weight of partial solution T_r in step r and w_{opt} be the weight of the exact EMST of P. The error function of our progressive algorithm is defined as follows:

$$err(T_r) = 1 - \frac{w_r}{w_{opt}}. \quad (1)$$

The Eq. (1) is a decreasing function. To prove this, we use the following lemma.

Lemma 1 *Let w_r be the weight of a partial solution T_r and w_{opt} be the weight of the exact EMST (T_{exact}) of P. We have $w_r \leq w_{opt}$.*

Proof. Let $V = \{v_1, \ldots, v_k\}$ and $E = \{e_1, \ldots, e_{k-1}\}$ be the set of vertices and edges of T_r respectively. Consider an edge $e_i = (v_t, v_l)$, and the corresponding sets S_t and S_l. We will show that p_t and p_l connect by a path in T_{exact} whose weight is greater or equal to e_i. Two cases

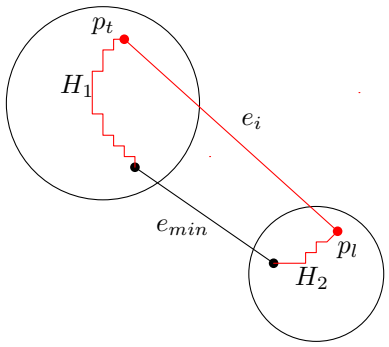


Figure 2: Illustration the first case of the proof of Lemma 1.

may be occured. First, the points of S_t and S_l connect by an edge e_{min} in T_{exact} which is the minimum length edge between the points in S_t and S_l. Let H_1 and H_2 be two paths that connect endpoints of e_{min} to p_t and p_l. By triangle inequality, the weight of e_i is less than or equal to the weight of the path $H_1 \bigcup H_2 \bigcup e_{min}$ (see Figure 2).

Second, there is a path H that connecting S_t and S_l through at least one subset S_k. In this case, by triangle inequality the weight of H is greater than e_i (see Figure 3). So we have $w_r < w_{opt}$. □

The idea of our progressive algorithm is the following: in any step r, we divide the points into 4^r disjoint sets and pick an arbitrary point from each set. Finally, EMST on the selected points is reported to the user as the partial solution. To divide the points into disjoint sets, we use $kd-tree$ approach to partition the plane by finding median vertical and horizontal lines with respect to the x and y coordinates.

So, in the first step of our progressive algorithm, we do as follows. Start by finding median points according to the x and y coordinates. Then draw vertical and horizontal lines passing through median points. These lines divide P into four subsets $S^1 = \{S_1, S_2, S_3, S_4\}$. For each $S_i \in S^1$ $i = 1, \ldots, 4$, assign a super vertex v_i that stores one point from S_i randomly. Construct complete graph induce by vertices v_1, v_2, v_3, v_4 and with edge weight correspond to the Euclidean distance between associate points of super vertices. Compute EMST of this graph (T_1) and report the weight of T_1 as the first partial solution.

In generic step r, we do as follows:

1. For each subset $S_i \in S^{r-1}$ do the following steps:
   (a) Divide S_i into four subsets with resepct to median lines according to x and y coordinates. Add these subsets to S^r.
   (b) Assign a super vertex for each new created subset and store a random point from corresponding subsets in it.

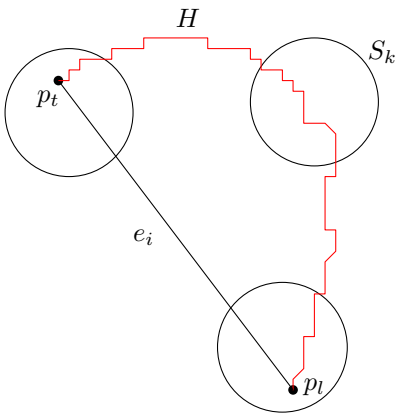


Figure 3: Illustration the second case of the proof of Lemma 1.

2. Let E_{new} be a set of edges that created by connecting each new vertex v to all vertices of T_{r-1}.

3. For each new super vertex v and corresponding edges E_{new}, T_r=*Update-EMST*(T_{r-1},v,E_{new}).

4. Report the weight of T_r to the user.

To describe how we can update T_{r-1} to obtain T_r in step 3, the following subproblem could be defined. Let T be an EMST of a graph G that has already been computed and v be a new vertex. Add v to T and connect it to all vertices of T and call the new graph by T' . The problem is designing an efficient algorithm to compute an EMST of T'. To this end, we use the following lemma..

Lemma 2 *Let e be an edge that has minimum weight among all new edges incident to a new vertex v. Then EMST of T' must contain e.*

Proof. Let T'_m denotes the EMST of T'. Assume to the contrary that T'_m does not contain e. So T'_m must contain an edge e' with the weight greater than e. By adding e to T'_m, a cycle that contains both e and e' will be obtained. By deleting e' from T'_m, we get a spanning tree where its weight is less than T'_m. This contradicts to our assumption that T'_m is an EMST. □

The algorithm uses the above lemma as the main strategy. Among all new edges, the algorithm finds the edge with minimum weight and adds it to T. According to above lemma, this edge belongs to EMST of T'. By this addition, T' will be a spanning tree but not necessarily with the minimum weight. Now, each new edge is added one by one to T. Suppose (v_i, v_j) is added to T and $w((v_i, v_j))$ denotes its weight. This graph is a spanning tree, so there is a unique path between v_i and v_j. In this path, the algorithm finds an edge e_{max} with maximum weight. If $w(e_{max}) > w((v_i, v_j))$ then e_{max} is deleted from T and we obtain a spanning tree with smaller weight. Otherwise, (v_i, v_j) is deleted from T' and the current spanning tree is preserved. This process is executed for all new edges. Finally, EMST of T' is computed. To obtain the unique path between v_i and v_j and update T, we can traverse T similar to depth-first traversal of a tree. The Algorithm *Update-EMST* is what we need to do in step 3 of our progressive algorithm.

Algorithm 1: *Update-EMST*(T,v,E)

Output: EMST of T
Find an edge (v, v_i) with minimum edges among E;
Add (v, v_i) to T;
$e_{max} = (v, v_i)$;
$DFS(v_i, e_{max}, T, E)$;
return T;

To implement the depth-first traversal of T, we use an array $Tmax[1, \ldots, n]$ such that $Tmax[i]$ denotes the edge with the maximum weight among the edges in the unique path between v and v_i. The vertex v is the first argument that $DFS(.)$ is invoked by it.

Algorithm 2: $DFS(v, e_{max}, T, E)$

```
Mark v as a visited vertex;
for each vertex v_i adjecent to v do
    if v_i is not marked as visited vetex then
        if w(v, v_i) > w(e_max) then
            Tmax[i] = (v, v_i);
        else
            Tmax[i] = e_max;
        if a new edge e_i exists in E then
            if w(e_i) < Tmax[i] then
                Add e_i to T;
                Delete Tmax[i] from T;
                Tmax[i] = e_i;
    Call DFS(v_i, Tmax[i], T, E);
```

In the following lemma, we analyze the convergence function of our progressive algorithm.

Lemma 3 *Let T_r be a partial solution in step r. Then $err(T_r) \in O(1 - \frac{4^r}{n-1}\alpha^{-1})$, where α is the aspect ratio of the input points.*

Proof. It is clear that, in step r of the algorithm, 4^r super vertices are created. So the partial solution T_r must contain $4^r - 1$ edges. Consider the error function defined in Eq. (1). Let C and D be the minimum pairwise and the maximum pairwise distance of points in the input point set P respectively. Trivially, $w_r \geq$

$(4^r-1)C$ and $w_{opt} \leq D(n-1)$. So, the upper bound of the error value of T_r is:

$$err(T_r) \leq 1 - \frac{(4^r-1)C}{D(n-1)} \leq 1 - \frac{4^r}{n-1}\alpha^{-1}.$$

□

Remark 1: Our progressive algorithm generates the *monotone* partial soltuions. It means that, if T_r and T_{r+1} be two partial solutions in two consecutive steps r and $r+1$, then $w_r \leq w_{r+1}$. Also,

$$f_{conv}(r) \leq f_{conv}(r+1).$$

This property obtain directly from Lemma 1 and Lemma 3.

Remark 2: The idea of our progressive algorithm is approximating the weight of EMST by choosing a subset of P. This idea is similar to a framework called *Coresets* introduced by Agarwal et al. [10]. One possible method is using this approach to pick a small subset of P in each step but Alvarez et al. [2] showed that this framework does not work when we want to approximate the weight of EMST.

Theorem 4 *There exists a progressive algorithm for computing EMST of a set of n points in the plane with $O(\log_4 n)$ steps. The algorithm takes $O(n^2)$ time in the most time-consuming step and the convergence function of the algorithm is $O(1-\frac{4^r}{n-1}\alpha^{-1})$ according to the error function defined in Eq. (1).*

Proof. The convergence function is obtained from Lemma 3. Let $S^r = \{S_1, \ldots, S_k\}$ be the decomposition of P in step r, where $k = 4^r$ and $|S_i|$ denotes the cardinality of S_i. By our approach for decomposition, each $S_i \in S^r$, is divided to four equal size subsets. So in step r, $|S_i| \leq \frac{n}{4^r}$. Therefore the number of steps is $O(\log n)$.

The lines 1 and 2 in step r of the progressive algorithm takes $\sum_{i=1}^{k} |S_i| = O(n)$ time, since $S_i \cap S_j = \emptyset$ holds for all $1 \leq i \neq j \leq k$.

For line 3, we know that T_{r-1} has $4^{r-1}-1$ edges. By adding a new vertex v to T_{r-1}, we have 4^{r-1} new edges. So, finding minimum edge takes $O(4^{r-1})$ time. To update T_{r-1} when v is added, we need to depth-first traversal of T_{r-1} that takes $O(4^{r-1})$ time. Therefore, updating T_{r-1} for each new added vertex takes $O(4^{r-1})$ time in total. We have $(4^r - 4^{r-1}) = 3 \times 4^{r-1}$ new vertices in step r. The total time of updating T_{r-1} is:

$$3 \times 4^{r-1} \times O(4^{r-1}) \in O(4^{2r-2}).$$

Since $r = O(\log_4 n)$, so step r never takes more than $O(n^2)$ time. □

3 Conclusion

In this paper, a progressive algorithm is proposed for computing the Euclidean minimum spanning tree of a set of n points in the plane with $O(\log_4 n)$ steps and the most time-consuming step takes $O(n^2)$ time. The upper bound on the error value of the generated partial solution in step r is $O(1-\frac{4^r}{n-1}\alpha^{-1})$, where α is the aspect ratio of points. One interesting future work is designing a progressive algorithm with convergence function that only depends on the size of input especially when α is unbounded. Developing the progressive algorithm for other problems with a massive input data are also interesting.

References

[1] S. P. A. Alewijnse, T. M. Bagautdinov, M. de Berg, Q. W. Bouts, A. P. ten Brink, K. Buchin, and M. A. Westenberg. Progressive geometric algorithms. *Journal of Computational Geometry*, 6(2):72–92, 2015.

[2] V. Alvarez and R. Seidel. Approximating the minimum weight spanning tree of a set of points in the hausdorff metric. *Computational Geometry*, 43(2):94–98, 2010.

[3] A. Andoni, A. Nikolov, K. Onak, and G. Yaroslavtsev. Parallel algorithms for geometric graph problems. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 574–583. ACM, 2014.

[4] P. Berenbrink, B. Krayenhoff, and F. Mallmann-Trenn. Estimating the number of connected components in sublinear time. *Information Processing Letters*, 114(11):639–642, 2014.

[5] B. Chazelle, R. Rubinfeld, and L. Trevisan. Approximating the minimum spanning tree weight in sublinear time. *SIAM Journal on computing*, 34(6):1370–1379, 2005.

[6] K. L. Clarkson. Fast expected-time and approximation algorithms for geometric minimum spanning trees. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 342–348. ACM, 1984.

[7] A. Czumaj, F. Ergün, L. Fortnow, A. Magen, I. Newman, R. Rubinfeld, and C. Sohler. Approximating the weight of the euclidean minimum spanning tree in sublinear time. *SIAM Journal on Computing*, 35(1):91–109, 2005.

[8] A. Gupta and J. Könemann. Approximation algorithms for network design: A survey. *Surveys in Operations Research and Management Science*, 16(1):3–20, 2011.

[9] R.-H. Li, L. Qin, J. X. Yu, and R. Mao. Efficient and progressive group steiner tree search. In *Proceedings of the 2016 International Conference on Management of Data*, pages 91–106. ACM, 2016.

[10] A. Mesrikhani and M. Farshi. Progressive sorting in the external memory model. In *48th Annual Iranian Mathematics Conference (AIMC48)*, August 2017.

A New Construction of the Greedy Spanner in Linear Space

Davood Bakhshesh* Mohammad Farshi†

Abstract

It is well known that the path-greedy spanner or greedy spanner for short is a high quality spanner in theory and practice. In this paper, we present a new construction of the greedy spanner in linear space. The our construction does not need to use the well-separated pair decomposition. The last construction of the greedy spanner in linear space due to Alewijnse et al. (*Algorithmica, volume 73, issue 3, pp 589–606, 2015*) uses the well-separated pair decomposition.

1 Introduction

Let S be a set points in $\mathbb{R}^d$ and $t > 1$ is a real number. A graph G with vertex set S is called *geometric*, if every edge $e = (p, q)$ in G is the straight line between p and q and the weight of e is the Euclidean distance between p and q, denoted by $|pq|$. We call a geometric graph G with vertex set S a *t-spanner* of S, if for each pair $p, q \in S$, there exists a path in G between p and q of length at most $t \times |pq|$. This path is called a *t-path* between p and q. The minimum value of $t > 1$ that a geometric graph $G = (S, E)$ is a t-spanner of S is called the *dilation* (or *stretch factor*) of G. Several studies have already been conducted on t-spanners. To study of an overview on the methods of constructing t-spanners for a given point set and their applications, see the book by Narasimhan and Smid [8].

One of the popular t-spanners is the *path-greedy spanner* or *greedy spanner* for short. To construct the greedy spanner of S, we start with a graph with vertex set S and empty edge set and consider all pairs of points in S in nondecreasing order of their distances. For a pair (p, q), it is tested whether there exists a t-path between p and q in the graph computed so far. If there is no such a t-path, the edge (p, q) is added to the graph (see Algorithm 1 that was presented in [8]). Using Dijkstra algorithm to find the shortest paths, a naive implementation of Algorithm 1 has the running time $O(n^3 \log n)$ and the space usage $O(n^2)$. For many years, computing the greedy spanner for a given point set in efficient time and space is a challenging problem. There are some researches on efficiently computing the greedy spanner [1, 2, 6]. In [6], Bose et al. presented an $O(n^2 \log n)$ time algorithm for computing the greedy spanner for metric spaces of bounded doubling dimension and therefore for Euclidean spaces. Their algorithm uses $O(n^2)$ space. It is notable that Farshi et al. [7] presented an algorithm, denoted by *FG-Greedy*, to compute the greedy spanner for a given point set in the plane that works well in practice and its running time tends to be near-quadratic, in practice. However, Bose et al. found a counterexample shows that the running time of FG-Greedy is $\Theta(n^3 \log n)$ [6].

Algorithm 1: PathGreedy(S, t)

input: a set S of n points in $\mathbb{R}^d$ and a real number $t > 1$.
output: t-spanner $G(S, E)$.

```
Sort (n choose 2) pairs of points in non-decreasing order of their distances (ties are broken arbitrarily), and store them in list L;
E := ∅;
G := (S, E);
foreach pair (u, v) ∈ L (in sorted order) do
    if ShortestPath(G, u, v) > t · |uv| then
        E := E ∪ {(u, v)};
    end
end
return G(S, E);
```

In [1], Alewijnse et al. presented an algorithm that computes the greedy spanner in $O(n^2 \log^2 n)$ using a linear space. They used *the well-separated pair decomposition* (*WSPD*) which is a well-known and a powerful data structure to solve the proximity problems, see [3, 4] for more details on the WSPD. In this paper, we present an algorithm similar to the algorithm in [1] that computes the greedy spanner in linear space without the need to the WSPD. Notably, implementing the WSPD for a given point set is not easy, see [8] to find an algorithm of computing the WSPD. It seems that the algorithm presented in the current paper is simpler than the way presented in [1] because it does not need to the WSPD.

*Department of Computer Science, University of Bojnord, Bojnord, Iran. d.bakhshesh@ub.ac.ir

†Combinatorial and Geometric Algorithms Lab., Department of Computer Science, Yazd University, Yazd, Iran. mfarshi@yazd.ac.ir

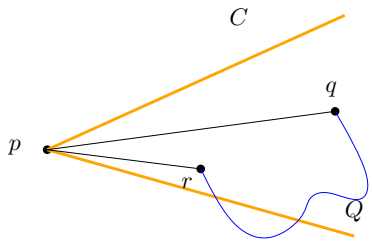


Figure 1: Illustrating the proof of Lemma 2

2 Fundamental Lemmas

Here, we provide some fundamental lemmas that is needed.

Lemma 1 ([5]) *Let a, b and c be three points such that $|ac| \leq |ab|$ and $\angle bac \leq \alpha < \pi$. Then $|bc| \leq |ab| - (1 - 2\sin(\alpha/2))|ac|$.*

Let $\theta > 0$ be a real number. Suppose that we partition the plain into $k = \lfloor \frac{2\pi}{\theta} \rfloor$ cones with apex in origin. We denote the set of resulting cones by $\mathcal{C}_k$. Suppose that S is a point set in the plain. For each $p \in S$ and cone $C \in \mathcal{C}_k$, we define C_p as $C_p := C + p$ (transformation of cone C to apex p).

Lemma 2 *Let $t > 1$ be a real number and let θ be a real number with $0 < \theta < \pi/3$ such that $1 - 2\sin(\theta/2) \geq 1/t$. Let $k = \lfloor \frac{2\pi}{\theta} \rfloor$ and let S be a set of points in the plane. For each $p \in S$ and $C \in \mathcal{C}_k$, the greedy spanner on S contains at most one edge $\{p, q\}$ with $q \in C_p$.*

Proof. By contradiction, for some point $p \in S$ and some cone $C \in \mathcal{C}_k$, suppose that the greedy spanner contains at least two edges $\{p, r\}$ and $\{p, q\}$ with $r, q \in C_p$ (see Figure 1). Suppose without loss of generality that $|pr| \leq |pq|$. By Lemma 1, we have $|rq| \leq |pq| - (1 - 2\sin(\theta/2))|pr| < |pq|$. So the greedy algorithm considers the pair $\{r, q\}$ before the pair $\{p, q\}$. Hence, there is a t-path between r and q that is denoted by Q. Now, consider the path $L := \{p, r\} \cup Q$ between p and q. So we have

$$\begin{aligned} |L| &= |pr| + |Q| \\ &\leq |pr| + t|rq| \\ &\leq |pr| + t(|pq| - (1 - 2\sin(\theta/2))|pr|) \\ &= t|pq| - (t(1 - 2\sin(\theta/2)) - 1)|pr| \leq t|pq|. \end{aligned}$$

So, the path L is a t-path between p and q. Hence, the greedy spanner does not add the edge $\{p, q\}$ which is a contradiction. This completes the proof. □

A straightforward conclusion of Lemma 2 is the following corollary.

Corollary 1 *Let $t > 1$ be a real number and let θ be a real number with $0 < \theta < \pi/3$ such that $1 - 2\sin(\theta/2) \geq 1/t$. The greedy t-spanner for a set of n points in the plane contains at most $O(\frac{1}{\theta}n)$ edges.*

Let $G = (S, E)$ be a geometric graph. For a cone $C \in \mathcal{C}_k$ and a point $p \in S$, let $x \in C_p$ be a point with the following properties:

1. there is no t-path between x and p,
2. for all points $y \in C_p$ with $|yp| < |xp|$, there is a t-path between y and p.

We denote x by Nearest(p, C_p). If there is no such x, we suppose that $x =$ **nil**.

Lemma 3 *Let $G = (S, E)$ be a geometric graph. For each cone $C \in \mathcal{C}_k$ and for each point $p \in S$, we can compute Nearest(p, C_p) in $O(|S| \log |S| + |E|)$ time and $O(|S|)$ space.*

Proof. We can use the Dijkstra algorithm to find the closest point to p in $C_p \cap S$ such that its dilation with respect to p is larger than t. This takes $O(|S| \log |S| + |E|)$ time and $O(|S|)$ space. Note that to check whether a point is in $C_p \cap S$ during the Dijkstra computation, we can preprocess an array A of size $|S|$ such that if a point r is in $C_p \cap S$ then $A[r]$ marked as true, otherwise marked as false. This takes $O(|S|)$ time and $O(|S|)$ space. It is noteworthy that we can reuse the space used for a cone for the next cone, too. □

3 First Algorithm

In this section, we present an algorithm to construct the greedy spanner in linear space. See Algorithm NewGreedyLinearSpace (Algorithm 3). The algorithm is similar to the algorithm due to Alewijnse et al. [1] just we do not use the data structure WSPD and instead we use the cones $\mathcal{C}_k$.

Now, we describe the algorithm NewGreedyLinearSpace in detail. Suppose that $t > 1$ and $\theta > 0$ with $0 < \theta < \pi/3$ such that $1 - 2\sin(\theta/2) \geq 1/t$. At first, the algorithm partitions the plain into k cones for $k = \lfloor \frac{2\pi}{\theta} \rfloor$. Denote the set of resulting cones by $\mathcal{C}_k$. The algorithm uses a procedure denoted by FillQueue that takes the priority queue Q and for each $C \in \mathcal{C}_k$ and $p \in S$ adds the pairs $\{p, x\}$ to Q with priority $|px|$, where $x =$ Nearest(p, C_p).

At first, the algorithm calls the procedure FillQueue. If Q is empty after a call of FillQueue, the algorithm terminates and returns E, otherwise it extracts the minimum from Q and adds the corresponding pair to the edges set E, see lines 5 and 6 of Algorithm 3. Then, the algorithm updates the priority queue Q.

Algorithm 2: FillQueue(Q)

```
1  foreach C ∈ C_k do
2      foreach p ∈ S do
3          x := Nearest(p, C_p);
4          if x is not nil then
5              Add {p, x} to Q with key |px|, and
                associate this entry with {p, C_p};
6          end
7      end
8  end
```

Algorithm 3: NewGreedyLinearSpace(S, t, θ)

```
1  Partition the plain into k cones for k = ⌊2π/θ⌋.
     Denote the set of resulting cones by C_k;
2  E := ∅;
3  Q := an empty priority queue;
4  FillQueue(Q);
5  while Q is not empty do
6      Extract the minimum from Q, let this be {u, v};
7      Add {u, v} to E;
8      foreach pair {p, C_p} with an entry in Q do
9          x := Nearest(p, C_p);
10         if x is nil then
11             Remove the entry in Q associated with
                {p, C_p} from Q;
12         end
13         else
14             Update the entry in Q associated with
                {p, C_p} to contain (p, x) and increase
                its key to |px|;
15         end
16     end
17 end
18 return E;
```

Theorem 4 *Algorithm* NewGreedyLinearSpace *works correctly, its running time is* $O(n^3 \log^2 n)$ *and uses* $O(kn)$ *space.*

Proof. Note that in the algorithm of constructing the original greedy spanner (Algorithm 1), during the process of a pair (p, q), if (p, q) is added to the edge set, then that is surely the closest pair of points among all pairs of points (x, y) such that there is no t-path between x and y in the graph computed so far. Algorithm 3 does exactly the same just the method used in this algorithm is based on lemmas 2 and 3.

Let k be a fixed number. The running time of the algorithm is computed as follows. Extracting from Q can be done in $O(\log n)$. Since by Corollary 1 the greedy spanner has $O(n)$ edges, lines 5 and 6 take $O(n \log n)$ time during total runtime. Now, by Lemma 3, loop **foreach** takes $O(n^2 \log n)$ time. Hence, overall running time is $O(n \log n \times n^2 \log n) = O(n^3 \log^2 n)$.

The space complexity of the algorithm can be computed as follows. All cones can be store in a linear space. Now, since the queue Q can be implemented by a linear space and by Lemma 3, the space complexity of the algorithm is $O(kn)$. □

4 Second Algorithm

In this section, we give an improved version of the algorithm NewGreedyLinearSpace. We call the improved algorithm by ModifiedGreedyLinearSpace. At first, we have the following observation.

Observation 1 ([1]) *Let E be some edge set for S. Let $(a, b) \in E$. Let $c, d \in S$ be points such that $|ac|, |ad|, |bc|, |bd| > t|cd|$. Then any t-path between c and d will not use the edge (a, b).*

Using Observation 1, we can improve the Line 8 in algorithm NewGreedyLinearSpace. In particular, we states that we do not need to update all entities in Q, but update some of them.

For each cone $C \in \mathcal{C}_k$ and for each $p \in S$, we consider the farthest neighbor of p denoted by $f_{p,C}$ such that $f_{p,C} \in C_p$. Furthermore, for each $C \in \mathcal{C}_k$, let $h_{1,C}$ and $h_{2,C}$ be the boundaries of the cone C. For a point $u \in S$, the minimum distance between u and the ray $h_{i,c}$ denoted by $|uh_{i,C}|$. Using the cone C_p, we define the cones C_p^-, C_p^{up} and C_p^{down} as is shown in Figure 2.

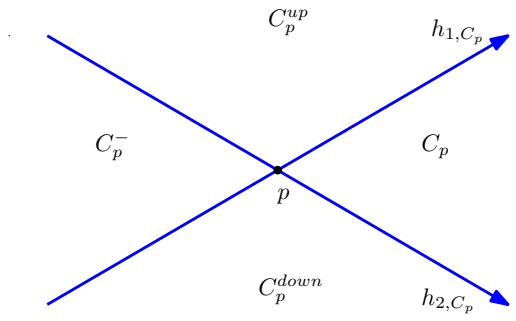


Figure 2: Cones C_p, C_p^-, C_p^{up} and C_p^{down}

Now, we present the idea of the algorithm ModifiedGreedyLinearSpace (Algorithm 4). The idea is based on what follows. Suppose that at a moment of execution of the algorithm, the pair (u, v) is added to the edge set E. Let C be the cone in $\mathcal{C}_k$ such that $v \in C_u$. Let p be an arbitrary point in S. Now, if $|pu| > t|pf_{p,C}|$, then since $f_{p,C}$ is the farthest point with respect to p in C_p, for each $q \in C_p$, we have $|pu| > t|pq|$. Hence, by Observation 1, any t-path between p and q will not use the edge (u, v). Depending on the positions of u and v, there are some cases that we are presented in Line 11.

Algorithm 4: ModifiedGreedyLinearSpace(S, t, θ)

1 Partition the plain into k cones for $k = \lfloor \frac{2\pi}{\theta} \rfloor$. Denote the set of resulting cones by $\mathcal{C}_k$;
2 **foreach** $C \in \mathcal{C}_k$ *and* $p \in S$ **do**
3 | Find the farthest neighbor of p in C_p denoted by $f_{p,C}$ with $f_{p,C} \in C_p$.
4 **end**
5 $E := \emptyset$;
6 $Q :=$ an empty priority queue;
7 FillQueue(Q);
8 **while** Q *is not empty* **do**
9 | Extract the minimum from Q, let this be $\{u, v\}$ and also suppose that C is a cone in $\mathcal{C}_k$ such that $v \in C_u$;
10 | Add $\{u, v\}$ to E;
11 | **foreach** *pair* $\{p, C_p\}$ *with an entry in* Q *such that* $|up| \leq t \times |pf_{p,C}|$ *or* $|vp| \leq t \times |pf_{p,C}|$ *or* $\left(|uh_{1,C_p}| \leq t \times |pf_{p,C}| \text{ or } |vh_{1,C_p}| \leq t \times |pf_{p,C}| \text{ when } u, v \in C_p^{up}\right)$ *or* $\left(|uh_{2,C_p}| \leq t \times |pf_{p,C}| \text{ or } |vh_{2,C_p}| \leq t \times |pf_{p,C}| \text{ when } u, v \in C_p^{down}\right)$ *or* $\left(|uh_{1,C_p}| \leq t \times |pf_{p,C}| \text{ or } |vh_{1,C_p}| \leq t \times |pf_{p,C}| \text{ or } |uh_{2,C_p}| \leq t \times |pf_{p,C}| \text{ or } |vh_{2,C_p}| \leq t \times |pf_{p,C}| \text{ when } u, v \in C_p^{-}\right)$ `/* There still exists some other cases that we are omitted them. The cases causes according to the position of` u `and` v `with respect to the cones` $C_p, C_p^-, C_p^{up}, C_p^{down}$ `(Number of all cases is 16). */`
12 | **do**
13 | | $x :=$ Nearest(p, C_p);
14 | | **if** x *is* **nil then**
15 | | | Remove the entry in Q associated with $\{p, C_p\}$ from Q;
16 | | **end**
17 | | **else**
18 | | | Update the entry in Q associated with $\{p, C_p\}$ to contain (p, x) and increase its key to $|px|$;
19 | | **end**
20 | **end**
21 **end**
22 **return** E;

Thus, by adding the edge (u, v) to E, we do not need to update the entry of Q corresponding to the pair $\{p, C_p\}$. Hence, the algorithm ModifiedGreedyLinearSpace dose not update all entries in Q. It just updates the entries of Q which have the conditions in Line 11.

Theorem 5 *Algorithm* ModifiedGreedyLinearSpace *works correctly and uses* $O(kn)$ *space.*

Proof. Proof is similar to the proof of Theorem 4. □

Note that the time complexity of Algorithm ModifiedGreedyLinearSpace is the same with the time complexity of Algorithm NewGreedyLinearSpace. We think that the time complexity of ModifiedGreedyLinearSpace can be improved because we have made an improvement on this algorithm. However, we did not find out how to improve this time.

5 Conclusion

In this paper, we considered the greedy spanner and focused on its construction using a linear space. In particular, we presented a new method to construct the greedy spanner in linear space without implementing the WSPD.

References

[1] S. P. A. Alewijnse, Q. W. Bouts, A. P. ten Brink, and K. Buchin. Computing the greedy spanner in linear space. *Algorithmica*, 73(3):589–606, Nov 2015.

[2] S. P. A. Alewijnse, Q. W. Bouts, A. P. ten Brink, and K. Buchin. Distribution-sensitive construction of the greedy spanner. *Algorithmica*, 78(1):209–231, May 2017.

[3] P. B. Callahan. *Dealing with Higher Dimensions: The Well-separated Pair Decomposition and Its Applications*. PhD thesis, Baltimore, MD, USA, 1995. UMI Order No. GAX95-33229.

[4] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM (JACM)*, 42(1):67–90, 1995.

[5] L. Barba, P. Bose, M. Damian, R. Fagerberg, W. L. Keng, J. O'Rourke, A. van Renssen, P. Taslakian, S. Verdonschot, and G. Xia. New and improved spanning ratios for Yao graphs. *Journal of Computational Geometry*, 6(2):19–53, 2015.

[6] P. Bose, P. Carmi, M. Farshi, A. Maheshwari, and M. Smid. Computing the greedy spanner in near-quadratic time. *Algorithmica*, 58(3):711–729, 2010.

[7] M. Farshi and J. Gudmundsson. Experimental study of geometric t-spanners. *Journal of Experimental Algorithmics (JEA)*, 14:3, 2009.

[8] G. Narasimhan and M. Smid. *Geometric spanner networks*. Cambridge University Press, 2007.

Approximate Hotspots of Orthogonal Trajectories

Ali Gholami Rudi*

Abstract

In this paper we study the problem of finding hotspots of polygonal two-dimensional trajectories, i.e. regions in which a moving entity has spent a significant amount of time. The fastest optimal algorithm, due to Gudmundsson, van Kreveld, and Staals (2013), finds an axis-parallel square hotspot of fixed side length in $O(n^2)$. We present an approximation algorithm with the time complexity $O(n \log n)$ and approximation factor 1/4 for orthogonal trajectories, in which the entity moves in a direction parallel either to the x or to the y-axis.

Keywords: Trajectory, Hotspot, Geometric algorithms

1 Introduction

Tracking technologies like GPS gather huge and growing collections of trajectory data, for instance for cars, mobile devices, and animals. The analysis of these collections poses many interesting problems, which has been the subject of much attention recently [1]. One of these problems is the identification of the region, in which an entity has spent a large amount of time. Such regions are usually called stay points, popular places, or hotspots in the literature.

We study polygonal trajectories, in which the trajectory is obtained by linearly interpolating the locations of the moving entity, recorded at specific points in time (this model, also called piecewise-linear trajectories [2], is very common in the literature [3]). For this model, Gudmundsson et al. define several problems about trajectory hotspots and present an $O(n^2)$ algorithm to solve the following [4]: defining a hotspot as an axis-aligned square of fixed side length, we wish to find the placement of such a square that maximizes the time the entity spends inside it (there are other models and assumptions about hotspots, for a brief survey of which, the reader may consult [4]; e.g. the assumption of pre-defined potential regions [5], counting only the number of visits or the number of visits from different entities [6], or based on the sampled locations only [7]). To solve this problem, they first show that the function that maps the location of the square to the duration the trajectory spends inside it, is piecewise linear and its breakpoints happen when a side of the square lies on a vertex, or a corner of the square on an edge of the trajectory. Based on this observation, they subdivide the plane into $O(n^2)$ faces and test each face for the square with the maximum duration.

Limiting ourselves to orthogonal trajectories, in which each edge is parallel to an axis of the coordinate system, we present an $O(n \log n)$ time approximation algorithm. Compared to the time the entity spends in the optimal placement of the square, the entity spends no less than 1/4 of that time in the square returned by our algorithm. It sweeps two parallel lines to find the best square, after decomposing the trajectory. Unlike Gudmundsson et al.'s algorithm, extending which to three-dimensional trajectories seems nontrivial, an extension of the algorithm presented in this paper can find approximate, axis-parallel, cube hotspots of trajectories in $\mathbb{R}^3$ with approximation factor 1/4 and the time complexity $O(n^2 \log n)$. This algorithm is explained in the extended version of this paper[1]. The extended version also describes a 1/2-approximation algorithm for two-dimensional trajectories with the time complexity $O(n \log^3 n)$.

The rest of this paper is organized as follows. In Section 2, we describe our model in more detail and introduce the notation used in this paper. We also prove results that we use in Section 3, in which we present our algorithm for finding approximate hotspots of orthogonal trajectories. Finally, in Section 4 we conclude this paper.

2 Preliminaries and Basic Results

A trajectory specifies the location of a moving entity through time. Therefore, it can be described as a function that maps each point in a specific time interval to a location in the plane. Similar to Gudmundsson et al. [4], we assume that trajectories are continuous and piecewise linear. The location of the entity is recorded at discrete time intervals, which we call the vertices of a trajectory. We assume that the entity moves in a straight line and with constant speed from a vertex to the next (this simplifying assumption is very common in the literature but there are other models for the move-

*Department of Electrical and Computer Engineering, Bobol Noshirvani University of Technology, Babol, Iran. Email: gholamirudi@nit.ac.ir.

[1] https://arxiv.org/abs/1710.05185

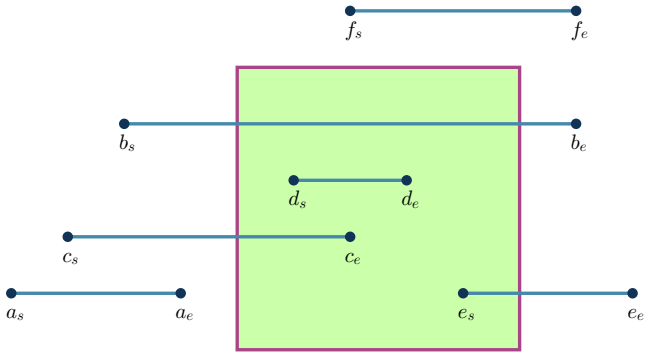


Figure 1: The entering rate of b and e and the leaving rate of b and c are nonzero

ment of the entity between vertices [8]); we call the sub-trajectory connecting two contiguous vertices, an edge of the trajectory.

In this paper, we relax the requirement that a trajectory is continuous. We assume that a trajectory T is a set of edges $\{e_1, e_2, ..., e_n\}$. For an edge e, we associate a weight w_e, which denotes the duration of the sub-trajectory through its end points (the difference between the time recorded for its end points). We also denote the left and the right vertex of an edge with e_s and e_e respectively. In orthogonal trajectories, all trajectory edges are parallel either to the x or to the y-axis. In horizontal (similarly vertical) trajectories all edges are parallel to the x-axis (y-axis).

For any axis-parallel square r with some fixed side length, we define the weight of r as the total duration in which the entity has spent inside it and denote it with $w_T(r)$, or if there is no confusion $w(r)$. A hotspot is an axis-parallel square with side length s and with the maximum possible weight. We denote the weight of a hotspot of trajectory T with $h_s(T)$.

Lemma 1 *Let H and V be a partition of an orthogonal trajectory T, in which H contains the horizontal edges and V contains the vertical edges of T. Let w be the maximum of $h_s(H)$ and $h_s(V)$. Then, w is at least $h_s(T)/2$.*

Proof. Let r be a hotspot in T. Every edge of T is either in H or in V and thus $w_H(r) + w_V(r)$ equals $h_s(T)$. Therefore, either $w_H(r) \geq h_s(T)/2$ or $w_V(r) \geq h_s(T)/2$. Since $h_s(H) \geq w_H(r)$ and $h_s(V) \geq w_V(r)$, we have $max(h_s(H), h_s(V)) \geq h_s(T)/2$ as required. □

Let r be an axis-parallel square and T be a horizontal trajectory. The entering rate of an edge e of T with respect to r is the rate at which the contribution of the weight of the edge to the weight of r increases, if the right side of r is moved to the right. Similarly, the leaving rate of an edge e with respect to r is the rate at which the contribution of the weight of the edge to the weight of r decreases, if the left side of r is moved to the right. We denote the former as $r_+(e)$ and the latter as $r_-(e)$. It is not difficult to see that $r_+(e)$ (and similarly $r_-(e)$) is either zero or the ratio of its duration to its length, which we denote as $d(e)$. In Figure 1, except the entering rate of b and e, and the leaving rate of b and c, the entering and leaving rates of all edges are zero.

The entering rate of horizontal trajectory T with respect to square r, denoted as $r_+(T)$, is defined as the sum of the entering rate of all edges of T. Similarly, the leaving rate of trajectory T with respect to square r is the sum of the leaving rate of all edges of T; this is denoted as $r_-(T)$.

Lemma 2 *Let T be a horizontal trajectory. There exists a square with side length s, whose weight equals $h_s(T)$ and one of its vertical sides contains a vertex of T.*

Proof. Let r be a square with weight $h_s(T)$ and suppose none of its vertical sides contains a vertex of T. Clearly, $r_+(T)$ cannot be greater than $r_-(T)$; otherwise, the weight of r increases by moving it to the right, which is impossible since it is a hotspot. Similarly, $r_-(T)$ cannot be greater than $r_+(T)$. Therefore, $r_+(T) = r_-(T)$ and by moving r to the right until one of its sides meets a vertex of T, its weight does not change. □

Theorem 3 *Let T be a horizontal trajectory and let h be the maximum weight of a square with side length s, one of whose corners coincides with one of the vertices of T. Then, $h \geq h_s(T)/2$.*

Proof. Let r be the square with weight $h_s(T)$, one of whose vertical sides contains a vertex v of T (such a square surely exists, as shown in Lemma 2). Suppose v is on the left side of r (the argument for the right side is similar). Let r' and r'' be the squares with side length s, whose lower left and upper left corners are on v respectively. Given that the union of r' and r'' covers r, $w(r') + w(r'')$ is at least $h_s(T)$ and therefore $max(w(r'), w(r''))$ is at least $h_s(T)/2$. Since $h \geq max(w(r'), w(r''))$, we have $h \geq h_s(T)/2$. □

3 A 1/4 Approximation Algorithm

In this section we present an approximation algorithm that, given an orthogonal trajectory T, finds an axis-aligned square, whose weight is at least $h_s(T)/4$. We start with Algorithm 3.1, which finds the square with the maximum weight among those whose lower right or upper right corners are on one of the vertices of the given horizontal trajectory. The algorithm assumes that the x coordinate of all vertices is at least 0; otherwise the trajectory may be shifted in the positive direction of the x-axis. Also, the algorithm uses the Fenwick tree data

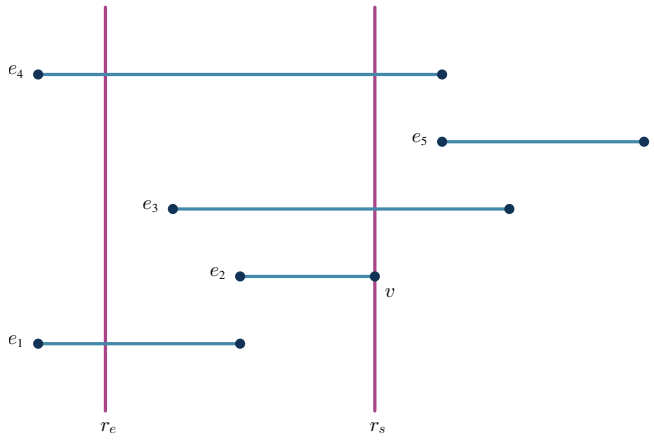


Figure 2: Sweeping two parallel lines in Algorithm 3.1

structure, supporting the computation of the prefix-sum of a sequence of n numbers and updating any of them in $O(\log n)$ [9]. For a Fenwick tree f with n elements, Add(f, i, c) increases the value of the i-th element by c, Sum(f, i, j) returns the sum of the elements i through j, and Get(f, i) returns the value of the i-th element. The algorithm maintains three Fenwick trees, each with size n: *fixed* for the fixed contributions of the edges to the weight of containing squares and *entering* (similarly *leaving*) for the entering (leaving) rate of the edges with respect to containing squares (the algorithm moves two parallel sweep lines and both vertical sides of the squares considered in the algorithm are on these lines).

One of the key ideas in Algorithm 3.1 is to maintain only the current entering and leaving rates of the edges and assume that each entering or leaving interval has begun from $x = 0$. Therefore, to compute the total contribution of entering and leaving edges, the algorithm simply multiplies the sum of their rate with the current value of x. To compensate for the intervals included in this computation before the actual entering or leaving intervals, the algorithm updates the *fixed* Fenwick tree. This frees the algorithm from storing a different starting position for the entering and leaving rates of each of the edges, which would make the computation of the weight of square r more complex. The correctness of this algorithm is shown in Theorem 4.

Theorem 4 *Among all axis-parallel squares with side length s and with a right corner on a vertex of a horizontal trajectory T, Algorithm 3.1 finds a square with the maximum weight and with time complexity $O(n \log n)$.*

Proof. The algorithm maintains the entering and leaving rates of all edges with respect to any square r that contains them and its left and right sides are on r_s and r_e respectively: the entering rate of an edge e_i is increased by $d(e)$ as its left vertex meets r_s and it is reset (decreased by $d(e)$) when it meets r_e (therefore,

Algorithm 3.1: RightCornerHotspots(T, s)

Input : A horizontal trajectory T and length s; n is the number of the edges of T.
Output: An axis-parallel square with side length s and with one of its right corners on a vertex of T, with the maximum weight.

fixed, *entering*, *leaving*: n-element Fenwick trees, with all elements zero initially.
Sort the edges of T increasingly by the value of their y coordinate to obtain the sequence $e_1, e_2, ..., e_n$ (note that all edges are horizontal and the height of some of the edges may be equal).
Move two parallel vertical sweep lines with distance s horizontally to the right. We denote the x-coordinate of the left sweep line with r_e and the right sweep line with r_s.
for *each event, i.e. a vertex v of T meeting any of these sweep lines (suppose v is an endpoint of e_i)* **do**
  if *v is the left vertex of e_i and is on $x = r_s$* **then**
    Add$(\textit{entering}, i, d(e_i))$
    Add$(\textit{fixed}, i, -x \cdot d(e_i))$
  if *v is the left vertex e_i and is on $x = r_e$* **then**
    Add$(\textit{entering}, i, -d(e_i))$
    Add$(\textit{fixed}, i, s \cdot d(e_i) - x \cdot d(e_i))$
  if *v is the right vertex e_i and is on $x = r_s$* **then**
    Add$(\textit{leaving}, i, d(e_i))$
    Add$(\textit{fixed}, i, x \cdot d(e_i))$
  if *v is the right vertex e_i and is on $x = r_e$* **then**
    Add$(\textit{leaving}, i, -d(e_i))$
    Add$(\textit{fixed}, i, -s \cdot d(e_i) - x \cdot d(e_i))$
  Find the maximum value of j, such that $j \geq i$ and the difference in the height of e_i and e_j is no more than s (this can be done with a simple binary search). The weight of the square whose lower right corner is at v is $ax + c$, in which a is the sum of Sum$(\textit{entering}, i, j)$ and Sum$(\textit{leaving}, i, j)$ and c is Sum$(\textit{fixed}, i, j)$.
  Record this as the best square, if this weight is the maximum so far.
  Do likewise for the square whose upper right corner is at v (for this case, the minimum value of j should be found such that $j \leq i$ and the difference between the height of e_i and e_j is at most s).
return the square with the maximum weight.

Get($entering, i$) is always $r_+(e_i)$). The leaving rate of the edges is updated similarly. Let r be the square considered in the loop and $e = e_k$, such that k is in the interval from i through j. To show that the algorithm finds the weight of r correctly, we show that the contribution of edge e to the weight of r is $contrib(e) = (\text{Get}(entering, k) - \text{Get}(leaving, k)) \cdot x + \text{Get}(fixed, k)$.

There are five cases to consider regarding the relative position of an edge and the two sweep lines. These cases are demonstrated in Figure 2: an edge may be outside (e_5) or inside (e_2) the region bounded by the two sweep lines, or it may intersect the right sweep line (e_3), the left sweep line (e_1), or both (e_4).

It is not difficult to show that the contribution of an edge e_i to the weight of r is equal to $contrib(e)$ in all five cases. If e is outside the region bounded by the two sweep lines, the entering and leaving rates and Get($fixed, k$) are all zero. When e_i intersects only the right sweep line (r_e), as e_3 in Figure 2, $contrib(e) = x \cdot d(e) - x(e_s) \cdot d(e)$, in which $x(p)$ is the value of the x coordinate of point p. This clearly is equal to the contribution of the edge e to the weight of any containing square r whose vertical sides are on r_s and r_e. We omit other cases for brevity. □

The following theorem shows how we can use Algorithm 3.1 and the theorems proved in the previous section to obtain an approximation algorithm for finding square hotspots of fixed side length for orthogonal trajectories.

Theorem 5 *There is an approximation algorithm for finding hotspots (axis-parallel squares with side length s) of orthogonal trajectories, such that the weight of the square found by the algorithm is at least $1/4$ of the optimal value ($h_s(T)$).*

Proof. Let T be an orthogonal trajectory. T can be partitioned into sets V and H containing the vertical and horizontal edges of T respectively. Then, Algorithm 3.1 finds a square r_H with the maximum possible weight, in which one of its corners is on a vertex of H (Algorithm 3.1 can be performed twice, once after rotating the plane 180 degrees to find the maximum-weight squares with one of its left corners on a vertex of H). The same algorithm can obtain a square r_V with the maximum possible weight for V, after rotating the plane 90 degrees. By Theorem 3, $w(r_H) \geq h_s(H)/2$ and $w(r_V) \geq h_s(V)/2$. Also, by Lemma 1, $max(h_s(H), h_s(V)) \geq h_s(T)/2$ implying that $max(w(r_H), w(r_V)) \geq h_s(T)/4$, as required. □

4 Discussion

The 1/4-approximation algorithm presented in this paper can be extended to three dimensions to find cube hotspots of fixed side length for orthogonal trajectories in $\mathbb{R}^3$; this algorithm is presented in the extended version of this paper. In the extended version, we also describe a 1/2-approximation algorithm with the time complexity $O(n \log^3 n)$, which maintains a kinetic tournament tree on a segment tree. It may be possible to improve this to obtain an exact algorithm. One of our motivations for studying orthogonal trajectories was finding efficient approximation algorithms for general trajectories; this goal requires further studies.

References

[1] Y. Zheng. Trajectory data mining - an overview. *ACM Transactions on Intelligent Systems and Technology*, 6(3):29:1–29:41, 2015.

[2] M. Buchin, A. Driemel, M. J. van Kreveld, and V. Sacristán. Segmenting trajectories - a framework and algorithms using spatiotemporal criteria. *Journal of Spatial Information Science*, 3(1):33–63, 2011.

[3] B. Aronov, A. Driemel, M. J. van Kreveld, M. Löffler, and F. Staals. Segmentation of trajectories on nonmonotone criteria. *ACM Transactions on Algorithms*, 12(2):26:1–26:28, 2016.

[4] J. Gudmundsson, M. J. van Kreveld, and F. Staals. Algorithms for hotspot computation on trajectory data. In *SIGSPATIAL/GIS*, pages 134–143, 2013.

[5] L. O. Alvares, V. Bogorny, B. Kuijpers, J. A. F. de Macêdo, B. Moelans, and A. A. Vaisman. A model for enriching trajectories with semantic geographical information. In *ACM International Symposium on Geographic Information Systems*, page 22. ACM, 2007.

[6] M. Benkert, B. Djordjevic, J. Gudmundsson, and T. Wolle. Finding popular places. *International Journal of Computational Geometry and Applications*, 20(1):19–42, 2010.

[7] S. Tiwari and S. Kaushik. Mining popular places in a geo-spatial region based on gps data using semantic information. In *Workshop on Databases in Networked Information Systems*, pages 262–276. Springer, 2013.

[8] H. J. Miller. Modelling accessibility using space-time prism concepts within geographical information systems. *International Journal of Geographical Information Science*, 5(3):287–301, 1991.

[9] P. M. Fenwick. A new data structure for cumulative probability tables - an improved frequency-to-symbol algorithm. *Software, Practice and Experience*, 26(4):489–490, 1996.

Knowledge Representation for the Geometrical Shapes

Abolfazl Fatholahzadeh[a,b*] Dariush Latifi[c†]

Abstract

This paper outlines the necessity of the knowledge representation for the geometrical shapes(KRGS). We advocate that KRGS for being powerful must contain at least three major components, namely (1) fuzzy logic scheme, (2) the machine learning technique, and (3) an integrated algebraic and logical reasoning. After arguing the need for using fuzzy expressions in spatial reasoning, then inducing the spatial graph generalized and maximal common part of the expressions is discussed. Finally, the integration of approximate references into spatial reasoning using absolute measurements is outlined. The integration here means that the satisfiability of a fuzzy spatial expression is conducted by both logical and algebraic reasoning.

Keywords: Knowledge representation, integrated algebraic and logical, fuzzy logic reasoning, machine learning.

1 Introduction

Referring in *practical* spatial description is seldom absolute. Sometimes, due to the lack of precise information, it is not possible to represent the $\mathtt{x}-\mathtt{y}$ coordinate of the vanishing points. In this case, symbolic knowledge can be used as mean of expression to situate the position of an absolute point with respect to a plane. For instance, in image processing by using `3D` projective space [6], one can use the relation between a point `P` and an object `O` via the `3D` line `PQ`, where `Q` is an ideal point. Often absolute measurements are unnecessary: if we want to know whether an object will pass through a hole, it is sufficient to know the *relative* size of the hole and object. Another example is the problem of soil classification, where the determination of some class is based on the `above` relation with respect to a particular line and the plasticity index.

The aim of this paper is to advocate in favor of three mentioned above components. Using concrete examples, We provide the evidences why these components are mandatory. The rest of the paper is organized as follows. Section 2 describes the representation of the fuzzy references. Section 3 sketches learning spatial graph. How the satisfiability process along with the integration of algebraic and logical reasoning, can be done, is explained in Section 4. Section 5 gives conclusions and future problems.

*a: University of CentraleSupélec, France. b: Institute for Research in Artificial Intelligence, University of Mohaghegh Ardabili, Iran, abolfazl.fatholahzadeh@centralesupelec.fr

†c: Department of Mathematics, University of Mohaghegh Ardabili, Iran, latifi@uma.ac.ir

2 Fuzzy References

We shall call fuzzy expressions those expressions including at least one approximate references like `above`, `below`, `over`, and `under` [2].

Their intuitive meanings can be depicted by Figure 1. To represent these predicates, we first describe how to map $\mathtt{F_1}=\{\mathtt{above},\mathtt{below}\}$ into algebraic reasoning. We then use these knowledge of F_1 with additional the predicates describing point and line relations to express the description of $\mathtt{F_2}=\{\mathtt{over},\mathtt{under}\}$.

For mapping $\mathtt{F_1}$ into algebraic reasoning, let us suppose that a fuzzy subset be characterized by a function, μ, called compatibility function, over a set of elements, called the universe of discourse, `U`, where $\mathtt{U}=\{\mathtt{u}_1,\mathtt{u}_2,\cdots,\mathtt{u}_n\}$ and $\mu:\mathtt{U}\rightarrow[0,1]$. A function μ is called $\prod-\mathtt{type}$ if there exists only one point at which monotonicity changes direction. The effect of `above` and `below` on a $\prod-\mathtt{type}$ can be best described by.

$$\mu_{above_x(U_i)}=\begin{cases}1-\mu_x(U_i) & \text{if } U_i\geq U_{max}\\ 0 & \text{otherwise}\end{cases}$$

$$\mu_{below_x(U_i)}=\begin{cases}1-\mu_x(U_i) & \text{if } U_i\leq U_{min}\\ 0 & \text{otherwise}\end{cases}$$

where $\mathtt{U}_{max}(\mathtt{U}_{min})$ is the value of `U`; where $\mu_x(\mathtt{U}_i)$ attains its maximum (minimum) value. It is worth to mention that in many practical applications including continuous domains, the data collected in real-world experiment are discrete. Therefore, presumably, there are appropriate segments that are representative knowledge of the domain. Consequently, this observation can be best combined by *Eshragh* and *Mamdani*'s idea [1]: the separation of fuzzy spreads into an appropriate number of segments with well defined characteristics.

Having the knowledge of $\mathtt{F_1}$, it now is possible to represent $\mathtt{F_2}$. Let us take `under` predicate, where by convention $\mathtt{under}(a,b)$ means that `a` is under `b`; where the variables `a` and `b` denote points. The representation of this predicate can be expressed by three other predicates, namely, *perpendicular* (or `perpen`

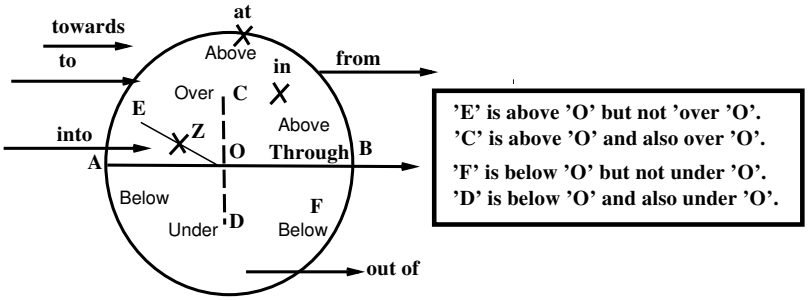


Figure 1: Points and approximate references.

for short), `below` and `online`, where `online`(P, A, B) mean that point `P` is on line segment `AB`; where `perpend`(A, B, C, D) represents the line segments AB and CD are perpendicular. Having the above definitions the logical representation of the predicate `under` can be defined.

Table 1 shows the definition of `under` predicate expressed in terms of those predicates taking points as their arguments. Those relations whose algebraic representation include inequalities are called *order relations*. As is clear from the definitions in Table 1, these predicates are non-order relations since they are defined in terms of non-order relations `on`, `eqseg` and `noteq`. Several redundant `noteq`s are included in Table 1 to clarify non-degenerated case specifications.

Note that the valid algebraic representations of the order relations cannot be obtained in the *Gröbner* basis method. This is also true in the geometric domain, where for instance, `between` and `eqang` whose meanings will be given later, are order relations. As pointed in [5], all geometric theorems proved so far by the Gröbner basis method do not include any order relations. This is also the case in Wu's method [7].

3 Learning Spatial Graph

Any geometrical shape can be expressed by a logical expression (Exp). In order to speed up the reasoning process, it is desirable to find a way for determining the common part of two or more geometrical shapes. In other words, learning the generalized common maximal (GCM) for the current expressions is required.

An n-ary predicate will be represented by $t_1(t_2, \cdots, t_n)$. Each t_i is a term, which may be either a constant, represented by lower case Roman letters, or a variable shown by upper case Roman letters. A literal is a list of terms, optionally prefixed by the logical negation ($\neg$) operator. For instance, on(o1,o2) and red(X) are both literals. If we consider

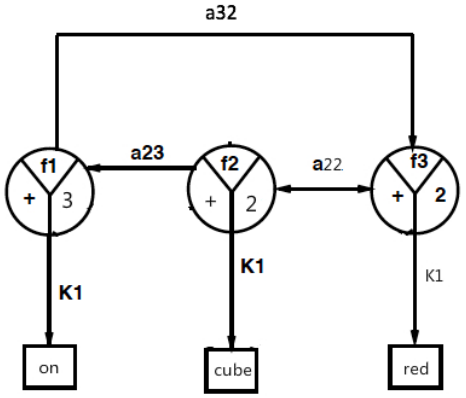


Figure 2: Generalized Spatial Graph of Exp_1 and Exp_2.

two following expressions:

$$\begin{aligned} Exp_1 =& on(o1, o2) \wedge sphere(o1) \wedge red(o1) \wedge \\ & cube(o2) \wedge red(o2) \\ Exp_2 =& on(o3, o4) \wedge pyram(o3) \wedge blue(o3) \wedge \\ & cube(o4) \wedge red(o4) \end{aligned}$$

where the predicate on(X,Y) means that Y is on X, the meaning other ones are self meaning. Then we obtain the following output expression: $GCM(Exp1, Exp2) = on(X, Y) \wedge red(Y) \wedge cube(Y)$ which is obtained by the linearization of the spatial graph shown in Figure 2.

An expression graph is a 6-tuple [8] $(L, C, \sigma, \theta, K, a)$ where L (resp. C) is a finite set of literal (resp. constant) nodes; σ is the literal dimension function $L \to Z_+$ (i.e. the set of positive integers; θ is the literal sign function $L \to \{+, -\}$; K is the literal partial content function $I \times Z_+ \to C$, such that, if $\ell, i, c) \in K$, then $i \in \{1, 2, \ldots, \sigma(\ell)\}$; and finally, a is the literal adjacency relation, a finite subset of $Z_+ \times Z_+ \times L \times L$ along with the following properties:

1. Symmetry: $(i_1, i_2, \ell_1, \ell_2) \in a$ iff $(i_2, i_1, \ell_2, \ell_1) \in a$

Table 1: Logical representation of the predicate `under`

under(D,C)	$(\exists A, B)$ below(D,C) $\wedge$ perpen(A,B,C,D)
perpen(A,B,C,D)	$noteq(A,B) \wedge noteq(C,D) \wedge ((\neg noteq(A,C) \wedge rangle(B,A,D)) \vee (noteq(A,C) \wedge online(A,C,D) \wedge rangle(B,A,C)) \vee (noteq(A,C) \wedge \neg online(A,C,D) \wedge online(C,A,B) \wedge rangle(A,C,D)) \vee (noteq(A,C) \wedge \neg online(A,C,D) \wedge \neg rangle(C,A,B) \wedge (\exists O)(online(O,A,B) \wedge online(O,C,D) \wedge online(A,O,C))))$
rangle(A,B,C)	$noteq(A,B) \wedge noteq(B,C)(\exists O)(midpoint(B,A,O) \wedge eqseq(A,C,C,O))$
midpoint(O,A,B)	$noteq(A,B) \wedge collinar(O,A,B)$
eqseg(A,B,C,D)	length(AB) = length(CD)
collinear(O,A,B)	$(\exists L)\ on(A,L) \wedge on(B,L) \wedge on(C,L) \wedge noteq(O,A) \wedge noteq(O,B) \wedge noteq(A,B)$
online(O,A,B)	$noteq(A,B) \wedge collinear(O,A,B)$

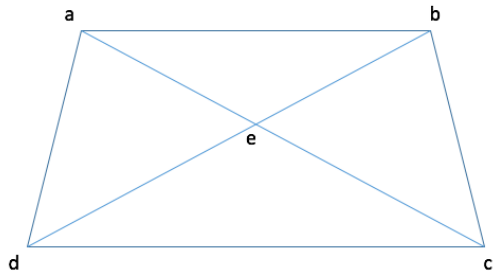


Figure 3: Hypotheses: eqang(e,a,d,e,b,c) and eqseg(e,a,e,b). Task: para(a,b,d).

2. Transitivity: if $(i_1, i_2, \ell_1, \ell_2) \in a$ and $(i_1, i_3, \ell_1, \ell_3) \in a$, then $(i_1, i_3, \ell_1, \ell_3) \in a$

3. Consistency: $\forall \ell_1, \ell_2 \in L$ and $i_1, i_2 \in Z_+$, if $(\ell_1, i_1, c_1) \in K$ and $(\ell_2, i_2, c_2) \in K$, then $(i_1, i_2, \ell_1, \ell_2) \in a$ iff $c_1 = c_2$.

In the method given in [8], the generlaization replaces just two expresions. We have developped a method, not reported here, to accepts more than two expresions. A common LISP software has been writen which implemnts and confirms the method.

It is intersting to point out that the expression graph can also be used for one expression, as in Figure 4 by a combination of ways, including above mentionned properties, done for the evaluation of the predicate para(a,b,c,d) of Figure 3 under the hypotheses depicted at the head part of Figure 4, except $\wedge \neg para(a,b,c,d)$.

4 Satisfiability of Fuzzy Spatial Expression

Definition: Let `Expr` be the set of spatial references of the following form: $\texttt{Expr} = \texttt{Pred}_1 \wedge \texttt{Pred}_2 \cdots \wedge \texttt{Pred}_n$, where $\texttt{Pred}_i$ for $i \leq 1 \leq n$ is a spatial predicate. If at least one above predicate is a fuzzy one, then the expression is called fuzzy one. An example of such expression is the following one.

$$Expr = \underbrace{\texttt{online}(Z,O,E)}_{Pred_1} \wedge \underbrace{\texttt{above}(Z,O)}_{Pred_2} \wedge \underbrace{\texttt{under}(D,O)}_{Pred_3}$$

where $\texttt{online}(Z,O,E)$ means that the point `Z` is on segment line `OE`. This example can be used in the *interpretation of laser-material experiments* where before perforating `Z`, we would like to be sure of the following information:

- 'Z' is above 'O' and also on Zapata's line.
- 'D' is under 'O'.

where *Zapata's line* is a nickname visualized in Figure 1 by $L = [O,E]$. Let us suppose `Expr` can be divided into two sub-expressions, such that $\texttt{Expr} \equiv \texttt{Expr}_h \wedge (\neg \texttt{Expr}_c)$. By convention $\texttt{Expr}_h$ and $\texttt{Expr}_c$ will be called problem hypotheses and conclusion, respectively.

Satisfiability: Let `Axioms` denote the set of application's axioms. Then the prof of domain-dependent property $\texttt{Expr}_c$ under a given set of hypotheses $\texttt{Expr}_h$ is formalized as follows.

(1): $\texttt{Axioms} \cup \texttt{Expr}_c \vdash \texttt{Expr}_c$
(2): $\texttt{Axioms} \models \texttt{Expr}_h \rightarrow \texttt{Expr}_c$
(3): $\texttt{Expr}_h \rightarrow \texttt{Expr}_c$
(4): $\neg(\texttt{Expr}_h \rightarrow \texttt{Expr}_c) \equiv \texttt{Expr}_h \wedge (\neg \texttt{Expr}_c) \equiv \texttt{Expr}$

The formula (1) is equivalent to the (2),which implies that all logical models of `Axioms` satisfy (3). In refutational reasoning, (4) is proved by showing that the negation of the `Expr` is not satisfied by any logical models of `Axioms`. However, since it is known that `Axioms`

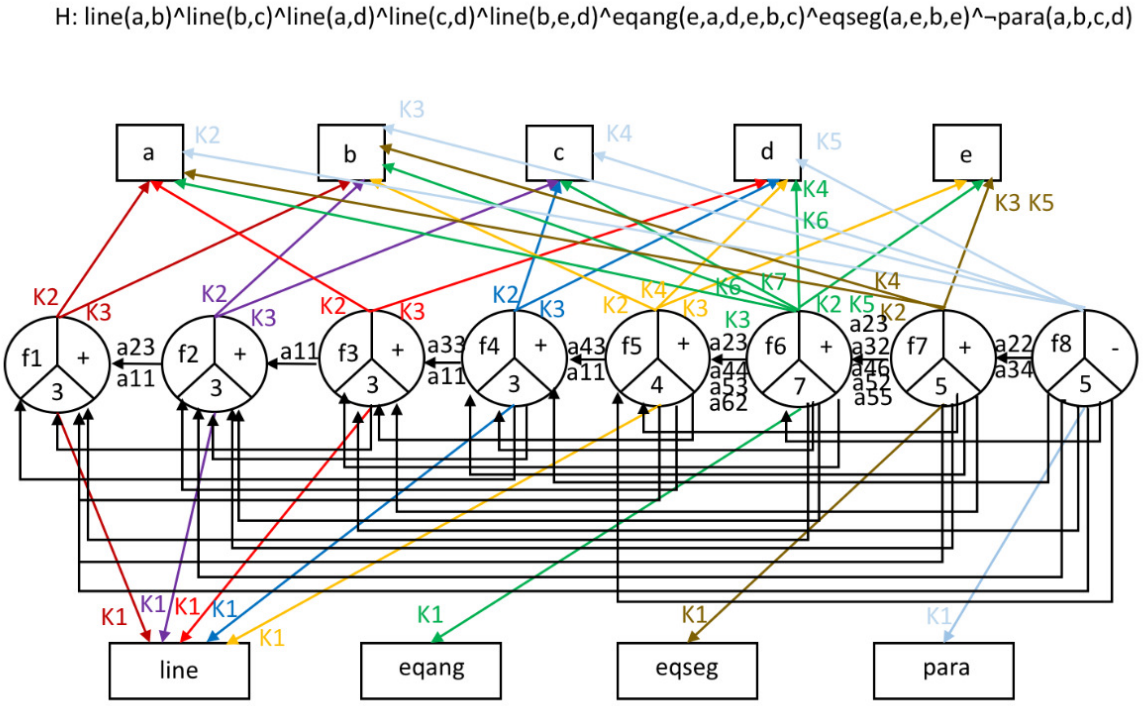


Figure 4: Expression graph for the task para(a,b,c,d).

is categorical and all its logical models are isomorphics, it is sufficient to show that logical formula of `Expr` is not satisfied by a specific logical model of `Axioms`. For complete details of the integrated algebraic and logical reasoning and termination/correctness proofs, as well as the limitations of that method, see [5].

Evaluation: In addition to our four fuzzy relations, in our work, nine predicates taking points as their arguments are used: `eqseg`, `eqang`, `collinear`, `online`, `midpoint`, `para`, `rangle`, `perpen` and `line`. The predicate `line` take a plain list of points and declare the existence of a straight line as well as the fact that the points in the list are aligned on that line. Furthermore, `line` is an order relation like `between`, where $\texttt{between}(P, A, B)$ means that point `P` is located between a pair points `A` and `B`.

Therefore, by this definition, `line` can appear only in `Expr` to maintain the soundness of the integrated reasoning. Moreover, since `line` subsumes non-order relations `collinear` and `online`, we do not use the latter predicate to describe the hypotheses; `collinear` and `online` are used within a spatial expression to describe the conclusion. Among the four following predicates `on`, `between`, `eqseg` and `eqang` that can be used to describe `Expr`, `between` is substituted by `line`.

The predicate `on` is excluded, because no information useful for reasoning is specified by describing either $\texttt{on}(A, L)$ or $\texttt{on}(A, L) \wedge \texttt{on}(B, L)$, and because the collinearity among more than three points, $\texttt{on}(A, L) \wedge \texttt{on}(B, L) \wedge \texttt{on}(C, L) \wedge \cdots$ can be described using the predicate `collinear`. The predicate $\texttt{on}(A, L)$ means that the point A is on line L and it is used internally by the evaluator.

To facilitate the evaluation process, it is often useful to define higher-level predicates. As pointed out in [4, 5], however, their meanings must be specified very strictly; careless loose definitions. Table 1 shows the strict definitions of six higher-level predicates, which are used in [5] following the method described in [4]. Here the predicate $\texttt{noteq}(x, y)$ implies that two points x and y are different. This predicate is non-order relation and is often used to specify subsidiary conditions to exclude degenerated case. Table 2 shows the algebraic representation of $\texttt{under}(x, y)$ predicate depending on the seven predicates of Table 1.

Among the nine mentioned predicates, `eqseg`, `eqang` and `para` are equivalent relations. It is often possible to express a fuzzy spatial expression using one of the mentioned predicate, like this one: $\texttt{para}(A, B, C, D) \wedge \texttt{below}(F, E)$, where the predicate $\texttt{para}(A, B, C, D)$ means that the line segments AB and CD are parallel. This predicate can be logically expressed by the following representation: $\neg(\exists P)(\texttt{online}(P, A, B) \wedge \texttt{online}(P, C, D))$. As pointed out in [5], this predicate

Table 2: Algebraic representations of the predicate `under`

under(D,C)	$(\exists A, B)$ below(D,C) $\wedge$ perpen(A,B,C,D)
below(A,B)	$(\exists \prod -type)(AB \subset \prod -type)\ (U_i \leq U_{min}) \wedge (1 - \mu_x(U_i))$
perpen(A,B,C,D)	$(x_A \neq x_B \vee y_A \neq y_B) \wedge (x_C \neq x_D \vee y_C \neq y_D) \wedge$ $(x_B - x_A)(x_D - x_C) + (y_B - y_A)(y_D - y_C) = 0$
noteq(A,B)	$x_A \neq x_B \wedge y_A \neq y_B$
eqseg(A,B,C,D)	$(x_B - x_A)^2 - (y_B - y_A)^2 = (x_D - x_C)^2 - (y_D - y_C)^2$
collinear(A,B,C)	$(y_B - x_A)x_C + (x_A - x_B)y_C + (x_B y_A - x_A y_B) = 0$
online(P,A,B)	$(x_A \neq x_B \vee y_A \neq y_B) \wedge (y_A - y_P y_B + x_A y_P - x_P y_A) = 0$
midpoint(P,A,B)	$(x_A \neq x_B \vee y_A \neq y_B) \wedge (2x_P - x_A - x_B = 0) \wedge$ $(2y_P - y_A - y_B = 0)$

is defined as a non-order relation. If `para` is used in the problem hypotheses, it must be possible to evaluate the directions of the pair of line segments AB and CD coincide with each other or not. To solve this, we report the definition of ordered parallel (`opara` for short)

$$\begin{aligned} \texttt{opara}(A,B,C,D) \Leftrightarrow \texttt{para}(A,B,C,D) \wedge \\ (\exists P)(\texttt{between}(P,A,D) \wedge \texttt{between}(P,B,C)) \\ \texttt{para}(A,B,C,D) \Leftrightarrow \neg(\exists P)(\texttt{online}(P,A,B) \\ \wedge \texttt{online}(P,C,D)) \end{aligned}$$

When `between` is included in the spatial expression the evaluator uses it as it is in the forward reasoning, while $\texttt{between}(P,A,B)$ is transformed into the non-order relations.

$$\begin{aligned} \texttt{between}(P,A,B) \equiv (\exists L) \wedge \texttt{on}(P,L) \wedge \texttt{on}(A,L) \\ \wedge \texttt{on}(B,L) \wedge (\texttt{A} \neq \texttt{B}) \wedge (\texttt{A} \neq \texttt{P}) \wedge (\texttt{B} \neq \texttt{P}) \end{aligned}$$

As appear from the above relation, the reasoning capacity about `between` is very limited. With the integration of the algebraic representation of $\texttt{para}(A,B,C,D)$ and $\texttt{between}(P,A,B)$ the above question can be done by the evaluation of the predicate $\texttt{opara}(A,B,C,D)$ of the following algebraic representation.

$$\begin{aligned} \texttt{para}(A,B,C,D) \Leftrightarrow (x_A \neq x_B \vee y_A \neq y_B) \wedge \\ (x_C \neq x_D \vee y_C \neq y_D) \wedge (y_B - y_A)x_C + \\ (x_A - x_B)y_C + (x_B y_A - x_A y_B) \neq 0 \wedge \\ (x_B - x_A)(y_D - y_C) - (x_D - x_C)(y_B - y_A) = 0 \end{aligned}$$

Since the evaluation of a fuzzy spatial expression `Expr` is by essence is equivalent to the problem of unsatisfiability in refutational logic then the expansions of completeness and the validation of soundness concerning the inclusions of four mentioned fuzzy relations into the reasoning method of Matsuyama and Nitta can be done provided that the domain-dependent axioms be carefully defined.

It is intersting to point out that the expression graph can also be used just for one expression as in Figure 4 by way of the mentionned properties, done just for the evaluation of the predicate para(a,b,c,d) of Figure 3 under the hypotheses depicted at the head part of Figure 4, except $\wedge\neg para(a,b,c,d)$.

5 Conclusion

The aim of this paper was to introduce the needs for the integration of three components of the knowledge represntation for the geometrical shapes. The experiments done on a limited numbers of simples shapes are satisfactory. Exploration of its capabilities is in progress.

This paper partially is based on *Eshragh* and *Mamdani*'s idea [1]: the separation of fuzzy spreads into an appropriate number of segments with well defined characteristics. Fortunately, in many practical applications including continuous domains, the data collected in real-world experiment are discrete. Therefore, presumably, there are appropriate segments that are representative knowledge of the domain. If the data are not discrete, by symbolic constraints, the knowledge represntation can be conduted [3].

The following are among future problems to be studied.

- Representation of approximative references like `toward`, `from` , etc. (See Fig. 1). To capture the intuitive meanings of these relations we have to analyze the fundamental conceptual structure of the shapes in which using these references make senses.
- Representation of complex shapes by means of logical combination of the simple ones.
- Elaboration of set of spatial axioms (`Axioms`) for the practical spatial applications like earthquake engineering, soil classification, etc.

Acknowledgments

The authors would like appreciatvely thank the support, help and initiative of Professor `Goudarz Sadeghi`

`Heshjin` the president of the University of Mohaghegh Ardabili, for the creation of the `new` Institute for Research in Artificial Intelligence, where this work is done.

References

[1] F. Eshragh and E.H. Mamdani. *A General Approach to Linguistic Approximation.* In: Mamdani et al. (Eds.), Academic Press, Computer and people series, pp. 169–187, 1981.

[2] A. Fatholahzadeh. *Reasoning with Exact and Approximate References in Scene Description.* In: ASME, Book VI, Energy Information Management, Vol. I, Computer in Engineering, George Brown Convention Center, Houston, Texas Jan. 29 - Feb. 2, pp. 80–88, 1996.

[3] A. Fatholahzadeh. *Traitement et Représentation des Connaissances: Méthodes, Algorithmes et Programmes.* Volumes, I and II, CentraleSupélec, France, 1993, 2006.

[4] B. Kutzler. *Algebraic Approaches to Automated Geometry Theorem Proving.* Ph.D. thesis, University of Linz, Austria, 1988.

[5] T. Matsuyama and T. Nitta. *Geometric Theorem Proving by Integrated Logical and Algebraic* Artificial Intelligence, pp. 93–114, 1995.

[6] R. Mohr et al. *Understanding Positioning from Multiple Images.* Artificial Intelligence, pp. 213–238, 1995.

[7] D. Kapur and J.L. Mundy, editors. *Special Volume on Geometric Reasoning.* Artificial Intelligence, pp. 1–412, 1998.

[8] Steven A. Vere. *Induction of Concepts in the Predicate Calculus.* IJCAI, pp. 281–287, 1975.

Increasing-Chord Planar Graphs for Points in Convex Position

Abolfazl Poureidi* Davood Bakhshesh† Mohammad Farshi‡

Abstract

In this paper we propose an algorithm that computes an increasing-chord planar graph for finite sets of points in convex position in the plane with the geometric dilation less than $\pi/2$ and with no Steiner point.

Keywords: Geometric graph, increasing-chord planar graph, convex position.

1 Introduction

Alamdari et al. [1] have defined self-approaching graphs. A geometric path from u to v is self-approaching if while a point a traversing the path from u to v, for any point b between a and v on the path, the Euclidean distance between a and b decreases, that is, for any three points a, b and c on P in this order from u to v, we have $|ac| \geq |bc|$. An increasing-chord path between u and v is self-approaching both from u to v and from v to u. It is shown that the dilation of an increasing-chord path is at most 2.094 [3]. A graph G is increasing-chord if, for any pair of distinct vertices u and v of G, there is an increasing-chord path between them in G.

Dehkordi et al. [2] have proposed the following open problems. Is it true that, for every convex point set P, there exists an increasing-chord planar graph $G = (P, E)$ [2]? Is it true that, for every set P of points lying on the boundary of an acute triangle, there exists an increasing-chord planar graph $G = (P, E)$ [2]? In this paper, we design a polynomial-time algorithm that for any set S of points in convex position in the plane computes an increasing-chord planar graph whose vertex set consists only of S with the dilation less than $\pi/2$.

Due to page restrictions, some lemmas and proofs are omitted. There is a full version of the paper in the appendix.

2 Definitions and Preliminaries

Let p and q be two points in the plane. Let (p, q) be the line segment joining p and q. Let C_{pq} be the closed disc with diameter (p, q). Let $\overrightarrow{pq}$ denote the ray that emanates from p and then passes through q. For points $p, q, r \in V$, such that $(q, r) \in E$, say r is the clockwise neighbor of p with respect to q, denoted by $CN_q(p)$, if there is no $s \in N_G(q)$ between rays $\overrightarrow{qp}$ and $\overrightarrow{qr}$ when $\overrightarrow{qp}$ is rotated clockwise around q by a positive angle to coincide with $\overrightarrow{qr}$ (possibly $(p, q) \notin E$). See Figure 1(a).

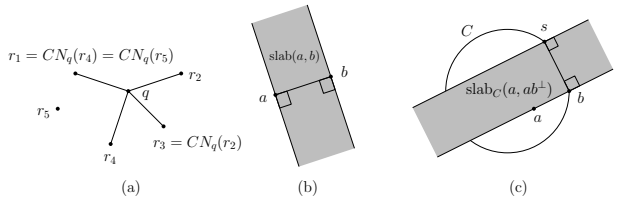


Figure 1: Illustrating (a) $CN_q(r_2)$, $CN_q(r_4)$ and $CN_q(r_5)$, when $N(q) = \{r_1, r_2, r_3, r_4\}$, (b) slab$(a, b)$, and (c) slab$_C(a, ab^{\perp})$.

Let S be a set of points in convex position in the plane, and let $CH(S)$ denote the convex hull of S. Let $P_c(p, q)$ and $P_{cc}(p, q)$ be the path between p and q passing through the boundary of $CH(S)$ in clockwise and in counterclockwise order from p to q, respectively. Let $S_c(p, q)$ and $S_{cc}(p, q)$ be the set of all points of S on $P_c(p, q)$ and on $P_{cc}(p, q)$, respectively, for some distinct points p and q in S. Say that P is a convex path between p_1 and p_n if for any edge (p_i, p_{i+1}) of P all the points of the path lie to the same side of the line through p_1 and p_n, for some $1 \leq i < n$.

Let a and b be points in the plane. Let slab(a, b) be the closed slab between a and b that is orthogonal to (a, b), see Figure 1(b). Let C be a closed disc in the plane. Assume that point b is on the boundary of C and point $a \in C$ is not on the diameter of C passing through b. Let s be the intersection point between the boundary of C and the line passing through b that is orthogonal to (a, b), that is, let s be a point on the boundary of C such that b is the orthogonal projection of s onto the line passing through a and b. We denote slab(b, s) by slab$_C(a, ab^{\perp})$, see Figure 1(c).

Lemma 1 *Let p and q be points in the plane. A convex path between p and q such that C_{pq} contains the whole path is increasing-chord with the dilation less than $\pi/2$.*

*Department of Applied Mathematics, Shahrood University of Technology, Shahrood, Iran, a.poureidi@shahroodut.ac.ir

†Department of Computer Science, University of Bojnord, Bojnord, Iran, dbakhshesh@gmail.com

‡Combinatorial and Geometric Algorithms Lab., Department of Computer Science, Yazd University, Yazd, Iran, mfarshi@yazd.ac.ir

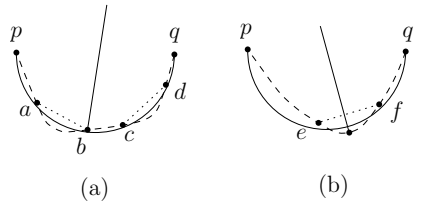


Figure 2: Constructing a planar convex path between p and q lying entirely in C_{pq} (a) it is possible to add (a, b) and (c, d) to the graph, but (b) if we add (e, f) to the graph, then the resulting graph is not planar.

3 Algorithm description

Let S be a finite set of points in convex position in the plane. In the following we describe an algorithm that computes an increasing-chord planar graph for S. The algorithm starts with a graph G having the vertex set S and the edge set E containing all edges on the boundary of the convex hull of S. Let L be the list of all the sorted pairs of distinct points of S in non-decreasing order of their distances, except pairs in E. Then, the algorithm considers all the pairs of L in non-decreasing order of their distances. For each pair $\{p, q\} \in L$, the algorithm first initializes both sets E_1 and E_2 to $\emptyset$, and then the algorithm considers to see whether it is possible to construct a convex path between p and q passing through some points of $S_c(p, q)$ (respectively, $S_{cc}(p, q)$) such that C_{pq} contains the whole path. To do this, the algorithm considers the shortest path between p and q in $G[S_c(p, q)]$ and the shortest path between p and q in $G[S_{cc}(p, q)]$. The algorithm starts at p (respectively, q) and then traverses the shortest path between p and q in $G[S_c(p, q)]$ (respectively, in $G[S_{cc}(p, q)]$) to reach q (respectively, p). While the shortest path between p and q in $G[S_c(p, q)]$ (respectively, in $G[S_{cc}(p, q)]$) lies in (the interior or on the boundary of) C_{pq}, the algorithm continues. (If the shortest path between p and q in $G[S_c(p, q)]$ or the shortest path between p and q in $G[S_{cc}(p, q)]$ lies entirely in C_{pq}, then the algorithm does nothing.) Assume that some part(s) of the shortest path between p and q in $G[S_{cc}(p, q)]$ (respectively, in $G[S_c(p, q)]$) are outside C_{pq}. Then, the algorithm checks whether it is possible to add some edge(s) to E_1 (respectively, to E_2) without crossing edges of E (except possibly at their endpoints) such that there is a convex path between p and q lying entirely in C_{pq}. See Figure 2. If it is possible, then the algorithm adds these edges to E_1 (respectively, to E_2), that is, if the algorithm adds an edge (a, b) to E_1 or E_2, then at least one point of S on the shortest path between a and b in $G[S_{cc}(p, q)]$ or $G[S_c(p, q)]$, respectively, is outside C_{pq}. See Figure 2(a). If both E_1 and E_2 are still the empty sets, then the algorithm adds (p, q) to E. Otherwise, if E_i, where $i = 1, 2$, is a nonempty set and the other set is an empty set or the following property does not hold for any $(a, b) \in E_i$, then the algorithm adds E_i to E.

Property 2 *(for edge (a, b)): There is a pair $\{u, v\} \in L$ with $u \in S_c(p, q)$, and $v \in S_{cc}(p, q)$, and $|pq| \leq |uv|$ such that both a and b are outside C_{uv}.*

The pseudocode of this algorithm is given in Algorithms 3.1 and 3.2. Let $(p, q) \in E$, where $\{p, q\} \in L$. As seen from Algorithm 3.1, there are two scenarios to add (p, q) to E. The *first scenario* that adds (p, q) to E occurs when Algorithm 3.2 on both inputs $(p, q, E_1(= \emptyset))$ and $(q, p, E_2(= \emptyset))$ returns the value $false$. So, Algorithm Increasing-Chord (in line #7) adds (p, q) to E. The *second scenario* that adds (p, q) to E occurs when Algorithm Find-Path on at least one of inputs $(p, q, E_1(= \emptyset))$ and $(q, p, E_2(= \emptyset))$ returns the value *true* and, therefore, Algorithm Increasing-Chord (in line #7) does not add (p, q) to E. Then, there is a pair $\{a, b\} \in L$, where $|pq| \leq |ab|$, such that just before the algorithm processes $\{a, b\}$, we have $(p, q) \notin E$, but when the processing of $\{a, b\}$ is completed, then (p, q) has been added to E. This occurs when Algorithm Find-Path on at least one of inputs $(a, b, E_1(= \emptyset))$ and $(b, a, E_2(= \emptyset))$ returns the value *true*. So, E_1 or E_2 is added to E in line #9 or #11 of Algorithm Increasing-Chord, respectively, where (p, q) is in E_1 or E_2. Clearly, both points p and q are in C_{ab}, that is, (p, q) lies entirely in C_{ab}, and both points a and b are outside slab(p, q).

4 Properties of the output of Algorithm 3.2

Lemma 3 *For any $\{p, q\} \in L$, Algorithm Find-Path on input $(p, q, E_1(= \emptyset))$ terminates.*

By Lemma 3, we obtain the following result.

Corollary 4 *Algorithm Increasing-Chord on input S terminates.*

Lemma 5 *Assume that just before processing $\{p, q\} \in L$ we have a planar graph $G = (S, E)$. If Algorithm Find-Path on input $(p, q, E_1(= \emptyset))$ returns the value true, then there is a convex path between p and q in $G = (S, E \cup E_1)$ such that C_{pq} contains the whole path and G is planar.*

Lemma 6 *Assume that just before processing $\{p, q\} \in L$ we have a planar graph $G = (S, E)$. Assume that there is (respectively, it is possible that we add some edges to E without crossing edges of E, except possibly at their endpoints, to construct) a convex path between p and q in $G[S_{cc}(p, q)]$ such that C_{pq} contains the whole path. Then, Algorithm Find-Path on input $(p, q, E_1(= \emptyset))$ returns the value true.*

Algorithm 3.1: INCREASING-CHORD(S)

Input: $S = \{p_1, \ldots, p_n\}$, where S is a finite set of points in convex position in the plane.
Output: An increasing-chord planar graph $G = (S, E)$.

1 $E := E(CH(S))$;
2 Sort the $\binom{n}{2}$ pairs of distinct points of S, except pairs of E, in non-decreasing order of their distances and store them in list L;
3 **foreach** $\{p, q\} \in L$ **do**
4 $\quad B_1 = B_2 := true$; $E_1 = E_2 := \emptyset$;
5 $\quad B_1 :=$FIND-PATH(p, q, E_1); $B_2 :=$FIND-PATH(q, p, E_2);
6 $\quad$ **if** $B_1 = B_2 = false$ **then**
7 $\quad\quad E := E \cup \{(p, q)\}$;
8 $\quad$ **else if** $(B_2 = false) \vee (\forall (a, b) \in E_1, \nexists \{u, v\} \in L$ *s.t.* $u \in S_c(p, q),\ v \in S_{cc}(p, q),\ |pq| \leq |uv|,\ a, b \notin C_{uv})$ **then**
9 $\quad\quad E := E \cup E_1$;
10 $\quad$ **else**
11 $\quad\quad E := E \cup E_2$;
12 $\quad$ **end**
13 **end**
14 **return** $G := (S, E)$;

Algorithm 3.2: FIND-PATH(p, q, E)

Input: A pair $\{p, q\}$ of distinct points in S, where S is a finite set of points in convex position in the plane.
Output: Either the value *true* and a (possibly empty) set of edges between points of S or the value *false* and an empty set.

1 $E' = \emptyset$; $r := CN_p(q)$; $a := p$; $b := r$;
2 **while** $b \neq q$ **do**
3 $\quad$ **if** $(b \in C_{pq}) \wedge (((p, q) \cap (a, b)) - \{p\} = \emptyset)$ **then**
4 $\quad\quad r := CN_b(a)$; $a := b$; $b := r$;
5 $\quad$ **else if** $(b \notin C_{pq}) \wedge (((p, q) \cap (a, b)) - \{p\} = \emptyset)$ **then**
6 $\quad\quad n_1 := a$; $r := CN_b(a)$;
7 $\quad\quad$ **while** $(r \notin C_{pq}) \wedge (((p, q) \cap (b, r)) = \emptyset)$ **do**
8 $\quad\quad\quad a := b$; $b := r$; $r := CN_b(a)$;
9 $\quad\quad$ **end**
10 $\quad\quad$ **if** $((p, q) \cap (b, r)) - \{q\} \neq \emptyset$ **then**
11 $\quad\quad\quad$ **return** *false*;
12 $\quad\quad$ **else if** $r \neq q$ **then**
13 $\quad\quad\quad E' := E' \cup \{(n_1, r)\}$; $b := CN_r(n_1)$;
14 $\quad\quad$ **else if** $n_1 = p$ **then**
15 $\quad\quad\quad$ **return** *false*;
16 $\quad\quad$ **else**
17 $\quad\quad\quad E' := E' \cup \{(n_1, q)\}$; $b := q$;
18 $\quad\quad$ **end**
19 $\quad$ **else if** $b \neq q$ **then**
20 $\quad\quad b := CN_a(b)$;
21 $\quad$ **end**
22 **end**
23 $E = E'$; **return** *true*;

Lemma 7 *Assume that just before processing $\{p, q\} \in L$ we have a planar graph $G = (S, E)$. Assume that Algorithm* FIND-PATH *on inputs $(p, q, E_1(= \emptyset))$ and $(q, p, E_2(= \emptyset))$ returns $E_1 \neq \emptyset$ and $E_2 \neq \emptyset$, respectively. There are no pairs $(a_1, b_1) \in E_1$ and $(a_2, b_2) \in E_2$ such that Property 2 holds for both (a_1, b_1) and (a_2, b_2), when the algorithm takes as input a finite set of points in convex position in the plane.*

As seen from Algorithm 3.1, if the condition in line #6 does not hold, then $B_1 = true$ or $B_2 = true$. If exactly one of variables B_1 and B_2 has the value $false$, then the condition in line #8 or #10, respectively, holds, otherwise $B_1 = B_2 = true$. By Lemma 7, it is impossible to hold Property 2 for an edge of E_1 and for an edge of E_2 when the algorithm processes one pair of L. So, we obtain the following result.

Corollary 8 *Assume that just before processing $\{p, q\} \in L$ we have a planar graph $G = (S, E)$. If Algorithm* FIND-PATH *on at least one of inputs $(p, q, E_1(= \emptyset))$ and $(q, p, E_2(= \emptyset))$ returns the value true, then the condition in line #8 or #10 of Algorithm* INCREASING-CHORD *holds.*

5 Properties of edges added by the first scenario

Lemma 9 *Let (p, q) be in E, where $\{p, q\} \in L$. Assume that just before processing $\{p, q\}$ we have a planar graph and there is $(a, b) \in E$ (with $|ab| \leq |pq|$) such that has a nonempty intersection with (p, q), except possibly at their endpoints. If the first scenario occurs for (p, q), then at least one of points a and b is in C_{pq}.*

Lemma 10 *Assume that just before adding (p, q) to E, where $\{p, q\} \in L$, we have a planar graph and there are at least two edges (a, b) and (c, d) of E (with lengths at most $|pq|$) such that points a and c are on $P_c(p, q)$, and points b and d are on $P_{cc}(p, q)$, and (a, b) has a nonempty intersection with (p, q), and (c, d) has a nonempty intersection with (p, q), except possibly at their endpoints. If $a, d \in C_{pq}$ and $b, c \notin C_{pq}$, then the first scenario cannot occur for (p, q).*

6 An increasing-chord planar graph returned by Algorithm 3.1

Lemma 11 *The output of Algorithm 3.1 on input S is a planar graph and there is a convex path in the output graph between any two distinct points $p, q \in S$ lying entirely in C_{pq}, when S is a finite set of points in convex*

position in the plane.

Proof. Clearly, any edge on the boundary of $CH(S)$ does not intersect the edge between any two points of S, except possibly at their endpoints, and for all edges on the boundary of $CH(S)$ the claim in the lemma holds. For distinct points $p, q \in S$, where (p, q) is no edge on the boundary of $CH(S)$, we prove the claim in the lemma by induction on the rank of $\{p, q\}$ in L, where L is the list of the $\binom{n}{2}$ sorted pairs of distinct points of S in non-decreasing order of their distances, except pairs that are endpoints of edges on the boundary of the convex hull of S.

Base case: Let $\{r, s\}$ be the first pair in L. If the first scenario occurs for (r, s), then clearly the resulting graph is planar and there is a convex path in the graph between points r and s lying entirely in C_{rs}. Otherwise, Algorithm FIND-PATH on at least one of inputs $(r, s, E_1(= \emptyset))$ and $(s, r, E_2(= \emptyset))$ (in line #5) returns the value *true*. By Corollary 8 at least one of the conditions in lines #8 and #10 of Algorithm INCREASING-CHORD holds. So, Algorithm INCREASING-CHORD (in line #9 or #11) adds E_1 or E_2, respectively, to E. By Lemma 5 the resulting graph is planar and there is a convex path between r and s in the graph such that C_{rs} contains the whole path.

Induction hypothesis: Assume that just before Algorithm INCREASING-CHORD processes $\{p, q\}$, where $|rs| \leq |pq|$, the graph $G = (S, E)$ is planar and there is a convex path between points of any pair $\{a, b\} \in L$ with $|ab| \leq |pq|$ such that C_{ab} contains the whole path.

The inductive step: Suppose that the algorithm processes $\{p, q\}$. There are two possible cases to consider, depending on whether or not (p, q) is added to E in line #7 of Algorithm INCREASING-CHORD.

Case 1: Edge (p, q) is not added to E in line #7 of Algorithm INCREASING-CHORD.

Similar to the base case the resulting graph is planar and there is a convex path between p and q in the graph such that C_{pq} contains the whole path.

Case 2: Edge (p, q) is added to E in line #7 of Algorithm INCREASING-CHORD, that is, the first scenario occurs for (p, q). Clearly, there is a convex path between p and q in the graph lying entirely in C_{pq}. We claim that any edge of E has an empty intersection with (p, q), except possibly at their endpoints.

Assume $(r_1, s_1), (r_2, s_2), \ldots, (r_k, s_k) \in E$, where $k \geq 1$, are all edges of E that have a nonempty intersection with (p, q), except possibly at their endpoints. Assume without loss of generality that each r_i, for $1 \leq i \leq k$, is on $P_c(p, q)$ with $|P_c(p, r_1)| \leq |P_c(p, r_2)| \leq \ldots \leq |P_c(p, r_k)|$. (Since just before processing $\{p, q\}$ we have a planar graph, each s_i is on $P_{cc}(p, q)$ with $|P_{cc}(p, s_1)| \leq |P_{cc}(p, s_2)| \leq \ldots \leq |P_{cc}(p, s_k)|$.) See Figure 3.

By Lemma 9, at least one of points r_i and s_i is in (the interior or on the boundary of) C_{pq}, for $1 \leq i \leq k$. We need only to consider two cases: (i) for all $1 \leq i \leq k$, every r_i is in C_{pq} or every s_i is in C_{pq} and (ii) there are edges (r_j, s_j) and (r_l, s_l) with $1 \leq j, l \leq k$ such that $r_j, s_l \in C_{pq}$ and $s_j, r_l \notin C_{pq}$. (Assume $(r_j, s_j) = (a, b)$ and $(r_j, s_j) = (c, d)$.) In the former case assume without loss of generality that all points $s_1, s_2, \ldots, s_k$ are in C_{pq}. By assumption we have a planar graph just before adding (p, q) to E. So, there is (respectively, it is possible that we add some edges to E without crossing edges of E, except possibly at their endpoints, to construct) a convex path between p and q passing through points $s_1, s_2, \ldots, s_k$ such that C_{pq} contains the whole path. As an example, consider path $(p, s_1, s_2, \ldots, s_k, q)$. It follows from Lemma 6 that Algorithm FIND-PATH on input $(p, q, E_1(= \emptyset))$ returns the value *true*, i.e., Algorithm INCREASING-CHORD (in line #7) does not add (p, q) to E. So, the first scenario cannot occur for (p, q), a contradiction. By Lemma 10 the latter case never occurs. Hence, the resulting graph is planar. □

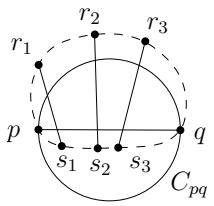


Figure 3: For proof of Lemma 11.

Clearly, the running time of the algorithm of this paper is polynomial in $|S|$. So, by Lemmas 1 and 11 we get the following result.

Theorem 12 *There is a polynomial-time algorithm that computes an increasing-chord planar graph for a set of points in convex position in the plane with no Steiner point and with the dilation less than $\pi/2$.*

References

[1] S. Alamdari, T. M. Chan, E. Grant, A. Lubiw, and V. Pathak. Self-approaching graphs. In *International Symposium on Graph Drawing*, pages 260–271. Springer, 2012.

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

[3] G. Rote. Curves with increasing chords. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 115, pages 1–12. Cambridge Univ Press, 1994.

Appendix

Abstract

In this paper we propose an algorithm that computes an increasing-chord planar graph for finite sets of points in convex position in the plane with the geometric dilation less than $\pi/2$ and with no Steiner point.

Keywords: Geometric graph, increasing-chord planar graph, convex position.

1 Introduction

A geometric graph is a weighted graph on a point set in $\mathbb{R}^d$ as the vertex set that has straight-line segments between vertices as the edge set, where the weight of an edge is the Euclidean distance between its endpoints. A straight-line drawing of a graph G in $\mathbb{R}^d$ is an embedding of G as a geometric graph in $\mathbb{R}^d$.

Rao et al. [14] have introduced the notion of greedy graph drawings. Geometric path $(p_1, p_2, \ldots, p_n)$ is a greedy path from p_1 to p_n if $|p_{i+1}p_n| < |p_ip_n|$, for all $1 \leq i < n$, where $|pq|$ denotes the Euclidean distance between points p and q. A geometric graph is greedy if for any ordered pair of distinct vertices u and v of the graph there is a greedy path from u to v in the graph. Greedy graph drawings have received a lot of attention in recent years [3, 4, 6, 7, 8, 10, 13]. The main motivation for studying greedy graph drawings is their application in a wide variety of ad hoc and sensor-net environments [14]. It is possible to have greedy graph drawings with an unbounded dilation, where the dilation of a geometric graph is the maximum of the ratio of the shortest path distance between every pair of distinct vertices in the graph to their actual Euclidean distance.

Alamdari et al. [1] have defined self-approaching graphs that not only have the properties of greedy graph drawings but also their geometric dilation is at most 5.3332 [9]. A geometric path from u to v is self-approaching if while a point a traversing the path from u to v, for any point b between a and v on the path (possibly b is an intermediate point on segments of the path), the Euclidean distance between a and b decreases, that is, for any three points a, b and c (not necessarily vertices) on P in this order from u to v, the inequality $|ac| \geq |bc|$ holds. An increasing-chord path between u and v is self-approaching both from u to v and from v to u. It is shown that the geometric dilation of an increasing-chord path is at most 2.094 [15]. A geometric graph G is increasing-chord (respectively, self-approaching) if, for any (respectively, ordered) pair of distinct vertices u and v of G, there is an increasing-chord path between u and v (respectively, a self-approaching path from u to v) in G.

Alamdari et al. [1] have designed polynomial-time algorithms to test whether a given path in the plane or $\mathbb{R}^3$ is self-approaching and Alamdari et al. [1] have also shown that recognizing self-approaching graph drawings in $\mathbb{R}^3$ is NP-hard. Alamdari et al. [1] have completely characterized trees that have self-approaching drawings. Furthermore, Alamdari et al. [1], when given a point set S in $\mathbb{R}^d$, have designed a polynomial-time algorithm to compute a linear sized self-approaching graph spanning S with $\mathcal{O}(|S|)$ Steiner points.

Mastakas and Symvonis [11] have designed a polynomial-time algorithm that for any set S of points in convex position in the plane computes a linear sized increasing-chord graph with at most one Steiner point spanning S.

Nöllenburg et al. [12] have proved that there is an increasing-chord drawing for planar triangulations and there is an increasing-chord planar drawing for planar 3-trees. Furthermore, Nöllenburg et al. [12] have completely characterized trees that have self-approaching or increasing-chord drawings in the hyperbolic plane.

Dehkordi et al. [5] have proposed a polynomial-time algorithm that, when given a point set S in the plane, computes an increasing-chord planar graph whose vertex set consists of S and $\mathcal{O}(|S|)$ Steiner points. Furthermore, Dehkordi et al. [5] have proved that, when given a set S of points in convex position in the plane, there is an increasing-chord graph with S as the vertex set and $\mathcal{O}(|S| \log |S|)$ edges.

Dehkordi et al. [5] have also proposed the following open problems. Is it true that, for every (convex) point set P, there exists an increasing-chord planar graph $G = (P, E)$ [5]? Is it true that, for every set P of points lying on the boundary of an acute triangle, there exists an increasing-chord planar graph $G = (P, E)$ [5]? In this paper, we design a polynomial-time algorithm that for any set S of points in convex position in the plane computes an increasing-chord planar graph whose vertex set consists only of S (i.e., with no Steiner point) with the geometric dilation less than $\pi/2$ answering two open problems of Dehkordi et al. [5]. Note that the best know geometric dilation of planar spanners for sets of points in convex position in the plane is 1.88 [2]. So, the algorithm also improves the geometric dilation of planar spanners for sets of points in convex position in the plane.

The organization of the paper is as follows. In Section 2 we begin with some preliminaries. In Section 3 we describe the algorithm and give the pseudocode of the algorithm (Algorithms 3.1 and 3.2). In Section 4 we state some properties of the output graph of the algorithm (i.e., Algorithm 3.2), when the algorithm (i.e., Algorithm 3.1) takes as input a set of points in convex position in the plane. In section 5 we state some properties for some edges of the output graph. In Section 6 we prove that the output of the algorithm for any set of points in convex position in the plane is an increasing-chord planar graph with the geometric dilation less than $\pi/2$. Finally, in Section 7 we conclude and close with some open problems.

2 Definitions and Preliminaries

Let p, q and r be three distinct points in the plane. Let (p, q) be the straight-line segment joining p and q, and let $|pq|$ denote the Euclidean distance between p and q. Let l_{pq} denote the line passing through p that is orthogonal to the line segment (p, q). Let C_{pq} be the closed disc with diameter (p, q). Let $\overrightarrow{pq}$ denote the ray that emanates from p and then passes through q, and let $ray(p, \overrightarrow{qr})$ denote the ray that emanates from p and is co-directed with $\overrightarrow{qr}$. We say that point p is the source of rays $\overrightarrow{pq}$ and $ray(p, \overrightarrow{qr})$. Let $\angle pqr$ denote the angle between rays $\overrightarrow{qp}$ and $\overrightarrow{qr}$. Let $\overrightarrow{r_1}$ and $\overrightarrow{r_2}$ be two distinct rays in the plane. The angle between $\overrightarrow{r_1}$ and $\overrightarrow{r_2}$ is the angle between translated rays $\overrightarrow{r_1}$ and $\overrightarrow{r_2}$

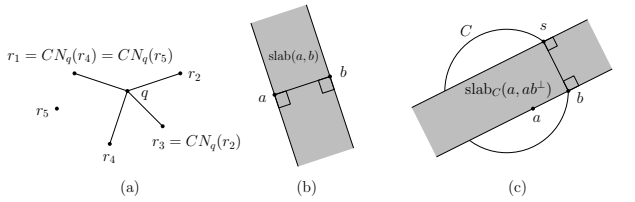


Figure 1: Illustrating (a) $CN_q(r_2)$, $CN_q(r_4)$ and $CN_q(r_5)$, when $N(q) = \{r_1, r_2, r_3, r_4\}$, (b) slab$(a, b)$, and (c) slab$_C(a, ab^{\perp})$.

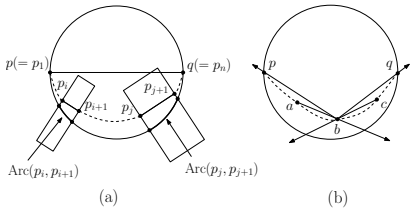


Figure 2: For proofs of (a) proposition 1 and (b) Lemma 2.

such that their sources are at the origin. Let $G = (V, E)$ be a graph, and for $p \in V$ let $N_G(p)$ denote the set of all incident vertices to p in G. For any $V' \subseteq V$ let $G[V']$ be a subgraph of G whose vertex set is V' and whose edge set consists of all edges in E that have both endpoints in V'. For points $p, q, r \in V$, such that $(q, r) \in E$, say r is the clockwise neighbor of p with respect to q, denoted by $CN_q(p)$, if there is no $s \in N_G(q)$ between rays $\overrightarrow{qp}$ and $\overrightarrow{qr}$ when $\overrightarrow{qp}$ is rotated clockwise around q by a positive angle to coincide with $\overrightarrow{qr}$ (possibly $(p, q) \notin E$). See Figure 1(a).

Let S be a set of points in convex position in the plane, and let $CH(S)$ denote the convex hull of S. Let $P_c(p, q)$ and $P_{cc}(p, q)$ be the path between p and q passing through the boundary of $CH(S)$ in clockwise and in counterclockwise order from p to q, respectively. Let $S_c(p, q)$ and $S_{cc}(p, q)$ be the set of all points of S on $P_c(p, q)$ and on $P_{cc}(p, q)$, respectively, for some distinct points p and q in S. Clearly, we have $S_c(p, q) \cup S_{cc}(p, q) = S$ and $S_c(p, q) \cap S_{cc}(p, q) = \{p, q\}$. Let $P = (p_1, p_2, \ldots, p_n)$ be a path between p_1 and p_n in the plane for some integer $n > 1$. The length of path P, denoted by $|P|$, is defined as the sum of the lengths of its edges. Say that P is a convex path between p_1 and p_n if for any edge (p_i, p_{i+1}) of P all the points of the path lie to the same side of the line through p_1 and p_n, for some $1 \leq i < n$.

Let a and b be points in the plane. Let slab(a, b) be the closed slab between a and b that is orthogonal to (a, b), see Figure 1(b). Let C be a closed disc in the plane. Assume that point b is on the boundary of C and point $a \in C$ is not on the diameter of C passing through b. Let s be the intersection point between the boundary of C and the line passing through b that is orthogonal to the line segment (a, b), that is, let s be a point on the boundary of C such that b is the orthogonal projection of s onto the line passing through a and b. We denote slab(b, s) by slab$_C(a, ab^{\perp})$, see Figure 1(c).

Proposition 1 *Let p and q be points in the plane. The geometric dilation of any convex path between p and q such that C_{pq} contains the whole path is less than $\pi/2$.*

Proof. Let $P = (p_1(= p), p_2, \ldots, p_n(= q))$, for some integer $n > 2$, be a convex path between p and q that C_{pq} contains the whole path. Assume (without loss of generality) that points p and q are on a horizontal line and p is to the left of q, and all points of P are not above (p, q). See Figure 2(a). Let Arc(p, q) denote the lower boundary of C_{pq} between points p and q. Let Arc(p_i, p_{i+1}) denote the intersection between slab(p_i, p_{i+1}) and Arc(p, q), for all $1 \leq i < n$. It is easy to see that the length of Arc(p_i, p_{i+1}) is less than $|p_i p_{i+1}|$. Clearly, for any two distinct indices $1 \leq i, j < n$ we have Arc$(p_i, p_{i+1}) \cap$Arc$(p_j, p_{j+1}) \subseteq \{u\}$, where u is either p_i or p_j. So, $|P|$ is less than the length of Arc(p, q). This completes the proof. □

Lemma 2 *Let p and q be points in the plane. A convex path between p and q such that C_{pq} contains the whole path is increasing-chord with the geometric dilation less than $\pi/2$.*

Proof. Let $P = (p_1(= p), p_2, \ldots, p_n(= q))$, for some integer $n > 2$, be a convex path between p and q that C_{pq} contains the whole path. Consider any three distinct points a, b, and c on P in this order from p to q. Clearly, points a, b, and c are in C_{pq}. Since P is a convex path between p and q, point a is between or on rays $\overrightarrow{bp}$ and $ray(b, \overrightarrow{cb})$ and point c is between or on rays $\overrightarrow{bq}$ and $ray(b, \overrightarrow{ab})$. See Figure 2. So, we get $\angle abc \geq \pi/2$ and, therefore, $|ac| > |bc|$ and $|ac| > |ab|$. Therefore, P is an increasing-chord path between p and q. By Proposition 1, the geometric dilation of P is less than $\pi/2$. This completes the proof. □

3 Algorithm description

Let S be a finite set of points in convex position in the plane. In the following we describe an algorithm that computes an increasing-chord planar graph for S. The algorithm starts with a graph G having the vertex set S and the edge set E containing all edges on the boundary of the convex hull of S. Let L be the list of all the sorted pairs of distinct points of S in non-decreasing order of their distances, except pairs in E. Then, the algorithm considers all the pairs of L in non-decreasing order of their distances. For each pair $\{p, q\} \in L$, the algorithm first initializes both sets E_1 and E_2 to $\emptyset$, and then the algorithm considers to see whether it is possible to construct a convex path between p and q passing through some points of $S_c(p, q)$ (respectively, $S_{cc}(p, q)$) such that C_{pq} contains the whole path. To do this, the algorithm considers the shortest path between p and q in $G[S_c(p, q)]$ and the shortest path between p and q in $G[S_{cc}(p, q)]$. The algorithm starts at p (respectively, q) and then traverses the shortest path between p and q in $G[S_c(p, q)]$ (respectively, in $G[S_{cc}(p, q)]$) to reach q (respectively, p). While the shortest path between p and q in $G[S_c(p, q)]$ (respectively, in $G[S_{cc}(p, q)]$) lies in (the interior or on the boundary of) C_{pq}, the algorithm continues. (If the shortest path between

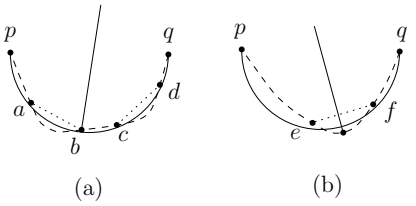


Figure 3: Constructing a planar convex path between p and q lying entirely in C_{pq} (a) it is possible to add (a, b) and (c, d) to the graph, but (b) if we add (e, f) to the graph, then the resulting graph is not planar.

p and q in $G[S_c(p, q)]$ or the shortest path between p and q in $G[S_{cc}(p, q)]$ lies entirely in C_{pq}, then the algorithm does nothing.) Assume that some part(s) of the shortest path between p and q in $G[S_{cc}(p, q)]$ (respectively, in $G[S_c(p, q)]$) are outside C_{pq}. Then, the algorithm checks whether it is possible to add some edge(s) to E_1 (respectively, to E_2) without crossing edges of E (except possibly at their endpoints) such that there is a convex path between p and q lying entirely in C_{pq}. See Figure 3. If it is possible, then the algorithm adds these edges to E_1 (respectively, to E_2), that is, if the algorithm adds an edge (a, b) to E_1 or E_2, then at least one point of S on the shortest path between a and b in $G[S_{cc}(p, q)]$ or $G[S_c(p, q)]$, respectively, is outside C_{pq}. See Figure 3(a). If both E_1 and E_2 are still the empty sets, then the algorithm adds (p, q) to E. Otherwise, if E_i, where $i = 1, 2$, is a nonempty set and the other set is an empty set or the following property does not hold for any $(a, b) \in E_i$, then the algorithm adds E_i to E.

Property 3 *(for edge (a, b)): There is a pair $\{u, v\} \in L$ with $u \in S_c(p, q)$, and $v \in S_{cc}(p, q)$, and $|pq| \leq |uv|$ such that both a and b are outside C_{uv}.*

The pseudocode of this algorithm is given in Algorithms 3.1 and 3.2. Let $(p, q) \in E$, where $\{p, q\} \in L$. As seen from Algorithm 3.1 (i.e., Algorithm Increasing-Chord), there are two scenarios to add (p, q) to E. The *first scenario* that adds (p, q) to E occurs when Algorithm 3.2 (i.e., Algorithm Find-Path) on both inputs $(p, q, E_1(= \emptyset))$ and $(q, p, E_2(= \emptyset))$ returns the value $false$. So, Algorithm Increasing-Chord (in line #7) adds (p, q) to E. The *second scenario* that adds (p, q) to E occurs when Algorithm Find-Path on at least one of inputs $(p, q, E_1(= \emptyset))$ and $(q, p, E_2(= \emptyset))$ returns the value *true* and, therefore, Algorithm Increasing-Chord (in line #7) does not add (p, q) to E. Then, there is a pair $\{a, b\} \in L$, where $|pq| \leq |ab|$, such that just before the algorithm processes $\{a, b\}$, we have $(p, q) \notin E$, but when the processing of $\{a, b\}$ is completed, then (p, q) has been added to E. This occurs when Algorithm Find-Path on at least one of inputs $(a, b, E_1(= \emptyset))$ and $(b, a, E_2(= \emptyset))$ returns the value *true*. So, E_1 or E_2 is added to E in line #9 or #11 of Algorithm Increasing-Chord, respectively, where (p, q) is in E_1 or E_2. Clearly, both points p and q are in (the interior or on the boundary of) C_{ab}, that is, (p, q) lies entirely in C_{ab}, and both points a and b are outside slab(p, q).

Algorithm 3.1: Increasing-Chord(S)

Input: $S = \{p_1, \ldots, p_n\}$, where S is a finite set of points in convex position in the plane.
Output: An increasing-chord planar graph $G = (S, E)$.

```
1  E := E(CH(S));        /* initialize E to the edges of CH(S) */
2  Sort the (n choose 2) pairs of distinct points of S, except pairs of E, in non-decreasing order of their distances and store them in list L;
3  foreach {p, q} ∈ L do
4      B1 = B2 := true; E1 = E2 := ∅;
5      B1 := Find-Path(p, q, E1);
       B2 := Find-Path(q, p, E2);
6      if B1 = B2 = false then
7          E := E ∪ {(p, q)};
8      else if (B2 = false) ∨ (∀(a, b) ∈ E1, ∄{u, v} ∈ L s.t. u ∈ Sc(p, q), v ∈ Scc(p, q), |pq| ≤ |uv|, a, b ∉ Cuv) then
9          E := E ∪ E1;
10     else if (B1 = false) ∨ (∀(a, b) ∈ E2, ∄{u, v} ∈ L s.t. u ∈ Sc(p, q), v ∈ Scc(p, q), |pq| ≤ |uv|, a, b ∉ Cuv) then
11         E := E ∪ E2;
12     end
13 end
14 return G := (S, E);
```

4 Properties of the output of Algorithm 3.2

Recall that S is a finite set of points in convex position in the plane. Also recall that L is the list of the $\binom{n}{2}$ sorted pairs of distinct points of S, except pairs that are endpoints of edges on the boundary of $CH(S)$, in non-decreasing order of their distances. Let $G = (S, E)$ be the output graph of Algorithm Increasing-Chord on input S. In this section we first prove that for any $\{p, q\} \in L$ Algorithm Find-Path on input $(p, q, E_1(= \emptyset))$ terminates. It follows that Algorithm Increasing-Chord on input S terminates. Next, we prove that Algorithm Find-Path adds edges to E_1 to construct an increasing-chord path between p and q in $G = (S, E \cup E_1)$ if it is possible, otherwise, returns the value $false$. Let $E_1 \neq \emptyset$, and assume that $(a, b) \in E_1$ with $|P_{cc}(p, a)| < |P_{cc}(p, b)|$. Then, we prove that all points on $P_{cc}(a, b)$, except points a and b, are outside C_{pq}. Finally, we prove that if Algorithm Find-Path on inputs $(p, q, E_1(= \emptyset))$ and $(q, p, E_2(= \emptyset))$ returns $E_1 \neq \emptyset$ and $E_2 \neq \emptyset$, respectively, then there are no pairs $(a_1, b_1) \in E_1$ and $(a_2, b_2) \in E_2$ such that Property 3 holds for both (a_1, b_1) and (a_2, b_2).

Lemma 4 *For any $\{p, q\} \in L$, Algorithm* Find-Path *on input $(p, q, E_1(= \emptyset))$ terminates.*

Proof. Clearly, if Algorithm Find-Path on input $(p, q, E_1(= \emptyset))$ (in line #11 or #15) executes the instruction "**return** $false$", then Algorithm Find-Path on input $(p, q, E_1(= \emptyset))$ terminates. So, we assume that Algorithm

Algorithm 3.2: FIND-PATH(p, q, E)

Input: A pair $\{p, q\}$ of distinct points in S, where S is a finite set of points in convex position in the plane.

Output: Either the value *true* and a (possibly empty) set of edges between points of S or the value *false* and an empty set.

```
1  E' = ∅; r := CN_p(q); a := p; b := r;
2  while b ≠ q do
3  |  if (b ∈ C_pq) ∧ (((p,q) ∩ (a,b)) − {p} = ∅) then
   |   /* b is on P_cc(p,q)                                  */
4  |  |  r := CN_b(a); a := b; b := r;
5  |  else if (b ∉ C_pq) ∧ (((p,q) ∩ (a,b)) − {p} = ∅)
   |   then/* b is on P_cc(p,q)                              */
6  |  |  n_1 := a; r := CN_b(a);
7  |  |  while (r ∉ C_pq) ∧ (((p,q) ∩ (b,r)) = ∅) do
   |  |   /* r is on P_cc(p,q)                               */
8  |  |  |  a := b; b := r; r := CN_b(a);
9  |  |  end
10 |  |  if ((p,q) ∩ (b,r)) − {q} ≠ ∅ then /* b ∉ C_pq
   |  |   is on P_cc(p,q) and r(≠ p,q) is on
   |  |   P_c(p,q)                                           */
11 |  |  |  return false;
12 |  |  else if r ≠ q then
13 |  |  |  E' := E' ∪ {(n_1, r)}; b := CN_r(n_1);
14 |  |  else if n_1 = p then/* All points
   |  |   P_cc(p,q), except points p and q, are
   |  |   outside C_pq                                       */
15 |  |  |  return false;
16 |  |  else
17 |  |  |  E' := E' ∪ {(n_1, q)}; b := q;
18 |  |  end
19 |  else if b ≠ q then /* a ∈ C_pq is on P_cc(p,q)
   |   and b is on P_c(p,q)                                  */
20 |  |  b := CN_a(b);
21 |  end
22 end
23 E = E';
24 return true;
```

FIND-PATH on input $(p, q, E_1(= \emptyset))$ does not execute the instruction "**return** *false*".

In the rest of the proof we compute a sequence of points in $S_{cc}(p, q)$. Elements of the sequence are points of $S_{cc}(p, q)$ which are generated by functions $CN_a(b)$, for some points a and b, in lines #1, #4, #6, #8, #13, and #20 of Algorithm FIND-PATH on input $(p, q, E_1(= \emptyset))$. We place these points in the sequence in the order that they are generated by Algorithm FIND-PATH on input $(p, q, E_1(= \emptyset))$. Next, we prove that the sequence is finite. This completes the proof.

Let $q_1 := p$ and $q_2 := CN_p(q)$, where $CN_p(q)$ is computed in line #1 of Algorithm FIND-PATH. Since $\{p, q\} \in L$, we have $q_2 \neq q$. It is easy to see that both q_1 and q_2 lie on $P_{cc}(p, q)$ and we have $|P_{cc}(q_1, q)| > |P_{cc}(q_2, q)|$. Since point q_2 lies on $P_{cc}(p, q)$, the condition in line #3 or #5 of Algorithm FIND-PATH holds. Consider $CN_{q_2}(q_1)$, where $CN_{q_2}(q_1)$ is computed in line #4 or #6 of Algorithm FIND-PATH. (It is easy to see that when $CN_{q_2}(q_1) = q$, then Algorithm FIND-PATH on input $(p, q, E_1(= \emptyset))$ terminates.) It is possible that $CN_{q_2}(q_1)$ lies on $P_c(p, q)$ or on $P_{cc}(p, q)$. If the condition in line #3, #5, #7, or #12 of Algorithm FIND-PATH holds for $CN_{q_2}(q_1)$, then $CN_{q_2}(q_1)$ lies on $P_{cc}(p, q)$ and if the condition in line #19 of Algorithm FIND-PATH holds for $CN_{q_2}(q_1)$, then $CN_{q_2}(q_1)$ lies on $P_c(p, q)$. (It is easy to see that when the condition in line #16 of Algorithm FIND-PATH holds for $CN_{q_2}(q_1)$, then Algorithm FIND-PATH on input $(p, q, E_1(= \emptyset))$ terminates.) If $CN_{q_2}(q_1)$ lies on $P_{cc}(p, q)$, then we define $q_3 := CN_{q_2}(q_1)$.

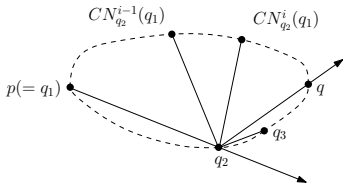


Figure 4: For proof of Lemma 4.

Suppose that $CN_{q_2}(q_1)$ lies on $P_c(p, q)$. We define $CN^1_a(b) := CN_a(b)$, and $CN^i_a(b) := CN_a(CN^{i-1}_a(b))$, where a and b are points in the plane and $i \geq 2$. It is easy to see that if both $CN^{i-1}_{q_2}(q_1)$ and $CN^i_{q_2}(q_1)$ lie on $P_c(p, q)$, then we have $|P_c(CN^{i-1}_{q_2}(q_1), q)| > |P_c(CN^i_{q_2}(q_1), q)|$, see Figure 4. Hence, there is an integer $2 \leq j < |N_G(q_2)|$ such that $CN^j_{q_2}(q_1)$ lies on $P_{cc}(p, q)$ and, for all $i < j$, point $CN^i_{q_2}(q_1)$, where $CN^i_{q_2}(q_1) \neq q$, lies on $P_c(p, q)$. (It is easy to see that when $CN^j_{q_2}(q_1) = q$, then Algorithm FIND-PATH on input $(p, q, E_1(= \emptyset))$ terminates.) Thus, we define $q_3 := CN^j_{q_2}(q_1)$. It is clear that q_3 is between or on rays $ray(q_2, \overrightarrow{q_1 q_2})$ and $\overrightarrow{q_2 q}$, see Figure 4. Hence, we have $|P_{cc}(q_2, q)| > |P_{cc}(q_3, q)|$. (The condition in line #19 of Algorithm FIND-PATH holds for $CN^i_{q_2}(q_1)$, when $i < j$.)

Then, Algorithm FIND-PATH on input (p, q, E_1) computes $CN_{q_3}(q_2)$ in line #4, #6, #8, #13, or #20. (It is easy to see that when $CN_{q_3}(q_2) = q$, then Algorithm FIND-PATH on input (p, q, E_1) terminates.) If $CN_{q_3}(q_2)$ lies on $P_{cc}(p, q)$, then we define $q_4 := CN_{q_3}(q_2)$. Otherwise, there is an integer j, where $2 \leq j < |N_G(q_3)|$, such that $CN^j_{q_3}(q_2)$ lies on $P_{cc}(p, q)$ and, for all $i < j$, point $CN^i_{q_3}(q_2)$, where $CN^i_{q_3}(q_2) \neq q$, lies on $P_c(p, q)$. Thus, we define $q_4 := CN^j_{q_3}(q_2)$. It is easy to see that $|P_{cc}(q_3, q)| > |P_{cc}(q_4, q)|$. Similarly, we define q_k for $k \geq 5$. We have $|P_{cc}(q_{k-1}, q)| > |P_{cc}(q_k, q)|$.

Let $P := (q_1, q_2, q_3, \ldots)$. Any q_i lies on $P_{cc}(p, q)$ and we have $|P_{cc}(q_i, q)| > |P_{cc}(q_{i+1}, q)|$, for all $i \geq 1$. It follows that all points $q_1, q_2, \ldots$ are pairwise distinct. Since S is finite, the sequence P is also finite, that is, we have $P = (q_1, q_2, \ldots, q_k(= q))$, for some $k < n$. □

By Lemma 4, we obtain the following result.

Corollary 5 *Algorithm* INCREASING-CHORD *on input S terminates.*

Lemma 6 *Assume that just before processing $\{p, q\} \in L$ we have a planar graph $G = (S, E)$. If Algorithm* FIND-PATH

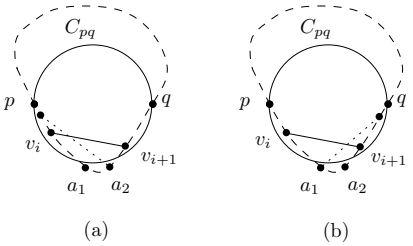


Figure 5: For proof of Lemma 6.

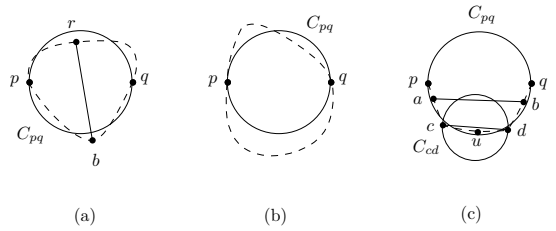


Figure 6: For proof of Lemmas 6 and 8.

on input $(p, q, E_1(= \emptyset))$ returns the value true, then there is a convex path between p and q in $G = (S, E \cup E_1)$ such that C_{pq} contains the whole path and G is planar.

Proof. By Lemma 4, Algorithm FIND-PATH on input $(p, q, E_1(= \emptyset))$ terminates. Let $P := (q_1(= p), q_2, \ldots, q_k(= q))$ be the sequence in the proof of Lemma 4, that is, the sequence contains points on $P_{cc}(p, q)$ that Algorithm FIND-PATH on input $(p, q, E_1(= \emptyset))$ considers these points. Compute sequence $P' := (v_1, v_2, \ldots, v_l)$ of points in S, a subsequence of P, as follows. Set $i := 1$. We consider all elements of P starting at p and ending at q. If we reach q_j, where $1 \leq j \leq k$, such that q_j is in C_{pq}, then v_i is set to q_j and i is set to $i + 1$. Clearly, we have $v_1 := p$, and $v_l := q$, and $|P_{cc}(v_i, q)| > |P_{cc}(v_{i+1}, q)|$, for all $1 \leq i < l$, and all points of P' lie on $P_{cc}(p, q)$. It is easy to see that if E_1 is a nonempty set, then for any edge (a, b) in E_1 there is an integer i, with $1 \leq i < l$, such that $\{a, b\} = \{v_i, v_{i+1}\}$. By assumption just before processing $\{p, q\}$ we have a planar graph. In the following we prove that, for all $1 \leq i < l$, there is an edge between v_i and v_{i+1} in $E \cup E_1$ without crossing edges of E, except possibly at their endpoints.

It is easy to see that, for all $1 \leq i < l$, we have either $(v_i, v_{i+1}) \in E$ or $(v_i, v_{i+1}) \in E_1$. Clearly, if $E_1 = \emptyset$, then there is nothing to be done. Let (v_i, v_{i+1}), for some $1 \leq i < l$, be in E_1, i.e., Algorithm FIND-PATH on input $(p, q, E_1(= \emptyset))$ (in line #13 or #17) adds (v_i, v_{i+1}) to E_1. Let $(a_1, a_2, \ldots, a_m)$, for some $m \geq 1$, be the sequence of all points in S which a_j is on $P_{cc}(v_i, v_{i+1})$, for all $1 \leq j \leq m$, and just before adding (v_i, v_{i+1}) to E_1, the condition in line #7 of Algorithm FIND-PATH holds for them starting at a_1 and ending at a_m. See Figure 5. As seen from Algorithm FIND-PATH, edges $(v_i, a_1), (a_1, a_2), \ldots, (a_{m-1}, a_m)$, and (a_m, v_{i+1}) are in E, and also there is no edge of E with an endpoint a_j, for some $1 \leq j \leq m$, having a nonempty intersection with (p, q), except possibly at their endpoints. If there is such edge, then the condition in line #10 of Algorithm FIND-PATH holds, a contradiction. Also, it is impossible to have an edge of E that crosses (v_i, v_{i+1}), but the edge does not cross (p, q). If there is such edge, then the algorithm does not consider at least one of points v_i and v_{i+1}, see Figure 5. In Figure 5(a), if the dotted line segment is an edge of E, then the algorithm does not consider v_i, and in Figure 5(b), if the dotted line segment is an edge of E, then the algorithm does not consider v_{i+1}. So, (v_i, v_{i+1}) is an edge of E_1 without crossing edges of E, except possibly at their endpoints. So, P' is a path between points p and q in $G = (S, E \cup E_1)$ and $G = (S, E \cup E_1)$ is a planar graph. Since all points of P' are on $P_{cc}(p, q)$ and in C_{pq}, it follows that P' is a convex path between points p and q such that C_{pq} contains the whole path. So, we are done. □

Lemma 7 *Assume that just before processing $\{p, q\} \in L$ we have a planar graph $G = (S, E)$. Assume that there is (respectively, it is possible that we add some edges to E without crossing edges of E, except possibly at their endpoints, to construct) a convex path between p and q in $G[S_{cc}(p, q)]$ such that C_{pq} contains the whole path. Then, Algorithm* FIND-PATH *on input $(p, q, E_1(= \emptyset))$ returns the value true.*

Proof. Assume (for a contradiction) that Algorithm FIND-PATH on input $(p, q, E_1(= \emptyset))$ returns the value $false$, that is, Algorithm FIND-PATH on input $(p, q, E_1(= \emptyset))$ executes the instruction "**return** $false$" in line #11 or #15. If Algorithm FIND-PATH on input $(p, q, E_1(= \emptyset))$ executes the instruction "**return** $false$" in line #11, then there are points $b \in S_{cc}(p, q)$ and $r \in S_c(p, q)$ such that b is outside C_{pq}, and (b, r) is in E, and (b, r) has a nonempty intersection with (p, q) (except possibly at their endpoints), a contradiction. See Figure 6(a). If Algorithm FIND-PATH on input $(p, q, E_1(= \emptyset))$ executes the instruction "**return** $false$" in line #15, then it is easy to see that all points of $S_{cc}(p, q)$ (except points p and q) are outside C_{pq}, a contradiction. See Figure 6(b). Thus, Algorithm FIND-PATH on input $(p, q, E_1(= \emptyset))$ returns the value *true*. □

Lemma 8 *Assume that just before processing $\{p, q\} \in L$ we have a planar graph $G = (S, E)$. Let Algorithm* FIND-PATH *on input $(p, q, E_1(= \emptyset))$ returns $E_1 \neq \emptyset$ and assume that $(a, b) \in E_1$, with $|P_{cc}(p, a)| < |P_{cc}(p, b)|$. Then, all points on $P_{cc}(a, b)$, except points a and b, are outside C_{pq}.*

Proof. Assume (without loss of generality) that points p and q are on a horizontal line and p is to the left of q. Assume for a contradiction that there is a point $u \neq a, b$ on $P_{cc}(a, b)$ which is in (the interior or on the boundary of) C_{pq}. If point u is considered by Algorithm FIND-PATH in the condition of line #7, then the condition in line #7 of Algorithm FIND-PATH does not hold. So, it is easy to see that (a, b) is not added to E_1 by the algorithm. So, point u is not considered by Algorithm FIND-PATH in the condition of line #7. This happens only when there is an edge (c, d) of E, with $|P_{cc}(a, c)| < |P_{cc}(a, d)|$, such that point u is on $P_{cc}(c, d)$. See Figure 6(c). Similarly, if one of points c and d is in (the interior or on the boundary of) C_{pq}, then (a, b) is not added to E_1 by the algorithm. So, both points

c and d are also outside C_{pq}. Clearly, point u is also in C_{cd}. Assume (without loss of of generality) that there is no edge $(r, s) \in E$ such that both points r and s are on $P_{cc}(c, d)$, and both points r and s are outside C_{pq}, and u is on $P_{cc}(r, s)$ (or $P_c(r, s)$). Since, (c, d) is not an edge on the boundary of the convex hull of S, the first or second scenario occurs for (c, d). In the following we consider these cases.

- The first scenario occurs for (c, d).

  By assumption just before processing $\{p, q\}$ we have a planar graph. Thus, there is (respectively, it is possible that we add some edges to E without crossing edges of E, except possibly at their endpoints, to construct) a convex path between c and d passing through u such that C_{cd} contains the whole path. As an example, consider path (c, u, d). So, by Lemma 6 Algorithm FIND-PATH on input $(c, d, E_1(= \emptyset))$ or $(c, d, E_2(= \emptyset))$ returns the value *true*. Therefore, edge (c, d) is not added to E in line #7 of Algorithm INCREASING-CHORD, that is, the first scenario cannot occur for (c, d), a contradiction.

- The second scenario occurs for (c, d).

  So, there are points c' and d' in S with $|c'd'| \leq |pq|$ such that Algorithm FIND-PATH on input $(c', d', E_1(= \emptyset))$ or $(c', d', E_2(= \emptyset))$ adds (c, d), where (c, d) lies entirely in $C_{c'd'}$, to E_1 or E_2, respectively, and then Algorithm INCREASING-CHORD in line #9 or #11 adds E_1 or E_2, respectively, to E. Since points S are in convex position and both points c' and d' are outside slab(c, d), it is impossible to lie both points c' and d' on $P_{cc}(c, d)$, that is, both points c' and d' lie only on $P_c(c, d)$. Assume that Algorithm FIND-PATH runs on $(c', d', E_1(= \emptyset))$ and we have $|P_c(c, c')| < |P_c(c, d')|$. Since (c, d) lies completely in $C_{c'd'}$, point u is in $C_{c'd'}$. Since by assumption just before processing $\{p, q\}$ we have a planar graph, it is easy to see that points c', c, u, d, and d' in this order are considered by Algorithm FIND-PATH on input (c', d', E_1). So, there is (respectively, it is possible to add some edges to E without crossing edges of E, except possibly at their endpoints, to construct) a convex path between c' and d' passing through u laying entirely in $C_{c'd'}$. As an example, consider path (c', c, u, d, d'). Therefore, (c, d) is not added to E_1 by Algorithm FIND-PATH on input $(c', d', E_1(= \emptyset))$, a contradiction.

Therefore, all points on $P_{cc}(a, b)$, except points a and b, are outside C_{pq} □

In the rest of this section we prove that at least one of the conditions in lines #8 and #10 of Algorithm INCREASING-CHORD holds, when the algorithm takes as input a finite set of points in convex position in the plane. To do this we first prove in the following lemma that it is impossible to hold Property 3 for an edge of E_1 and for an edge of E_2 when the algorithm processes a pair of L. Recall that l_{xy} denotes the line passing through x that is orthogonal to the line segment (x, y) for distinct points x and y in the plane.

Lemma 9 *Assume that just before processing $\{p, q\} \in L$ we have a planar graph $G = (S, E)$. Assume that Algorithm* FIND-PATH *on inputs $(p, q, E_1(= \emptyset))$ and $(q, p, E_2(= \emptyset))$ returns $E_1 \neq \emptyset$ and $E_2 \neq \emptyset$, respectively. There are no pairs $(a_1, b_1) \in E_1$ and $(a_2, b_2) \in E_2$ such that Property 3 holds for both (a_1, b_1) and (a_2, b_2), when the algorithm takes as input a finite set of points in convex position in the plane.*

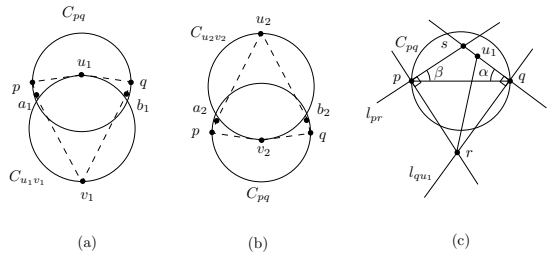


Figure 7: For proof of Lemma 9.

Proof. Let $i = 1, 2$. Recall that Property 3 for (a_i, b_i) is as follows. There is a pair $\{u_i, v_i\} \in L$ with $u_i \in S_c(p, q)$, and $v_i \in S_{cc}(p, q)$, and $|pq| \leq |u_iv_i|$ such that both a_i and b_i are outside $C_{u_iv_i}$. Assume (without loss of generality) that points p and q are on a horizontal line and p is to the left of q. Assume Algorithm FIND-PATH runs on inputs $(p, q, E_1(= \emptyset))$ and $(q, p, E_2(= \emptyset))$ and then the algorithm returns $E_1 \neq \emptyset$ and $E_2 \neq \emptyset$.

Assume (for a contradiction) that Property 3 holds for both $(a_1, b_1) \in E_1$ and $(a_2, b_2) \in E_2$ with $|P_{cc}(p, a_1)| < |P_{cc}(p, b_1)|$ and $|P_c(p, a_2)| < |P_c(p, b_2)|$. See Figure 7(a),(b). Since both a_1 and b_1 are outside $C_{u_1v_1}$ and on $P_{cc}(p, q)$, point v_1 is on $P_{cc}(a_1, b_1)$. See Figure 7(a). Similarly, point u_2 is on $P_c(a_2, b_2)$. See Figure 7(b). Clearly, we have $v_1 \neq a_1$, $v_1 \neq b_1$, $u_2 \neq a_2$, and $u_2 \neq b_2$. So, by Lemma 8, points v_1 and u_2 are outside C_{pq} and ,therefore, points u_1 and v_2 are inside C_{pq}.

It is clear that points p and q are outside $C_{u_1v_1}$ and $C_{u_2v_2}$. It follows that $v_2 \neq p$ and $v_2 \neq q$. Therefore, it is possible that v_2 is a point on (i) $P_{cc}(p, a_1)$, except for point p, or (ii) $P_{cc}(a_1, b_1)$, except for points a_1 and b_1, or (iii) $P_{cc}(b_1, q)$, except for point q. In the rest of the proof we consider these cases.

- Assume that $v_2 \neq p$ is a point on $P_{cc}(p, a_1)$.

  Clearly, point u_1 is above or on (p, q). Let u_1 be an arbitrary point above or on (p, q) and inside C_{pq}. Let r be the point on l_{qu_1} and below (p, q) such that $|u_1r| = |pq|$. See Figure 7(c). Since point q is outside $C_{u_1v_1}$, it is easy to see that point v_1 is to the left of (u_1, r) and to the left of l_{qu_1} and we have $|u_1v_1| \geq |pq|$. Let point v_1 is to the left of (u_1, r) and to the left of l_{qu_1} such that $|u_1v_1| \geq |pq|$. By assumption point v_2 is on $P_{cc}(p, a_1)$. So, point v_2 is below the line passing through points p and r. Let point v_2 be below the line passing through points p and r. Since point p is outside $C_{u_2v_2}$, point u_2 is below l_{pr}. It is easy to see that there is an intersection point between l_{pr} and the line passing through points u_1 and q. Let s be the intersection point between l_{pr} and the line passing through points u_1 and q. Since the algorithm takes as input a set of points in convex position in the plane, point u_2 is below the line passing through points q and u_1. It follows that point u_2 is below line segments (p, s) and (s, q). Define $\alpha := \angle sqp$

and $\beta := \angle spq$. Since point u_1 is inside C_{pq}, we have $0 \leq \alpha < \pi/2$.

Let α be a fixed angle in interval $[0, \pi/2)$. It is easy to see that when point u_1 is far from point q, then β increases. So, the maximum angle of β happens when point u_1 is very close to the boundary of C_{pq}. If we place point u_1 on the boundary of C_{pq}, then point r is also on the boundary of C_{pq} and, therefore, we have $\alpha + \beta = \pi/2$. So, when point u_1 is inside C_{pq}, then we have $\alpha + \beta < \pi/2$. Since point u_2 is below line segments (p, s) and (s, q), it follows that point u_2 is inside C_{pq}, contradicting that point u_2 is outside C_{pq}.

- Assume that $v_2 \neq a_1, b_1$ is a point on $P_{cc}(a_1, b_1)$.

  So, by Lemma 8, point v_2 is outside C_{pq}, contradicting that point v_2 is inside C_{pq}.

- Assume that $v_2 \neq q$ is a point on $P_{cc}(b_1, q)$.

  Similar to the first case (i.e., point $v_2 \neq p$ is on $P_{cc}(p, a_1)$) it follows that point u_2 is inside C_{pq}, a contradiction.

Hence, It is impossible to hold Property 3 for both (a_1, b_1) and (a_2, b_2). □

As seen from Algorithm 3.1 (i.e., Algorithm Increasing-Chord), if the condition in line #6 does not hold, then $B_1 = true$ or $B_2 = true$. If exactly one of variables B_1 and B_2 has the value $false$, then the condition in line #10 or #8, respectively, holds, otherwise $B_1 = B_2 = true$. By Lemma 9, it is impossible to hold Property 3 for an edge of E_1 and for an edge of E_2 when the algorithm processes one pair of L. So, we obtain the following result.

Corollary 10 *Assume that just before processing $\{p, q\} \in L$ we have a planar graph $G = (S, E)$. If Algorithm* Find-Path *on at least one of inputs $(p, q, E_1(= \emptyset))$ and $(q, p, E_2(= \emptyset))$ returns the value true, then the condition in line #8 or #10 of Algorithm* Increasing-Chord *holds.*

5 Properties of edges added by the first scenario

Recall that $G = (S, E)$ is the output graph of Algorithm Increasing-Chord on S, where S is a finite set of points in convex position in the plane. In this section we state some properties for edges of G which the first scenario adds these edges to E, that is, these edges are added to E in line #7 of Algorithm Increasing-Chord. These properties help us to prove that G is a planar graph.

Lemma 11 *Let (p, q) be in E, where $\{p, q\} \in L$. Assume that just before processing $\{p, q\}$ we have a planar graph and there is $(a, b) \in E$ (with $|ab| \leq |pq|$) such that has a nonempty intersection with (p, q), except possibly at their endpoints. If the first scenario occurs for (p, q), then at least one of points a and b is in (the interior or on the boundary of) C_{pq}.*

Proof. Suppose that the first scenario occurs for (p, q). Assume (without loss of generality) that points p and q are on a horizontal line and p is to the left of q. Suppose for a contradiction that (a, b) is an edge of E with $|ab| \leq |pq|$ such that has a nonempty intersection with (p, q), except possibly at their endpoints, and both a and b are outside C_{pq}. So, at least one of points p and q is in (the interior or on the boundary of) C_{ab}. Assume without loss of generality that p is in C_{ab} and p is on $P_{cc}(a, b)$. See Figure 8. Suppose that (a, b) has the leftmost intersection point between all edges of E that have a nonempty intersection with (p, q), except possibly at their endpoints, and both theirs endpoints are outside C_{pq}. Clearly, edge (a, b) is not an edge on the boundary of the convex hull of S. So, the first or second scenario occurs for (a, b). In the following we consider these cases.

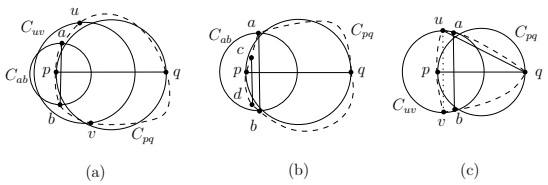


Figure 8: For proof of Lemma 11.

- The first scenario occurs for (a, b).

  Suppose that there is no edge of E having a nonempty intersection with (p, q) to the left of the intersection point between (p, q) and (a, b), except possibly at their endpoints. By assumption just before processing $\{p, q\}$ we have a planar graph. Thus, there is (respectively, it is possible that we add some edges to E without crossing edges of E, except possibly at their endpoints, to construct) a convex path between a and b passing through p such that C_{ab} contains the whole path. As an example, consider path (a, p, b). So, by Lemma 6 Algorithm Find-Path on input $(a, b, E_1(= \emptyset))$ or $(a, b, E_2(= \emptyset))$ returns the value *true*. Therefore, edge (a, b) is not added to E in line #7 of Algorithm Increasing-Chord, that is, the first scenario cannot occur for (a, b), a contradiction. See Figure 8(a). Otherwise, suppose that edge $(c, d) \in E$ (with $|cd| \leq |pq|$) has a nonempty intersection with (p, q) to the left of the intersection point between (p, q) and (a, b) such that there is no intersection point between these two intersections points on (p, q). Since (a, b) has the leftmost intersection point between all edges of E that have a nonempty intersection with (p, q) (except possibly at their endpoints) and both theirs endpoints are outside C_{pq}, at least one of points c and d is in C_{pq}. Assume (without loss of generality) that point c is in C_{pq}. Since just before adding (p, q) to E we have a planar graph, therefore, point c is also in C_{ab}. See Figure 8(b). So, there is (respectively, it is possible that we add some edges to E without crossing edges of E, except possibly at their endpoints, to construct) a convex path between a and b passing through c such that C_{ab} contains the whole path. As an example, consider path (a, c, b). It follows from Lemma 6 that Algorithm Find-Path on input $(a, b, E_1(= \emptyset))$ or $(a, b, E_2(= \emptyset))$ returns the value *true*. Therefore, the first scenario cannot occur for (a, b), a contradiction.

- The second scenario occurs for (a, b).

  So, there are points u and v in S with $|uv| \leq |pq|$ such

that Algorithm FIND-PATH on input $(u, v, E_1(= \emptyset))$ or $(v, u, E_2(= \emptyset))$ adds (a, b), where (a, b) lies entirely in C_{uv}, to E_1 or E_2, respectively, and then Algorithm INCREASING-CHORD in line #9 or #11 adds E_1 or E_2, respectively, to E. (Clearly, Algorithm FIND-PATH on at least one of inputs $(u, v, E_1(= \emptyset))$ and $(v, u, E_2(= \emptyset))$ returns the value *true*.) There are two possible cases for u and v.

Case 1: Both u and v are on $P_c(a, b)$ (with $|P_c(a, u)| < |P_c(a, v)|$), and Algorithm FIND-PATH on input $(u, v, E_1(= \emptyset))$ or $(u, v, E_2(= \emptyset))$ returns the value *true*, and (a, b) is in E_1 or E_2, respectively. See Figure 8(a).

Case 2: Both u and v are on $P_{cc}(a, b)$ (with $|P_{cc}(a, u)| < |P_{cc}(a, v)|$), and Algorithm FIND-PATH on input $(v, u, E_1(= \emptyset))$ or $(v, u, E_2(= \emptyset))$ returns the value *true*, and (a, b) is in E_1 or E_2, respectively. See Figure 8(c).

We first consider Case 1. Since $p \in C_{ab}$, and $|uv| \leq |pq|$, and (a, b) lies entirely in (the interior or on the boundary of) C_{uv}, and both points a and b are outside C_{pq}, it is easy to see that p is in C_{uv}. See Figure 8(a). Clearly, $p \neq a$ and $p \neq b$. But, by Lemma 8, point p is outside C_{uv}, a contradiction.

We now consider Case 2. Assume that Algorithm FIND-PATH runs on inputs $(v, u, E_1(= \emptyset))$ and $(u, v, E_2(= \emptyset))$. So, (a, b) is in E_1. Clearly, point p is in C_{uv}. It is easy to see (similar to the previous case of the proof, i.e., the first scenario occurs for (a, b)) that there is (respectively, it is possible to add some edges to E without crossing edges of E, except possibly at their endpoints, to construct) a convex path between u and v in $G[S_{cc}(u, v)]$ laying entirely in C_{uv}. See Figure 8(c). By Lemma 6, Algorithm FIND-PATH on input $(u, v, E_2(= \emptyset))$ returns the value *true*. Hence, Algorithm FIND-PATH on both inputs $(v, u, E_1(= \emptyset))$ and $(u, v, E_2(= \emptyset))$ returns the value *true*. Since Property 3 holds for $(a, b) \in E_1$, by Lemma 9, Property 3 does not holds for any edge of E_2. This, together with that Algorithm FIND-PATH on both inputs $(v, u, E_1(= \emptyset))$ and $(u, v, E_2(= \emptyset))$ returns the value *true*, implies that E_2 is added to E in line #11 of Algorithm INCREASING-CHORD. Since $(a, b) \in E_1$, edge (a, b) is not added to E, a contradiction.

Therefore, at least one of points a and b is in C_{pq}. □

Lemma 12 *Let (p, q) be in E, where $\{p, q\} \in L$. Assume that just before processing $\{p, q\}$ we have a planar graph and there is $(a, b) \in E$ with $|ab| \leq |pq|$ such that has a nonempty intersection with (p, q), except possibly at their endpoints. If C_{ab} contains p or q, then it is impossible to occur the first scenario for (a, b).*

Proof. Assume for a contradiction that the first scenario occurs for (a, b). So, by Lemma 11, there is no edge of E (with length at most $|pq|$) which has a nonempty intersection with (p, q) (except possibly at their endpoints) and that both its endpoints are outside C_{pq}. Assume (without loss of generality) that points p and q are on a horizontal line and p is to the left of q. Suppose that $(a, b) \in E$ (with $|ab| \leq |pq|$) has the leftmost intersection point between all edges of E that have a nonempty intersection with (p, q), except possibly at their endpoints. Assume (without loss of generality) that a is in C_{pq}, and by assumption C_{ab} contains p, and point p is on $P_{cc}(a, b)$. See Figure 9(a). Also, by assumption just before processing $\{p, q\}$ we have a planar graph. Thus, there is (respectively, it is possible that we add some edges to E without crossing edges of E, except possibly at their endpoints, to construct) a convex path between a and b passing through p such that C_{ab} contains the whole path. As an example, consider path (a, p, b). So, by Lemma 6 Algorithm FIND-PATH on input $(a, b, E_1(= \emptyset))$ or $(a, b, E_2(= \emptyset))$ returns the value *true*. Therefore, edge (a, b) is not added to E in line #7 of Algorithm INCREASING-CHORD, that is, the first scenario cannot occur for (a, b). □

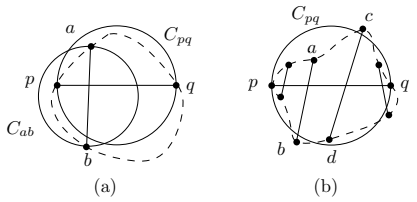


Figure 9: For proof of Lemmas 12 and 13 from left to right, respectively.

Lemma 13 *Assume that just before adding (p, q) to E, where $\{p, q\} \in L$, we have a planar graph and there are at least two edges (a, b) and (c, d) of E (with lengths at most $|pq|$) such that points a and c are on $P_c(p, q)$, and points b and d are on $P_{cc}(p, q)$, and (a, b) has a nonempty intersection with (p, q), and (c, d) has a nonempty intersection with (p, q), except possibly at their endpoints. If $a, d \in C_{pq}$ and $b, c \notin C_{pq}$, then the first scenario cannot occur for (p, q).*

Proof. Assume without loss of generality that p and q are on a horizontal line and p is to the left of q and that the intersection point between (p, q) and (a, b) is to the left of the intersection point between (p, q) and (c, d). See Figure 9(b). Clearly, points a and d are above and below (p, q), respectively. Assume (for a contradiction) that the first scenario occurs for (p, q). Clearly, edges (a, b) and (c, d) are not edges on the boundary of the convex hull of S. So, the first or second scenario occurs for every one of edges (a, b) and (c, d). In the following before we complete the proof, we explain what happens when the second scenario occurs for (a, b) (respectively, (c, d)).

Suppose that the second scenario occurs for (a, b) (respectively, (c, d)). So, there is $\{a', b'\} \in L$ with $|a'b'| \leq |pq|$ (respectively, $\{c', d'\} \in L$ with $|c'd'| \leq |pq|$) such that when the algorithm processes $\{a', b'\}$ (respectively, $\{c', d'\}$), then the algorithm adds (a, b), where (a, b) lies entirely in $C_{a'b'}$ (respectively, (c, d), where (c, d) lies entirely in $C_{c'd'}$), to E. Clearly, both a' and b' (respectively, both c' and d') lie on $P_c(a, b)$ or $P_{cc}(a, b)$ (respectively, $P_c(c, d)$ or $P_{cc}(c, d)$). Since points of S are in convex position, both a' and b' (respectively, both c' and d') are not in (the interior or on the boundary of) slab(a, b) (respectively, slab(c, d)). There are

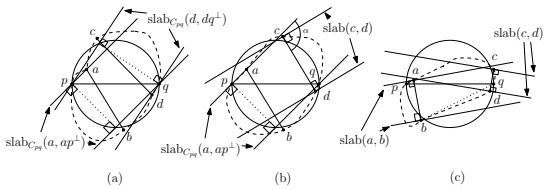


Figure 10: For proof of Lemma 13.

the following cases to consider. See Figure 10. Recall that by assumption both a and c are in (the interior or on the boundary of) C_{pq} and both b and c are outside C_{pq}.

- Suppose that the first scenario occurs for both (a, b) and (c, d).

  Assume that a is an arbitrary point in (the interior or on the boundary of) C_{pq} and above (p, q). By Lemma 12, both C_{ab} and C_{cd} contain none of points p and q. So, it is easy to see that b is below $\text{slab}_{C_{pq}}(a, ap^{\perp})$, and, therefore, d is also below $\text{slab}_{C_{pq}}(a, ap^{\perp})$. Clearly, point q is on the boundary of $\text{slab}_{C_{pq}}(a, ap^{\perp})$. It follows that c is above $\text{slab}_{C_{pq}}(d, dq^{\perp})$, and, therefore, c is above $\text{slab}_{C_{pq}}(a, ap^{\perp})$. Hence, a is below the line passing through points p and c, contradicting the fact that points of S are in convex position. See Figure 10(a).

- Assume (without loss of generality) that the first scenario occurs for (a, b) and the second scenario occurs for (c, d).

  Assume that a is an arbitrary point in C_{pq} and above (p, q). By Lemma 12, C_{ab} contains none of points p and q. Hence, b is below $\text{slab}_{C_{pq}}(a, ap^{\perp})$, and, therefore, d is also below $\text{slab}_{C_{pq}}(a, ap^{\perp})$. Clearly, point q is on the boundary of $\text{slab}_{C_{pq}}(a, ap^{\perp})$. There are two possibilities for c' and d'. Both c' and d' are either on $P_c(c, d)$ or on $P_{cc}(c, d)$. Assume without loss of generality that c' is above $\text{slab}(c, d)$ and, therefore, d' is below $\text{slab}(c, d)$. Since points of S are in convex position, point c is below or on the line passing through points a and p, that is, point c is in (the interior or on the boundary of) $\text{slab}_{C_{pq}}(a, ap^{\perp})$. Clearly, in the former case point c' is below or on the line passing through points p and c, and, therefore, point c' is also in (the interior or on the boundary of) $\text{slab}_{C_{pq}}(a, ap^{\perp})$. Let α be the angle between rays $\overrightarrow{pa}$ and $\overrightarrow{cd}$. It is clear that $\alpha > \pi/2$. Since points b and d are below $\text{slab}_{C_{pq}}(a, ap^{\perp})$, and point d' is below $\text{slab}(c, d)$, and point d' is on $P_c(c, d)$, it follows that point d' is below the line passing through points b and d, contradicting the fact that points of S are in convex position. Clearly, $q \neq c$ and $q \neq d$. In the latter case, by Lemma 8, point q is outside C_{cd}. So, similar to the case which the first scenario occurs for both (a, b) and (c, d) (i.e., the first case) this contradicts the fact that points of S are in convex position. See Figure 10(b).

- Suppose that the second scenario occurs for both (a, b) and (c, d).

  It suffices to consider the case that both a' and b' are on $P_{cc}(a, b)$ and both c' and d' are on $P_c(c, d)$. Assume that a is an arbitrary point in C_{pq} and above (p, q). Assume without loss of generality that a' is above $\text{slab}(a, b)$ and, therefore, b' is below $\text{slab}(a, b)$. Since points of S are in convex position, both points c and d are inside $\text{slab}(a, b)$. Assume without loss of generality that c' is above $\text{slab}(c, d)$ and, therefore, d' is below $\text{slab}(c, d)$. Clearly, point c' is below or on the line passing through points a and c and, therefore, point c' is inside $\text{slab}(a, b)$. Let α be the angle between rays $\overrightarrow{ab}$ and $\overrightarrow{cd}$. Clearly, $\alpha > 0$. Since point d' is below $\text{slab}(c, d)$ and point d' is on $P_c(c, d)$, it follows that point d' is below the line passing through points b and d, contradicting the fact that points of S are in convex position. See Figure 10(c).

Therefore, all the above cases never occur. This completes the proof. □

6 An increasing-chord planar graph returned by Algorithm 3.1

In this section we prove that the output of Algorithm Increasing-Chord on input S, where S is a finite set of points in convex position in the plane, is an increasing-chord planar graph. To do this, we prove by induction on the rank of L that just after processing each pair $\{p, q\} \in L$ by Algorithm Increasing-Chord there is a convex path between p and q in the resulting graph such that C_{pq} contains the whole path and that the resulting graph is planar.

Lemma 14 *The output of Algorithm 3.1 on input S is a planar graph and there is a convex path in the output graph between any two distinct points $p, q \in S$ lying entirely in C_{pq}, when S is a finite set of points in convex position in the plane.*

Proof. Clearly, any edge on the boundary of $CH(S)$ does not intersect the edge between any two points of S, except possibly at their endpoints, and for all edges on the boundary of $CH(S)$ the claim in the lemma holds. For distinct points $p, q \in S$, where (p, q) is not an edge on the boundary of $CH(S)$, we prove the claim in the lemma by induction on the rank of $\{p, q\}$ in L, where L is the list of the $\binom{n}{2}$ sorted pairs of distinct points of S in non-decreasing order of their distances, except pairs that are endpoints of edges on the boundary of the convex hull of S.

Base case: Let $\{r, s\}$ be the first pair in L. If the first scenario occurs for (r, s), then clearly the resulting graph is planar and there is a convex path in the graph between points r and s lying entirely in C_{rs}. Otherwise, Algorithm Find-Path on at least one of inputs $(r, s, E_1(= \emptyset))$ and $(s, r, E_2(= \emptyset))$ (in line #5) returns the value *true*. By Corollary 10 at least one of the conditions in lines #8 and #10 of Algorithm Increasing-Chord holds. So, Algorithm Increasing-Chord (in line #9 or #11) adds E_1 or E_2, respectively, to E. By Lemma 6 the resulting graph is planar and there is a convex path between r and s in the graph such that C_{rs} contains the whole path.

Induction hypothesis: Assume that just before Algorithm Increasing-Chord processes $\{p, q\}$, where $|rs| \leq |pq|$, the graph $G = (S, E)$ is planar and there is a convex

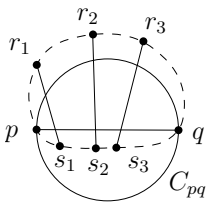


Figure 11: For proof of Lemma 14.

path between points of any pair $\{a, b\} \in L$ with $|ab| \leq |pq|$ such that C_{ab} contains the whole path.

The inductive step: Suppose that the algorithm processes $\{p, q\}$. There are two possible cases to consider, depending on whether or not (p, q) is added to E in line #7 of Algorithm Increasing-Chord.

Case 1: Edge (p, q) is not added to E in line #7 of Algorithm Increasing-Chord.

Similar to the base case the resulting graph is planar and there is a convex path between p and q in the graph such that C_{pq} contains the whole path.

Case 2: Edge (p, q) is added to E in line #7 of Algorithm Increasing-Chord, that is, the first scenario occurs for (p, q). Clearly, there is a convex path between p and q in the graph lying entirely in C_{pq}. We claim that any edge of E has an empty intersection with (p, q), except possibly at their endpoints.

Assume that $(r_1, s_1), (r_2, s_2), \ldots, (r_k, s_k) \in E$, where $k \geq 1$, are all edges of E that have a nonempty intersection with (p, q), except possibly at their endpoints. Assume without loss of generality that each r_i, for $1 \leq i \leq k$, is on $P_c(p, q)$ with $|P_c(p, r_1)| \leq |P_c(p, r_2)| \leq \ldots \leq |P_c(p, r_k)|$. (Since just before processing $\{p, q\}$ we have a planar graph, each s_i is on $P_{cc}(p, q)$ with $|P_{cc}(p, s_1)| \leq |P_{cc}(p, s_2)| \leq \ldots \leq |P_{cc}(p, s_k)|$.) See Figure 11.

By Lemma 11, at least one of points r_i and s_i is in (the interior or on the boundary of) C_{pq}, for $1 \leq i \leq k$. We need only to consider two cases: (i) for all $1 \leq i \leq k$, every r_i is in C_{pq} or every s_i is in C_{pq}, see Figure 11, and (ii) there are edges (r_j, s_j) and (r_l, s_l) with $1 \leq j, l \leq k$ such that $r_j, s_l \in C_{pq}$ and $s_j, r_l \notin C_{pq}$, see Figure 9(b). (Assume $(r_j, s_j) = (a, b)$ and $(r_j, s_j) = (c, d)$.) In the former case assume without loss of generality that all points $s_1, s_2, \ldots, s_k$ are in C_{pq}. By assumption we have a planar graph just before adding (p, q) to E. So, there is (respectively, it is possible that we add some edges to E without crossing edges of E, except possibly at their endpoints, to construct) a convex path between p and q passing through points $s_1, s_2, \ldots, s_k$ such that C_{pq} contains the whole path. As an example, consider path $(p, s_1, s_2, \ldots, s_k, q)$. It follows from Lemma 6 that Algorithm Find-Path on input $(p, q, E_1(= \emptyset))$ returns the value *true*, that is, Algorithm Increasing-Chord (in line #7) does not add (p, q) to E. So, the first scenario cannot occur for (p, q), a contradiction. By Lemma 13 the latter case never occurs. Hence, the resulting graph is planar, and we are done. □

Clearly, the running time of the algorithm of this paper (i.e., Algorithms Increasing-Chord and Find-Path) is polynomial in $|S|$. So, by Lemmas 2 and 14 we get the following result.

Theorem 15 *There is a polynomial-time algorithm that computes an increasing-chord planar graph for a finite set of points in convex position in the plane with no Steiner point and with the geometric dilation less than $\pi/2$.*

7 Conclusions

Let S be a finite set of points in convex position in the plane. We studied the problem of computing an increasing-chord planar graph for S. We proposed a polynomial-time algorithm that computes an increasing-chord planar graph spanning S with the geometric dilation less than $\pi/2$ and with no Steiner point. A direct implementation of the algorithm has running time $\mathcal{O}(|S|^5)$. A natural problem is to improve the running time of the algorithm. Also, we guess it is possible to extend the algorithm to an algorithm that computes an increasing-chord planar graph for any point set in the plane with no Steiner point.

References

[1] S. Alamdari, T. M. Chan, E. Grant, A. Lubiw, and V. Pathak. Self-approaching graphs. In *International Symposium on Graph Drawing*, pages 260–271. Springer, 2012.

[2] M. Amani, A. Biniaz, P. Bose, J. De Carufel, A. Maheshwari, and M. Smid. A plane 1.88-spanner for points in convex position. *Journal of Computational Geometry*, 7(1):520–539, 2016.

[3] P. Angelini, F. Frati, and L. Grilli. An algorithm to construct greedy drawings of triangulations. *Journal of Graph Algorithms and Applications*, 14(1):19–51, 2010.

[4] P. Bose, R. Fagerberg, A. van Renssen, and S. Verdonschot. Competitive routing in the half-θ 6-graph. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 1319–1328. Society for Industrial and Applied Mathematics, 2012.

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

[6] M. T. Goodrich and D. Strash. Succinct greedy geometric routing in the euclidean plane. In *International Symposium on Algorithms and Computation*, pages 781–791. Springer, 2009.

[7] X. He and H. Zhang. On succinct convex greedy drawing of 3-connected plane graphs. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 1477–1486. Society for Industrial and Applied Mathematics, 2011.

[8] X. He and H. Zhang. On succinct greedy drawings of plane triangulations and 3-connected plane graphs. *Algorithmica*, 68(2):531–544, 2014.

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

[10] T. Leighton and A. Moitra. Some results on greedy embeddings in metric spaces. *Discrete & Computational Geometry*, 44(3):686–705, 2010.

[11] K. Mastakas and A. Symvonis. On the construction of increasing-chord graphs on convex point sets. In *Information, Intelligence, Systems and Applications (IISA), 2015 6th International Conference on*, pages 1–6. IEEE, 2015.

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

[13] C. H. Papadimitriou and D. Ratajczak. On a conjecture related to geometric routing. *Theoretical Computer Science*, 344(1):3–14, 2005.

[14] A. Rao, S. Ratnasamy, C. Papadimitriou, S. Shenker, and I. Stoica. Geographic routing without location information. In *Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 96–108. ACM, 2003.

[15] G. Rote. Curves with increasing chords. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 115, pages 1–12. Cambridge Univ Press, 1994.

Kinetic Nearest Neighbor Search in Black-Box Model

Bahram Sadeghi Bigham* Maryam Nezami † Marziyeh Eskandari‡

Abstract

Proximity problems is a class of important problems which involve estimation of distances between geometric objects. The nearest neighbor search which is a subset of proximity problems, arises in numerous fields of applications, including Pattern Recognition, Statistical classification, Computer vision and etc. In this study, a nearest neighbor search is presented to move points in the plane, while query point is static.

The proposed method works in the black-box KDS model, in which the points location received at regular time steps while at the same time, an upper bound d_{max} is known on the maximum displacement of any point at each time step. In this paper, a new algorithm is presented for kinetic nearest neighbor search problem in the black-box model, under assumptions on the distribution of the moving point set P. It has been shown how the kinetic nearest neighbor will be updated at each time step in $O(k\Delta_k \log n)$ amortized time, where Δ_k is the k-spread of a point set P.

Key words: Computational Geometry, Black Box Model, Kinetic, Nearest Neighbor.

1 Introduction

In recent years, there has been a growing amount of research dealing with moving, or kinetic, data. Algorithms dealing with objects in motion traditionally discretized time and recompute the structure of interest at every time step from scratch. This can be wasteful, especially if time steps are small. Since objects will move more slightly, and the structure may not change at all. Ideally an object receives attention if and only if, its new location triggers an actual change in the structure. A basic assumption in the Kinetic Data Structure (KDS), which is introduced by Basch et al. [1], is that the object trajectories are known. This assumption severely limits the applications of KDS framework.

*Corresponding author: Department of Computer Science and Information Technology, Institute for Advanced Studies in Basic Sciences (IASBS), Prof. Yousef Sobouti Blvd., Zanjan, Iran. b_sadeghi_b@iasbs.ac.ir

†Department of Computer Science and Information Technology, Institute for Advanced Studies in Basic Sciences (IASBS), Prof. Yousef Sobouti Blvd., Zanjan, Iran. mnezami@gmail.com

‡Department of Computer Science and Information Technology, Alzahra University, Tehran Province, Tehran, Vanak Village Street, eskandari@alzahra.ac.ir

Sometimes in an online manner, the object locations are available only at time steps. So when future tracking moving object is not available, the basic assumption in the KDS model is not always valid (refer to surveys by Guibas for more information about KDSs [5, 6]). Certainly, there is a need for a hybrid model, which combines ideas from the KDS model with a traditional time-slicing approach. De Berge et al. [2] introduced kinetic data structures in black-box model with fewer restrictions, instead of knowing knowledge of the trajectories. They only assume that they know upper bounds on objects' speed and can compute their positions at regular time steps. They develop KDSs in the black-box model that consider under certain assumptions on the trajectories which is proved to be more efficient than recomputing the structure from scratch. Additionally, in recent years, some problems (e.g., convex hull, 2-center) have been studied in black box model [2, 3].

Proximity problems is a class of important problems in computational geometry which involve estimation of distances between geometric objects. The nearest neighbor search problem arises in numerous fields of applications, including Pattern Recognition, Statistical classification, Computer vision and etc.

In this paper, a black-box KDSs has been studied to find the nearest neighbor in a set P of n points moving to a static query point q in the plane. There is a need to make assumptions on the point movements and time steps to obtain proved efficient solutions. Time steps should be small enough to have coherency between the objects' positions in consecutive time steps. Otherwise, there is no better way than recomputing the structure from scratch. Also, it has been assumed that in most of our results that P is fairly distributed at each time step.

In the following, firstly, the black-box model which was introduced by de Berg et al. [2] and the concept of k-spread will be described. Furthermore, an algorithm that updates the nearest neighbor search at each time step in $O(k\Delta_k \log n)$ amortized time, where Δ_k is the k-spread of a point set P will be presented.

The Back-Box model: Let P be a set of moving points in plane. In black-box model, the exact motions of the points are unknown and a new location $p(t)$ for each point $p \in P$ is available at regular time steps $t =$

$t_1, t_2, t_3, ...$ The goal is to update the nearest neighbor to q at each time step.

For any point $p \in P$, it is assumed a maximum displacement d_{max} that indicates how far the point can move between two consecutive time steps. For a (static) set S of points and a point $s \in S$, let $NN_k(s, S)$ denote the k-th nearest neighbor of s in $S \backslash \{s\}$. Let $dist(p_1, p_2)$ denotes the Euclidean distance between two points p_1 and p_2, and

$$mindist_k(S) = min_{s \in S} dist(s, NN_k(s.S)).$$

At any time step t, d_{max} is assumed the most $mindist_k(P(t))$.

Displacement Assumption: There is a maximum displacement d_{max}, such that at any time step t_i:

- $d_{max} \leq mindist_k(P(t_i))$, and
- $dist(p(t_i), p(t_{i+1})) \leq d_{max}$ for each point $p \in P$.

This means that there are no more than k points within a distance d_{max} of each other.

The k-spread of a point set: In this model, a conception called k-spread is utilized, as introduced by Erickson [4] for distribution of a set of point. The k-spread, Δ_k of a set P of static points is defined as:

$$\Delta_k(P) = \frac{diam(P)}{mindist_k(P)}$$

in which $diam(P)$ is diameter of P.

Lemma 1 *Let P be a set of points and R be a region in the plane such that $diam(R) < mindist_k(P)$, then R contains at most k points of P [2].*

2 Maintaining the nearest neighbor search

Let P be a set of n moving points in the plane and q be static query point that adheres to the Displacement Assumption with parameters k and d_{max}. The goal is to compute the nearest neighbor to q from moving points P at regular time steps $t = t_1, t_2, t_3, \cdots$. Let $NNS(P(t))$ denotes the nearest neighbor in $P(t) := \{p(t) : p \in P\}$ to q. $NNS(t)$ is used as a shorthand for $NNS(P(t))$.

The maintenance of some structures always depends on all the points and the others are defined by only a subset of the points. The nearest neighbor search is latter type; so as previously mentioned, the algorithm does not need to ask for all new positions at every time step. It may ignore some points, if the new locations of these points cannot change the structure. Thus, an efficient algorithm is potentially exist. A subset $Q(t) \subseteq P(t)$ is introduced in this paper, so that $NNS(Q(t)) = NNS(t)$. A point in $P(t)$ is called active if it is part of $Q(t)$ at time step t; and the points that are not in $Q(t)$ are called inactive. $NNS(t)$ can be computed using only points from Q. If Q is much smaller than P, this may be much faster than computing $NNS(t)$ directly. In a kinetic setting, knowledge from previous time steps can be used to find Q quickly. The coherency between structures at successive time steps are to find Q quickly at each time step. To this end, a bound can be computed for each point, on the number of time steps that must pass before it can become part of Q.

Let P be a set of moving points and t be a time step. A function $\tau_{NNS}(p, t)$ is called an inactivity function if the point p is inactive at any time t_i with $t < t_i \leq t + \tau_{NNS}(p, t)$.

For the nearest neighbor search problem, it is defined

$$\tau_{NNS}(p, t) = \lfloor \frac{dist(p, q) - dist(q, NNS(t))}{2d_{max}} \rfloor$$

where $dist(p_1, p_2)$ denotes the minimum Euclidean distance p_1 to p_2. The following lemma shows that this is a valid inactivity function.

Lemma 2 *For any point $p \in P$ and any time step t, $p(t_i)$ is inactive at any time step t_i with $t < t_i \leq t + \tau_{NNS}(p, t)$.*

Proof. In the black-box model, each point p can move at most d_{max} per time step. This includes the nearest neighbor of q, therefore the distance between a point p and q can decrease by at most $2d_{max}$ per time step. Also, the distance between $NNS(t)$ and q can increase by at most $2d_{max}$ in each time step; as $dist(p, q) - dist(q, NNS(t))$ can decrease by at most $2d_{max}$ per time step. As a result, p cannot be active point at any time step t_i with $t < t_i \leq t + \tau_{NNS}(p, t)$. □

For each point $p \in P$, a time stamp T_p maintained that indicates the first time in the future at which p can be active. At the time step $t = T_p$, we say that the time stamp of p has expired.

The general algorithm thus works as follows: $NNS(t_0)$ initially is computed by scratch. Then each point $p \in P$ enter in a queue with the time stamp $T_p = t_0 + \tau_{NNS}(p, t) + 1$ as its key. At each time step t, the set $Q(t)$ of all points with key t retrieved from the queue. Then $NNS(Q(t))$ is computed and, hence, $NNS(t)$. Finally each point $p \in Q(t)$ reinserted into the queue with its new time stamp $T_p = t + \tau_{NNS}(p, t) + 1$.

To implement this algorithm, an array A is used where $A[t_i]$ contains the points which time stamps expire at time t_i. To restrict the amount of storage, an array $A[0 \cdots n-1]$ is used with n entries, where $A[i]$ stores all points with time stamp $i = T_p \mod n$. The time stamps are bounded to be at most n steps. This approach enables us to add and remove points from the queue in $O(1)$ time per point as opposed to $O(\log n)$ with a standard priority queue structure. The proposed

approach is made explicit in Algorithm 1. Note that the algorithm needs to know only d_{max} to work properly and it does not need to know bounds on the k-spread.

Algorithm 1 UPDATENNS

$Q(t) \leftarrow$ set of points stored in $A[t]$
Compute $NNS(Q(t))$ and set $NNS(P(t)) \leftarrow NNS(Q(t))$
for each $p \in Q(t)$ **do**
  compute $\tau_{NNS}(p,t)$
  Add p to $A[t + \min(\tau_{NNS}(p,t)+1, n) \mod n]$
end for
$t \leftarrow (t+1) \mod n$

It is worth mentioning that how can compute $NNS(t)$ and τ_{NNS}. Computing $NNS(Q(t))$ can be done by nearest neighbor search algorithm at static mode in $|Q(t)|$ time and clearly computing τ_{NNS} is done in $O(1)$ time, so the time to update algorithm 1 at time step t, depends on the size of $Q(t)$. In the worst case, all points may expire in a single time step, but when the k-spread of P is low, it is possible to show that on average only a small number of points expire.

Lemma 3 *At each time step t, UPDATENNS updates the nearest neighbor in P to q in $O(|Q(t)|)$ time.*

The number of points that can expire in a single time step -the size of $Q(t)$- can be bounded by amortized analysis using $\gamma_{NNS}(P)$ as a parameter, which will define a bound on the maximum number of points $p \in P$ at any time t which τ_{NNS} has the same value.

Lemma 4 *The number of points $p \in P$ for which $\tau_{NNS}(p,t) = i$ is bounded by $\gamma_{NNS}(P) = O(k\Delta_k)$ for any $0 \le i \le n$.*

Proof. Let G_i denotes the set of points $p \in P$ for which $\tau_{NNS}(p,t) = i$, according to the definition $\tau_{NNS}(p,t)$ for any point $p \in P$ can conclude the following statement:

$$i \cdot 2d_{max} \le dist(p,q) - dist(q, NNS(t)) < (i+1) \cdot 2d_{max}$$

So, the number of points which distance $dist(p,q) - dist(q, NNS(t))$ is between $i \cdot 2d_{max}$ and $(i+1) \cdot 2d_{max}$ should be bounded. The points in G_i reside in area C that which are centered at the same point q, but differ in radius length $2d_{max}$ as illustrated in Figure 1. Now consider overlaying this area with an axis-aligned grid $\mathcal{G}$ of which each cell has edge length $mindist_k(P)$. From Lemma 1, it is clear that each cell contains $O(k)$ points. This region C can be subdivided into four axis aligned rectangles R_n, R_w, R_s and R_e with edge lengths at most $2d_{max}$ and $dam(P)$ (see Figure 2). Firstly, the maximum number of cells in R_n are computed. From the definition of the k-spread and the Displacement Assumption, it is obvious that each rectangle of R_n intersects at most $O(\Delta_k)$ cells of this grid and similar argument can be made for R_w, R_s and R_e and so, it follows that $R = R_n \cup R_s \cup R_e \cup R_w$ contains $O(\Delta_k)$ cells of this grid. Then, according the areas defined in Figure 3, should be fixed out the maximum number of cells in C is less than R.

Let N_C denotes the maximum number of cells of C (and similarly for other regions). The areas of r_{ne}, r_{se}, r_{nw} and r_{sw} are pink regions, c_{ne}, c_{se}, c_{nw} and c_{sw} are yellow regions and red regions in Figure 3 are s_{ne}, s_{se}, s_{nw} and s_{sw}. In the following, we define:

$N_{\mathcal{C}} = N_R + N_{c_{nw}} + N_{c_{ne}} + N_{c_{se}} + N_{c_{sw}} - N_{r_{nw}} - N_{r_{sw}} - N_{r_{se}} - N_{r_{ne}} - N_{s_{nw}} - N_{s_{sw}} - N_{s_{se}} - N_{s_{ne}}$.

It is clear that the area of c_{nw} is less than the area of r_{nw}; therefore $N_{c_{nw}} < N_{r_{nw}}$. Similarly it can be seen that $N_{c_{ne}} < N_{r_{ne}}$, $N_{c_{se}} < N_{r_{se}}$ and $N_{c_{sw}} < N_{r_{sw}}$. So, according to statement , the maximum number of cells in C is less than R. As a result, C contains $O(\Delta_k)$ cells. As already mentioned, each cell contains $O(k)$ points from P, so set G_i contains $O(k\Delta_k)$ points. □

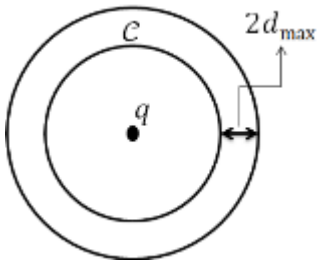


Figure 1: The area of $\mathcal{C}$ include set of points $p \in P$ which $\tau_{NNS}(p,t) = i$

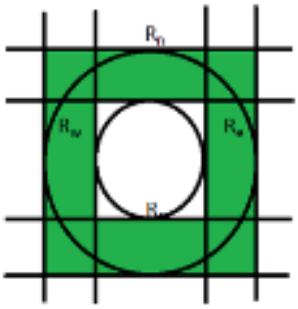


Figure 2: Rectangles R_n, R_s, R_w and R_e

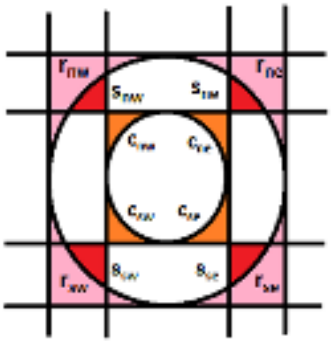


Figure 3: The areas of r_{nw}, r_{ne}, r_{sw} and r_{se}, s_{nw}, s_{ne}, s_{sw} and s_{se}, c_{nw}, c_{ne}, c_{sw} and c_{se}

In the worst case, all points will expire in a single time step. However, using an amortization argument and Lemma 4 it can be shown that on average, only $O(k\Delta_k \log n)$ point will expire in each time step.

Lemma 5 *If the number of points $p(t) \in P(t)$ with $\tau_{NNS}(p,t,P) = i$ is at most $\gamma_{NNS}(P)$ for any $0 \leq i \leq n$ and any t, then on average only $O(\gamma_{NNS}(P) \log n)$ points expire per time step [2].*

From Lemma 3 and 5 the following theorem can be concluded:

Theorem 6 *Under the Displacement Assumption, the nearest neighbor in a set P of n points moving to static query point q in the plane, can be maintained in the black-box model in $O(k\Delta_k \log n)$ amortized time per time step, where Δ_k is the maximum k-spread of P at any time.*

3 Conclusion

In this paper, an algorithm is presented to maintain the nearest neighbor in a set points moving to static query point in the KDS black-box model. The algorithm does not require knowledge of k or Δ_k. It only needs to know d_{max}, the maximum displacement of any point in one time step and also does not need to know the point trajectories. It is also shown that the proposed algorithm can update nearest neighbor in $O(k\Delta_k \log n)$ amortized time at each time step. Interesting open problems arise when someone talks about time bound in worst-case rather than amortized.

References

[1] J. Basch, L. J. Guibas and J. Hershenberger, *Data structure for mobile Data*, Journal of algorithms, 31(1): 1-28, 1999.

[2] M. de Berg, M. Roeloffzen and B. Speckmann, *Kinetic convex hulls, delaunay triangulations and connectivity structures in the black-box model*, Journal of Computational Geometry, 3 (1), 222-249, 2012.

[3] M. de Berg, M. Roeloffzen and B. Speckmann, *Kinetic 2-center in the black-box model*, Proceedings of the 29th annual symposium on Symposium on computational geometry, 145-154, 2013.

[4] J. Erickson, *Dense point sets have sparse delaunay triangulations or ... but not too nasty*, Discrete and Computational Geometry, Volume 33, Issue 1, pp 83115, 2005.

[5] L.J. Guibas, *Kinetic data structures a state-ofthe-art report*, In Proc. 3rd Workshop Algorithmic Found. Robot., pages 191-209, 1998.

[6] L.J. Guibas. Kinetic data structures. In: D. Mehta and S. Sahni, *Handbook of Data Structures and Applications*, Chapman and Hall/CRC, 2004.

[7] Agarwal, Pankaj K and Kaplan, Haim and Sharir, Micha. *Kinetic and dynamic data structures for closest pair and all nearest neighbors*, ACM Transactions on Algorithms (TALG), Volume 5 Issue 1, Article No. 4, 2008.

[8] Rahmati, Zahed and Abam, Mohammad Ali and King, Valerie and Whitesides, Sue and Zarei, Alireza, *A simple, faster method for kinetic proximity problems*, Journal of Computational Geometry: Theory and Applications, Volume 48, Issue 4, 342-359, 2015.

Exploring Rectangular Grid Environments

Mansoor Davoodi * Mehdi Khosravian Ghadikolaei† Mohammad Mehdi Malekizadeh‡

Abstract

In this paper we study the problem of grid exploring which is finding a shortest possible tour for a mobile robot moving in an unknown environment. We focus on a rectangular grid with m columns and n rows, denoted by $R(m, n)$, as the environment. We assume the robot can see only the four cells adjacent to its current cell. Under such conditions, we investigate different variants of exploration problem, and propose efficient bounds and algorithms for the shortest tour in which visits all the cells of $R(m, n)$. We show that for odd $R(m, n)$, $mn+1$ and for even $R(m, n)$, mn are the tight lower bounds for the length of the minimum tour, and propose efficient algorithms for finding such a tour. We show that the algorithms are optimal if the starting cell lies on the boundary of $R(m, n)$. Finally we propose a $(1 + \frac{4}{mn})$-competitive algorithm when the robot starts at non-boundary cell.

1 Introduction

Exploration problems is one of the challenges of robot motion planning. Based on type of the environment and the characteristics of the robot, different kinds of exploring problems have been considered (e.g., blindness or visibility of the robot, type of environment and different services). Exploration refers to the task of finding a path, such that every point in the environment is seen from at least one point on the path [4]. In the exploration tour problem, path length minimization is studied in Manhattan and Euclidean metrics. The results have applications in lawn mower, searcher and cleaner robots.

Two models of environment are defined as follows: one without hole and barrier that is called *simple* environment, and one with hole and barrier [7]. When exploring is in a continuous environment, the visibility of the robot could be finite or infinite; but actually robots are usually blind and understand their surroundings using proximity sensors. Also, some tasks in the framework of exploring are considered for robots which necessitates their presence in the environment like lawn mowers that need to move all over the environment to cut the grass.

The grid exploration problem consists of finding the shortest possible tour, which visits every cell of a grid at least once [7]. We call two cells adjacent, if they share a common edge. At each step, the robot can move to an adjacent cell along the four main directions –north, south, east and west–. We assume that the cells have unit size, therefore, the length of the path is equal to the number of steps from one cell to the another and the robot has enough memory to store a map of known cells.

There are two variants of the grid exploration problem. In the *offline* variant, the robot gets a map as an input and computes a tour. In the *online* variant, the robot has limited visibility and can recognize only the four adjacent, without any prior knowledge [7]. A classical graph theory problem named Hamiltonian cycle is closely related to the exploration problem which it consists in determining whether or not a given graph contains a Hamiltonian cycle, i.e., a cycle which visits every vertex exactly once. This problem is well known to be $NP-complete$ [9]. Umans and Lenhart [12] presented an $O(n^4)$ time algorithm for finding Hamiltonian cycles in grids without hole, where n is the number of cells. Salman [11] introduced alphabet grid graphs and determined classes of alphabet grid graphs which contain Hamiltonian cycles, moreover, it is demonstrated that if an $R(m, n)$ (rectangle with m columns and n rows in which $m, n \geq 2$) is odd ($m \times n$ is odd), then the rectangle is non Hamiltonian and also is presented a pattern to find a Hamiltonian cycle for the rectangular grid graph for any even number m or n. Itai et al. [6] presented necessary and sufficient conditions for existence of Hamiltonian paths in rectangular grid graphs and proved that the problem for general grid graphs (can contains holes) is $NP - complete$, which implies that the grid exploration problem is $NP - hard$ as well [6]. This result is particularly interesting because it demonstrates that allowing holes in the input grids can make the problem much harder. Arkin et al. [1] presented an offline algorithm which achieves an approximation factor of $\frac{6}{5} = \frac{48}{40}$ in grids without holes and a factor of $\frac{53}{40}$ if it contains holes.

For online variant of grid exploration, Gabriely and Rimon [3] presented an upper bound $C + B$ for the

*Department of Computer Science and Information Technology, Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan, Iran, mdmonfared@iasbs.ac.ir

†LAMSADE Paris Dauphine University, mehdi.khosravian-ghadikolaei@dauphine.fr

‡Department of Computer Science and Information Technology, Institute for Advanced Studies in Basic Sciences (IASBS), Zanjan, Iran, malekizadeh.mehdi90@gmail.com

length of exploration tour that C is the number of cells and B is the number of boundary cells. Icking et al. [5] presented an upper bound $C + \frac{1}{2}E - 3$ for length of exploration tour where E denotes the number of boundary edges and showed that the best possible competitive ratio is 2 for general grids. Miyazaki et al. [10] proved that an online version of depth-first search achieves this ratio, therefore, the focus moved to exploration of grids without holes. Icking et al. [5] described an online $\frac{4}{3}-$*competitive* algorithm which assumes that the robot starts from the boundary of the grid. This ratio was improved to $\frac{5}{4} = \frac{20}{16}$ by Kolenderska et al. [8], who also showed that the best possible ratio is at least $\frac{20}{17}$. In this paper we show that the length of exploration tour in the grid environment which is always 2-colorable is: $S \geq C + \big||V^B| - |V^W|\big|$, where V^B is the set of black cells and V^W is the set of white ones. We also show that the length of optimal exploration tour in an $R(m,n)$ (rectangle with m columns and n rows in which $m, n \geq 2$) is equal to $C+1$ if $R(m,n)$ is odd ($m \times n$ is odd) and C if R(m,n) is even ($m \times n$ is even), where $C = m \times n$. For the offline variant of exploring, we present an algorithm to compute the optimal exploration tour in a grid rectangle. This algorithm takes time linear in the length of the path, *O(mn)*.

We prove that, every strategy in the online variant for exploring of an $R(m,n)$ with C cells needs at least $C+2$ steps, therefore, the lower bound of competitive ratio is equal to $1+\frac{2}{C}$. We also present an optimal algorithm to compute the exploration tour in the grid rectangle by an assumption that the starting cell lies on the boundary. Finally we present a $(1+\frac{4}{C})-$*competitive* algorithm when the robot starts at a non-boundary cell.

2 A Lower Bound

The grid graph corresponding to a grid environment consists of one node for every cell in the grid environment. Two nodes are connected by an edge, if their corresponding cells are adjacent.

A graph is bipartite if its nodes can be partitioned into two sets, so that all edges have one endpoint in each set. Every bipartite graph is 2-colorable and has no cycle with odd length.

Proposition 1 *Grid graphs are bipartite.*

Every bipartite graph is 2-colorable. Moreover we can set white color to each odd node of the grid graph and black to the even nodes. This implies that each node in a grid graph has at most four adjacent with different color.

Theorem 2 *Every strategy for exploring a grid environment with C cells needs at least $C + \big||V^B| - |V^W|\big|$ steps, where $|V^B|$ and $|V^W|$ are the number of black and white cells, respectively.*

Proof. The grid environment is bipartite. Therefore, to explore all cells we have a cycle with consequently movement of black and white cells, because in each step of exploration we enter to a cell which has different color from the previous cell. Thus, if $\big||V^B| - |V^W|\big| = k \neq 0$, to explore all the cells in the larger set, we need to explore an equal number of cells in the other set which means k extra visited cells. □

3 Optimal Exploration Tour in $R(m,n)$

In the offline variant of the grid exploration problem, the entire grid is provided as input and the goal is to determine a shortest exploration tour. Even though we know that it is $NP-hard$ to solve grid exploration in general grids [6], the difficulty of the problem seems to vary greatly depending on whether or not the grids are allowed to have some holes. In the grid environment without hole, exploration problem is still open. however, we present an algorithm to explore an $R(m,n)$ in time linear in the length of the path, $O(mn)$.

Lemma 3 *(see [2]). $R(m,n)$ has a Hamiltonian cycle if and only if $m \times n$ is even.*

So the optimal exploration tour in $R(m,n)$ is Hamiltonian cycle too.

Lemma 4 *(see [11]). $R(m,n)$ contains no Hamiltonian cycle if $m \times n$ is odd.*

So, the length of exploration tour in $R(m,n)$ is $S \geq (m \times n) + 1$.

Lemma 5 *(see [6]). $(R(m,n), s, t)$ has a Hamiltonian path with started node s and final node t if:*

Necessary conditions:

1. *R is even ($|V^B| = |V^W|$) and s and t have different color or*
2. *R is odd ($|V^B| = |V^W| + 1$) and s and t are colored by majority color.*

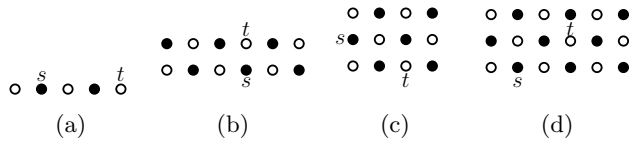
   And each the following conditions for the graph to have no s,t Hamiltonian path:

3. *$R(m,1)$ is a 1-rectangle and either s or t is not a corner vertex (Figure 1(a)).*
4. *$R(m,2)$ is a 2-rectangle and (s,t) is a non-boundary edge, that (s,t) is an edge and it is not on the outer face (Figure 1(b)).*
5. *$R(m,3)$ is a isomorphic to a 3-rectangle $R'(m,3)$ such that s and t are mapped to s' and t' and all of the following three conditions hold:*

(a) *m is even,*

(b) *s' is black, t' is white,*

(c) *${s'}_y = 2$ and ${s'}_y < {t'}_x$ (Figure 2(c)) or ${s'}_y \neq 2$ and ${s'}_y < {t'}_x - 1$(Figure 1(d)).*



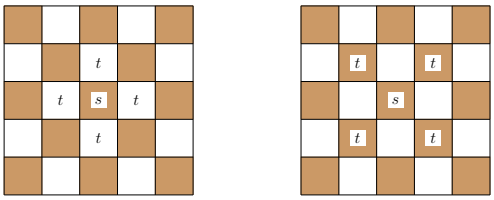
(a) (b) (c) (d)

Figure 1: Rectangular grid graphs in which there is no Hamiltonian path between s and t.

Then finding Hamiltonian path is done in liner time.

Corollary 6 *The length of the optimal exploration tour is $m \times n$ if $R(m,n)$ is even, and $(m \times n)+1$ if $R(m,n)$ is odd.*

Proof. By considering Lemma 3, if $R(m,n)$ is even, we have a Hamiltonian path which starts from s and ends at one of four adjacent of s (Figure 2(a)). Thus, we can change our Hamiltonian path to exploration tour by moving toward s in one step. If $R(m,n)$ is odd, we have a Hamiltonian path which starts from s and ends at t, one of four diagonal neighbors of s (Figure 2(b)). Therefore, we can change our Hamiltonian path to exploration tour by moving toward s using traversing one common adjacent between t and s. □



(a) Hamiltonian path in even rectangle (b) Hamiltonian path in odd rectangle

Figure 2: Conditions for Hamiltonian path in $R(m,n)$.

Corollary 7 *Finding an optimal exploration tour problem in an $R(m,n)$ without hole can be solved in time linear in the length of the path, $O(mn)$.*

4 Competitive Complexity

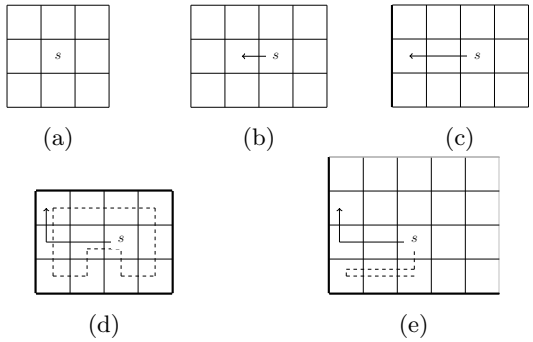
In the online variant of the grid exploration problem the robot has a limited visibility and must explore the environment from a starting cell with no prior knowledge. Thus, the first question is whether the robot is still able to find the optimal solution or has to approximate the solution with a constant factor. There is a quick answer to this question.

Theorem 8 *Every strategy in the online variant for the exploration of an $R(m,n)$ with C cells needs at least $C+2$ steps.*

Proof. Since the robot knows nothing about the dimensions of $R(m,n)$ and its position in the environment, generally there are two different strategies for the robot's movements in two prior steps (Figure 3(a)):

1. First Strategy

This strategy decides to walk two steps to the west and by these movements robot meets the boundary of the environment (Figure 3(c)). The robot has two choices for the next step, move toward either the north adjacent or south adjacent. Without loss of generality, assume the robot moves to the north one (Figure 3(d)). In this state, the robot needs at least two additional steps for exploring the environment (Figure 3(d)). We can easily extend this pattern to build any rectangular environment of arbitrary size by extending height and width toward the north and east, respectively (Figure 3(e)). We can show easily that if the two first steps are toward another directions (north, south or east), the result is hold as well.



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

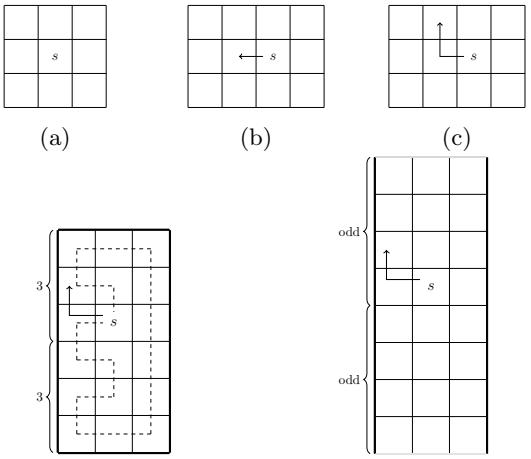
Figure 3: Tight example for two additional steps in $R(m,n)$ with First Strategy. Dashed lines is the optimal exploration tour.

2. Second Strategy

This strategy decides to move two steps toward perpendicular direction (Figure 4(c)). We close our rectangle as shown in Figure 4(d). The robot must continue its exploration in two odd rectangles with width 3. Considering Corollary 1, we know the length of exploration tour is $C+1$ in each *odd* rectangle, where C is the number of cells. So, the strategy needs at least two additional steps for exploring the whole environment. We can easily extend this pattern to build rectangular environment of arbitrary size by extending the height toward the north and south as the height of each new rectangle is odd (Figure 3(e)). We can easily show that if the two first steps are toward the other perpendicular directions (east-south, west-north,...), our final result is hold.

By these two cases, we have shown that using any strategies in online variant, it is need at least $C+2$ steps to explore $R(m,n)$, whereas the optimal strategy in offline variant needs C steps (Figure 3(d), Figure 4(d)). □

Corollary 9 *Every strategy for the exploration of a rectangular grid environment with C cells is at least $1+\frac{2}{C}$-competitive.*



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

Figure 4: Tight example for two additional steps in $R(m,n)$ with Second Strategy. Dashed lines is optimal exploration tour.

Proof. The tight example is obtained by exploring the environment which is shown in Figure 3(d) and Figure 4(d). The length of the optimal tour is C, but the length of robot's tour which leads by an online strategy is at least $C+2$.

$$\frac{S_{online}}{S_{optimal}} = \frac{C+2}{C} = 1 + \frac{2}{C}$$

□

5 Algorithms of Exploration

5.1 Patterns of exploration in offline variant

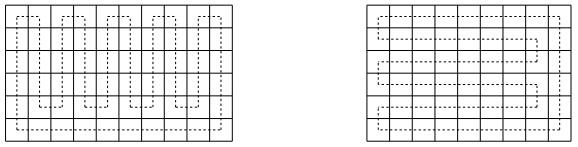
In Section 3, we proved that the exploration tour in the offline variant can be found in linear time in the length of the path, $O(mn)$. In this section, we present two patterns to explore each rectangular grid. Depending on even or odd rectangular grid these patterns can be algorithmically extended to $R(m,n)$, for any m and n.

1. *R(m,n)* is even:

Considering Lemma 1, if $R(m,n)$ is even, we have an exploration tour with the length C, where C is $m \times n$. Figure 5 gives an illustration of exploration tour with 2 examples $R(10,6)$ and $R(9,6)$. The patterns in this figure can be used for finding an optimal exploration tour of $R(m,n)$ for any even number m or n.

2. *R(m,n)* is odd:

Considering Corollary 1, if $R(m,n)$ is odd, we have an exploration tour of length $C+1$. An optimal exploration tour for $R(9,7)$ is shown in Figure 6. The pattern in this figure can be used for finding an optimal exploration tour of $R(m,n)$ for any odd number m and n, where $m,n \geq 3$.



(a) Exploration tour in $R(10,6)$. (b) Exploration tour in $R(9,6)$.

Figure 5: Patterns of exploration tour.

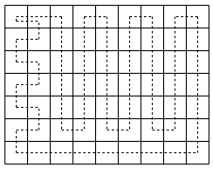
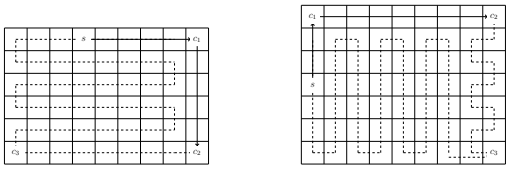


Figure 6: Exploration tour in $R(9,7)$.

5.2 Algorithms of exploration in online variant

In this section we present two algorithms for exploring $R(m,n)$ in online variant. If the robot starts at a boundary cell, we present an optimal exploration algorithm and when the starting cell is a non-boundary cell we present an $(1+\frac{4}{C})-$*competitive* exploration algorithm.



(a) Exploration tour in $R(9,6)$. (b) Exploration tour in $R(9,7)$.

Figure 7: Examples for algorithm's output in the online variant of exploring rectangular grid environment.

1. The starting cell lies on the boundary:

We present an algorithm to compute exploration tour of $R(m,n)$ when the starting cell lies on the boundary. The robot is able to recognize either the number of passing cells is even or odd. There are four possible directions (north, south, east and west) for the robot to move from one cell to an adjacent cell. Command CW denotes rotated clockwise in the environment. Every corner of the environment has only two adjacent and the robot can recognize them. The robot begins his exploration from starting cell in CW direction until reaches the first corner. Then he moves to the next corner and determines the number of cells between the corners is odd or even. If it is even, the robot must walk CW to reach the third corner and explores the remaining cells by the zigzag form between columns (rows) to reach the starting cell (Figure 7(a)). Otherwise, the robot explore between two rows (columns) by the zigzag form to reach

the third corner and after that continue his exploration by the zigzag form between columns (rows) to reach the starting cell (Figure 7(b)).

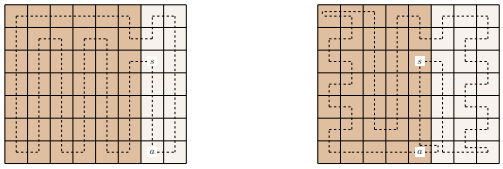
Algorithm 1 BOUNDRAY CONSTRAINT

1: Walk CW to reach the first corner–called c_1.
2: Walk CW to reach the second corner–called c_2.
3: **if** $|c_1c_2|$ is even **then**
4: Walk CW to reach the third corner–called c_3.
5: Explore all rows (columns) by the zigzag form parallel to c_2c_3 to reach the starting cell.
6: **else**
7: Walk between two last rows (columns) by the zigzag form parallel to c_1c_2 until reach third corner–called c_3.
8: Explore the rest rows (columns) by the zigzag form parallel to c_2c_3 to reach the starting cell.
9: **end if**

We can easily prove that our algorithm finds optimal exploration tour. If the number of cells from c_1 to c_2 (denoted by $|c_1c_2|$) is even, then $R(m,n)$ is even, hence we can continue our exploration using defined pattern in the offline variant. In the other case, $|c_1c_2|$ is odd, hence $R(m,n)$ can be either odd or even. If $|c_2c_3|$ is odd, our first zigzag strategy ends at c_3. In this state, for continuation of exploration, we have to explore a visited cell again and after that it is easy explore the remaining rows (columns) by the zigzag form to reach the starting cell without visiting any extra cells. Therefore the algorithm finds the optimal exploration tour.

2. The robot starts at non-boundary cell:

We present a new algorithm for the case in which the robot starts at a non-boundary cell. The main idea in this algorithm is to subdivide the whole environment into two smaller rectangular environments based on the column of starting cell. In this algorithm the robot begins his movement toward the south until reaches the boundary cell, denoted by a. Then the robot continues his exploration by one step toward east. If this cell is a boundary cell, the robot continues his movement to explore the right rectangle and after that explores the left rectangles using Algorithm 1 (Figure 8(a)). Otherwise, the robot goes back to a and explores the left and right rectangles, respectively, using Algorithm 1 (Figure 8(b),(c)).



(a) Optimal exploration. (b) A tight example.

Figure 8: Examples for algorithm's output in the online variant of exploring rectangular grid environment.

Algorithm 2 GENERAL START POSITION

1: Walk to the south to reach a boundary cell–called a.
2: Walk to the east cell.
3: **if** is not a corner cell **then**
4: Go back to a.
5: Explore rectangle induced by s column and west cells by Algorithm 1 until reach the cell adjacent to the north s.
6: Walk to east cell.
7: Explore the remaining cells by Algorithm 1 until reach s.
8: **else**
9: Walk to the north until reach boundary.
10: Walk to the west cell.
11: Walk to the south until reach the cell adjacent to the north s.
12: Walk to the west cell.
13: Explore the remaining cells by Algorithm 1 until reach s.
14: **end if**

Theorem 10 *The algorithm Genaral Start Position is* $(1+\frac{4}{C})-$*competitive.*

Proof. Suppose $R(m,n)$ is even in which m is even and n is odd. Suppose s lies on the m'-th column of the rectangle. In the worst case, if m' is odd, we have two odd rectangles $R(m',n)$ and $R(m-m',n)$ such that . However, both subdivided rectangles are odd and we can explore them with one additional step by using Algorithm 1. Also, as it is shown in Figure 8(c), the robot visits cell a and the eastern adjacent cell of a twice. So, we have four extra cell, in general.

$$\frac{S_{online}}{S_{optimal}} = \frac{C+4}{C} = 1+\frac{4}{C}$$

□

6 Conclusion

Different variants of online exploring in a rectangular grid $R(m,n)$ for a single robot have been studied in this paper. Efficient bounds and algorithms have been proposed depending on the odd or even size of $R(m,n)$ and also locus of starting position. In all of these cases, we propose almost optimal online algorithms linear to the length of the output path. As a future work, investigating the problem for two or more robots is suggested. In fact in this paper, we assumed a very limited visibility for the robot, while it seems efficient collaborating robots under such assumption is challenging.

References

[1] E. M. Arkin, S. P. Fekete, and J. S. Mitchell. Approximation algorithms for lawn mowing and milling. *Computational Geometry*, 17(1):25–50, 2000.

[2] S. D. Chen, H. Shen, and R. Topor. An efficient algorithm for constructing hamiltonian paths in meshes. *Parallel Computing*, 28(9):1293–1305, 2002.

[3] Y. Gabriely and E. Rimon. Competitive on-line coverage of grid environments by a mobile robot. *Computational Geometry*, 24(3):197–224, 2003.

[4] C. Icking, T. Kamphans, R. Klein, and E. Langetepe. Exploring an unknown cellular environment. In *EuroCG*, pages 140–143, 2000.

[5] C. Icking, T. Kamphans, R. Klein, and E. Langetepe. Exploring simple grid polygons. In *Computing and Combinatorics*, pages 524–533. Springer, 2005.

[6] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

[7] T. Kamphans. *Models and algorithms for online exploration and search*. PhD thesis, University of Bonn, 2006.

[8] A. Kolenderska, A. Kosowski, M. Małafiejski, and P. Żyliński. An improved strategy for exploring a grid polygon. In *Structural Information and Communication Complexity*, pages 222–236. Springer, 2010.

[9] J. A. McHugh. *Algorithmic graph theory*. Prentice Hall Englewood Cliffs, 1990.

[10] S. Miyazaki, N. Morimoto, and Y. Okabe. The online graph exploration problem on restricted graphs. *IEICE transactions on information and systems*, 92(9):1620–1627, 2009.

[11] A. Salman, E. Baskoro, and H. Broersma. A note concerning hamilton cycles in some classes of grid graphs. *Journal of Mathematical and Fundamental Sciences*, 35(1):65–70, 2003.

[12] C. Umans and W. Lenhart. Hamiltonian cycles in solid grid graphs. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 496–505. IEEE, 1997.

Shortest Path Problem among Imprecise Obstacles

Zahra Gholami* Mansoor Davoodi†

Abstract

In this paper, we consider the shortest path problem among a set of imprecise segments as obstacles. In the precise context, each segment is defined by its endpoints. Here, we assume an uncertainty region for the endpoints of the segments and study the problem of arranging the obstacles by placing a point inside each uncertainty region in such a way that maximizes the possible shortest path. We prove the NP-completeness of the maximum shortest path problem for both the Euclidean and Manhattan metrics and propose a $\frac{1}{2}$-approximation algorithm for it when the uncertainty regions are modelled as independent disjoint disks. If the regions are dependent on separation factor k, we obtain the approximation factor of $1 - \frac{2}{k+4}$.

1 Introduction

Regarding the widespread applications of shortest path problems in motion planning, VLSI design and network wire-routing, the necessity of investigation over shortest path problems has currently become evident. It is unsurprising, therefore, that finding a shortest path for robots or points within a workspace with obstacles has been under investigation as an intriguingly applicable problem. Taking some constraints and properties of obstacles or workspaces into consideration, various approaches like visibility graph [8] have been suggested for the shortest path problem. These approaches assume the workspace, data processing, and motions completely in a precise manner, however, this assumption is not realistic. For instance, in the robot motion planning, many uncertainties such as robot's sensing and acting are inevitable. In addition to the mechanical constraints of robots, a raft of data is inaccessible due to the different sources of error such as collecting real data about the world and its dynamical properties. Clearly, these uncertainties make these approaches inefficient under uncertainty and, consequently, imprecision consideration will draw a more complete and accurate picture of finding shortest paths.

Region-based models [3, 12], are popular approaches to model the imprecision. In this models, the precise point may appear anywhere in the region with a uniform probability. The goal of the region-based models for handling imprecision is to find the critical point for each geometric region in order to minimize or maximize specific values. For example, Löffler and van Kreveld discussed the convex hull of imprecise points in various types of regions which maximize or minimize area/perimeter of the convex hull [12]. For each variant, they either provide an NP-hardness proof or a polynomial-time algorithm. As another example, the problem of finding Minimum Spanning Tree (MST) for imprecise points, turn out to be the problems of finding the Minimum and Maximum-weight MST. This problem has been studied by Dorrigiv et al. [6] under disk-shaped uncertainty regions.

Regarding to the applications of the shortest path problem, it has been considerably studied under different parameters close to uncertainty conditions. For a sequence of simple polygons and two points s and e, the Touring Polygons Problem (TPP) is looking for a tour from s to e so that all polygons are visited in the given order. A more general form of TPP is the Shortest Path Problem (hereafter: SPP) for imprecise points. In this problem, a graph of polygons is given instead of an order of polygons. In a directed graph, traversal between vertices is only allowed through the edges. The aim of SPP is to find a placement of the vertices which minimizes the shortest distance between s and e. The maximum variant of SPP has been studied which searched for such a placement that maximizing the shortest path length.

For this problem in the case of significant uncertainty of the arc lengths, Yu et al. [14] presented exact and heuristic solutions. Dror et al. in [7] showed that for convex and disjoint polygons TPP is solvable in polynomial time. NP-hardness of such a problem has been proved for any metric, L_p, $p \geq 1$ in case of non-convex polygons (i.e. they are disjoint [1] or overlapping polygons [7]). Moreover, some approximation algorithms are given for TPP in cases for which the polygons are non-convex [13]. Also, the maximum variant of TPP is explored by Disser et al. in [5] that provided a polynomial time algorithm for computing a maximum placement.

In general, SPP is NP-hard for any metric L_p, $p \geq 1$. Disser et al. in [4] showed that for axis-aligned rectilinear polygons (not necessarily convex) under Manhattan metric, proposing a polynomial time algorithm is feasible. Their study in [5] shows that the problem is hard to

*School of Computer Science and Software Engineering, The University of Western Australia, eli.gholami@uwa.edu.au

†Department of Computer Science and Information Technology, Institute for Advanced Studies in Basic Sciences, Zanjan, mdmonfared@iasbs.ac.ir

approximate for any approximation factor $(1-\epsilon)$ with $\epsilon < 1/4$, even when the polygons consist of only vertically aligned segments.

In this paper, we define an imprecise segment as an obstacle whose endpoints are some regions instead of points. We study the problem of maximum possible shortest paths' length. Our goal in Maximum Shortest Path Problem (hereafter: Max-SPP) is placing a point inside each region (a placement) in order to arrange the obstacles such that the shortest path from s to e becomes maximum. In other words, Max-SPP is SPP in continuous space instead of the graph.

This paper makes the following contributions:

1. We prove NP-completeness for the decision version of *Max-SPP* when the uncertainty regions are modelled as segments in the region-based models.

2. When the uncertainty regions have been modelled as disjoint disks, we propose a $\frac{1}{2}$-approximation algorithm for Max-SPP. Also in cases where the regions are *k-separable* disks (see Definition 4) we show that the approximation factor of the algorithm is $1-\frac{2}{k+4}$.

2 Problem Formulation

Free Space: In this work, we assume points of s and e to be located within the free space. In the precise manner, the free space is introduced as all points in the workspace that do not belong to any obstacles. However, in the imprecise manner, the free space refers to all points that do not belong to any obstacles for all possible placements. So, there are no placements for which obstacles contain points of s and e. In other words, s and e are not allowed to be located in the obstacles for any possible placement.

For a robot, we consider a workspace containing start point s, endpoint e in free space and a set of imprecise segments as obstacles. We define the imprecise points or regions as a set

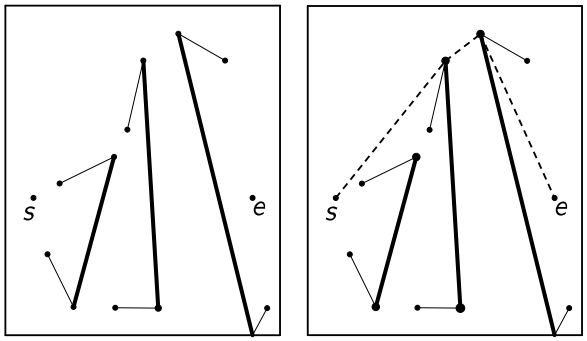
$\mathcal{R} = \{R_1, R_2, R_3, ..., R_n\}; R_i \subset \mathbb{R}^2, 1 \leqslant i \leqslant n$. Where n is the number of obstacles' endpoints in the workspace. Suppose $\mathcal{I}$ to be a set of points that we achieve by placing a point or *instance* inside each region of $\mathcal{R}$, like the placement

$$\mathcal{I} = \{I_1, I_2, I_3, \ldots, I_n\}; I_i \in R_i, 1 \leqslant i \leqslant n. \qquad (1)$$

If $\mathcal{L}(\mathcal{I})$ refers to the length of the shortest path from s to e for the placement $\mathcal{I}$, then in the **Max-SPP**, the goal is to maximize $\mathcal{L}(\mathcal{I})$ by setting a placement like

$$\mathcal{I}^{max} = \{I_1^{max}, I_2^{max}, I_3^{max}, \ldots, I_n^{max}\}; I_i^{max} \in R_i, 1 \leqslant i \leqslant n \qquad (2)$$

$(\mathcal{I})^{max}$ represents a placement which maximizes the shortest path length between s and e.



(a) (b)

Figure 1: (a) A workspace containing start and endpoint of shortest path, segments as obstacles and their uncertain endpoints which are modeled as a line segments. (b) Placing a point for each region of uncertainty in order to maximize the distance between s and e.

The Decision Version of Max-SPP:

Input: $\mathcal{R}$ as a set of imprecise points, points of s and e, and a length value of B.

Output: YES if there exists a placement like $\mathcal{I}$ that $\mathcal{L}(\mathcal{I}) \geqslant B$, NO otherwise.

The **Existence Path Problem** (hereafter: **EPP**) is a problem whose answer is YES if there exists a path from s to e for at least a set of $\mathcal{I} \subset \mathcal{R}$, NO otherwise.

3 Maximum Shortest Path Problem (Max-SPP)

In this section, with the help of reduction from the SAT problem to Max-SPP, we show the hardness results for Max-SPP. We assume a simple case of Max-SPP in which the imprecise regions are modelled by segments. Since in this case the approach for NP-hardness proof of the Largest Convex Hull problem in [12] is not applicable for Max-SPP, we add some crucial obstacles to the workspace.

For a given SAT instance (formula ψ), we construct a Max-SPP instance. For this, we setup $\mathcal{R}(\psi)$ including imprecise obstacles's endpoint. Then, we prove that the decision version of Max-SPP returns YES if and only if the SAT formula ψ is satisfiable.

As illustrated in Fig. 2(a), for converting the SAT formula to the Max-SPP instance, we divide a circle into $M = c + q$ arcs, where c and q are the numbers of clauses and variables in formula ψ, respectively. The c clauses and q variables of ψ are characterized as arcs in the Max-SPP instance. This unique circle contains one arc for each clause and one arc for each variable as well as two points s, e and M separator points that separate arcs from each other. We locate the points s and e by some sufficiently small $\epsilon > 0$ above and below the separator point of z (Fig. 2(a)). In addition, to insert

some obstacles we draw segments from circle center at o to all separator points and a segment from point of z to the workspace boundary.

Variable Arcs Configuration: As Fig. 2(b) shows, for each variable in ψ like v, we have an arc that contains: a segment parallel to $\overline{lr}$ (shown as $\overline{tf}$), and two sets of points, P_v and Q_v, with the same number of elements equal to $3c$.

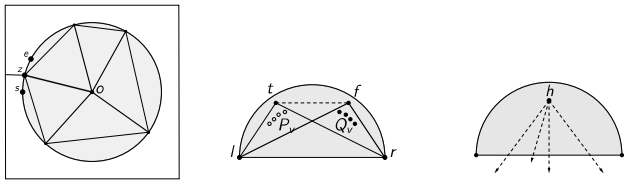
Notably, although the points in P_v corresponding to each variable like v are placed such that they are all on the convex hull of $\{l, r, f\}, P_v$ and Q_v, they are not on the convex hull of $\{l, r, t\}, P_v$ and Q_v. As shown in Fig. 2(a), points in Q_v are symmetrical with P_v.

Clause Arcs Configuration: As Fig. 2(c) shows, for each clause in ψ like c we have an arc containing a point h_c. If the variable v appears in clause c as a positive literal, we connect the point h_c to a member of P_v. If the variable v appears in clause c as a negative literal, we connect the point h_c to a member of Q_v. In this way, the obstacles as precise and imprecise regions would be produced.

- the connection between the workspace boundary and the separator point of z as a precise obstacle.
- the connection between the circle center at o and all separator points as precise obstacles.
- the connection between the circle center at o and the imprecise regions with an endpoint h_c and the other in P_v or Q_v sets as imprecise obstacles.
- the connection between the circle center at o and the imprecise regions with endpoints at t and f as imprecise obstacles.

Finally, in the workspace constructed by formula ψ, maximizing the shortest path between s and e is now equal to the sum of the maximized shortest paths between two separator points. So, in order to maximize the shortest path between two separator points (which locate on a single arc) for every variable arc like v, the selected endpoint should be either t together with all points in Q_v or f together with all points in P_v. Moreover, for the optimal placement of $\mathcal{I}^{max}$ point h_c should be selected in each clause arc such as c.

Theorem 1 *Suppose we are given a workspace containing a set of segment obstacles and a set of imprecise points as obstacles' endpoints. Obstacles are assumed to be arbitrary segments that can have common intersections only at their endpoints. For such a workspace with these imprecise obstacles, Max-SPP is NP-hard under the Euclidean metric and its decision version is NP-complete.*



(a) (b) (c)

Figure 2: (a) A circle is divided into arcs in which all the segments passing through o represent obstacles. The points s and e located in some sufficiently small $\epsilon > 0$ above and below the separator point of z. (b) A variable arc. (c) A clause arc.

4 Approximation Algorithm for Max-SPP

Regarding the NP-hardness of Max-SPP, the approximation algorithms could be used for estimating the solution. For approximating the optimal placement of Max-SPP, we focus on those workspaces which their obstacles are just segments and their imprecise endpoints are disjoint disks. Our approximation algorithm simply selects center of disks as placement $\mathcal{I}$ (i.e. as an approximate placement). Similar to [6], we have proved that the result of this simple algorithm, $\mathcal{L}(\mathcal{I})$, is not smaller than half of that in optimal placement.

Definition 1 *Let* $\boldsymbol{\mathcal{L}(\mathcal{I}^{center})}$ *and* $\boldsymbol{SP(\mathcal{I}^{center})}$ *denote respectively the solution value and the shortest path of the approximation algorithm that selects the disks' centers as the placement* $\mathcal{I}$.

Definition 2 *We define* $\boldsymbol{SP(\mathcal{I}^{max})}$ *and* $\boldsymbol{\mathcal{L}(\mathcal{I}^{max})}$ *as the shortest path and its length in the optimal placement for Max-SPP, respectively.*

Definition 3 *We suppose* $\boldsymbol{SP'(\mathcal{I}^{max})}$ *is the path from* s *to* e *with the same topology*[1] *as* $SP(\mathcal{I}^{center})$ *and with the point of* $(\mathcal{I}^{max})$. *Noticeably, this path is not necessarily the shortest path.*

Let $\mathcal{L}(\mathcal{I}^{max})$ and $\mathcal{L}(SP'(\mathcal{I}^{max}))$ stand for the length of paths from s to e for paths $SP(\mathcal{I}^{max})$ and $SP'(\mathcal{I}^{max})$, respectively. Then we have

$$\mathcal{L}(\mathcal{I}^{max}) \leqslant \mathcal{L}(SP'(\mathcal{I}^{max})) \tag{3}$$

Theorem 2 *Consider a workspace such that the imprecise obstacles' endpoints are disjoint disks. Now, in the approximation algorithm assuming the center of all disks as the placement* $\mathcal{I}^{center}$ *for Max-SPP, we have*

$$\frac{1}{2}\mathcal{L}(\mathcal{I}^{max}) \leqslant \mathcal{L}(\mathcal{I}^{center}) \tag{4}$$

[1]The sequence of obstacles and their endpoints throughout a path.

If the disks are sufficiently far from each other, the approximation factor of the algorithm in Theorem 2 will be improved to more accurate values. So, in the following, we prove that the larger the distances between disks, the better the approximation factor we get (i.e. closer to 1).

Definition 4 *As defined by [10], a given set of disks with the largest radius r_{max} are **k-separable** when for the maximum value of k the minimum distance between each pair of disks is at least $k.r_{max}$.*

Theorem 3 *Consider a workspace such that the imprecise obstacles' endpoints are k-separable disks with $k > 0$. Now, in the approximation algorithm assuming the center of all disks as the placement $\mathcal{I}^{center}$ for Max-SPP, we have*

$$(1 - \frac{2}{k+4})\mathcal{L}(SP(\mathcal{I}^{max})) \leqslant \mathcal{L}(SP(\mathcal{I}^{center})) \quad (5)$$

Proof. See the Appendix. □

Now, obviously, farther disks (i.e. larger value of k) leads the algorithm to more accurate approximation factors (i.e. closer to 1).

5 Conclusion

In this paper, we modelled the imprecise points by using some geometric approaches and proved that the Maximum Shortest Path Problem (Max-SPP) is NP-hard and its decision version is NP-complete. For this proof, we considered the obstacles to be segments and their endpoints to be imprecise points modelled as segments. Remarkably, the obstacles can only be intersected at their endpoints. In addition, we presented an approximation algorithm with approximation factors of $1/2$ and $1-\frac{2}{k+4}$ for disk and *k-separable* disk as imprecise points, respectively.

A possible future work includes the investigation of the hardness of Max-SPP for different shapes which imprecise points could be modelled with.

References

[1] Ahadi, A., Mozafari, A., Zarei, A.:Touring disjoint polygons problem is NP-hard. In Combinatorial Optimization and Applications (pp. 351-360). Springer International Publishing (2013)

[2] Choset, H., M., Ed.: Principles of robot motion: Theory, Algorithms, and Implementation MIT press (2005)

[3] Davoodi, M., Mohades, A.: Data imprecision under λ-geometry model: Range searching problem. Scientia Iranica, 20(3), pp. 663-669 (2013).

[4] Disser, Y., Mihalk, M., Montanari, S., Widmayer, P.: Rectilinear Shortest Path and Rectilinear Minimum Spanning Tree with Neighborhoods. In Combinatorial Optimization. Springer International Publishing, 208-220 (2014)

[5] Disser, Y., Mihalk, M., Montanari.: Max Shortest Path for Imprecise Points. In EuroCG, (2015)

[6] Dorrigiv, R., Fraser, R., He, M., Kamali, S., Kawamura, A., López-Ortiz, A., and Seco, D.: On Minimum and Maximum-weight Minimum Spanning Trees with Neighborhoods. In: Approximation and Online Algorithms. Springer Berlin Heidelberg, 93-106 (2013)

[7] Dror, M., Efrat, A., Lubiw, A., Mitchell, J. S.: Touring a sequence of polygons. In Proceedings of the thirty-fifth annual ACM symposium on Theory of computing, 473-482 (2003)

[8] Ghosh, S. K., Mount, D. M.: An output-sensitive algorithm for computing visibility graphs. SIAM Journal on Computing, 20(5), 888-910 (1991)

[9] LaValle, S., M.: Planning Algorithms. Cambridge university press (2006)

[10] Lichtenstein, D.: Planar Formulae and Their Uses. SIAM journal on computing, 11(2), 329-343 (1982)

[11] Löffler, M., van Kreveld, M.: Largest Bounding Box, Smallest Diameter, and Related Problems on Imprecise Points. In: Algorithms and Data Structures, Springer Berlin Heidelberg, pp. 446-457 (2007)

[12] Löffler, M., van Kreveld, M.: Largest and smallest convex hulls for imprecise points. Algorithmica, 56(2), 235-269 (2010)

[13] Pan, X., Li, F., Klette, R.: Approximate shortest path algorithms for sequences of pairwise disjoint simple polygons, 175-178 (2010)

[14] Yu, G., Yang, J.: On the robust shortest path problem. Computers and Operations Research, 25(6), 457-468 (1998)

[15] Surmann, H., Huser, J., Wehking, J.: Path Planning far a Fuzzy Controlled Autonomous Mobile Robot. In: Fifth IEEE International Conference on Fuzzy Systems, Vol. 3, pp. 1660-1665. IEEE (1996)

On some VC-combinatorial notions in computational geometry

Alireza Mofidi*

Abstract

We study some VC-combinatorial and asymptotic aspects of certain set systems, in particular those with more importance in the sense of applications in computational geometry, extremal combinatorics and learning theory.

1 Introduction

Several aspects of interactions between combinatorial features of set systems and computational geometrical properties of them have been explored in the context of VC-theory and discrepancy theory. For example many connections between notions of VC-dimension, VC-density, ϵ-net theorem and (p,q)-theorem from combinatorial sides and sampling methods and geometric algorithms from computational geometric side has been vastly investigated. The theory of Vapnik-Chervonenkis (abbreviated by VC-theory) is a powerful combinatorial theory discovered by Vapnik and Chervonenkis (see [12]) and has been investigated since the early seventies. This theory is developed by both combinatorists and computer scientists in order to capture certain complexities in some mathematical objects which are called set systems and naturally appear in several situations in both disciplines. Note that different strong connections between this theory and many other fields such as logic and model theory has been discovered. Nowadays this theory is a foundation and essential ingredient of several fields and have found numerous applications in learning theory, statistics, extremal combinatorics, model theory and computational geometry. One can see for example chapter 44 of [3] (with the title "the discrepancy method in computational geometry (B. Chazelle)") for a survey of applications of this theory in computational geometry and also algorithms related to it.

By a set system $(X, \mathcal{F})$ we mean a set X which is called the domain of the set system and a family $\mathcal{F}$ of its subsets. For a set system $(X, \mathcal{F})$ and $Y \subseteq X$ define

$$\mathcal{F} \cap Y := \{A \cap Y : A \in \mathcal{F}\}.$$

We call the new set system $\mathcal{F} \cap Y$ on the domain Y the *trace* of the set system $\mathcal{F}$ on the set Y. In a set system $(X, \mathcal{F})$ a subset $Y \subseteq X$ is called *shattered* if $\mathcal{F} \cap Y = \mathcal{P}(Y)$. The VC dimension of $\mathcal{F}$, $VC(\mathcal{F})$, is the largest integer n (if exists) such that there exists some subset of X of size n which is shattered by $\mathcal{F}$. If such n does not exist, then $VC(\mathcal{F}) = \infty$. By a *subsystem* of a system $(X, \mathcal{F})$ we mean the trace of some subset of $\mathcal{F}$ on some subset $Y \subseteq X$. The following fundamental theorem was proved in [10] and [11]. Also there are many other proofs for this theorem.

Theorem 1 (Sauer-Shelah lemma) *Assume that $(X, \mathcal{F})$ is a set system with $VC(\mathcal{F}) = d$. Then for every finite $Y \subseteq X$ we have*

$$|\mathcal{F} \cap Y| \leqslant \sum_{i=0}^{d} \binom{|Y|}{i}.$$

The following theorem is also an important known result about set systems.

Theorem 2 *Let $(X, \mathcal{F})$ be a set system where X is finite. Then*

$$|ssht(\mathcal{F})| \leqslant |\mathcal{F}| \leqslant |sht(\mathcal{F})|$$

where by $sht(\mathcal{F})$ and $ssht(\mathcal{F})$ we mean the set of all subsets of X shattered and strongly shattered by $\mathcal{F}$ respectively.

Geometric examples are source of many ideas in VC-theory. One can see for example [2] to see some specific structures studied by VC dimension. Also the following theorem and its extensions are important results which connects many ideas in VC-theory to computational geometry.

Theorem 3 *Let $S_1, \ldots, S_n$ be convex subsets of $\mathbb{R}^d$ with $n > d$. If every $d+1$ of these sets have non-empty intersection, then the intersection of all is non-empty ($\bigcap_{i=1}^{n} S_i \neq \emptyset$).*

From another point of view, VC-dimension is in direct relation with both PAC learnability and also existence of compression scheme. In fact a system $\mathcal{F}$ is PAC learnable if and only if $VC(\mathcal{F})$ is finite. Also if $(X, \mathcal{F})$ admits a compression scheme, then it has bounded VC-dimension. More precisely if $(X, \mathcal{F})$ admits a k-compression scheme then $vc(\mathcal{F}) \leq k$. The question of the converse side was an important problem. Existence of compression schemes of size depending on VC was proven in [9] in the following way.

*Department of Mathematics and Computer Science, Amirkabir University of Technology and Institute for research in fundamental sciences IPM mofidi@aut.ac.ir

Theorem 4 *Assume that $VC(\mathcal{F}) = d$ and the dual set system $\mathcal{F}^*$ satisfies $VC(\mathcal{F}^*) = d^*$. Then $\mathcal{F}$ has a compression scheme of size $O(dd^*)$. In particular (as $VC(\mathcal{F}) \leq d$ implies $VC(\mathcal{F}^*) < 2^{d+1}$), only assuming that $VC(\mathcal{F}) \leq d$ we get a compression scheme of size $2^{O(d)}$.*

By using tools from model theory, several aspects of interactions between combinatorial features of set systems, learning theoretic properties of them and certain computational geometric features of them have been explored in different works in recent years such as [4] and [6]. For example in [6] several techniques from model theory was used in order to construct compression schemes in certain situations and examples.

2 Results

One can enrich the classes of set systems by assuming extra conditions (in particular some extremal properties in terms of Sauer-Shelah lemma) and then study the VC-theoretic and computational geometrical impacts of them. Among the important classes, one can consider the class of maximum systems. We call a set system $(X, \mathcal{F})$ with $VC(\mathcal{F}) = d$ a d-maximum system if for any $Y \subseteq X$ the inequality of the Sauer-Shelah lemma turns to be equality for $\mathcal{F} \cap Y$. There are also other extremal combinatorial assumptions which impose very nice combinatorial structures on the systems. For example in [1] investigation of classes with certain extremal properties was started and then later was generalized in different directions. Restricting to classes with above extra conditions enables us to study certain problems in computational geometry and statistical learning theory from the point of view of the techniques existing in such classes. For example construction of a sample compression scheme of size equal to their VC-dimension for some classes with extremal properties was recently done by Moran and Warmuth in [8]. This is an improvement of Theorem 4 in the restricted case of classes with extremal properties studied in [1]. Note that these classes include maximum systems. A property which plays important role in many situation in the study of these systems (and also maximum systems) is that these classes are very well behaved in terms of some set system operations such as restrictions and derivations.

In this work we continue studying above mentioned systems with extremal properties and consider some properties of them in particular those with some rich examples in the sense of computational geometry and learning theory. We mostly focus on a particular class containing the maximum systems and build a combinatorics-model theoretical framework appropriate for them. For example we investigate ultraproduct construction of them and build very large systems which possess many nice properties of the element of product. This method gives some understanding of limit behaviour of the systems as well as combinatorial insights about their relations to each other. Also this construction could be considered as a method for building certain systems with extremal properties in infinitary settings. In fact we construct some new (infinite) families of extremal classes using the above constructions. We also use some logical methods to study certain VC-combinatorial invariants defined in [7]. The following theorem is an instance of some asymptotic results obtained for analysing the combinatorial structure of certain limits of systems.

We state and prove the following result:

Theorem 5 *Let $(X, \mathcal{F})$ be a countable set system and Y a limit ultraproduct structure (w.r.t some indexing set λ and suitable incomplete ultrafilter). Let X^Y (which we denote it here by Z for simplicity) be the image of embedding of X in Y. Also for every family $\{f_i : i < \lambda\}$ of elements of the system with the property that $f := [f_i]$ has value 1 on Z, define $O_{[f_i]} := Z^c \cap f^{-1}(1)$. Similarly, for every family $\{g_i : i < \lambda\}$ of elements of the system with the property that $g := [g_i]$ has value 0 on Z, define $P_{[g_i]} := Z^c \cap g^{-1}(0)$. Then $O_{[f_i]}$'s and $P_{[g_i]}$'s are mutually intersecting.*

Proof. In the first step we mention that letting $X = \{a_i : i < \omega\}$ and $Y := X^\lambda/\mathcal{D}$ where λ is a cardinal and $\mathcal{D}$ is a non-principal ultrafilter on λ, we have that $Y \backslash X^Y \neq \emptyset$ if and only if $\mathcal{D}$ is a σ_1-incomplete ultrafilter.

Assume that $X = \{a_i : i < \omega\}$ and $Y := X^\lambda/\mathcal{D}$. It is not difficult to see that every $b \in Y$ with representation $b := [a_{n_i}]$ can be seen as a partition $P^b = \{P^b_n : n < \omega\}$ of λ with $P^b_n := \{i < \lambda : n_i = n\}$. Also conversely it is easy to see that every partition $P = \{P_n : n < \omega\}$ of λ or any $A \in \mathcal{D}$ gives rise to a unique element $b_P := [a_{n_i}] \in Y$ where $n_i = n \Leftrightarrow i \in P_n$ for each $i < \lambda$. Moreover, $b \in Y \setminus X^Y$ if and only if for every n, $P^b_n \notin \mathcal{D}$.

Claim 1: Assume that $\{f_i : i < \lambda\}$ are a sequence of subsets of X such that $f := [f_i]$ takes value 1 on X^Y. For every $n < \omega$ define $B_n := \{i < \lambda : f_i(a_n) = 1\}$. Then for every $b := [a_{n_i}]$ in Y, $f(b) = 1$ if and only if $\bigcup_{n<\omega}(B_n \cap P^b_n) \in \mathcal{D}$. Similarly, the same holds if one replaces all values 1 to 0.

Proof of claim 1:

Let $b := [a_{n_i}] \in Y$ and let P_b be its corresponding partition. Define $C_b := \{i < \lambda : f_i(a_{n_i}) = 1\}$. So $f(b) = 1$ if and only if $C_b \in \mathcal{D}$. On the other hand we have

$$C_b = \bigcup_{n<\omega}(C \cap P^b_n) = \bigcup_{n<\omega}(\{i < \lambda : f_i(a_n) = 1, n \in P^b_n\}) = \bigcup_{n<\omega}(B_n \cap P^b_n).$$

Combining these, we have $f(b) = 1$ if and only if $\bigcup_{n<\omega}(B_n \cap P^b_n) \in \mathcal{D}$ as desired.

One can easily see that by using the above analysis there is a one to one correspondence between elements of $f^{-1}(1) \setminus X^Y$ and partitions of λ with the mentioned property.

Claim 2:

Assume that X and Y and $\{f_i : i < \lambda\}$ be a as above such that $f := [f_i]$ takes value 1 on X^Y. Let $B_n := \{i < \lambda : f_i(a_n) = 1\}$. Then the following hold:

Firstly, if $\bigcap_{n<\omega} B_n \in \mathcal{D}$ then $f(b) = 1$ for every $b \in Y$.

Secondly, if $\bigcap_{n<\omega} B_n \notin \mathcal{D}$ then there exists some $b \in Y \setminus X^Y$ such that $f(b) = 1$.

Similarly, the same above results hold if one replaces all values 1 to 0.

Proof of claim 2:

Fix an arbitrary $b \in Y \setminus X^Y$. Assume that $b = [a_{n_i}]$ where $a_{n_i} \in X$ for each i. Let P^b and P^b_n's be as defined above By using analysis we have done before and since $b \notin X^Y$, we have $P^b_n \notin \mathcal{D}$ for each n. Define $L := \bigcap B_n$. By our assumption we have $L \in \mathcal{D}$. Hence

$$\bigcup_{n<\omega}(P^b_n \cap B_n) \supseteq \bigcup_{n<\omega}(P^b_n \cap L) = L \in \mathcal{D}.$$

Now again by some arguments in above we have $f(b) = 1$. Since b was arbitrary, therefore f takes value 1 on whole of Y. This proves the first part.

Now for the second part define $H_n := \bigcap_{i=1,\dots,n} B_i$. So $H_n \in \mathcal{D}$ and also $H_{n+1} \subseteq H_n$ for each n and $\bigcap_{n<\omega} H_n = \bigcap_{n<\omega} B_n \notin \mathcal{D}$. Define $E_n := H_n \setminus H_{n+1}$ and $E := \bigcup_{n<\omega} E_n$. Now E_n's are mutually disjoint and since $E_n \subseteq H^c_{n+1}$ then $E_n \notin \mathcal{D}$. On the other hand we have

$$E = H_1 \setminus \bigcap_{n<\omega} H_n \in \mathcal{D}$$

where we are using the fact that $\bigcap_{n<\omega} H_n \notin \mathcal{D}$. Note that for each n we have $B_n \cap E_n \supseteq H_n \cap E_n = E_n$. So

$$\bigcup_{n<\omega}(B_n \cap E_n) \supseteq \bigcup_{n<\omega} E_n = E \in \mathcal{D}, \quad (*).$$

Now let $P := \{P_n : n < \omega\}$ be a partition of E (as an element of $\mathcal{D}$) where $P_n := E_n$. Also let $b_P \in Y$ be the element corresponding to this partition defined previously. Each member of P does not belong to $\mathcal{D}$. So $b_P \in Y \setminus X^Y$. One can see that $P^{b_P} = P$ and each $P^{b_P}_n$ coincides with E_n. Therefore, by using the fact $(*)$ we have $f(b_P) = 1$ as desired in the second part.

Now we use the above analysis in order to prove the following claim:

Claim 3:

Assume that X and Y be a as before and assume that $\{f_i : i < \lambda\}$ and $\{g_i : i < \lambda\}$ be families of some elements of $\mathcal{U}$ such that $f := [f_i]$ has value 1 on X^Y and $g := [g_i]$ has value 0 on X^Y. Then there is some $b \in Y \setminus X^Y$ such that $f(b) = 1$ and $g(b) = 0$.

Proof of claim 3:

For every $n < \omega$ define $B_n := \{i < \lambda : f_i(a_n) = 1\}$ and $C_n := \{i < \lambda : g_i(a_n) = 0\}$. For every n we have $B_n \in \mathcal{D}$ and $C_n \in \mathcal{D}$. We divide the rest of proof to three cases.

(I: $\bigcap_{n<\omega} B_n \in \mathcal{D}$ and $\bigcap_{n<\omega} C_n \in \mathcal{D}$):

By using results proved earlier, for both B_n's and C_n's one concludes that for every $b \in Y$, $f(b) = 1$ and $g(b) = 0$.

(II: Exactly one of $\bigcap_{n<\omega} B_n \in \mathcal{D}$ or $\bigcap_{n<\omega} C_n \in \mathcal{D}$ happens):

Without loss we may assume that $\bigcap_{n<\omega} B_n \in \mathcal{D}$ and $\bigcap_{n<\omega} C_n \notin \mathcal{D}$. By using above claims for B_n's, one concludes that f takes value 1 on whole of Y. Now there is some $b \in Y \setminus X^Y$ such that $g(b) = 0$. So combining these we have $f(b) = 1$ and $g(b) = 0$.

(III: $\bigcap_{n<\omega} B_n \notin \mathcal{D}$ and $\bigcap_{n<\omega} C_n \notin \mathcal{D}$):

For every $n < \omega$ define $H_n := B_n \cap C_n$. So H_n's belong to $\mathcal{D}$. Also $\bigcap_{n<\omega} H_n \notin \mathcal{D}$. Define $H'_n := \bigcap_{i=1,\dots,n} H_i$. So $H'_n \in \mathcal{D}$ and also $H'_{n+1} \subseteq H'_n$ for each n and $\bigcap_{n<\omega} H'_n = \bigcap_{n<\omega} H_n \notin \mathcal{D}$. Define $E_n := H'_n \setminus H'_{n+1}$ and $E := \bigcup_{n<\omega} E_n$. Now E_n's are mutually disjoint and since $E_n \subseteq {H'_{n+1}}^c$ then $E_n \notin \mathcal{D}$. On the other hand we have

$$E = H'_1 \setminus \bigcap_{n<\omega} H'_n \in \mathcal{D}$$

where we are using the fact that $\bigcap_{n<\omega} H'_n \notin \mathcal{D}$. Note that for each n we have $B_n \cap E_n \supseteq H'_n \cap E_n = E_n$ and $C_n \cap E_n \supseteq H'_n \cap E_n = E_n$. So

$$\bigcup_{n<\omega}(B_n \cap E_n) \supseteq \bigcup_{n<\omega} E_n = E \in \mathcal{D}, \quad (1)$$

$$\bigcup_{n<\omega}(C_n \cap E_n) \supseteq \bigcup_{n<\omega} E_n = E \in \mathcal{D}. \quad (2)$$

Now let $P := \{P_n : n < \omega\}$ be a partition of E (as an element of $\mathcal{D}$) where $P_n := E_n$. Also let $b_P \in Y$ be the element corresponding to this partition defined earlier. Each member of P does not belong to $\mathcal{D}$. So $b_P \in Y \setminus X^Y$. One can see that $P^{b_P} = P$ and each $P^{b_P}_n$ coincides with E_n. Therefore, by using the facts (1) and (2) above and previous claims we have $f(b_P) = 1$ and $g(b_P) = 0$ as desired. This completes the proof of claim 3.

Now Theorem 5 can be obtained by a careful using of claim 3. □.

As a corollary of Theorem 5 one can mention the following. Let $(X, \mathcal{F})$ be a set system, and $\mathcal{D}$ a σ-incomplete ultrafilter on some index set λ. Assume that $\mathcal{V} \subseteq \mathcal{F}^{\lambda}/\mathcal{D}$. Then $\mathcal{V}$ can not strongly shatter image of embedding of X in limit product structure.

Note that it can be seen that for every finite number n the restriction of the ultraproduct system to every subset of size n of the domain is isomorphic to restrictions of almost all members of the product to some finite subsets of size n of their domains. More precisely, for every finite number n, the restriction of the ultraproduct system to every subset of size n of the domain is isomorphic to the restrictions of almost all (w.r.t ultrafilter) members of the product to some finite subsets of size n of their domains.

3 Discussion

In a related direction to the above results, one may consider model theoretical aspects related to combinatorial configurations encoded in systems as well as first order expressibility of extremal notions. Note that model theoretical tools are proven (in works such as [6]) to be very useful in constructing and analysing systems which forbid certain configurations such as order. Such systems include many nice geometrical examples. One can consider an analogue of compression scheme and Theorem 4 in this context which is called "uniform definability over finite sets". One can also explore their impacts on geometrical complexities and also give several examples in different situations.

References

[1] B. Bollobas and A. J. Radcliffe. Defect sauer results. *J. Comb. Theory, Ser. A*, 72(2):189208, 1995.

[2] B. Gartner and E. Welzl. Vapnik-chervonenkis dimension and pseudo-hyperplane arrangements. *Discrete and Computational Geometry (DCG)*, 12:399432, 1994.

[3] J. Goodman, J. O'Rourke *Handbook of Discrete and Computational Geometry*, Chapman Hall-CRC, 2004.

[4] H. Johnson Some new maximum VC classes *https://arxiv.org/abs/1309.2626.*

[5] D. Kuzmin and M. K. Warmuth. Unlabeled compression schemes for maximum classes. *Journal of Machine Learning Research*, 8:20472081, 2007.

[6] R. Livni, P. Simon Honest Compressions and Their Application to Compression Schemes, *Proceedings of the 26th Conference on Learning Theory (COLT)* , 2013.

[7] A. Mofidi On some dynamical aspects of NIP theories, *Arch. Math. Logic*, 2017, to appear.

[8] S. Moran, M. Warmuth, Labeled compression schemes for extremal classes, https://arxiv.org/abs/1506.00165, 2016.

[9] S. Moran, A. Yehudayoff Sample Compression Schemes for VC Classes. *J. ACM* 63(3): 21:1-21:10 ,2016.

[10] N. Sauer. On the density of families of sets. *J. Comb. Theory, Ser. A*, 13:145147, 1972.

[11] S. Shelah. A combinatorial problem; stability and order for models and theories in infinitary languages. *Pac. J. Math*, 41:247261, 1972.

[12] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probability. *Theory Probab. Appl*, 16:264280, 1971.

Covering Points by Triangles

Sima Hajiaghaei Shanjani* Alireza Zarei†

Abstract

Given a set of n points on the plane, the problem considered in this paper are to compute two triangles that cover all the points and the largest triangle or total area of triangles is minimized. First, we study the case of two isosceles triangles with fixed apex angles. For this case, we propose an exact algorithm running in $O(n^5 \log n)$ time to compute triangles with the minimum largest or triangles with the minimum total area, and propose a PTAS in $O(\epsilon^{-1} n^2 \log^3 n)$ time for this problem. Then we consider the case of arbitrary triangles, and present an algorithm to compute a $(4+\epsilon)$ approximation of the minimum total area triangles in $O(\epsilon^{-6} n \log^3 n)$ time and a PTAS for the problem of minimizing the largest area of the triangles. Finally, we show that, having a constant factor approximation of the longest side of the optimum solution, for minimizing the total area of the triangles and minimizing the largest area of them PTAS exist.

1 Introduction

Covering problems are primary concerns in computational geometry. In one classic type of these problems, for a given set of points, the goal is to find a set of one or more geometric object that encloses the points, while optimizing an objective function such as area, perimeter, surface area, or volume. Here the location of the geometric object is to be computed as part of the goal. Minimum Enclosing Circle and Minimum Enclosing Box are well-known problems of this type. Many covering problems have arisen in this context of applications like Geographic Information System (GIS) and wireless networks [1]. This includes, the problem of covering a set of points by two rectangles, known as two-rectangle Covering, seeks to minimize the area of the largest rectangles or the total area of the rectangles, which has been studied extensively for different constraints [3, 9, 10].

Related work: To our knowledge, this is the first time that the problem of covering points by more than one triangle has been studied. However, there are two different threads of research that relate to this problem. The first thread relates to covering points with one triangle, i.e. variants of the Minimum Enclosing Triangle problem [2, 5, 6, 7, 8]. The second thread relates to covering points with more than one object, e.g. covering points with two disjoint arbitrary rectangles with the minimum total area [3, 1, 14, 13, 11, 12, 9, 10].

two axis-parallel rectangles	minimize largest area or total area	$O(n \log n)$, 2000 [13]
two parallel and equal size squares	minimize largest area	$O(n^2)$, 1996 [11]
two arbitrarily oriented squares	minimize largest area	$O(n^3 \log^2 n)$, 2000 [12]
two parallel rectangles	minimize largest area	$O(n^3)$, 2009 [3]
two disjoint rectangles	minimize largest area	$O(n^2 \log n)$, 2011 [10]
two disjoint and parallel rectangles	minimize largest area	$O(n^2 \log n)$, 2011 [10]
two disjoint rectangles (one axis-aligned and the other arbitrarily oriented)	minimize largest area	$O(n^2 \log n)$, 2011 [10]
two arbitrarily oriented rectangles	$(1+\epsilon)-$approx minimize total area	$O(\epsilon^{-6} n \log^3 n)$, 2010 [9]

Table 1: Related results to covering points by two object

• First Thread: Minimum Enclosing Triangle The problem of finding the smallest enclosing triangle has been studied widely with various constraints and goals. Examples of this category are Minimum Area Enclosing Triangle [5], the Minimum Enclosing Area Triangle with a Fixed Angle [6], and the Minimum Isosceles Enclosing Triangle with a Fixed Apex Angle [8]. In 1985, the problem of covering points with a triangle appeared in the form of enclosing a convex polygon with the minimum area triangle [15]. The best known algorithm, proposed in [5], runs in linear time in the size of the polygon and can be applied on point set of size n in $O(n \log n)$ time, is proposed in [5]. In [8, 6, 7], several variants of this problem were studied, particularly where the enclosing triangle is an isosceles triangle with fixed apex angle. For this problem, all optimum triangles for a set of points were computed in $O(n \log n)$ time and for a simple polygon the running time was $O(n)$.

• Second Thread: Covering Points with more than One Object Different types of objects have been studied for the problem of covering points with more than one object, particularly rectangles. See [1] for a comprehensive survey that includes a list of these problems and see [14] for a PhD thesis on this topic. In 2000, Two-Rectangle covering problem was studied for the case, in which the rectangles are axis-aligned, and an algorithm in $O(n \log n + n^{d-1})$ time was proposed for the problem in d-dimensional space [13]. Many other versions

*Departments of Computer Science, University of Victoria, Sima@uvic.ca

†Department of Mathematical Science, Sharif University of Technology , zarei@sharif.edu

of these problems exist, and Table 1 summarize the results that relate to covering with squares and rectangles, such as covering with two parallel rectangles, two parallel squares, and two arbitrary oriented squares.

Our Contribution: In this paper we study covering problems with two triangles for a given point set. The goals of the problem is to minimize the area of largest triangles or to the total area of two triangles. Different constraints also can be considered for these problems, such as triangles with a fixed angle or isosceles triangles. Approximation and exact algorithms are found for the problem of covering a set point by two isosceles triangles with a fixed apex angle and the problem of covering points by two arbitrary triangles.

The problem of finding an exact solution for covering a set of points with two arbitrary triangles seems a difficult problem in terms of time complexity. Thus, we are interested in approximate solutions and exact constrained solutions.

First, we propose an exact algorithm for the problem of covering points with two isosceles triangles with fixed apex angles and minimum total/largest area in $O(n^5 \log n)$ time. Then we give a PTAS that solves the problem for the case of minimizing total area in $O(\epsilon^{-1}n^4)$ time, then a PTAS for minimum largest area in $O(\epsilon^{-1}n^2 \log^3 n)$ time. For the case of arbitrary triangles, we first provide a $(4+\epsilon)$-approximation algorithm in $O(\epsilon^{-6}n \log^3 n)$ time, which is mainly based on [9]. Then we give a PTAS to minimize the largest area in $O(\epsilon^{-3}n^2 \log^2 n)$ time and for the case of minimizing total areas a PTAS in $O(\epsilon^{-3}n^4)$ time. Finally, if we have a constant factor approximation of the longest side of the optimum solution, we provide a PTAS algorithm in $O(\epsilon^{-5}n^2)$ time to minimize total area and a PTAS in $O(\epsilon^{-3} \log^2 \epsilon^{-1}n^2)$ time to minimize the largest area.

2 Isosceles Triangles with fixed apex angle

In this section, we study the problem where the covering triangles are not arbitrary triangles, but isosceles triangles with fixed apex angles.

Theorem 1 *Given a set P of points in the plane and a positive angle $\alpha < \pi$, the two enclosing isosceles triangles with apex alpha with the minimum total area can be computed in $O(n^5 \log n)$ time.*

Proof. Each side of the minimum enclosing triangles should be in contact with at least one point of P; Otherwise, the size of the triangle can be reduced. In order to have this property in the first triangle, consider two points p_i and p_j on the equal sides of the triangle. Then, the third point should be chosen to lie on the base(the edge opposite the apex) of the triangle. Consider the locus of apices of minimal α-wedges for P that has p_i and p_j on it's sides. This locus is a subset of the locus

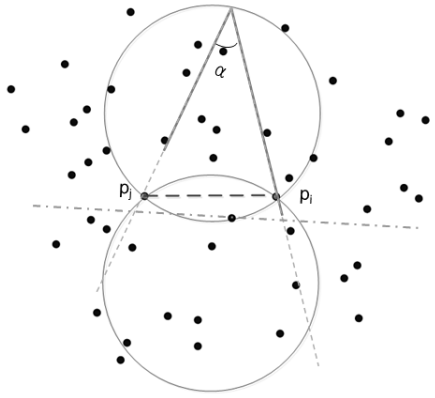


Figure 1: Locus of all points at which a given p_ip_j subtends a fixed angle α

of all points at which a given segment p_ip_j subtends a fixed angle α, which is a circular arc with the segment as a chord (See Figure 1).

For the third point, at each location of the apex, any point that is inside the minimal α-wedge can be chosen to lie on the base of the triangle. Then, for a chosen points as a third points, compute the minimum enclosing isosceles triangle of the points that are inside of these three edges (i.e. sides of the α-wedge and a base). Note that the triangle that formed based on these three edges is not necessarily the minimum area triangle, which covers all the points that lie inside the triangle. Then, compute the points that lie outside of the first minimum enclosing triangle, and find the minimum area triangle that covers the rest of the points.

Time analysis: There is a quadratic pair of points to choose as p_i and p_j to lie on equal sides of the first triangle. Then, we should find the number of times that the subset of points lying inside the α-wedge is different, while the location of the apex changes on the circular arc. This number is linear, as each point comes to the inside of α-wedge and goes out at most once for each circular arc. So, there is a linear different subset of points for each pair of p_i and p_j . The number of choices for the third point is also linear, as the number of the points in each subset is linear. Therefore, there are $O(n^4)$ different subsets of points for the first triangle. Computing minimum enclosing isosceles triangle of these points takes $O(n \log n)$ time. Therefore, the time complexity of this algorithm is $O(n^5 \log n)$. □

Theorem 2 *Given a set P of points in the plane and a positive angle $\alpha < \pi$, a $(1+\epsilon)$-approximation of the two enclosing isosceles triangles with apex alpha with the minimum total area can be computed in $O(\epsilon^{-1}n^4)$ time.*

Proof. Consider D a set of $O(\delta^{-1})$ uniformly spread directions on the unit circle, where $\delta = \theta(\epsilon)$. In such a way that for any direction d on the unit circle there are some directions $d^* \in D$, which the angle between d and d^* is less than δ.

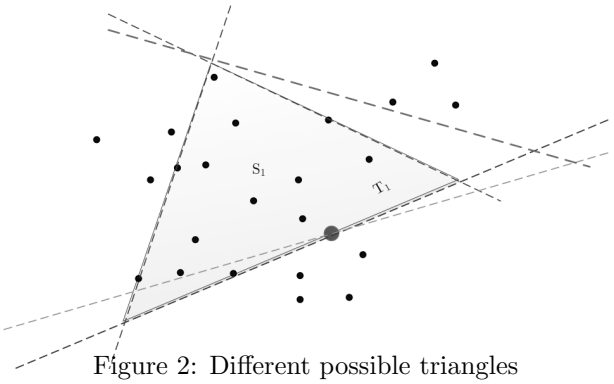


Figure 2: Different possible triangles

Assume T is one of the triangles in optimum solution and the direction of its base is d. First, we want to show that there is a triangle T' with a side in $d^* \in D$ direction that contains T and approximates the area of it; i.e., $area(T') \leq (1+\epsilon)area(T)$. Then,we should show how to compute such a triangle T'.

For showing that the T' exists, the reasoning is similar to the analysis of PTAS algorithm for computing the smallest enclosing triangle a set of points in [8]. Consider the smallest enclosing isosceles triangle of T with apex angle α and the base direction d^* where $|\vec{d} - \vec{d^*}| \leq \delta$. Let h is the height of T and h' is the height of T'. Then, we have that $h' \leq h(\frac{(2-\cos\alpha)\sin\delta}{\sin\alpha} + \cos\delta)$. By considering $\delta = \frac{\epsilon \sin\alpha}{2-\cos\alpha}$, we have $h' \leq h(\cos\delta + \epsilon) \leq h(1+\epsilon)$. So, as the triangles are similar and $h' \leq h(1+\epsilon)$ the area of T' is at most $(1+\epsilon)$ area of T.

For finding such triangles to cover all the points, Consider set of D directions for each point p in P. Compute all optimal enclosing isosceles triangles with apex angle α and base directions in D, which p is on the base of this triangle (Figure 2). For each of these optimal triangles compute the points that lies outside of the triangle, and compute a $(1+\epsilon)$- approximation solution for this subset of points by using the PTAS in [8].

Time analysis: There is a linear number of points and $O(\delta^{-1})$ directions for each point, where $\delta = \theta(\epsilon)$. For each base direction of first triangle there is a quadratic number of possibilities for different optimal triangles. This is because, each side is limited by a point and the number of these points is linear. So, there are $O(\epsilon^{-1}n^3)$ different choices for the first triangle. For computing the points that lie outside of the first triangle we can do it easily in linear time. Besides, computing a $(1+\epsilon)$-approximation of the solution for the points that lie outside of the first triangle takes $O(n)$ time. So, the total time complexity of this algorithm is $O(\epsilon^{-1}n^4)$. □

Corollary 2.1 *Given a set P of points in the plane and a positive angle $\alpha < \pi$, a $(1+\epsilon)$-approximation of the two enclosing isosceles triangles with apex alpha with the minimum largest area can be computed in $O(\epsilon^{-1}n^2 \log^3 n)$ time.*

Proof. The algorithm for minimizing the largest triangle is similar to the one for minimizing total area of triangles. In order to minimize the largest area we do not need to consider all the possibilities for the first triangle. For one particular point and one direction $d^* \in D$, first we will sort all the point in the perpendicular directions of the triangle's sides. Then, instead of checking all the $O(n^2)$ possibilities, we can use binary search for both sides to minimize the largest area. Thus, the running time of this algorithm reduces to $O(\epsilon^{-1}n^2 \log^3 n)$. □

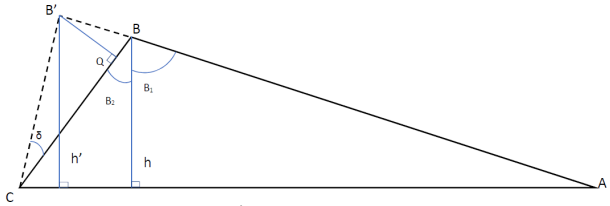


Figure 3: Triangle T' with a side in $d^* \in D$, and T' contains T and approximates the area of it

3 Arbitrary Triangles

Theorem 3 *Given a set P of points in the plane, a $(4+\epsilon)$-approximation of the two enclosing arbitrary triangles with the minimum total area can be computed in $O(\epsilon^{-6}n \log^3 n)$ time.*

Proof. In [9] a $(1+\epsilon)$- approximation algorithm proposed for the problem of covering points with two arbitrary oriented rectangles. We want to show how the answer of that problem can help us to find a good approximation for the problem of covering points by two triangles. let R'_1 and R'_2 are the optimum solution for the problem of covering points by rectangles, and respectively P_1 and P_2 are the subset of point that they cover. Compute the minimum enclosing triangles of R'_1 and R'_2, and call them T'_1 and T'_2. Next, we want to show that the total area of T'_1 and T'_2 is at most $(4+\epsilon)$ of area of the total area of optimum solution.

The area of the smallest enclosing triangle of a rectangles is twice the area of rectangle. On the other hand R'_1 and R'_2 are $(1+\epsilon)$-approximation of the two R_1 and R_2 rectangles in optimum solution. Now, we want to show that the total area of optimum solution T_1 and T_2 for the problem of covering points by two arbitrary triangles is at least half of the total area of R_1 and R_2. This is because, if the total area of T_1 and T_2 is smaller, then minimum enclosing rectangles of T_1 and T_2 are better choices than R_1 and R_2 for covering points by rectangles. □

Theorem 4 *Given a set P of points in the plane and a positive angle $0 < \theta$, a $(1+\epsilon)$-approximation of the two enclosing arbitrary triangles with the minimum total area while all the angles are grater than θ can be computed in $O(\epsilon^{-3}n^4)$ time.*

Proof. The idea of this algorithm is similar to the theorem 2, but the analysis of the approximation ratio is not the same.

Consider D a set of $O(\delta^{-1})$ uniformly spread directions on the unit circle, where $\tan\delta = \theta(\epsilon)$. In such a way that for any direction d on the unit circle there are some directions $d^* \in D$, which the angle between d and d^* is less than δ.
Assume T is one of the triangles in optimum solution and the direction of its base is d. First, we want to show that there is a triangle T' with all sides on directions $d^* \in D$ that contains T and approximates the area of it. For showing that the T' is exists, consider triangle T in Figure 3, and triangle T_1, which is the smallest covering triangle of T with base direction d^* where $|\vec{d} - \vec{d^*}| \leq \delta$ and the direction of $\bar{BC}$ is d.
Let B' is the intersection point of $\bar{AB}$ direction and the direction d', which the angle between d and $\bar{CB}$ is less than δ. Triangle $AB'C$ is a triangle that has a side in direction d', and we will show that $area(T_1) \leq (1+\epsilon)area(T)$. Let h is the height of T and h_1 is the height of T'. Then, we have that, $h_1 = \bar{AB'}.\cos B_1 = (\bar{AB} + \bar{BB'}).\cos B_1 = h + \bar{BB'}.\cos B_1 = h + \frac{\bar{B'Q}}{\sin(A+C)}.\cos B_1 = h + \frac{\tan\delta.\bar{CQ}.\cos B_1}{\sin B} \leq h + \frac{\tan\delta.\bar{CB}.\cos B_1}{\sin B} \leq h + \frac{\tan\delta.h.\cos B_1}{\cos B_2 \sin B}$
As $B_2 \leq B$, so $\cos B_2 \geq \cos B$, and
$h_1 \leq h + \frac{\tan\delta.h}{1/2\sin 2B} \leq h(1+\epsilon_1)$ if $\tan\delta \leq \frac{\epsilon_1 \sin 2\theta}{2}$
Therefore, we have $h_1 \leq h(1+\epsilon_1)$ and $area(T_1) \leq (1+\epsilon_1)area(T)$.
Next, we can have same reasoning for T_2 on AB' side of T_1, and $T' = T_3$ on AC side of T_2, when $\tan\delta = \Theta(\epsilon_2)$ and $\tan\delta = \Theta(\epsilon_3)$ respectively. Then, by adjusting $\epsilon_1, \epsilon_2, \epsilon_3$, we can have a T' triangle such that $area(T') \leq (1+\epsilon)area(T)$ and $\tan\delta = \Theta(\epsilon)$.
For computing all the triangles in D directions, similar to the Theorem 2, consider all directions in D for each point. For each side, there are $O(1/\epsilon)$ directions, and for each direction, there are $O(n)$ points. Thus, for each point there are $O(\epsilon^{-3}n^2)$ different possible triangles. Therefore, the total running time of the algorithm in order to minimize the total area of the triangles is $O(\epsilon^{-3}n^4)$. □

Corollary 4.1 *Given a set P of points in the plane and a positive angle $0 < \theta$, a $(1+\epsilon)$-approximation of the two enclosing arbitrary triangles with the minimum largest area while all the angles are grater than θ can be computed in $O(\epsilon^{-3}n^2\log^2 n)$ time.*

Theorem 5 *Given a set P of points in the plane and a constant factor approximation of the longest side of the optimum solution, a $(1+\epsilon)$-approximation of the two enclosing arbitrary triangles with the minimum total area can be computed in $O(\epsilon^{-5}n^2)$ time.*

Proof. If we have Δ, which is a constant factor approximation of the longest side of the triangles, we do not need to consider all the $O(\epsilon^{-2}n^2)$ possible triangles for each direction of each point. Instead, it is enough to consider a grid with size $O(\epsilon^{-1}\Delta)$ on base direction for each point and direction. Then, for each side there are $O(\epsilon^{-2})$ possible choices, and in total $O(\epsilon^5)$ choices for each point. The rest of the algorithms is similar to the algorithm in Theorem 2, i.e. find the points lie outside of the first triangle, and find $(1+\epsilon)$-approximation of the minimum area covering triangles of them in $O(n)$ time. Therefore, the total running time of this algorithm is $O(\epsilon^{-5}n^2)$. □

Corollary 5.1 *Given a set P of points in the plane and a constant factor approximation of the longest side of the optimum solution, a $(1+\epsilon)$-approximation of the two enclosing arbitrary triangles with the minimum largest area can be computed in $O(\epsilon^{-3}\log^2\epsilon^{-1}n^2)$ time.*

References

[1] P. Aggarwal and M. Sharir Efficient algorithms for geometric optimization. ACM Comput. Surveys 1998.

[2] B. Bhattacharya and A. Mukhopadhyay On the minimum perimeter triangle enclosing a convex polygon. JCDCG, 2002.

[3] C. Saha and S. Das Covering a set of points in plane using two parallel rectangles. ICCTA, 2009.

[4] S.Har-peled No coreset, no cry. In 24th Conf. Found. Soft. Tech. Theoretical Computer Science, 2004.

[5] J. O Rourke, A. Aggarwal, S. Maddila and M. Baldwin An optimal algorithm for finding minimal enclosing triangles. Journal of Algorithms, 1986.

[6] P. Bosea and J. L. De Carufel Minimum enclosing area triangle with a fixed angle. CCCG, 2010.

[7] P. Bose and J. L. De Carufel Isoperimetric triangular enclosure with a fixed angle. CCCG 2011.

[8] P. Bose, C. Seara, S. Sethia On computing enclosing isosceles triangles and related problems. International Journal of Computational Geometry and Application, 2009.

[9] S. Har-Peled and N. Kumar On covering points by rectangles. Note, 2010

[10] S. Kim, S. Bae, H. K. Ahn Covering a point set by two disjoint rectangles. International Journal of Computational Geometry and Applications, 2011.

[11] J.W. Jaromczyk and M. Kowaluk Orientation independent covering of point sets in R^2 with pairs of rectangles or optimal squares. EuroCG, 1996.

[12] M. Kats, K. Kedem, M.Segal Discrete rectilinear 2-center problem. computational geometry theory and applications, 2000.

[13] S. Bespamyatnikh and M. Segal Covering a set of points by two axis-parallel boxes. Inform. *journal Information Processing Letters*, 75:95100, 2000.

[14] M. Segal Covering point sets and accompanying Problems. Ph.D. dissertation, Ben-Gurion University, 1999.

[15] V. Klee and M. Laskowski Finding the smallest triangle containing a given convex polygon. Algorithms, 6:359375, 1985.