The Third Iranian Conference on

Computational Geometry

(ICCG 2020)

Proceedings

Department of Mathematics and Computer Science
Amirkabir University of Technology
Tehran, Iran, February 16, 2020

Foreword

The third Iranian Conference on Computational Geometry was held on February 16, 2020 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 nine refereed papers and an invited talk that were presented during the conference, in four sessions. I would like to thank my pc co-chair Mohammad Farshi, the invited speaker Alexander Wolff, all the pc members, and members of the local organizing committee (Sepehr Moradi and Simin Yazdandoost). I also want to thank the sponsors: Amirkabir University of Technology for financial supports and Islamic World Science Citation Center (ISC) for indexing the conference (#ISC 98190-51487).

Zahed Rahmati
General Chair

Invited Speaker

Alexander Wolff University of Würzburg, Germany

General Chair

Zahed Rahmati Amirkabir University of Technology

Program Committee

Mohammad Ali Abam	Sharif University of Technology, Iran
Davood Bakhshesh	University of Bojnord, Iran
Mansoor Davoodi	Institute for Advanced Studies in Basic Sciences, Iran
Marzieh Eskandari	Al-Zahra University, Iran
Mohammad Farshi	Yazd University, Iran (co-chair)
Amin Gheibi	Amirkabir University of Technology, Iran
Xavier Goaoc	Université de Lorraine, France
Anil Maheshwari	Carleton University, Canada
Saeed Mehrabi	Carleton University, Canada
Debajyoti Mondal	University of Saskatchewan, Canada
David Mount	University of Maryland, USA
Mostafa Nouri Baygi	Ferdowsi University of Mashhad, Iran
Martin Nöllenburg	Vienna University of Technology, Austria
Zahed Rahmati	Amirkabir University of Technology, Iran (co-chair)
André van Renssen	The University of Sydney, Australia
Don Sheehy	North Carolina State University, USA
Farnaz Sheikhi	K. N. Toosi University of Technology, Iran
Carola Wenk	Tulane University, USA
Alexander Wolff	Universität Würzburg, Germany
Alireza Zarei	Sharif University of Technology, Iran
Hamid Zarrabi-Zadeh	Sharif University of Technology, Iran

Local Organizers

Sepehr Moradi	Amirkabir University of Technology, Iran
Zahed Rahmati	Amirkabir University of Technology, Iran, (Chair)
Simin Yazdandoost	Amirkabir University of Technology, Iran

Conference Program

Drawing Graphs and Hypergraphs in 2D and 3D

Alexander Wolff*

In this talk we review the basic concepts and the methodology of Graph Drawing, which is an active research area in the intersection of Graph Theory, Computational Geometry, and Information Visualization. Graph Drawing is about finding algorithms that map abstract, combinatorial objects (graphs or hypergraphs) to drawings, that is, "tangible" geometric objects. The goal is to find algorithms that guarantee a provable geometric quality measure in the worst case. For example, there are algorithms that draw any planar graph on a grid whose size is quadratic in the number of vertices of the graph [7, 13]. Other than in areas such as Information Visualization, the evaluation is usually *not* task-driven.

The Graph Drawing problem has numerous incarnations: supported graph classes (e.g., trees, outerplanar, planar, or bipartite graphs), drawing styles (e.g., orthogonal, straight-line, Bézier), quality measures (e.g., number of bends, number of crossings, crossing resolution), the embedding space (2D or 3D), and the type of representation (node–link diagrams, contact or intersection representations). Often, optimizing one measure leads to drawings that are bad in other measures. There is a lack of algorithms that are "pretty good" or at least "not too bad" in many aspects.

There is a (surprisingly small) set of standard techniques for drawing graphs. For example, if the graph class for which we want to design a drawing algorithm has a recursive definition, an obvious approach is to construct drawings recursively. A prominent example are orthogonal straight-line drawings of binary trees. It turns out that they can be drawn in a compact way; on a grid of size $O(n \log n)$, where n is the number of vertices of the given tree [6]. Similarly, if a graph class has an inductive definition, we may try to draw the given graph of that class inductively. Such an approach is used to show that every n-vertex planar 3-tree can be drawn using $2n - 4$ segments [8]. Finally, there are two at first glance very different approaches for drawing planar graphs on a grid of quadratic size. One approach, the shift algorithm [7], constructs the drawing incrementally; the other approach [13] counts some combinatorial objects (using a Schnyder wood) and then turns the resulting numbers into coordinates. A more careful analysis, however, reveals structural similarities between the two approaches.

Topics that have received considerable attention over the last few years are simultaneous embedding, morphing of graphs, drawings with large crossing angles, drawings of beyond-planar graphs, and visual complexity. The visual complexity of a drawing is measured by the number of geometric objects needed to compose or cover the drawing. For example, the *segment number* of a planar graph is the smallest number of straight-line segments whose union represents a straight-line drawing of the given graph [8]. The *arc number* [14] is defined accordingly with respect to circular-arc drawings, which often allow for less complex or more compact representations. Another recent generalization [12] are variants of the segment number for nonplanar graphs, either admitting crossings or embedding in 3D. The *plane cover number* [3, 5] asks for the smallest number of planes needed to cover a straight-line drawing of a given graph in 3D. Accordingly, one can define the *line cover number* in 2D (for planar graphs) or in 3D (for arbitrary graphs), which is obviously upperbounded by the corresponding segment number. Also weak versions of these numbers have been studied where only the vertices of a crossing-free straight-line drawing of the graph need to covered. While it is not hard to see that any outerplanar graph has weak line cover number 2 [4], even some cubic, 3-connected, bipartite planar graphs have unbounded weak line cover number (exceeding $\sqrt[3]{n}$, where n is the number of vertices) [9]. On the other hand, every 4-connected plane triangulation on n vertices has weak line cover number at most $\sqrt{2n}$ [11].

Recently, there is some effort to better understand the drawing of graphs and hypergraphs in 3D. For example, we now know that every graph has a contact representation in 3D where vertices are represented by pairwise interior-disjoint convex polygons and edges by vertex–vertex contacts between the corresponding polygons [10]. (Unfortunately, our construction needs exponential space.) If we insist that the (convex) polygons that correspond to two adjacent vertices must share an edge rather than a vertex, not all graphs can be represented. For example, it is not difficult to see that K_5 does not admit a representation with convex quadrilaterals. On the other hand, $K_{4,4}$ admits such a representation, and there is an unbounded family of graphs with n vertices and $6n - o(n)$ edges that admit edge-contact representations with convex polygons in 3D [1]. Hence, such graphs can be considerably denser than pla-

*Institute of Computer Science, University of Würzburg, Germany. alexander.wolff@uni-wuerzburg.de

nar graphs.

For hypergraphs, the situation is difficult even in the setting with vertex–vertex contacts. For example, the Fano plane (which is the smallest Steiner triple system $S(2,3,7)$) and the Steiner triple system $S(2,3,9)$ admit a representation where vertices are represented by points and hyperedges by (pairwise interior-disjoint) triangles connecting the corresponding points.

On the other hand, the 3-uniform complete hypergraph with six (or more) vertices does not admit such a representation [2]. This can be shown by analyzing the *link graph* of an arbitrary vertex v of this hypergraph. This is the graph that lives on the boundary of a small sphere around v. It has a vertex for each vertex other than v and an edge for each hyperedge incident to v. This graph must be planar, but it would have too many edges for large 3-uniform complete hypergraphs. Mostly with the same technique, one can show that no Steiner quadruple systems $S(3,4,n)$ admits a representation with touching *convex* quadrilaterals [10]. Now consider the smallest projective plane $PG(3)$ $(= S(2,4,13))$. It has 13 vertices and 13 hyperedges of size 4. We don't know whether it admits a representation with (convex) quadrilaterals or with tetrahedra.

An earlier version of this talk was joint work with André Schulz [15].

References

[1] Elena Arseneva, Linda Kleist, Boris Klemz, Maarten Löffler, André Schulz, Birgit Vogtenhuber, and Alexander Wolff. Representing graphs by polygons with edge contacts in 3D. In Steven Chaplick, Philipp Kindermann, and Alexander Wolff, editors, *Proc. 36th European Workshop on Computational Geometry (EuroCG'20)*, pages 53:1–8, 2020.

[2] Johannes Carmesin. Embedding simply connected 2-complexes in 3-space – I. A Kuratowski-type characterisation. ArXiv report, 2019. URL: `http://arxiv.org/abs/1709.04642`.

[3] Steven Chaplick, Krzysztof Fleszar, Fabian Lipp, Alexander Ravsky, Oleg Verbitsky, and Alexander Wolff. Drawing graphs on few lines and few planes. In Yifan Hu and Martin Nöllenburg, editors, *GD 2016*, volume 9801 of *LNCS*, pages 166–180. Springer, 2016. URL: `http://arxiv.org/abs/1607.01196`, `doi:10.1007/978-3-319-50106-2_14`.

[4] Steven Chaplick, Krzysztof Fleszar, Fabian Lipp, Alexander Ravsky, Oleg Verbitsky, and Alexander Wolff. Drawing graphs on few lines and few planes. In Yifan Hu and Martin Nöllenburg, editors, *Proc. 24th Int. Symp. Graph Drawing & Network Vis. (GD'16)*, volume 9801 of *LNCS*, pages 166–180. Springer, 2016. `doi:10.1007/978-3-319-50106-2_14`.

[5] Steven Chaplick, Krzysztof Fleszar, Fabian Lipp, Alexander Ravsky, Oleg Verbitsky, and Alexander Wolff. The complexity of drawing graphs on few lines and few planes. In Faith Ellen, Antonina Kolokolova, and Jörg-Rüdiger Sack, editors, *WADS 2017*, volume 10389 of *LNCS*, pages 265–276. Springer, 2017. URL: `http://arxiv.org/abs/1607.06444`, `doi:10.1007/978-3-319-62127-2_23`.

[6] Pierluigi Crescenzi, Giuseppe Di Battista, and Adolfo Piperno. A note on optimal area algorithms for upward drawings of binary trees. *Comput. Geom. Theory Appl.*, 2(4):187–200, 1992. `doi:10.1016/0925-7721(92)90021-J`.

[7] Hubert de Fraysseix, János Pach, and Richard Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10(1):41–51, 1990. `doi:10.1007/BF02122694`.

[8] Vida Dujmović, David Eppstein, Matthew Suderman, and David R. Wood. Drawings of planar graphs with few slopes and segments. *Comput. Geom. Theory Appl.*, 38(3):194–212, 2007. `doi:10.1016/j.comgeo.2006.09.002`.

[9] David Eppstein. Cubic planar graphs that cannot be drawn on few lines. In *Proc. 35th Int. Symp. Comp. Geom. (SoCG'19)*, volume 129 of *LIPIcs*, pages 32:1–32:15, 2019. `doi:10.4230/LIPIcs.SoCG.2019.32`.

[10] William Evans, Paweł Rzążewski, Noushin Saeedi, Chan-Su Shin, and Alexander Wolff. Representing graphs and hypergraphs by touching polygons in 3D. In Daniel Archambault and Csaba D. Tóth, editors, *Proc. 27th Int. Symp. Graph Drawing & Network Vis. (GD'19)*, volume 11904 of *LNCS*, pages 18–32. Springer, 2019. `doi:10.1007/978-3-030-35802-0_2`.

[11] Stefan Felsner. 4-connected triangulations on few lines. In Daniel Archambault and Csaba D. Tóth, editors, *Proc. 24th Int. Symp. Graph Drawing & Network Vis. (GD'16)*, volume 11904 of *LNCS*, pages 395–408. Springer, 2019. URL: `https://arxiv.org/abs/1908.04524`, `doi:10.1007/978-3-030-35802-0_30`.

[12] Yoshio Okamoto, Alexander Ravsky, and Alexander Wolff. Variants of the segment number of

a graph. In Daniel Archambault and Csaba D. Tóth, editors, *Proc. 27th Int. Symp. Graph Drawing & Network Vis. (GD'19)*, volume 11904 of *LNCS*, pages 430–443. Springer, 2019. doi:10.1007/978-3-030-35802-0_33.

[13] Walter Schnyder. Embedding planar graphs on the grid. In David S. Johnson, editor, *Proc. 1st ACM-SIAM Symp. Discrete Algorithms (SODA'90)*, pages 138–148, 1990. URL: https://dl.acm.org/citation.cfm?id=320191.

[14] André Schulz. Drawing graphs with few arcs. *J. Graph Alg. Appl.*, 19(1):393–412, 2015. doi:10.7155/jgaa.00366.

[15] André Schulz and Alexander Wolff. Survey on graph and hypergraph drawing. In Maarten Löffler, Anna Lubiw, Saul Schleimer, and Erin Moriarty Wolf Chambers, editors, *Computation in Low-Dimensional Geometry and Topology (Dagstuhl Seminar 19352)*, volume 9 (number 8) of *Dagstuhl Reports*, pages 87–89. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/DagRep.9.8.84.

Angle-Monotonicity of Delaunay Triangulation

Davood Bakhshesh* Mohammad Farshi†

Abstract

Given an angle $\gamma > 0$, a geometric path $(v_1, \ldots, v_k)$ is called *angle-monotone with width* γ if, for any two integers $1 \leq i, j < k$, the angle between the two vectors $\overrightarrow{v_iv_{i+1}}$ and $\overrightarrow{v_jv_{j+1}}$ is at most γ. Let S be a set of n points in the plane. A geometric graph G with vertex set S is called *angle-monotone with width* γ, if there exists an angle-monotone path with width γ between every pair of vertices of G. In this paper, we show that the Delaunay triangulation of a given point set in the plane is not necessarily angle-monotone with width α, for $0 < \alpha \leq 140°$. This makes significant progress towards solving an open problem posed by Bonichon et al. (*Bonichon et al., Gabriel triangulations and angle-monotone graphs: Local routing and recognition, Graph Drawing and Network Visualization, 2016*).

1 Introduction

Let S be a set of points in the plane. A weighted graph $G = (S, E)$ is called *geometric*, if any edge $(p, q) \in E$ is the straight line between p and q and the weight of the edge (p, q) is the Euclidean distance between p and q denoted by $|pq|$. Let $t > 1$ be a real number. The graph G is called a t-spanner for S, if for every pair of points $p, q \in S$, there exists a path Q between p and q in G such that $|Q| \leq t|pq|$, where $|Q|$ is called the length of Q which is the sum of the weight of all edges of Q. The smallest value of t such that G is a t-spanner for S is called the *stretch factor* (*dilation*) of G.

In 2014, Dehkordi et al. [2] introduced the concept of *angle-monotone* graphs as a new family of geometric graphs that have an angle-monotone path as a "good" path between each pair of vertices. A geometric path $(v_1, \ldots, v_n)$ is called angle-monotone, if there exists a $90°$ closed wedge such that for each $1 \leq i < n$, the vector $\overrightarrow{v_iv_{i+1}}$ of the edge (v_i, v_{i+1}) lies in this wedge. A geometric graph is called angle-monotone, if for any two vertices, there is an angle-monotone path in the graph between them. In 2016, Bonichon et al. [1], generalized the concept of angle-monotone graphs. Let $0 < \gamma < \pi$ be a real number. Bonichon et al. [1] introduced the *angle-monotone graphs with width* γ. A geometric path $(v_1, \ldots, v_n)$ is called angle-monotone with width γ, if there is a closed wedge of angle γ such that for each $1 \leq i < n$, the vector $\overrightarrow{v_iv_{i+1}}$ lies in this wedge. This definition is equivalent to the following definition: a geometric path $(v_1, \ldots, v_n)$ is called angle-monotone with width γ if, for any two integers $1 \leq i, j < k$, the angle between the two vectors $\overrightarrow{v_iv_{i+1}}$ and $\overrightarrow{v_jv_{j+1}}$ is at most γ. A geometric graph is called angle-monotone with width γ, if for any two vertices of the graph, there is an angle-monotone path with width γ connecting them.

Bonichon et al. [1], posed the following open problem.

Open Problem [1]. Is there a constant $0 < \gamma < \pi$ such that the standard Delaunay triangulation is angle-monotone with width γ?

In this paper, we prove there exist Delaunay triangulations that are not angle monotone with width $< 140°$. This makes significant progress towards solving the above open problem.

2 Preliminaries

Let $n > 12$ be an integer such that $n - 12$ is divisible by 18. Throughout the paper, suppose that $S := \{s_0, s_1, \ldots, s_{n-1}\}$ is a set of n points in the plane placed at vertices of a regular n-gon. We assume that the points of S are placed in clockwise direction, and s_0 and $s_{n/2}$ are on a horizontal line and s_0 is to the left of $s_{n/2}$ (see Figure 1(a)). Let C be the circumcircle of S and O its center. We may assume, without loss of generality, that the radius of C is one. Let $P = S \cup \{A, B\}$, where A and B are two points in the plane such that $|OA| = |OB|$ and s_0, A and O are collinear and $s_{\frac{n-6}{3}}, B$ and O are collinear. Let $f = \frac{2n-3}{3}$. Let D be the circumcircle of the triangle $\triangle ABs_f$ and let O' be the center of D. Throughout this paper, the subscripts are taken modulo n.

Now, we have the following observation.

Observation 1 *For each $0 \leq a \leq n-1$ and $k \geq 0$, the segment s_as_{a+1} is parallel to the segment $s_{a-k}s_{a+1+k}$*

Now, we have the following lemma.

Lemma 1 *The points O, O' and s_f are collinear and the radius of D is less than the radius of C.*

*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

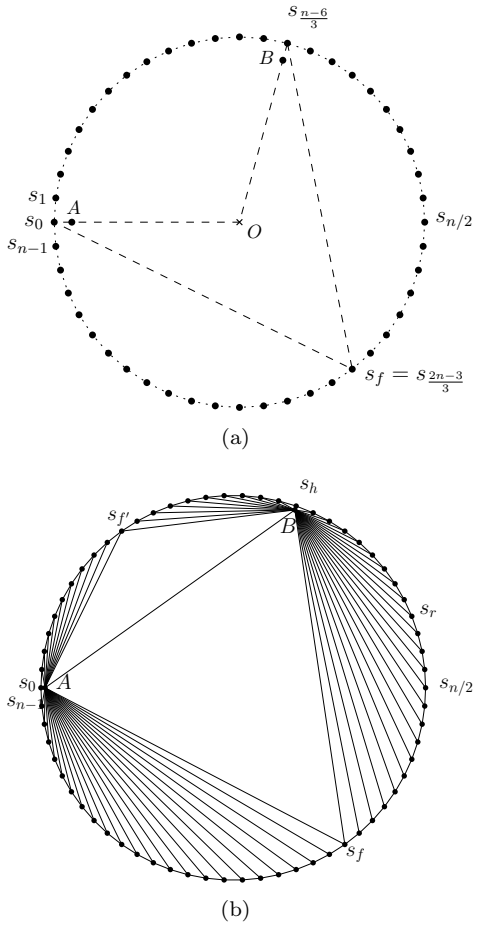


Figure 1: (a) The point set of S for $n = 48$. The center of the circumcircle of S is shown by O. (b) $DT(P)$ for $n = 48$, where $P = S \cup \{A, B\}$.

Let $f \leq k \leq n-1$ be an integer. Let F_k be the circumcircle of the triangle $\triangle s_k s_{k+1} A$ and let O_k be the center of F_k. Let m_k be the midpoint of the segment $s_k s_{k+1}$. Now, we have the following result.

Lemma 2 *The points O, O_k and m_k are collinear and the radius of F_k is less than the radius of C.*

3 Delaunay triangulation of the point set P

Let $DT(P)$ be the Delaunay triangulation of the point set P. Note that we added the points A and B to P, because we want $DT(P)$ to be unique. Now, we prove the following lemma.

Lemma 3 *$DT(P)$ contains the triangle $\triangle AB s_f$.*

Lemma 4 *For each $f \leq k \leq n-1$, $DT(P)$ contains the triangle $\triangle s_k s_{k+1} A$.*

Let $f' = \frac{n-6}{6}$. Note that the midpoint of the segment ff' is the point O. Similar to the proof of Lemma 3 and Lemma 4, by symmetric arguments, we can prove the following lemmas.

Lemma 5 *$DT(P)$ contains the triangle $\triangle AB s_{f'}$.*

Lemma 6 *For each $0 \leq k < f'$, $DT(P)$ contains the triangle $\triangle s_k s_{k+1} A$.*

Lemma 7 *For each $f' \leq k < f$, $DT(P)$ contains the triangle $\triangle s_k s_{k+1} B$.*

In Figure 1(b), using Lemmas 3, 4, 5, 6 and 7, $DT(P)$ is shown for $n = 48$.

4 Angle-monotonicity property

In this section, we present a point set P' in the plane and we show that for every angle $\alpha \leq 140°$, the Delaunay triangulation of P' is not angle-monotone with width α. The point set P' is same as the point set P except that in P' we assume that $|OA| = |OB| = 1 - \epsilon$, where $\epsilon > 0$ is a small enough real number.

Let $Q = (v_1, \ldots, v_k)$ be a directed geometric path from v_1 to v_k. We may assume, without loss of generality, that v_1 and v_k are on a horizontal line and v_k is to the right of v_1. Let $m(Q) = \max_{1 \leq i < j \leq n} \{angle(v_i v_{i+1}, v_j v_{j+1})\}$, where $angle(v_i v_{i+1}, v_j v_{j+1})$ is the angle between $\overrightarrow{v_i v_{i+1}}$ and $\overrightarrow{v_j v_{j+1}}$. According to the definition of an angle-monotone path with width γ, we have the following lemma.

Lemma 8 *For every $\gamma < m(Q)$, Q is not angle-monotone with width γ.*

Let $h = \frac{n-6}{3}$. Let r be an integer such that $\frac{n+3}{3} < r < \frac{n-2}{2}$. Consider five paths Q_1, Q_2, Q_3, Q_4 and Q_5 from s_{n-1} to s_r as follows:

- $Q_1 = (s_{n-1}, A, B, s_r)$,
- $Q_2 = (s_{n-1}, s_{n-2}, s_{n-3}, \ldots, s_r)$,
- $Q_3 = (s_{n-1}, s_{n-2}, s_{n-3}, \ldots, s_f, B, s_r)$,
- $Q_4 = (s_{n-1}, A, s_f, B, s_r)$.
- $Q_5 = (s_{n-1}, A, s_f, s_{f-1}, \ldots, s_r)$.

Now, we prove the following result.

Lemma 9 *For paths $Q_1, \ldots, Q_5$, it holds that*

$$m(Q_1) = \frac{(n+3r-3)\pi}{3n}, m(Q_2) = \frac{(2n-2r-4)\pi}{n},$$
$$m(Q_3) = \pi,$$
$$m(Q_4) = \frac{(n+3r+3)\pi}{3n}, m(Q_5) = \frac{(5n-6r-6)\pi}{3n}.$$

Proof. Note that according to the construction of P', we may assume that $\angle s_0 s_{n-1} A = \angle s_h s_{h+1} B = 0$. This assumption does not affect our results because we can move the points A and B to s_0 and s_h, respectively, as much as we need. So, even though the points A and s_0 are different, we can assume that $A = s_0$. Also, we may assume that $B = s_h$. In the following, our calculations are based on this assumption.

- **Calculation of** $m(Q_1)$. By Observation 1, the segment $s_{n-1}s_{h+1}$ is parallel to the segment $s_0 s_{s_h}$ (see Figure 2). Then, $angle(s_{n-1}s_0, s_0 s_h) = angle(s_0 s_h, s_h s_{h+1})$. Since $r > h$, clearly $angle(s_0 s_h, s_h s_{h+1}) \leq angle(s_0 s_h, s_h s_r)$. Then,

  $$m(Q_1) = \max\{angle(s_0 s_h, s_h s_r), angle(s_{n-1}s_0, s_h s_r)\}.$$

  It is not hard to see that $angle(s_0 s_h, s_h s_r) \leq angle(s_{n-1}s_0, s_h s_r)$. Hence, $m(Q_1) = angle(s_{n-1}s_0, s_h s_r)$. By Observation 1, segment $s_{n-1}s_0$ is parallel to the segment $s_{n-h-1}s_h$. Then, we have $angle(s_{n-1}s_0, s_h s_r) = angle(s_{n-h-1}s_h, s_h s_r)$, and therefore $m(Q_1) = angle(s_{n-h-1}s_h, s_h s_r)$. Since $h = \frac{n-6}{3}$ and for any $0 \leq a \leq n-1$, $\angle s_a O s_{a+1} = \frac{2\pi}{n}$, we have

  $$m(Q_1) = \pi - \frac{(n-h-1-r)\pi}{n} = \frac{(n+3r-3)\pi}{3n}.$$

- **Calculation of** $m(Q_2)$. It is not hard to see that the maximum angle between the edges on the path Q_2 is obtained by two edges (s_{n-1}, s_{n-2}) and (s_{r+1}, s_r) (see Figure 2).

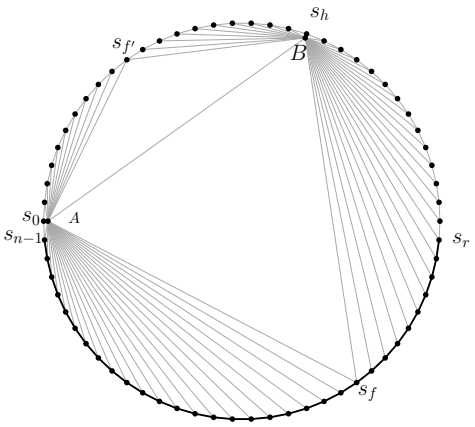


Figure 2: Illustrating proof of Lemma 9.

  By Observation 1, the segment $s_{n-r-4}s_{r+1}$ is parallel to the segment $s_{n-2}s_{n-1}$. Then, we have $angle(s_{n-1}s_{n-2}, s_{r+1}s_r) = angle(s_{n-r-4}s_{r+1}, s_{r+1}s_r)$. Hence,

  $$m(Q_2) = \pi - \frac{(r-(n-r-4))\pi}{n} = \frac{(2n-2r-4)\pi}{n}.$$

- **Calculation of** $m(Q_3)$. Note that $f = \frac{2n-3}{3}$ and $h = \frac{n-6}{3}$. By Observation 1, the segment $s_{n-2}s_{n-1}$ is parallel to the segment $s_f s_{\frac{n-6}{3}}$ (see Figure 2). Note that $\frac{4n-6}{3}$ in modulo n is equal to $\frac{n-6}{3}$. Then, $angle(s_{n-1}s_{n-2}, s_f s_h) = \pi = 180°$. Now, it is not hard to see that the maximum angle between the edges on the path Q_3 is obtained by two edges (s_{n-1}, s_{n-2}) and (s_f, s_h). Hence, $m(Q_3) = \pi$.

- **Calculation of** $m(Q_4)$. To compute $m(Q_4)$, we need to compute the four angles $angle(s_{n-1}s_0, s_0 s_f), angle(s_{n-1}s_0, Bs_r), angle(s_0 s_f, s_f B)$ and $angle(s_f B, Bs_r)$ (see Figure 2). Note that we do not need to compute the other angles on the path Q_4 such as $angle(s_{n-1}s_0, s_f s_h)$, because $angle(s_{n-1}s_0, s_f s_h)$ is almost equal to $angle(s_{n-2}s_{n-1}, s_f s_h)$. This is because that $angle(s_{n-2}s_{n-1}, s_f s_h) = \pi - angle(s_{n-1}s_{n-2}, s_f s_h) = 0$ (see calculation of $m(Q_3)$). Therefore, obviously these angles are smaller than the four angles. Now, we compute these four angles.

  1.
  $$\begin{aligned} angle(s_{n-1}s_0, s_0 s_f) &= \pi - \frac{(n-1-f)\pi}{n} \\ &= \pi - \frac{\left(n-1-\frac{2n-3}{3}\right)\pi}{n} \\ &= \frac{2\pi}{3}. \end{aligned}$$

  2. By Observation 1, the segment $s_{n-1}s_0$ is parallel to the segment $s_{n-h-1}s_h$. Therefore, $angle(s_{n-1}s_0, s_h s_r) = angle(s_{n-h-1}s_h, s_h s_r)$. Now, since $h = \frac{n-6}{3}$, we have

  $$\begin{aligned} angle(s_{n-h-1}s_h, s_h s_r) &= \pi - \frac{(n-h-1-r)\pi}{n} \\ &= \frac{(n+3r-3)\pi}{3n}. \end{aligned}$$

  3.
  $$angle(s_0 s_f, s_f s_h) = \pi - \frac{h\pi}{n} = \frac{(2n+6)\pi}{3n}$$

  4.
  $$angle(s_f s_h, s_h s_r) = \pi - \frac{(f-r)\pi}{n} = \frac{(n+3r+3)\pi}{3n}.$$

  Since we assumed that $r > \frac{n+3}{3}$, it is easy to show that $\frac{n+3r+3}{3n} > \frac{2n+6}{3n}$, $\frac{n+3r+3}{3n} > \frac{n+3r-3}{3n}$ and $\frac{n+3r+3}{3n} > \frac{2}{3}$. Hence, we have

  $$m(Q_4) = \frac{(n+3r+3)\pi}{3n}.$$

- **Calculation of $m(Q_5)$.** It is not hard to see that to compute $m(Q_5)$ on the path Q_5, it is sufficient to calculate $angle(s_{n-1}s_0, s_0s_f)$ and $angle(s_0s_f, s_{r+1}s_r)$ (see Figure 2).

  Similar to the calculation of $m(Q_4)$, we have $angle(s_{n-1}s_0, s_0s_f) = \frac{2\pi}{3}$. Now, we compute $angle(s_0s_f, s_{r+1}s_r)$. By Observation 1, the segment s_fs_0 is parallel to the segment $s_{r+1}s_{f-r-1}$. Then, $angle(s_0s_f, s_{r+1}s_r) = angle(s_{f-r-1}s_{r+1}, s_{r+1}s_r)$. Now, since $f = \frac{2n-3}{3}$, we have

$$angle(s_{f-r-1}s_{r+1}, s_{r+1}s_r) = \pi - \frac{(r-(f-r-1))\pi}{n} = \frac{(5n-6r-6)\pi}{3n}.$$

  Since $r < \frac{n-2}{2}$, we have $\frac{(5n-6r-6)\pi}{3n} > \frac{2\pi}{3}$. Hence,

$$m(Q_5) = \frac{(5n-6r-6)\pi}{3n}. \qquad \square$$

Theorem 10 *For any $\alpha \leq \frac{(7n-12)\pi}{9n}$, $DT(P')$ is not angle-monotone with width α.*

Proof. Let $r = \frac{4n-3}{9}$. It is easy to see that $\frac{n+3}{3} < r < \frac{n-2}{2}$. Now, let Q be a path from s_{n-1} to s_r such that $m(Q)$ is minimum. According to the shape of $DT(P')$, it is not hard to see that the paths Q_1, Q_2, Q_3, Q_4 and Q_5 are only candidates for the path Q. Note that some of the paths from s_{n-1} to s_r such as $(s_{n-1}, A, s_{f'}, B, s_r)$ cannot be the path Q because some of the edges on the path increase the value of $m(Q)$. Hence,

$$m(Q) = \min\{m(Q_1), m(Q_2), m(Q_3), m(Q_4), m(Q_5)\}.$$

By Lemma 9, we have

$$m(Q_1) = \frac{(n+3r-3)\pi}{3n}, m(Q_2) = \frac{(2n-2r-4)\pi}{n},$$
$$m(Q_3) = \pi, m(Q_4) = \frac{(n+3r+3)\pi}{3n},$$
$$m(Q_5) = \frac{(5n-6r-6)\pi}{3n}.$$

Now, since $m(Q_3) = \pi$ and $m(Q_i) \leq \pi$, we have

$$m(Q) = \min\{m(Q_1), m(Q_2), m(Q_4), m(Q_5)\} = \min\{m(Q_1), m(Q_2), m(Q_5)\}.$$

On the other hand, it is easy to see that $\frac{5n-6r-6}{3n} < \frac{2n-2r-4}{n}$. Therefore, $m(Q_5) < m(Q_2)$. Since $r < \frac{n-2}{2} < \frac{5n-9}{9}$, it is easy to see that $\frac{n+3r-3}{3n} < \frac{2n-2r-4}{n}$. Then, $m(Q_1) < m(Q_2)$. Hence, $m(Q) = \min\{m(Q_1), m(Q_5)\}$.

It is easy to see that for $r = \frac{4n-3}{9}$, we have

$$\frac{(n+3r-3)\pi}{3n} = \frac{(5n-6r-6)\pi}{3n}.$$

Hence, by substituting $r = \frac{4n-3}{9}$ in $m(Q_1)$ and $m(Q_5)$, we have $m(Q) = \frac{(7n-12)\pi}{9n}$. Then, by Lemma 8, for any $\alpha \leq \frac{(7n-12)\pi}{9n}$, Q is not angle-monotone with width α. This completes the proof. $\square$

Since

$$\lim_{n\to\infty} \frac{(7n-12)\pi}{9n} = \frac{7\pi}{9} = 140^\circ,$$

using Theorem 10, we easily conclude the following result.

Corollary 1 *The Delaunay triangulation of a given point set in the plane is not necessarily angle-monotone with width less than or equal to* 140°.

5 Conclusion

In this paper, we showed that for any $\alpha \leq \frac{7\pi}{9} = 140^\circ$, the standard Delaunay triangulation of a given point set is not necessarily angle-monotone with width α. We leave the following open problems.

1. Is there a constant $140^\circ < \alpha < 180^\circ$ such that the standard Delaunay triangulation is angle-monotone with width α?

2. Are there other types of triangulations that are known to be angle-monotone with some width $< 140^\circ$?

References

[1] N. Bonichon, P. Bose, P. Carmi, I. Kostitsyna, A. Lubiw, and S. Verdonschot. Gabriel triangulations and angle-monotone graphs: Local routing and recognition. In *Graph Drawing and Network Visualization*, pages 519–531, 2016.

[2] H. R. Dehkordi, F. Frati, and J. Gudmundsson. Increasing-chord graphs on point sets. In *Graph Drawing and Network Visualization*, pages 464–475, 2014.

Never Absent for Long and Never Far Away

Ali Gholami Rudi* Fatemeh Golchin†

Abstract

A stay region of a moving entity is a region in the plane in which it spends a significant amount of time. In this paper we find stay regions with the following definition. Let α and g be constants, such that $\alpha > 1$ and $g > 0$. A stay region of a trajectory is an axis-aligned square, such that the moving entity is never outside it, unless for durations of at most g. Also, if the side length of the stay region is s, the entity should never leave an axis-aligned square with the same centre but of side length αs. The goal is finding a stay region of the minimum size. We present an algorithm that, for any real constant ϵ where $0 < \epsilon < 1$, finds a stay region of side length at most $(1 + \epsilon) \cdot P$ in almost linear time, in which P is the side length of the optimal answer.

Keywords: Geometric algorithms, trajectory analysis, stay regions.

2010 Mathematics subject classification: 68U05.

1 Introduction

The movement of various objects such as cars, animals, and mobile devices are collected to extract valuable information, either about the object itself or about its surrounding environment [10, 3, 5, 1]. Performing such analyses on trajectories, which can be very large in available trajectory databases, requires efficient algorithms. One of the analyses performed on trajectories is finding the regions in which the moving entity has spent a significant amount of time [9].

Different definitions for stay regions (also called stay points, popular or interesting places, or hotspots) has been examined and algorithms have been presented for identifying them [4, 9, 8, 11, 2, 12, 13]. Among the firsts, Benkert et al. [4] defined a popular place to be an axis-aligned square of fixed side length in the plane which is visited by the most number of distinct trajectories. They modelled a visit either as containing of a trajectory vertex or containing of any portion of a trajectory edge, and presented optimal algorithms for finding them with time complexities $O(n \log n)$ and $O(n^2)$, respectively, in which n is the total number of trajectory vertices. Gudmundsson et al. [9] introduced several different definitions of trajectory hotspots. In some of these definitions, a hotspot is an axis-aligned square that contains a continuous sub-trajectory with the maximum duration and in others it is an axis-aligned square in which the entity spends the maximum possible duration but its presence may not be continuous. For hotspots of fixed side length, for the former they presented an $O(n \log n)$ algorithm and for the latter they presented an algorithm with the time complexity $O(n^2)$ to find the hotspots of a trajectory with n vertices. For the case where the edges of a trajectory are orthogonal, Rudi [13] presented an 1/2-approximation algorithm with the time complexity $O(n \log^3 n)$ to find non-continuous hotspots of a trajectory. Heuristic algorithms have also been published to identify hotspots with a definition similar to the ones considered by Gudmundsson et al. [9], such as the one presented by Damiani et al. [6].

There are applications in which we need to identify regions that are regularly visited. Djordjevic et al. [7] concentrated on a limited form of this problem and presented an algorithm to decide if a region is visited almost regularly (in fixed periods of time) by an entity.

However, in many applications that require spatio-temporal analysis, we are interested in finding regions that are never left for a long time. Examples include bird nests, animal resting place, player posts in sports, and bus stations. For this problem, Arboleda et al. [2] assumed that the input consists, in addition to the trajectories, of a set of polygons as potential stay points or interesting sites. They presented a simple algorithm to identify stay points among the given interesting sites; their algorithm computes the longest sub-trajectory visiting each interesting site, while allowing the entity to leave the site for some predefined amount of time. They also mentioned motivating real world examples to show that in some applications, it makes sense to allow the entity to leave the site for short periods of time, like leaving a cinema for the bathroom.

Rudi [12] presented a $(1+\epsilon)$-approximation algorithm to find all axis-aligned squares, in which the entity is always present, except for short periods of time. His algorithm runs in $O(kn^2)$ time, in which k depends on ϵ and the length of trajectory edges. The main idea of his approximation algorithm is considering snapshots of the regions that can be a stay regions and computing

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

†Department of Computer Engineering, Sharif University of Technology

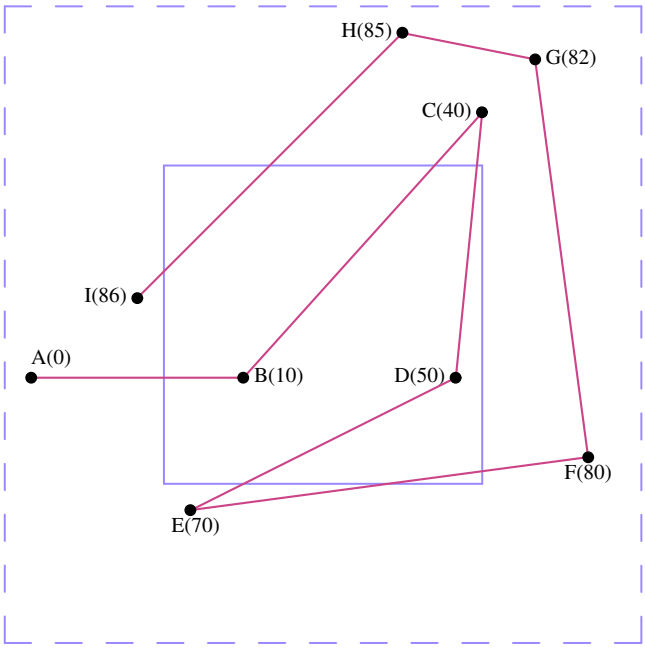


Figure 1: An example trajectory stay region for $\alpha = 2$ and $g = 10$.

their intersection. We study a similar problem, with the difference that we assume the side length of a stay region is not known beforehand and our goal is finding a stay region of the minimum possible size. Also, we assume that the entity never moves very far from the stay region.

This paper is organised as follows. In Section 2 we introduce the notation of the paper. In Section 3 we present the approximation algorithm, and finally, in Section 4 we present our experimental results and conclude this paper.

2 Preliminaries

A trajectory is a sequence of points on the plane, which we call its vertices. Each vertex has a timestamp, indicating the time at which the entity was observed at that point. The entity is assumed to have moved from each vertex to its next in a straight line and at constant speed (the timestamps of the sequence of vertices of a trajectory is nondecreasing). The segments connecting contiguous vertices of a trajectory are its edges.

Throughout this paper, we assume α and g are input constants such that $\alpha \geq 1$ and $g > 0$. A *stay region* is defined as follows.

Definition 1 *An axis-aligned square R with side length s is a stay region of trajectory T, if the entity is never outside it for time more than g and never leaves the axis-aligned square with side length αs and with the same centre of gravity as R.*

Figure 1 demonstrates a stay region for an example trajectory. The trajectory starts from the vertex marked as A and ends at I. The numbers inside parentheses are timestamps. The solid square is a stay region. Our goal in this paper is finding the smallest possible stay region, as defined in Definition 2.

Definition 2 *An optimal stay region of a trajectory T is a stay region with the minimum possible size. A $(1+\epsilon)$-approximate stay region is a stay region whose side length is at most $(1+\epsilon)$ times the side length of an optimal one, in which $\epsilon > 0$.*

Optimal (exact) stay regions of a trajectory may be found using the algorithm presented by Gudmundsson et al. [9], with the worst-case time complexity $\Omega(n^2)$.

3 Finding the Stay Regions

In this section we present an algorithm for finding approximate stay regions of a trajectory. Lemma 1 shows that the minimum size of a stay region is bounded.

Lemma 1 *Let t be the side length of a square R of minimum size containing all edges of trajectory T. Also, let s be the side length of an optimal stay region. We have*

$$t/\alpha \leq s \leq t.$$

Proof. Obviously, s cannot be larger than t, because R is a stay region by Definition 1. On the other hand, because R is the smallest square containing T, the side length of any stay region cannot be smaller than t/α; otherwise, there cannot exist a square of side length αs containing the whole trajectory, as required by Definition 1. □

Let G_s be a grid, formed by parallel vertical lines $x = i \cdot s$ and parallel horizontal lines $y = j \cdot s$, for every possible integral values of i and j. Each $s \times s$ cell of G_s is identified using a tuple $G_s(i, j)$, whose lower left corner is at (is, js). For any point $p = (p_x, p_y)$ on the plane, the cell that contains p is $G_s(\lfloor p_x/s \rfloor, \lfloor p_y/s \rfloor)$; we also use $G_s(p)$ to refer to this cell. We assume that the computation model is strong enough to compute the floor function in constant time.

Lemma 2 *Let ϵ be a real constant such that $\epsilon > 0$. Consider any value of s ($s > 0$) and define s' as $s + \epsilon s$. Define a set of vectors V as follows:*

$$V = \{(-i\epsilon s, -j\epsilon s) \mid \forall \text{ integers } i, j \text{ such that } 0 \leq i, j \leq 1/\epsilon\}$$

For any axis-aligned square R of side length s there exists a member v of V such that $R + v$ is contained in a single cell of grid $G_{s'}$.

Proof. Obviously, the size of V is $O(1/\epsilon^2)$. Let $p = (x, y)$ be the lower left corner of R, such that $x = as' + a'$ and $y = bs' + b'$ for integers a and b, such that $0 \leq a', b' < s'$. This implies that point (x, y) is in cell $G_{s'}(a, b)$ of $G_{s'}$. Let $i = \lfloor a'/\epsilon s \rfloor$ and $j = \lfloor b'/\epsilon s \rfloor$. The vector $(-i\epsilon s, -j\epsilon s)$ is in V. The distance of $p + v$ and the left side of $G_{s'}(a, b)$ is $a' - i\epsilon s$, which is smaller than ϵs, based on the definition of i. Similarly, the distance from $p+v$ and the lower side of $G_{s'}(a, b)$ is smaller than ϵs. This implies that R (with dimensions $s \times s$) is contained in $G_{s'}(a, b)$ (with dimensions $s' \times s'$). □

When the side length of stay regions is specified, Theorem 3 uses Lemma 2 to find a stay region, which may be slightly larger.

Theorem 3 *Let ϵ be any positive, non-zero real constant. For a trajectory T with n vertices, if there exists a stay region of side length s, a stay region of side length $s + \epsilon s$ can be computed with the time complexity $O(n\alpha^2/\epsilon^2)$.*

Proof. Let $P = \langle p_1, p_2, ..., p_n \rangle$ be the sequence of the vertices of T, ordered by their timestamps. Define s' as $s + \epsilon s$ and consider the grid $G_{s'}$. Let V be the set of vectors defined in Lemma 2. For each vector v in V, we repeat the following steps for trajectory $T' = T + v$ (each vertex of T is moved by vector v).

1. We can find the minimum and maximum values of x and y coordinates of these vertices in $O(n)$ time. Let C be the set of all cells of $G_{s'}$ containing any part of the smallest square R containing the whole of T' such that the whole trajectory is contained in the square with the same centre but of side length αs.

2. For each cell c of C we store two numbers: c_{last} is the timestamp of the last visit to this cell for trajectory T', and c_{stay} specifies if this cell can be a stay region or not. For every cell c of C, initialise c_{last} as the time stamp of p_1 and c_{stay} as true.

3. We consider each trajectory cell c intersected by the trajectory T' as follows.

   (a) Compute the time t at which the trajectory enters c. If $c_{\text{last}} - t$ is greater than g, c cannot be a stay point and we assign false to c_{stay}.

   (b) Compute the time t at which the trajectory leaves c and update the value of c_{last}.

4. Let t be the timestamp of p_n. For each cell c in C, if $c_{\text{last}} - t$ is greater than g, c cannot be a stay point and we assign false to c_{stay}.

5. Report any cell c of C such that c_{stay} is true.

The above procedure is repeated $O(1/\epsilon^2)$ times, once for each vector v in V. Since the side length of R is at most αs (otherwise, no stay region can exist by Lemma 1), the size of C is $\lceil \alpha s/s \rceil \times \lceil \alpha s/s \rceil$, or $O(\alpha^2)$, asymptotically. Given that each edge of the trajectory visits at most $O(\alpha)$ cells of C, the time complexity of step 3 is $O(\alpha n)$. The steps 2, 4, and 5 check each cell of C, and therefore, have time complexity $O(\alpha^2)$. Thus, the time complexity of the whole algorithm is $O(\alpha^2/\epsilon^2 + \alpha n/\epsilon^2)$. □

We present our approximation algorithm for finding an approximate stay region of a trajectory in Theorem 4.

Theorem 4 *Let ϵ be any positive, non-zero real constant. We can find a $(1 + \epsilon)$ stay region of a trajectory T with n vertices with the time complexity $O(n \cdot \alpha^2/\epsilon^2 \log \epsilon/\alpha)$.*

Proof. Let t be the side length of the smallest square containing all vertices of trajectory T. Let o be the side length of an optimal stay region. Lemma 1 implies that o is between t/α and t. Define ϵ' as $\epsilon/3$. Let o' be the smallest value of s for which Theorem 3 finds a stay region when the value of ϵ in the theorem is specified as ϵ'. Based on Theorem 3, we have

$$o' - o \leq \epsilon' o. \tag{1}$$

We perform a binary search with b steps to find the smallest value x for s such that there exists a stay region of size at least x, in which b is defined as follows:

$$b = \left\lceil \log \frac{\alpha}{\epsilon'} \right\rceil$$

Given that in the i-th step of the binary search, the distance between the selected value of s and o is at most $2^{-i}t$, after b steps, the difference between x and o' would be $t/b \leq t\epsilon'/\alpha$. Given that o' is at most t/α, we have

$$x - o' \leq \epsilon' o'. \tag{2}$$

By combining Equations 1 and 2, we get

$$x - o \leq (x - o') + (o' - o),$$

and

$$x - o \leq \epsilon' o + (\epsilon' + \epsilon'^2)o \leq 3\epsilon' o \leq \epsilon o.$$

This implies that the binary search finds a $(1 + \epsilon)$-approximate stay region of T.

Since the algorithm of Theorem 3 is repeated b times, the time complexity of the whole algorithm is $O(n \cdot \alpha^2/\epsilon^2 \log \epsilon/\alpha)$. □

Note that the result Theorem 4 can be extended to handle other fat objects like disks as stay regions, or to find stay regions in higher dimensions.

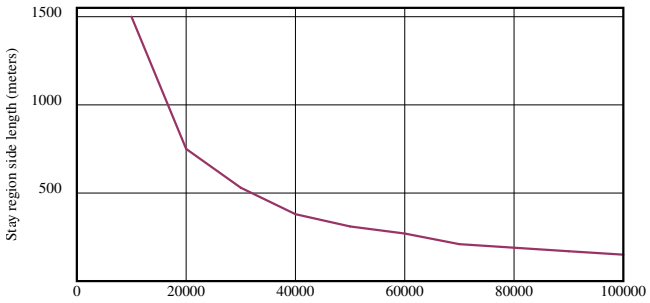


Figure 2: The effect of the value of α on stay region side length (ϵ is .5 and g is five hours)

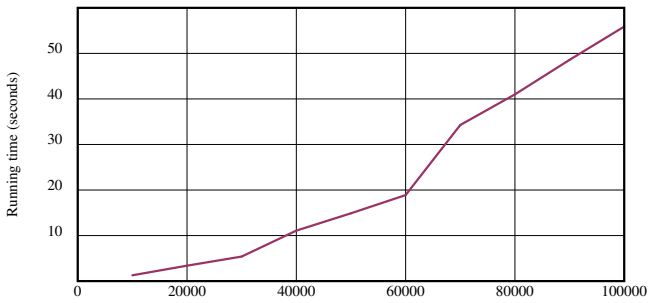


Figure 3: The effect of the value of α on the running time (ϵ is .5 and g is five hours)

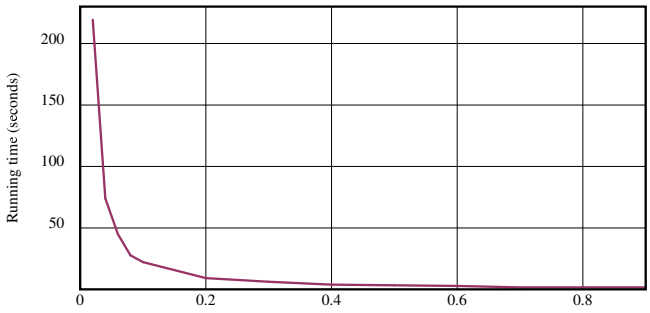


Figure 4: The effect of the value of ϵ on the running time (α is $20k$ and g is five hours)

4 Discussion

To experimentally evaluate the presented algorithm, we implemented it and tested it on real-world trajectories. We used the T-Drive dataset [14], which contains one-week trajectories of more than ten thousand taxis. The algorithm was implemented in C programming language and the experiments were performed in the Ubuntu 14.04 distribution of Linux operating system using an Intel Core i5-4670 3.4GHz processor.

Among the trajectories in T-Drive dataset, we chose the one with the most vertices (about 154 thousand). Figure 2 visualises the size of the computed stay region for different values of α. The implementation, for instance, finds a $1.5km \times 1.5k$ square, to which the trajectory returns every 5 hours and never leaves the containing square of dimension $1500km \times 1500km$.

Figure 3 shows how α affects the running time of the algorithm and Figure 4 shows the effect of ϵ. The implementation seems very fast for $\epsilon = .1$, though, as expected, the running time increases quadratically when it is decreased. Also, increasing α seems to increase the running time of the algorithm, though quadratically, at at a slow rate. Note that in this implementation, in each stop of the binary search of Theorem 4, we stop when for any vector v in V of Theorem 3 a stay region is found and this improves the running time of the implementation.

References

[1] S. P. A. Alewijnse, K. Buchin, M. Buchin, S. Sijben, and M. A. Westenberg. Model-based segmentation and classification of trajectories. *Algorithmica*, 80(8):2422–2452, 2018.

[2] F. J. M. Arboleda, V. Bogorny, and H. Patiño. Smot+ncs - algorithm for detecting non-continuous stops. *Computing and Informatics*, 3(2):283–306, 2017.

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

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

[5] K. Buchin, M. Buchin, M. J. van Kreveld, B. Speckmann, and F. Staals. Trajectory grouping structure. *Journal of Computational Geometry*, 6(1):75–98, 2015.

[6] M. L. Damiani, H. Issa, and F. Cagnacci. Extracting stay regions with uncertain boundaries from GPS trajectories - a case study in animal ecology. In *ACM International Conference on Advances in Geographic Information Systems*, pages 253–262, 2014.

[7] B. Djordjevic, J. Gudmundsson, A. Pham, and T. Wolle. Detecting regular visit patterns. *Algorithmica*, 60(4):829–852, 2011.

[8] M. Fort, J. A. Sellarès, and N. Valladares. Computing and visualizing popular places. *Knowledge and Information Systems*, 40(2):411–437, 2014.

[9] J. Gudmundsson, M. J. van Kreveld, and F. Staals. Algorithms for hotspot computation on trajectory data. In *ACM International Conference on Advances in Geographic Information Systems*, pages 134–143, 2013.

[10] H. J. Miller, S. Dodge, J. A. Miller, and G. Bohrer. Towards an integrated science of movement - converging research on animal movement ecology and human mobility science. *International Journal of Geographical Information Science*, 33(5):855–876, 2019.

[11] R. Pérez-Torres, C. Torres-Huitzil, and H. Galeana-Zapién. Full on-device stay points detection in smartphones for location-based mobile applications. *Sensors*, 16(10):1693, 2016.

[12] A. G. Rudi. Looking for bird nests: Identifying stay points with bounded gaps. In *The Canadian Conference on Computational Geometry*, pages 334–339, 2018.

[13] A. G. Rudi. Approximate hotspots of orthogonal trajectories. *Fundamenta Informaticae*, 167(4):271–285, 2019.

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

Competitive Strategies for Walking in Streets for a Simple Robot Using Local Information

Azadeh Tabatabaei* Mohammad Aletaha† Mohammad Ghodsi‡

Abstract

We consider the problem of walking in an unknown street, for a robot that has a minimal sensing capability. The robot is equipped with an abstract sensor that only detects the discontinuities in depth information (gaps) and can locate the target point as it enters its visibility region. First, we propose an online deterministic search strategy that generates an optimal search path for the simple robot to reach the target t, starting from s. The path created by this strategy is 9-competitive which is proven to be optimal. In contrast with previously known research, the path is designed without memorizing any portion of the scene that has been seen so far. The robot using local information about the location of some gaps achieves the target t starting from s in a street. Then, we present a randomized search strategy, based on the deterministic strategy. Also, a randomized lower bound on the competitive ratio has been proved.

1 Introduction

Path planning is a basic problem to almost all scopes of computer science; such as computational geometry, online algorithms, robotics, and artificial intelligence [3]. Especially, path planning in an unknown environment for which there is no geometric map of the scene is interesting in many real-life cases. Robot sensors are the only tool for gathering information in an unknown street. The amount of information derived from the environment depends on the capability of the robot. Due to the importance of using a simple robot, including low cost, less sensitive to failure, robust against sensing errors and noise, many types of path planning for simple robots have been studied [1, 5, 9].

In this paper, we consider the problem of walking a simple robot in an unknown street. A simple polygon P with two separated vertices s and t is called a street if the left boundary chain L_{chain} and the right boundary chain R_{chain} constructed on the polygon from s to t are mutually weakly visible. In other words, each point on the left chain can see at least one point on the right chain and vice versa [6], see Figure 1(a). A point robot which its sensor has a minimal capability that can only detect discontinuities in depth information (gaps) and the target point t, starts searching the street. The robot can locate the target as soon as it enters its visibility region. Also, the robot cannot measure any angles or distances, or infer its position, see Figure 1. The goal is to reach the target t using the information gathered through its sensor, starting from s such that the traversed path by the robot is as short as possible.

To evaluate the efficiency of a search strategy for the robot, we use the notion of competitive of the competitive analysis. The competitive analysis for a strategy that leads the robot is the ratio of the (expected) distance traversed by the robot over the shortest distance from s to t, in the worst case.

In this paper, first, we present a deterministic strategy using local information about the location of two special gaps which are updated during the walking. The robot achieves the target, without memorizing environment and without using pebbles, in contrast with previously known research [10]. The search path is optimal; the length of the generated path is at most 9 times longer than the shortest path. Then, we present a randomized strategy that generates a search path similar to the deterministic one. We introduced the deterministic strategy and the idea of randomization of that previously in [12].

Related Works: Klein proposed the first competitive algorithm for walking in streets problem for a robot that was equipped with a 360 degrees vision system [6]. Also, Icking, et al. presented an optimal search strategy for the problem with the competitive factor of $\sqrt{2}$ [4]. Many online strategies for patrolling unknown environments such as streets, generalized streets, and star polygons are presented in [3, 7, 13].

The limited sensing model (gap sensor) that our robot is equipped with, in this research, was first introduced by Tovar, et al. [14]. They offered Gap Navigation Tree (GNT) to maintain and update the gaps seen along a navigating path. Some strategies, using GNT for exploring unknown environments, presented in [8, 15].

Tabatabaei, et al. gave a deterministic algorithm for the simple robot to reach the target t in a street and a generalized street, starting from s. The robot us-

*Department of Computer Engineering, University of Science and Culture, Tehran, Iran, a.tabatabaei@usc.ac.ir

†Department of Computer Engineering, Sharif University of Technology, Tehran, Iran, mohammadaletaha@ce.sharif.edu

‡Sharif University of Technology and Institute for Research in Fundamental Sciences (IPM), Tehran, Iran, ghodsi@sharif.edu

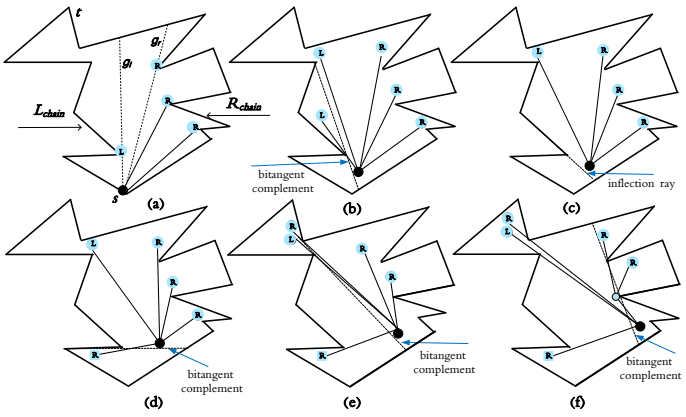


Figure 1: Street polygons and the dynamical changes of the gaps as the robot walks towards a gap in street polygons. The dark circle is the location of the robot, and squares and other circles denote primitive and non-primitive gaps respectively. (a) Existing gaps at the start point. (b) A split event. (c) A disappearance event. (d) An appearance event. (e) Another split event. (f) A merge event.

ing some pebbles and memorizing some portion of the streets has seen so far, explores the street. The target t is achieved such that the traversed path is at most 11 times longer than the shortest path by using one pebble. Also, they showed, allowing the use of many pebbles reduces the factor to 9 [10, 11].

Another minimal sensing model was presented by Suri, et al. [9]. They assumed that the simple robot can only sense the combinatorial (non-metric) properties of the environment. The robot can locate the vertices of the polygon in its visibility region and can report if there is a polygonal edge between them. Despite the minimal ability, they showed that the robot can accomplish many non-trivial tasks. Then, Disser et al. empowered the robot with a compass to solve the mapping problem in polygons with holes [2].

2 Preliminaries

2.1 The Sensing Model and Motion Primitives

The robot has an abstract sensor that reports a cyclically order list of discontinuities in the depth information (gaps) in its visibility region, see Figure 1(a). All the gaps and the target can be located by the robot as they enter in the robots omnidirectional and unbounded field of view. Each gap has a label of L (left) or R (right) which displays the direction of the part of the scene that is hidden behind the gap, see Figure 1.

The robot can orient its heading to each gap and moves towards the gap in an arbitrary number of steps, e.g., two steps towards gap g_x. Each step is a constant distance which is already specified for the robot by its manufacturer, it puts a stepper motor on the robot that specifies its step size, for example 1mm, 2mm,..., also, the robot moves towards the target as it enters its visibility region.

While the robot moves, combinatorial changes occur in the visibility region of the robot called critical events. There are four types of critical events: appearances, disappearances, merges and splits of gaps. Appearance and disappearance events occur when the robot crosses inflection rays. Each gap that appears during the movement, corresponds to a portion of the environment that was already visible, but now is not visible. such gaps are called primitive gaps and all the others are non-primitive gaps. Merge and split events occur when the robot crosses bitangent, as illustrated in Figure 1.

2.2 Known Properties

At each point of the search path, if the target is not visible, the robot reports a set of gaps with the labels of L or R (l-gap and r-gap for abbreviation) cyclically. Let g_l be a non-primitive l-gap that is on the right side of the other left gaps, and g_r be a non-primitive r-gap that is in the left side of the other right gaps, see Figure 1(a). Each of the two gaps is called the most advanced gap. The two gaps have a fundamental role in path planning for the simple robot.

Theorem 1 *[4, 10] While the target is not visible, it is hidden behind one of the two gaps, g_l or g_r.*

From Theorem 1, if there exist only one of the two gaps (g_r and g_l) then the goal is hidden behind the gap. Thus, there is no ambiguity and the robot moves towards the gap, see Figure 2(a). When both of g_r and g_l exist, a funnel case arises, see Figure 2(b). At each funnel case, the robot does not know that the shortest path is along which ofg_r and g_l. So, usually, a detour from the shortest path is unavoidable.

2.3 Essential Information

All we maintain during the search strategy is the location of g_l and g_r. As the robot moves in the street, the critical events that change the structure of the robot's visibility region may dynamically change g_l and g_r. Also, by the robot movement, a funnel case may end or a new funnel may start. We refer to the point, in which a funnel ends a *critical point* of the funnel.

The following events update the location of g_l and g_r as well as a funnel situation when the robot moves towards g_l or g_r.

1. When g_r/g_l splits into g_r/g_l and another r-gap/l-gap, then g_r/g_l will be replaced by the r-gap/l-gap, (point 1 in Figure 2(b)).

2. When g_r/g_l splits into g_r/g_l and another l-gap/r-gap, then l-gap/r-gap will be set as g_l/g_r. This point is a critical point in which a funnel situation ends, (point 2 in Figure 2(b)).

3. When g_l or g_r disappears, the robot may achieve a critical point in which a funnel situation ends, (point 3 in Figure 2(a)).

Note that the split and disappearance events may occur concurrently, (point 3 in Figure 2(b)). Furthermore, by moving towards g_r and g_l, these gaps never merge with other gaps.

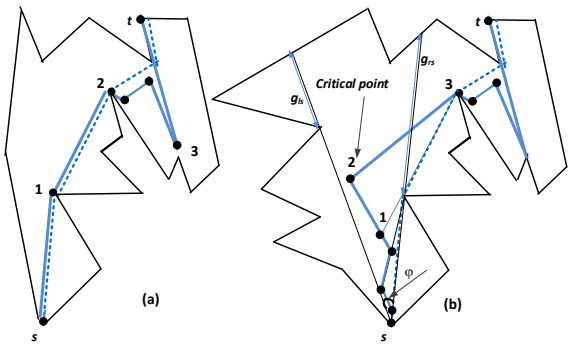


Figure 2: The bold path is the robot search path, the dotted path is shortest path, and v_l and v_r are the corresponding reflex of g_l and g_r respectively. (a) There is only g_r. (b) g_r and g_l are the two most advanced gaps at the start point s, in which a funnel case arises. The angle between the gaps, φ, is the opening angle at the start point.

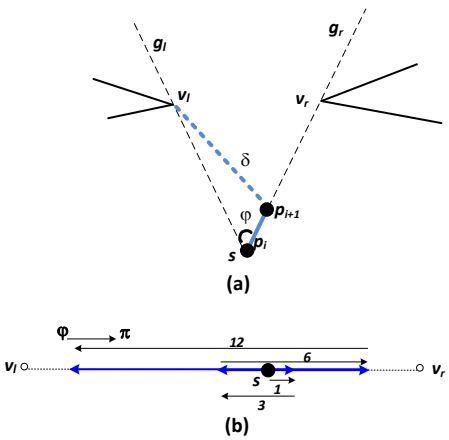


Figure 3: (a) p_ip_{i+1} is a detour from the shortest path. (b) The worst case.

3 Algorithm

Now, we present our strategy for searching the street, from s to t. Since the target is constantly behind one of g_r and g_l, during the search, the location of the two gaps is maintained and dynamically updated as explained in the previous section.

3.1 A Deterministic Strategy

At each point of the search path, especially at the start point s, there are two cases:

- If only one of the two gaps (g_r and g_l) exists, or they are collinear then the goal is hidden behind the gap. The robot moves towards the gap until the target is achieved or a funnel situation arises, see Figure 2(a).
- If there is a funnel case, to bound the detour, the robot moves towards g_r and g_l alternatively, as follows:

  Move towards g_r up to one step;
  $d \leftarrow 3$;
  repeat
  Move towards g_l up to d steps;
  if Critical point not achieved **then**
  $d \leftarrow 2.d$;
  Move towards g_r up to d steps;
  end if
  $d \leftarrow 2.d$;
  until Critical point of the funnel achieved;

  At the critical point, one of g_r or g_l disappears, or g_r and g_l are collinear. So, the robot moves along the existing gap direction until the target is achieved or a new funnel situation arises, as illustrated in Figure 2(b).

3.2 The Randomized Strategy

Now, we present a randomized search strategy based on the above deterministic strategy. The difference between them is using a random variable at the beginning of the above algorithm (in the funnel case). We choose random variable X from $\{0, 1\}$ u.a.r to lead the robot towards g_r or g_l at the first movement while in the deterministic strategy, the robot moves towards g_r.

3.3 Correctness and Analysis

Throughout the search, the robot path coincides with the shortest path unless a funnel case arises. Then, to prove the competitive ratio of our strategy, we compare the length of the path and the shortest path in a funnel case. In the case, the angle between g_r and g_l that is always smaller than π is called the opening angle [4], see Figure 2(b). In lemma 3, we show that our robot

detour from the shortest path depends on the size of the angle.

Also, we inspire from the doubling strategy by Baeza-Yates, et al. [1] to compute the competitive ratio of our strategy. In the strategy, a robot moves back and forth on a line such that the distance to the start point doubles at each movement until the target is reached.

Theorem 2 *[1] The doubling strategy for searching a point on a line has a competitive factor of 9, and this is optimal.*

Lemma 3 *By our strategy, the detour from the shortest path for a small opening angle, in the funnel case, is shorter than detour for a large opening angle.*

Theorem 4 *Our deterministic strategy guarantees a path at most 9 times longer than the shortest path. Also, the strategy is optimal.*

The proof of Theorem 4 shows that our deterministic strategy to reach the goal in street is a planar generalization of the doubling strategy for search a point on a line.

Theorem 5 *The randomized strategy generates a search path to achieve target t in the street, starting from s, with an expected competitive ratio of 7.*

3.4 Randomized Lower Bound

To achieve a randomized lower bound of the competitive ratio we consider a special funnel case which it's opening angle is very closed to π. so we can consider it as a problem of searching on the line, see Figure 3(b). Kao, Reif, and Tate [5] proved that the randomized lower bound of the competitive ratio for searching on the line is $1 + (1 + r)/\ln r$ where r is the multiplication factor of the randomized SmartCow algorithm and it is optimal. If we let r=3.59112 we can achieve the expected competitive factor of 4.59112 which is optimal and no other strategy can achieve this bound.

Theorem 6 *There is no on-line randomized strategy for walking in the streets for a simple robot that achieves an expected competitive ratio of less than 4.59112.*

4 Conclusions

In this paper, we have improved the previously known strategy for walking in streets for a simple robot. The point robot can only detect the gaps and the target in the environment. The robot using local information about the location of some gaps, along a 9-competitive optimal path achieves the target t starting from s in a street. Also, based on the improved strategy, a randomized strategy that has better performance is proposed. The expected length of the generated path by the random strategy is 7 times longer than the shortest path. Moreover, a randomized lower bound of 4.59112 is proved. It would be absorbing if there are competitive search strategies for more general classes of polygons.

References

[1] Baeza-yates, R. A., Culberson, J. C., Rawlins, G. J. Searching in the plane. *Information and Computation*, 106(2):234–252, 1993.

[2] Disser, Y., Ghosh, S. K., Mihalk, M., Widmayer, P. Mapping a polygon with holes using a compass. *Theoretical Computer Science*,553, 106-113. 2013.

[3] Ghosh, S., Klein, R. Online algorithms for searching and exploration in the plane. *Computer Science Review*, 4(4):189–201, 2010.

[4] Icking, C., Klein, R., Langetepe, E. An optimal competitive strategy for walking in streets. *In STACS 99, Springer Berlin Heidelberg*, 110–120, 1999.

[5] Kao, Mi., Reif, J., Tate, S. Searching in an unknown environment: An optimal randomized algorithm for the cow-path problem. *Information and Computation*, 131(1):63–79, 1996.

[6] Klein, R. Walking an unknown street with bounded detour. *Computational Geometry*, 1(6):325–351, 1992.

[7] Lpez-Ortiz, A., Schuierer, S. Lower bounds for streets and generalized streets. *International Journal of Computational Geometry and Applications*, 11(04):401–421, 2001.

[8] Lopez-Padilla, R., Murrieta-Cid, R., LaValle, S. M. Optimal Gap Navigation for a Disc Robot. *In Algorithmic Foundations of Robotics, Springer Berlin Heidelberg*, 123–138, 2012.

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

[10] Tabatabaei, A., Ghodsi, M. Walking in Streets with Minimal Sensing. *Journal of Combinatorial Optimization*, 30(2):387–401, 2015.

[11] Tabatabaei, A., Ghodsi, M., Shapouri, F. A Competitive Strategy for Walking in Generalized Streets for a Simple Robot. *CCCG*, (pp. 75-79), 2016.

[12] Tabatabaei, A., Ghodsi, M. Randomized Strategy for Walking in Streets for a Simple Robot. *arXiv:1512.01784v2*, 2015

[13] Tabatabaei, A., Ghodsi, M. and Shapouri, F. Competitive Strategy for Walking in Streets for an Empowered Simple Robot. *ICCG*, (pp. 59-62), 2019.

[14] Tovar, B., Murrieta-Cid, R., LaValle, S. M. Distance-optimal navigation in an unknown environment without sensing distances. *Robotics IEEE Transactions*, 23(3):506–518, 2007.

[15] Wei, Q., Ta, X. Walking an Unknown Street with Limited Sensing *International Journal of Pattern Recognition and Artificial Intelligence*, 1959042, 2019.

Appendix

Proof of lemma 3

In each funnel case, the robot moves some steps towards g_r or g_l, alternatively.

In the alternative movement, one of the directions is correct and the other is a deviation. Assume that at point p_i when a funnel case arises the robot moves toward g_r while the target is behind g_l. The robot achieves point p_{i+1}. In order to achieve the target, it should traverse at least distance $\delta = \sqrt{p_i p_{i+1}^2 + p_i v_l^2 - 2p_i p_{i+1} p_i v_l \cos\varphi}$, by the law of cosines, see Figure 3(a). It can be verified that δ is strictly increasing as a function of φ by taking the derivative with respect to φ where $0 \leq \varphi < \pi$.

Proof of Theorem 4

In a funnel case, when the opening angle φ is adequately near to π, the simple robot can only move towards left or right. Searching the target in the street in the limited case is similar to searching a line. So walking in street is at least as hard as searching a point on a line. Then, the competitive ratio of 9 is the lower bound for leading the robot in street, see Figure 3(b). From Lemma 3, there is a further deviation from the shortest path for large opening angles. The angle never exceeds π. Then, for computing a competitive factor, we consider it equals π. Starting from s, the robot moves one step towards g_r, then moves $1+2$ steps towards g_l, and again moves forth $2+2^2$ steps towards g_r, moves back 2^2+2^3 steps towards g_l, and so on. In other words, the robot moves back and forth on the line that contains g_l and g_r such that the distance to the start point s doubles until the critical point is reached. By Theorem 2, the competitive factor for the search strategy is 9. Then, the problem of walking in street polygons for a simple robot in the worst-case coincides with the searching a point on a line problem. So, the ratio of 9 is optimal.

Proof of Theorem 5

As shown in Theorem 4, in the worst case, when φ comes close to π, our problem is similar to the problem of searching on the line and our deterministic strategy coincides with the doubling strategy. In the first randomized strategy by choosing the direction of the first movement u.a.r, we have two cases depend on which direction is selected and each of which makes the robot traveling different distances. In the worst case, the critical point is on the $n = 2^k + \delta$ (where k is an integer and δ is a real value satisfying $0 < \delta < 1$) from the origin and the greatest distance for search is taken. Let m be the first stage where robot travels distance at least 2^k on the same path as the critical point exists. The value m satisfies $m \in \{k, k+1\}$. At the beginning of the search, the algorithm chooses a random direction, so $Prob(m = c) = \frac{1}{2}$ for $c = k, k+1$. If D is the random variable denoting the distance traveled by the randomized strategy, then it is easy to see that when $m = c$ we have

$$D = 2\sum_{i=0}^{c} 2^i + n$$

and the expected values calculated as

$$E[D|m = c] = D$$

$$E[D] = \sum_{c=k}^{k+1} Prob(m = c)E[D|m = c]$$

Thus, the resulting expected distance traveled is

$$E[D] = \frac{1}{2}[2\sum_{i=0}^{k+1} 2^i + n] + \frac{1}{2}[2\sum_{i=0}^{k} 2^i + n]$$

$$= \frac{1}{2}[2(2^{k+2} - 1) + 2^k + \delta] + \frac{1}{2}[2(2^{k+1} - 1) + 2^k + \delta]$$

$$= \frac{1}{2}[(9)2^k - 2 + \delta] + \frac{1}{2}[(5)2^k - 2 + \delta]$$

$$= \frac{1}{2}[(14)2^k + 2\delta - 4] = (7)2^k + \delta - 2$$

$$\leq 7(2^k + \delta) = 7n$$

On Connecting with Neighborhoods: Complexity and Algorithms

Arash Ahadi* Alireza Zarei†

Abstract

Uncertainty of the location of a point can be modeled by a region. In this case, a routing problem on imprecise points is equivalent to finding a tour over their corresponding regions. Finding shortest paths and minimum spanning trees are two well-known such problems. We consider the worst and best cases of some versions of these two problems and prove their NP-Hardness in sharp cases. Also, we discuss their accurate polynomial-time approximation algorithms.

1 Introduction

Erroneous measurement offered by real measurement sensors is always a challenge in designing optimal algorithms on such imprecise data. In the literature, the term imprecise data is also used instead of uncertainty by researchers. As a natural model of uncertainty in geometric inputs, an imprecise point is represented by a region which is assumed to contain the point with high probability.

We denote by $[N]$ the set of numbers $\{1, 2, ..., N\}$. A set of imprecise points $P = \{p_i, \ | i \in [k]\}$ is given by a set of regions $\mathcal{R} = \{R_i | i \in [k]\}$, where R_i contains p_i. In this paper, we study the minimum spanning tree and the shortest path problems on such imprecise data.

The *minimum spanning tree with neighborhoods* problem (MSTN), is to locate the position of each point $p_i \in P$ on a point of R_i, such that the minimum spanning tree on these points has the minimum possible cost. MSTN was introduced by Yang *et al.* in [11], who proved NP-hardness of the problem and gave several approximation algorithms and a PTAS when the regions $\mathcal{R}$ are disjoint unit disks. This NP-hardness result was also proved by Löffler and van Kreveld [8] in a different technique. Disser *et al.* proved that MSTN does not admit an FPTAS, even if the regions are horizontally or vertically aligned line segments [4], and gave some approximation results for the rectilinear MSTN. They also proposed considering MSTN in the worst case as an interesting open problem, which was then considered by Dorrigiv *et al.* [5]. The *maximizing minimum spanning tree with neighborhoods* (max-MSTN) problem is to compute the worst case of MSTN; that is to locate the positions of points P, such that the minimum spanning tree has the maximum possible cost. In view of uncertainty, the answers of MSTN and max-MSTN are respectively lower and upper bounds on the weight of the minimum spanning tree of the graph regardless of the actual locations of the imprecise points. Dorrigiv *et al.* proved the non-existence of any FPTAS for max-MSTN, when the underlying regions are disjoint disks [5] and appreciated studying the problem for other shapes like line segments, as future research problems. See [12] as a recent research on this problem in operation research field.

In the above discussion, the underlying graph of the MSTN problem was implicitly considered as a complete graph, in which there is an edge between any pair of points (regions). MSTN and max-MSTN problems have been considered in general cases when the underlying graph is not complete [2, 3, 4]. In such cases, a graph $G = (\mathcal{R}, E)$ is given, where its vertex set is a set of imprecise points $P = \{p_i | i \in [k]\}$, represented by the regions $\mathcal{R} = \{R_i | i \in [k]\}$, and each edge $(R_i, R_j) \in E$ means that traversing from a point in R_i to a point in R_j, or vice versa, is allowed. However, the preceding definition of the MSTN or max-MSTN problems are special cases of this one in which the graph is complete.

In this paper, we prove that max-MSTN is NP-hard, when the regions are horizontally or vertically aligned unit line segments and the underlying graph G is 3-regular. Note that if the maximum degree of G is 2, the problem has a simple polynomial time algorithm for line segments even for arbitrary polygonal uncertainty regions (subsection 5.3 in [9]). Thus, 3-regularity in this problem is a sharp bound.

The shortest path is another problem which we consider on imprecise data. The *minimum shortest path with neighborhoods* problem between two imprecise points p_i and p_j on a graph $G = (\mathcal{R}, E)$, is to locate the position of any point $p_l \in P$ on a point in R_l for every l, such that the shortest path from p_i to p_j (supporting the edges of E) has the minimum possible length.

A special case of this problem was considered more than one decade earlier as *touring polygons problem* (TPP); that was introduced by Dror *et al.* in [6]. In this case, the graph G is a path from R_1 to R_k; the first and the last regions are respectively single points s and t (have no uncertainty). Then, the minimum shortest path problem is to find a shortest tour from s to t vis-

*Department of Mathematical Science, Sharif University of Technology, arash.ahadi1@student.sharif.ir

†Department of Mathematical Science, Sharif University of Technology, zarei@sharif.ir

iting R_i's in order. They proved that TPP is NP-hard for general polygons (non-convex and non-disjoint) and can be solved in polynomial time when polygons of $\mathcal{R}$ are convex and disjoint [6]. Pan *et al.* [10] in 2010 proposed a linear time approximation algorithm for disjoint non-convex TPP; while the complexity of TPP for disjoint non-convex polygons was still open. Finally, in [1] it has been proved that TPP is NP-hard for disjoint (non-convex) polygons in any L_p norm, and it is asked whether disjoint TPP remains NP-hard, when the sizes of all polygons differ polynomially with respect to each other.

Here and because of the limited space, we only give the NP-hardness of the max-MSTN on line segments, but we have already proved that disjoint TPP remains NP-hard, even if each polygon is made by one or two unit segments, each of them make a multiple of $\frac{\pi}{4}$ angle with the line $x = 0$, and our proof is shorter and simpler than both proofs in [6] and [1].

2 Maximizing MSTN on Line Segments

We reduce *min-SAT* problem to max-MSTN to show its complexity class. In an input $\Psi = (\mathcal{X}, \mathcal{C})$ of *min-SAT* problem, we have a set of boolean variables $\mathcal{X} = \{x_i | i \in [m]\}$ and a set of clauses $\mathcal{C} = \{c_i | i \in [n]\}$ on these variables in disjunctive normal form. The goal is to obtain a *true/false* assignment to the variables of $\mathcal{X}$, such that the minimum possible number of clauses in $\mathcal{C}$ are satisfied. *min-SAT* is NP-hard, even if each clause contains at most two literals [7]. By changing every clause α to $\alpha \vee \alpha \vee \alpha$ and changing every clause $\alpha \vee \beta$ to $\alpha \vee \alpha \vee \beta$, the answer of *min-SAT* does not change (α and β may be negative literals). So, *min-SAT* is NP-hard, when each clause contains exactly three literals.

For a given $\Psi = (\mathcal{X}, \mathcal{C})$, we construct an input $G_\Psi = (\mathcal{R}, E)$ of max-MSTN as follows. For each $c_i \in \mathcal{C}$, we add a horizontal unit segment R_{c_i}, whose left endpoint lies sufficiently close to the point $(0, -\frac{1}{4})$. Therefore, these segments are parallel with sufficiently small vertical distance between any pair of consecutive ones. This distance is only to make them disjoint, and, for simplicity one can consider them all on a unique segment from point $(0, -\frac{1}{4})$ to $(1, -\frac{1}{4})$.

For every $x_j \in \mathcal{X}$, we put $2n$ vertical unit segments

$$R_{x_1^j}, R_{\bar{x}_1^j}, R_{x_2^j}, R_{\bar{x}_2^j}, \ldots, R_{x_n^j}, R_{\bar{x}_n^j}$$

from left to right, such that their upper endpoints lie sufficiently close to the point $(L, \frac{1}{2})$. The value of L will be determined later. Similar to R_{c_i}'s, the horizontal distance between each pair of consecutive segments in $R_{x_1^j}, \ldots, R_{\bar{x}_n^j}$ is sufficiently small; and is considered only to make them disjoint. Again, for the sake of simplicity, assume that they all have the same locations.

Finally, for every $j \in [m-1]$, we add a unit horizontal segment R_j on the line $y = 0$ and in the right side of the segments $R_{x_i^j}$'s and $R_{\bar{x}_n^j}$'s(See Figure 1).

These unit segments are the vertices of G_Ψ. For every $i \in [n]$ and $j \in [m]$, if $x_j \in c_i$, there is an edge in G_Ψ between R_{c_i} and $R_{x_i^j}$, and if $(\bar{x}_j) \in c_i$, then there is an edge in G_Ψ between R_{c_i} and $R_{\bar{x}_i^j}$. Moreover, the induced subgraph of G_Ψ on $\{R_j | j \in [m-1]\} \cup \{R_{x_i^j} | j \in [m], i \in [n]\} \cup \{R_{\bar{x}_i^j} | j \in [m], i \in [n]\}$ is the path

$$H := R_{x_1^1} R_{\bar{x}_1^1} R_{x_2^1} \ldots R_{\bar{x}_n^1} R_1, R_{x_1^2} \ldots R_{\bar{x}_n^2} R_2, \ldots R_{\bar{x}_n^m}.$$

Therefore, in the graph $G_\Psi = (\mathcal{R}, E)$, $\mathcal{R}$ is the set of these unit segments and E is the above edges connecting these vertices. The maximum degree in G_Ψ is 3.

Assume that OPT is an optimal solution for G_Ψ and $\mathcal{T}$ is a minimum spanning tree on this graph for vertex set OPT.

Lemma 1 *Degree of R_{c_i} in $\mathcal{T}$ is 1, and all edges of H exist in $\mathcal{T}$.*

Proof. At least one of the three joined edges to R_{c_i} exists in $\mathcal{T}$. We show that exactly one of these three edges exists in $\mathcal{T}$. Let L^* be an upper bound for total length of all edges in chain H for all possible positions of points in regions of H. Recall that L is the horizontal distance between the left endpoint of R_{c_i}'s and the vertical segments. If the value of L is sufficiently large (in fact if $L > L^*$), the length of all three edges of R_{c_i} are too long; such that regardless of the actual positions of points in OPT, only one edge of each vertex R_{c_i} is selected in $\mathcal{T}$. This along with connectivity requirement of $\mathcal{T}$ imply that all edges of H exist in $\mathcal{T}$. □

The following lemma shows how any point $p \in OPT$ is selected from its corresponding region $R' \in \mathcal{R}$ to maximize the total cost of $\mathcal{T}$.

Lemma 2 *In OPT,*

(i) if $R' \in \{R_{c_i} | i \in [n]\}$, p is the left end point of R', and, if $R' \in \{R_j | j \in [m-1]\}$, p is the right end point of R'.

(ii) each point p is an endpoint of its corresponding region.

(iii) opposite endpoints of consecutive vertical segments of H are selected, i.e., if the upper (resp. lower) endpoint of a vertical segment is selected the lower (resp. upper) endpoint of its next adjacent vertical segment must be selected in OPT. Therefore, for every $j \leq m$, either "lower endpoint of $R_{x_i^j}$ and upper endpoint of $R_{\bar{x}_i^j}$" or "upper endpoint of $R_{x_i^j}$ and lower endpoint of $R_{\bar{x}_i^j}$" are selected, for all $i \in [n]$.

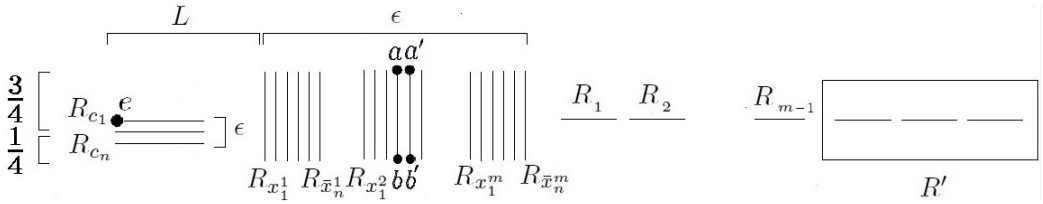


Figure 1: Reducing *min-SAT* to max-MSTN. ϵ is a sufficiently small number. In fact, if segment intersections are permitted in the problem, one can consider $\epsilon = 0$. However, and for the sake of simplicity, inside proofs, we assume that $\epsilon = 0$.

Proof. (i) In G_Ψ, each region of $R' \in \{R_{c_i} | i \in [n]\} \cup \{R_i | i \in [m-1]\}$ is only connected to some regions in $\{R_{x_1^1}, ..., R_{\bar{x}_n^m}\}$. Based on the locations of these regions, the cost of $\mathcal{T}$ is maximized when the selected point of each region in $\{R_{c_i} | i \in [n]\}$ lies on its left endpoint and the selected point of each region in $\{R_i | i \in [m-1]\}$ lies on its right endpoint.

(ii) Having proved (i), we need to prove this for vertical segments only. For the sake of a contradiction, assume that the selected point p of the region R' is not an endpoint of R'. There may be an edge between R' and a region $R'' \in \{R_{c_i} | i \in [n]\}$. Let q be the selected point on R'' in OPT. Moreover, based on the position of R' in H, either R' in H is adjacent to two vertical segments or it is adjacent to a vertical segment and a segment in $\{R_i | i \in [m-1]\}$. The only exceptions are $R_{x_1^1}$ and $R_{\bar{x}_n^m}$ which have only one vertical adjacent segment in H. Let p_- and p_+ be respectively the selected points in OPT for the preceding and the succeeding regions of R' in H. The structure of G_Ψ forces that all edge p_-p and pp_+ must exist in $\mathcal{T}$. Assuming that the positions of p_- and p_+ are fixed, and, b is the intersection point of p_-p_+ and R'. By moving from b in both sides along R' the lengths of both p_-b and bp_+ are increased. Therefore, $|p_-p|+|pp_+|$ is maximized when p is an endpoint of R'. Moreover, when pq also exists in $\mathcal{T}$, it is geometrically simple to verify that the maximum value of $|p_-p|+|pp_+|+|pq|$ occurs at an endpoint of R'.

The cases where R' is either $R_{x_1^1}$ or $R_{\bar{x}_n^m}$ we only need to maximize $|pp_+| + |pq|$ or $|p_-p| + |pq|$, respectively which trivially happens at an endpoint as well.

(iii): To the contrary, assume that for some $j \leq m$ there are two adjacent vertical segments ab and $a'b'$ in H such that either both their upper, a and a', or lower b and b' endpoints exist in OPT. Also, assume that these segments are the first pair on H with this property. If these segments correspond to regions $R_{x_i^j}$ and $R_{\bar{x}_i^j}$, then at most one of them is connected to a region R_{c_i} in G_Ψ. Assume that $R_{x_i^j}$ is this segment. Then we can select the other endpoint if $R_{\bar{x}_i^j}$ in OPT. This selection increases the total cost of $\mathcal{T}$ by

- 1 if the next adjacent region of $R_{\bar{x}_i^j}$ is R_j, or $i = n$ and $j = m$,
- 2 if the next adjacent region of $R_{\bar{x}_i^j}$ is $R_{x_{i+1}^j}$ and the same endpoints of $R_{\bar{x}_i^j}$ and $R_{x_{i+1}^j}$ have been selected in OPT,
- 0 if the next adjacent region of $R_{\bar{x}_i^j}$ is $R_{x_{i+1}^j}$ and the opposite endpoints of $R_{\bar{x}_i^j}$ and $R_{x_{i+1}^j}$ exist in OPT.

However, this may decrease the length of the edge that connects R_{c_j} to the vertical segment $R_{\bar{x}_i^j}$ (if such an edge exists in $\mathcal{T}$) from $\sqrt{L^2 + (\frac{3}{4})^2}$ to $\sqrt{L^2 + (\frac{1}{4})^2}$. But, this decrement ($d = \sqrt{L^2 + (\frac{3}{4})^2} - \sqrt{L^2 + (\frac{1}{4})^2}$) of cost is smaller than 1 and 2 and the set of new selected points results a MST with greater cost than OPT in the first two cases which is a contradiction.

For the last case, if at most one of the upper endpoints of the segments from $R_{\bar{x}_i^j}$ to $R_{\bar{x}_n^j}$ is connected to R_{c_j} we reverse all selected endpoints in OPT for segments from $R_{x_{i+1}^j}$ to $R_{\bar{x}_n^j}$ as done for $R_{\bar{x}_i^j}$. Then, the total cost of H which completely exists in $\mathcal{T}$ is increased by 1 and as the first two cases, the probable cost decrement (d) is smaller than 1. Therefore, in this case the set of new selected points results a MST with greater cost than OPT which is again a contradiction.

Finally, for the last case, if more than one of the upper endpoints of the segments from $R_{\bar{x}_i^j}$ to $R_{\bar{x}_n^j}$ is connected to R_{c_j} we do not change the position of the selected point on $R_{\bar{x}_i^j}$, but reverse all selected endpoints in OPT for segments from $R_{x_1^j}$ to $R_{x_i^j}$. Then the total cost of H is increased by 1. Because we have three literals in c_j, at most one upper endpoint of these segments is connected to R_{c_j} which means that as previous cases, the probable cost decrement (d) is smaller than 1. Therefore, again in this case the set of new selected points results a MST with greater cost than OPT which is a contradiction.

Therefore, it is impossible to have two points on the same side of two consecutive vertical segments in OPT. □

The third part of Lemma 2 implies that for every $j \in$

$[m]$, either lower endpoint of $R_{x_i^j}$'s and upper endpoints of $R_{\bar{x}_i^j}$'s, or upper endpoint of $R_{x_i^j}$'s and lower endpoints of $R_{\bar{x}_i^j}$'s are selected in OPT, for all i.

Now, we describe how to obtain an optimal solution for Ψ in min-SAT problem from an OPT solution of max-MSTN on G_Ψ.

For every $i \in [m]$, the value of x_i is set to *true* if and only if the lower endpoint of $R_{x_j^i}$ is selected in OPT. The third part of Lemma 2 shows that the assignment to x_i is well-defined. We prove that this assignment is an optimal assignment for Ψ in *min-SAT* if and only if OPT is an optimal solution for G_Ψ of max-MSTN.

Let e_i for every $i \in [n]$ be the selected edge adjacent to its corresponding region R_{c_i}. The weight of $\mathcal{T}$ is equal to the sum of the lengths of the edges of H, denoted by $length(H)$, plus the lengths of $e_1, ..., e_n$. Trivially, $length(H)$ is a fixed value and independent of the lengths of e_i's. Let e be the left endpoint of R_{c_i} and ab is the other side region of e_i(a vertical region with a as its upper endpoint). The length of e_i is either $|ea|$ or $|eb|$ which depends on the selected endpoint of the corresponding vertical segment. The key point is that for each $i \in [n]$, if the assignment satisfies clause c_i, then the length of e_i is $|ea|$. Otherwise, this length is $|eb|$.

Therefore, the weight of $\mathcal{T}$ is $length(H)+(n-k)|ea|+k|eb|$, where k is the number of satisfied clauses. Since $|ea| > |eb|$, the maximum weight of $\mathcal{T}$ is determined by the minimum number of clauses that can be satisfied. Since *min-SAT* is NP-hard and our reduction is polynomial, max-MSTN problem is also NP-hard.

Theorem 3 *max-MSTN is NP-hard, even if each region is a horizontal or a vertical unit line segment, and the underlying graph is 3-regular.*

Proof. The above discussion proves the theorem for graphs with the maximum degree $\Delta = 3$. In order to make a cubic (3-regular) graph, one can add some new horizontal unit segments on the line $y = 0$ in region R' as shown in Figure 1. Here, R' is sufficiently far from the main structure. Any one of these segments is connected to exactly 3 segments of the main structure whose degrees are 2. Note that the degree of each vertex R_{c_i} is 3. It is 2 for each vertex R_i. Also, the degree of each vertex $R_{x_i^j}$ and each vertex $R_{\bar{x}_i^j}$ is 2 or 3.

Finally, if the number of necessary edges is not a multiple of 3, we add several new segments in R' with sufficiently large distance from other segments, such that the resulting graph is 3-regular.

In every optimal answer of this graph, since R' is far from the other segments, the degree of each segment of R' is 1; and the right endpoint of each of them should be selected. Any one of these points have equal distances from both endpoints of its corresponding region. Therefore, these edges add a constant value to the optimal solution $\mathcal{T}$. This means that the optimal answer of this instance of max-MSTN is equivalent to the optimal answer of the corresponding min-SAT problem. □

References

[1] A. Ahadi, A. Mozafari and A. Zarei, Touring a sequence of disjoint polygons: Complexity and extension, Theoretical Computer Science, 556: pp. 45-54, 2014.

[2] V. Blanco, E. Fernandez and J. Puerto, Minimum spanning trees with neighborhoods, Arxiv, 1611.02918, 2016.

[3] E. Chambers, A. Erickson, S. Fekete, J. Lenchner, J. Sember, S. Venkatesh, U. Stege, S. Stolpner, C. Weibel, and S. Whitesides, Connectivity graphs of uncertainty regions, in Proceedings of the International Symposium on Algorithms and Computation (ISAAC), LNCS 6507, pp. 434-445, 2010.

[4] Y Disser, M Mihalk, S Montanari and P. Widmayer, Rectilinear Shortest Path and Rectilinear Minimum Spanning Tree with Neighborhoods, Combinatorial Optimization. ISCO 2014, p. 208-220, 2014.

[5] R. Dorrigiv, R. Fraser, M. He, S. Kamali, A. Kawamura, A. Lopez-Ortiz and D. Seco, On Minimum- and Maximum-Weight Minimum Spanning Trees with Neighborhoods, Theory of Computing Systems volume 56, p. 220-250, 2015.

[6] M. Dror, A. Efrat, A. Lubiw and J.S.B. Mitchell, Touring a sequence of polygons, in Proc. 35th Annu. ACM Sympos. Theory Comput., California, USA, pp. 473-482, 2003.

[7] R. Kohli, R. Krishnamurti and P. Mirchandani, The Minimum Satisfiability Problem, SIAM J. Discrete Mathematics, 7, pp. 275-283, 1994.

[8] M. Löffler and M. van Kreveld, Largest and smallest convex hulls for imprecise points, Algorithmica 56, pp. 235-269, 2010.

[9] S. Montanari, Computing routes and trees under uncertainty, PhD Dissertation, ETH Zurikh, No. 23042, 2015.

[10] X. Pan, F. Li and R. Klette, Approximate shortest path algorithms for sequences of pairwise disjoint simple polygons, CCCG'10, pp. 175-178, 2010.

[11] Y. Yang, M. Lin, J. Xu, and Y. Xie, Minimum Spanning Tree with Neighborhoods, Proceedings of Algorithmic Aspects in Information and Management: Third International Conference, AAIM, pp. 306-316, 2007.

[12] V. Blancoa E. Fernandezb and J. Puertoc, Minimum Spanning Trees with neighborhoods: Mathematical programming formulations and solution methods, European Journal of Operational Research 262 (3), p. 863-878, 2017.

Planar Euclidean TSP via Snowflake Tree

Sepideh Aghamolaei* Mohammad Ghodsi†

Abstract

Given a set of points P in the Euclidean plane, the Euclidean Steiner tree is the embedded tree of minimum weight that connects the points of P, but is allowed to have vertices other than the points of P. The Euclidean traveling salesman problem (ETSP) asks for the tour of minimum length that visits every vertex once.

We define the snowflake tree as the Euclidean Steiner tree with orthogonal edges. Using this tree in the TSP construction algorithm based on doubling the edges of MST, we compute a constant factor approximation ETSP tour with a planar embedding, which the original doubling algorithm does not guarantee.

1 Introduction

The traveling salesman problem (TSP) takes as input a set of points P and computes a cycle on P that visits all the vertices and the sum of all the edge weights is minimized [11]. TSP is NP-hard in general case, in the metric case, and in the Euclidean case [11]. Metric TSP has a 2-approximation based on doubling the edges of a MST and shortcutting them, a 3/2-approximation based on doubling and shortcutting the union of the edges of a MST and a matching.

Euclidean TSP (ETSP) has a PTAS ($(1+\epsilon)$-approximation) [1]. The nonself-intersecting tours in the algorithm of [1] guarantees the resulting tour does not have any intersections between its edges. This result was later improved to have a polynomial time in ϵ and n [2, 9].

The Steiner tree problem takes as input a point set P and a set of Steiner points S, and computes a tree with the minimum cost that contains all the vertices of P and any subset of S [11]. A *geometric Steiner tree* or *Euclidean Steiner tree* is an embedded tree that connects a set of points P with the minimum cost, where it is allowed to add arbitrary points of the Euclidean plane as Steiner points. There are PTAS algorithms for Euclidean Steiner tree [1, 2, 9].

In Figure 1, a snowflake tree is shown, and the TSP built from that tree is shown in Figure 4. An MST of the same point set is drawn in Figure 2, with the TSP constructed from it in Figure 3.

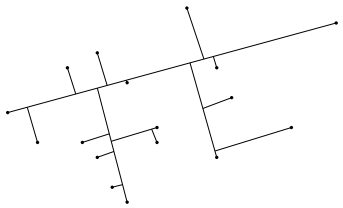


Figure 1: A snowflake tree.

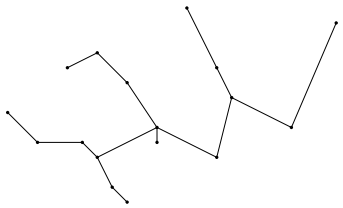


Figure 2: A minimum spanning tree.

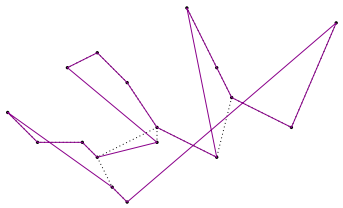


Figure 3: A TSP built from MST.

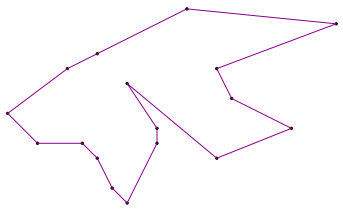


Figure 4: A TSP built using a snowflake tree.

We design a deterministic approximation algorithm for ETSP such that the edges of the TSP tour do not intersect in their original embedding in the Euclidean plane. Our algorithm takes $O(n^2 \log n)$ time, and has approximation factor 52.1. The time complexities of the

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

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

previous algorithms (PTASs) were $n(\log n)^{O(1/\epsilon)}$ (randomized), $n(\log n)^{O(1/\epsilon)}+O(n^2)$ (deterministic) [2], and $O(n\log n)+n(\sqrt{2}/\epsilon)^{O(\sqrt{2}/\epsilon)}$ (randomized) [9].

2 Preliminaries

Segments Voronoi diagram (SVD). Given a set of segments, their Voronoi diagram is a diagram where each cell represents the points with the same nearest segment. The segments Voronoi diagram can be constructed in $O(n\log n)$ time and $O(n)$ space, and a nearest neighbor query takes $O(\log n)$ time [4]. The incremental construction of a SVD takes $O(n\log n)$ time [3].

Diameter of a point-set. The diameter of a point-set is the length of the segment connecting two points from this set with maximum distance. Computing the diameter takes $O(n\log n)$ time, by computing the convex hull of points [4] and then using the algorithm of [10].

Steiner tree (ST). Given two sets of vertices R and S, the goal is to connect the vertices of R with the smallest total length, using any number of vertices from S. This problem is NP-hard and has a 2-approximation algorithm [11].

Minimum rectilinear Steiner tree problem (MRST). In the MRST problem, we are given a set of points and the goal is to connect these points using axis-aligned (vertical or horizontal) line segments such that the overall length of the line segments is minimized. The Steiner points are the points of the Hanan grid, which is the irregular grid constructed by drawing lines parallel to the axes [6]. The decision version of this problem is NP-complete [7]. Other than its PTASs [1, 2, 9], it has a $(1.267+\epsilon)$-approximation algorithm with $O(n\log^2 n)$ time [8].

Rectilinear minimum spanning tree (RMST). The minimum spanning tree using ℓ_1 distance is called the rectilinear minimum spanning tree problem, and it can be constructed in $O(n\log n)$ time [5].

The difference between MRST and RMST is that in RMST a common segment between two rectilinear ℓ_1 distances is summed twice, while in RMST it is summed once. The Steiner ratio is the maximum ratio of the weight of MST to the weight of the Steiner tree. The Steiner ratio of MRST and RMST is at most 3/2 [7].

Doubling and Shortcutting for TSP and ST. Replacing each edge of a tree with two edges is called doubling, which converts the tree into an Eulerian graph [11]. This Eulerian tour is then used as a 2-approximation for TSP. Shortcutting is removing the Steiner or repeated vertices of this tour [11].

3 Snowflake Tree

The difference between the snowflake tree and the minimum rectilinear Steiner tree is that the edges of the snowflake tree are not required to be parallel to one of the axes of the coordinate system.

3.1 The Snowflake Tree of 3 Points

The snowflake tree of 3 points can add at most 3 Steiner vertices (Figure 5).

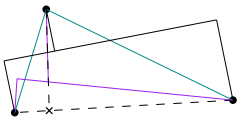


Figure 5: The 4 cases of the snowflake trees of 3 points.

Theorem 1 *The snowflake tree of the vertices of a right triangle is the tree consisting of the sides of the right angle.*

Proof. Assume the catheti have lengths a and b.

When no Steiner vertices are used, the two shorter edges of the triangle are the solution, so the cost is $a+b$. Using one Steiner point, because one edge of the tree has to be an edge of the triangle, there is only one new case: when we draw the altitude opposite to the hypotenuse. Let h be the length of this perpendicular. The cost of this case is: $\sqrt{a^2+b^2}+\frac{ab}{\sqrt{a^2+b^2}}$, since the length of the altitude is $\frac{ab}{\sqrt{a^2+b^2}}$, using the formulas for the area of a triangle: $\frac{ab}{2}$ and $\frac{h\sqrt{a^2+b^2}}{2}$. Computing the square of this length gives the following value: $(\sqrt{a^2+b^2}+\frac{ab}{\sqrt{a^2+b^2}})^2 = a^2+b^2+2ab+(\frac{ab}{\sqrt{a^2+b^2}})^2$, which is more than the square of the cost of the solution with no Steiner points, which is $(a+b)^2 = a^2+b^2+2ab$. We discuss the case with 2 Steiner points as a special case of the case with 3 Steiner points, where one of the Steiner points lies on a vertex of the triangle. The case with 3 Steiner points, we name the altitudes as shown in Figure 6.

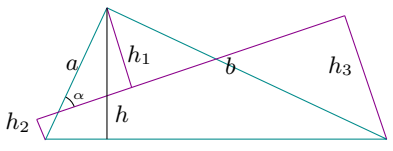


Figure 6: The case with 3 Steiner points (Theorem 1).

Using triangle similarity and trigonometry, the sum of the lengths of the edges in this case is:
$h_1+h_2+h_3+h_2|\cot(\alpha)|+h_1|\cot(\alpha)|+h_1|\tan(\alpha)|+h_3|\tan(\alpha)|$
$= h_1(1+|\cot(\alpha)|+|\tan(\alpha)|)+h_2(1+|\cot(\alpha)|)+h_3(1+|\tan(\alpha)|)$.

The length of the catheti can be written in terms of h_1, h_2, h_3 and α as follows: $a = \frac{h_1}{|\sin(\alpha)|} + \frac{h_2}{|\sin(\alpha)|}$, $b = \frac{h_1}{|\cos(\alpha)|} + \frac{h_3}{|\cos(\alpha)|}$. We use these equalities to remove h_2 and h_3 from the formula: $h_2 = a|\sin(\alpha)| - h_1$, $h_3 = b|\cos(\alpha)| - h_1$. Substituting these values in the cost of this case gives:

$$\begin{aligned}
&h_1(1 + |\cot(\alpha)| + |\tan(\alpha)|) \\
&+ (a|\sin(\alpha)| - h_1)(1 + |\cot(\alpha)|) \\
&+ (b|\cos(\alpha)| - h_1)(1 + |\tan(\alpha)|) \\
&= -h_1 + a(|\sin(\alpha)| + |\cos(\alpha)|) + b(|\cos(\alpha)| + |\sin(\alpha)|) \\
&= -h_1 + (a+b)(|\sin(\alpha)| + |\cos(\alpha)|).
\end{aligned}$$

Assume the notation $h_M = \min(a|\sin(\alpha)|, b|\cos(\alpha)|)$. Taking the minimum over parameters h_1 and α gives us the answer to this case:

$$\begin{aligned}
&\min_{\substack{\alpha\in[0,\pi],\\ h_1\le h_M}} -h_1 + (a+b)(|\sin(\alpha)| + |\cos(\alpha)|) \\
&= \min_{\alpha\in[0,\pi]} -h_M + (a+b)(|\sin(\alpha)| + |\cos(\alpha)|)
\end{aligned}$$

where the inequality uses the fact that h_1 is independent from α, other than its range. If $a|\sin(\alpha)| \le b|\cos(\alpha)|$:

$$\min_\alpha a|\cos(\alpha)| + b(|\sin(\alpha)| + |\cos(\alpha)|)$$

$$\min_\alpha |\cos(\alpha)|(a+b) + |\sin(\alpha)|b = a + b.$$

Since $a + b \ge b$, then $|\cos(\alpha)| = 1, |\sin(\alpha)| = 0$ is the optimal solution and $\alpha = 0$.

Otherwise ($a|\sin(\alpha)| \ge b|\cos(\alpha)|$):

$$\min_\alpha a(|\sin(\alpha)| + |\cos(\alpha)|) + b|\sin(\alpha)|$$

$$\min_\alpha (a+b)|\sin(\alpha)| + a|\cos(\alpha)| = a + b.$$

Since $a + b \ge a$, then $|\sin(\alpha)| = 1, |\cos(\alpha)| = 0$ is the optimal solution and $\alpha = \frac{\pi}{2}$. So, the optimal solution of this case is the same as the first case (with 0 Steiner points). □

3.2 Approximate Snowflake Tree

In Algorithm 1, *weight* denotes the weight of the tree, which is the sum of the lengths of its edges.

Theorem 2 *Algorithm 1 is a $(\beta+\epsilon)$-approximation for snowflake tree, using an β-approximation MRST.*

Proof. Since all the edges of a snowflake tree are perpendicular, if the direction of one edge is determined, the two possible directions of all the edges are also determined. Therefore, computing the MRST in the rotated coordinates finds the optimal solution, if all the angles are checked. However, we use an α-approximation of MRST. We only check k angles in the algorithm, so the length of each edge is at most $\frac{1}{\cos(\frac{2\pi}{k})}$ times its value in the direction of the optimal solution. Using the Taylor series expansion of the cosine function, $\cos(\theta) \ge 1 - \frac{\theta^2}{2}$, for small enough θ. To guarantee a $(1+\epsilon)$-approximation, the value of k must satisfy the following inequality: $\frac{1}{\cos(\frac{2\pi}{k})} \le 1 + \epsilon$. Relaxing the inequality by the bound on the cosine function and rearranging the terms gives: $\frac{1}{1+\epsilon} \ge \cos(\frac{2\pi}{k}) \ge 1 - \frac{1}{2}(\frac{2\pi}{k})^2$. So, $\frac{\sqrt{2}\pi\sqrt{1+\epsilon}}{\sqrt{\epsilon}} \le k$. Using $\sqrt{1+\epsilon} \le 1 + \sqrt{\epsilon}$, we have $\frac{\sqrt{2}\pi}{\sqrt{\epsilon}} + \sqrt{2}\pi \le k$. This approximation factor is multiplied by β, which gives $\beta + O(\epsilon)$. □

Algorithm 1 Approximate Snowflake Tree

Input: A point set P, a constant $\epsilon > 0$
Output: T: an approximate snowflake tree of P

```
cost = ∞, k = ⌈√2π/√ε + √2π⌉
for i = 1, ..., k do
    Rotate the coordinates by i·2π/k clockwise.
    Compute an approximate MRST.
    if cost > weight(MRST) then
        T ← MRST, cost ← weight(MRST)
return T
```

The time complexity of Algorithm 1 using RMST is $O(kn \log n)$ and its approximation factor is $\frac{3}{2} + \epsilon$, since the time complexity of computing a RMST is $O(n \log n)$ and the Steiner ratio of MRST is $\frac{3}{2}$ [7]. Using the MRST of [8], the approximation factor is $1.267 + \epsilon$ and the time complexity is $O(kn^2 \log n)$.

3.3 Greedy Approximation Algorithm

Algorithm 2 builds an approximate snowflake tree, and we prove its approximation factor in Theorem 3.

Algorithm 2 Snowflake Tree

Input: A point set P
Output: T, the snowflake tree of P

```
Add the diameter of P to T
while graph T is disconnected do
    p = arg min_{q∈P} min_{e∈T} d(q, e)
    Add p to T with the perpendicular from p to T.
```

Theorem 3 *Algorithm 2 builds a $\frac{32\sqrt{2}+24\sqrt{3}}{5}$-approximation snowflake tree.*

Proof. We convert the optimal snowflake tree to the output of Algorithm 2 by first replacing its longest edge e with the diameter d of the point set, and connecting the adjacent vertices of e to d, and the rest of the vertices

to the previous edges, rotated in the direction of d or perpendicular to d. Using triangle inequality, the length of the edges other than e is at most multiplied by $\sqrt{2}$. Then, we flip the longest edge of the optimal tree on either side of d recursively to get the snowflake tree. Each flip adds a factor at most as much as the ratio of the cases with 0 Steiner points and 1 Steiner point in the proof of Theorem 1:

$$\frac{\sqrt{a^2+b^2}+\frac{ab}{\sqrt{a^2+b^2}}}{a+b} = \sqrt{1+\frac{a^2b^2}{(a+b)^2(a^2+b^2)}} \leq \sqrt{\frac{9}{8}}.$$

(happens at $a = b$). The approximation factor is at most $\sqrt{2}\sum_{i=0}^{n}((\sqrt{\frac{9}{8}})\frac{\sqrt{3}}{2})^i \leq \frac{32\sqrt{2}+24\sqrt{3}}{5} \leq 17.4$, since the maximum distance of the points in a subproblem of v, to T is at most $\frac{\sqrt{3}}{2}$ times the diameter of v. □

Theorem 4 *Algorithm 2 takes $O(n \log n)$ expected time and $O(n^2 \log n)$ worst-case time.*

Proof. Based on the construction of a snowflake tree, each vertex other than the diameter endpoints can add at most one vertex. So, the number of Steiner vertices is at most $n-2$. By keeping a SVD of T at each step of the algorithm, the point with maximum distance from T can be found in $O(\log n)$ time. The incremental construction of a SVD takes $O(n \log n)$ time. If we reconstruct the SVD after each insertion, the time complexity of the algorithm becomes $O(n^2 \log n)$. □

Without an incremental SVD, it takes $O(n|T|)$ time to compute the farthest point to existing centers by checking all the center-point distances. So, the overall time complexity of the algorithm becomes $\sum_{i=1}^{n} i \cdot n = O(n^3)$.

4 TSP via Snowflake Tree

Algorithm 3 uses the doubling and shortcutting algorithm of [11] to compute an approximate TSP.

Algorithm 3 TSP via Snowflake Tree

Input: A point set P
Output: T: a TSP tour of P
1: Build a snowflake tree of P (Algorithm 2)
2: Double the edges of the snowflake tree and compute a TSP by shortcutting.

Theorem 5 *The TSP constructed from an Euclidean Steiner tree is a 2-approximation.*

Proof. By doubling the edges of the tree and shortcutting, it is possible to prove that the weight of the TSP tour is at most twice the weight of the Euclidean Steiner tree. The weight of the Euclidean Steiner tree is less than the optimal TSP, since the optimal TSP minus an edge is one of the possible solutions of the Euclidean Steiner tree, and the minimum of that set is the Euclidean Steiner tree. Therefore, the weight of the optimal TSP is between the weight of the optimal Euclidean Steiner tree and twice the weight of the optimal Euclidean Steiner tree. □

Using Theorems 3 and 5 and the Steiner ratio, the ETSP constructed via doubling the snowflake tree of Algorithm 2 has approximation factor $2 \cdot \frac{3}{2}\frac{32\sqrt{2}+24\sqrt{3}}{5} \leq 52.1$.

Theorem 6 *The TSP constructed from a snowflake tree of Algorithm 2 is a planar embedding.*

Proof. The diameter is a single segment, so it satisfies the conditions. Points on each side of the diameter are closer to the diameter than segments on the other side. So, we can solve the problem by only considering points on each side. The theorem holds for the perpendiculars from the points to the diameter, so, each of these subproblems can be considered as an instance of the original problem. By induction on the size of T, the proof is concluded. □

References

[1] S. Arora. Polynomial time approximation schemes for Euclidean TSP and other geometric problems. In *Proceedings of the 37th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 2–11. IEEE, 1996.

[2] S. Arora. Nearly linear time approximation schemes for Euclidean TSP and other geometric problems. In *Proceedings of the 38th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 554–563. IEEE, 1997.

[3] J.-D. Boissonnat and M. Yvinec. *Algorithmic geometry*. Cambridge university press, 1998.

[4] M. De Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 1997.

[5] L. J. Guibas and J. Stolfi. On computing all north-east nearest neighbors in the L1 metric. *Inform. Process. Lett.*, 17(4):219–223, 1983.

[6] M. Hanan. On Steiner's problem with rectilinear distance. *SIAM J. Appl. Math.*, 14(2):255–265, 1966.

[7] F. K. Hwang, D. S. Richards, and P. Winter. The Steiner tree problem. number 53 in annals of discrete mathematics, 1992.

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

[9] S. B. Rao and W. D. Smith. Approximating geometrical graphs via "spanners" and "banyans". In *Proceedings of the 30th Annu. ACM Sympos. Theory Comput.*, pages 540–550. ACM, 1998.

[10] M. Shamos. Computational geometry Ph. D. *O Thesis, Yale University*, 1978.

[11] V. V. Vazirani. *Approximation Algorithms*. Springer Science and Business Media, 2013.

Fréchet Distance Queries in Trajectory Data

Joachim Gudmundsson* André van Renssen* Zeinab Saeidi† Sampson Wong*

Abstract

Let π be a trajectory with n vertices in the plane. We show how to preprocess π such that given any two points u and v on π (not necessarily vertices of π) and a horizontal query segment Q, one can quickly determine the **exact** Fréchet distance between Q and the subtrajectory from u to v. We present a data structure that requires $\mathcal{O}(n^2 \log^2 n)$ construction time and space such that queries can be answered in $\mathcal{O}(\log^8 n)$ time.

1 Introduction

The Fréchet distance is a popular measure of similarity between curves as it takes into account the location and ordering of the points along the curves, and it was introduced by Maurice Fréchet in 1906. Measuring the similarity between curves is an important problem in many areas of research, including computational geometry, computational biology [12], data mining, image processing [11] and geographical information science [9].

The Fréchet distance is most commonly described as the dog-leash distance; consider a man standing at the starting point of one trajectory and the dog at the starting point of another trajectory. A leash is required to connect the dog and its owner. Both the man and his dog are free to vary their speed, but they are not allowed to go backward along their trajectory. The cost of a walk is the leash length required to connect the dog and its owner from the beginning to the end of their trajectories. The Fréchet distance is the minimum length of the leash that is needed over all possible walks. More formally, for two curves A and B each having complexity n, the Fréchet distance between A and B is defined as:

$$\delta_F(A, B) = \inf_{\mu} \max_{a \in A} dist(a, \mu(a))$$

where $dist(a, b)$ denotes the Euclidean distance between point a and b and $\mu : A \to B$ is a continuous and non-decreasing function that maps every point in $a \in A$ to a point in $\mu(a) \in B$.

*School of Computer Science, University of Sydney, Australia, {joachim.gudmundsson, andre.vanrenssen, swon7907}@sydney.edu.au

†Combinatorial and Geometric Algorithms Lab, Department of Mathematical Sciences, Yazd University, Yazd, Iran zsaeidi2007@gmail.com

Since the early 90's the problem of computing the Fréchet distance between two polygonal curves has received considerable attention. In 1992 Alt and Godau [1] were the first to consider the problem and gave an $\mathcal{O}(n^2 \log n)$ time algorithm for the problem. The only improvement since then is a randomized algorithm with running time $\mathcal{O}(n^2(\log \log n)^2)$ in the word RAM model by Buchin *et al.* [4]. In 2014 Bringmann [2] showed that, conditional on the Strong Exponential Time Hypothesis (SETH), there cannot exist an algorithm with running time $\mathcal{O}(n^{2-\varepsilon})$ for any $\varepsilon > 0$. Even for realistic models of input curves, such as c-packed curves [7], exact distance computation requires $n^{2-o(1)}$ time under SETH [2]. Only by allowing a $(1 + \varepsilon)$-approximation can one obtain near-linear running times in n and c on c-packed curves [3, 7].

Querying the Fréchet distance between a given trajectory and a query trajectory has been studied [5, 7, 8], but due to the difficult nature of the query problem, data structures only exist for answering a restricted class of queries. The two most relevant results are the following. The first is De Berg *et al.*'s [6] data structure, which answers Fréchet distance queries between a horizontal query segment and a vertex-to-vertex subtrajectory of a preprocessed trajectory. Their data structure can be constructed in $\mathcal{O}(n^2 \log^2 n)$ time using $\mathcal{O}(n^2 \log^2 n)$ space such that queries can be answered in $\mathcal{O}(\log^2 n)$ time. The second is Driemel and Har-Peled's [7] data structure, which answers approximate Fréchet distance queries between a query trajectory of complexity k and a vertex-to-vertex subtrajectory of a preprocessed trajectory. The data structure can be constructed in $\mathcal{O}(n \log^3 n)$ using $\mathcal{O}(n \log n)$ space, and a constant factor approximation to the Fréchet distance can be answered in $\mathcal{O}(k^2 \log n \log(k \log n))$ time. In the special case when $k = 1$, the approximation ratio can be improved to $(1+\varepsilon)$ with no increase in preprocessing or query time with respect to n. Other query versions for the Fréchet distance have also been considered [5, 8].

In this paper, we answer exact Fréchet distance queries between a subtrajectory (not necessarily vertex-to-vertex) of a preprocessed trajectory and a horizontal query segment. The data structure can be constructed in $\mathcal{O}(n^2 \log^2 n)$ time using $\mathcal{O}(n^2 \log^2 n)$ space such that queries can be answered in $\mathcal{O}(\text{polylog}\, n)$ time. We use Megiddo's parametric search technique [10] and De Berg *et al.*'s [6] data structure to compute the Fréchet distance.

2 Preliminaries

Let $p_1, \ldots, p_n$ be a sequence of n points in the plane. We use $\pi = (p_1, p_2 \ldots, p_n)$ to denote the polygonal trajectory defined by this sequence. Let $x_0 \leq x_1$ and $y \in \mathbb{R}$, and define $p = (x_0, y)$ and $q = (x_1, y)$ so that $Q = pq$ is a horizontal segment in the plane. Let u and v be two points on the trajectory π, $\pi[u, v]$ denotes the subtrajectory of π that starts at u and ends at v. Following De Berg *et al.* [6], the Fréchet distance between $\pi[u, v]$ and Q can be computed by using the formula:

$$\delta_F(\pi[u,v], pq) = \max\{\|up\|, \|vq\|, \delta_{\overrightarrow{h}}(\pi[u,v], pq), B(\pi[u,v], y)\}.$$

The first two terms are simply the distances between the starting points of the two trajectories, and the ending points of the two trajectories. The third term is the directed Hausdorff distance between $\pi[u, v]$ and Q, which can be computed as follows:

$$\delta_{\overrightarrow{h}}(\pi[u,v], Q) = \max\{\max_{p_i.x \in (-\infty, x_0]} \|p - p_i\|, \max_{p_i.x \in [x_1, \infty)} \|q - p_i\|, \max_i \|y - p_i.y\|\}.$$

where p_i in the formula above is a vertex of the subtrajectory $\pi[u, v]$, and $p_i.x$ is its x-coordinate. The formula handles three cases for mapping every point of $\pi[u, v]$ to its closest point on Q. The first term describes mapping points of $\pi[u, v]$ to the left of p to their closest point p. The second term describes mapping points of $\pi[u, v]$ to the right of q analogously. The third term describes mapping points of $\pi[u, v]$ that are in the vertical strip between p and q to their orthogonal projection onto Q.

The fourth term in our formula for the Fréchet distance is the maximum backward pair distance over all backward pairs. A pair of vertices (p_i, p_j) (with $j > i$) is a backward pair if p_j lies to the left of p_i. The backward pair distance of $\pi[u, v]$ can be computed from:

$$B(\pi[u,v], y) = \max_{\forall p_i, p_j \in \pi[u,v]: i \leq j, p_i.x \geq p_j.x} B_{(p_i,p_j)}(y),$$

where $B_{(p_i,p_j)}(y)$ is the backward pair distance for a given backward pair (p_i, p_j) and is defined as

$$B_{(p_i,p_j)}(y) = \min_{x \in \mathbb{R}} \max\{\|p_i - (x, y)\|, \|p_j - (x, y)\|\}.$$

The distance terms in the braces compute the distance between a given point (x, y) and the farthest of p_i and p_j. Let us call this the backward pair distance of (x, y). Then the function $B_{(p_i,p_j)}(y)$ denotes the minimum backward pair distance of a given backward pair (p_i, p_j) over all points (x, y) which have the same y-coordinate. Taking the maximum over all backward pairs gives us the backward pair distance for $\pi[u, v]$.

In Figure 1, we show for each y-coordinate the point with the minimum backward pair distance (left), and the magnitude of this minimum distance (right). We see in the figure that the function $B_{(p_i,p_j)}(y)$ consists of two linear functions joined together in the middle with a hyperbolic function.

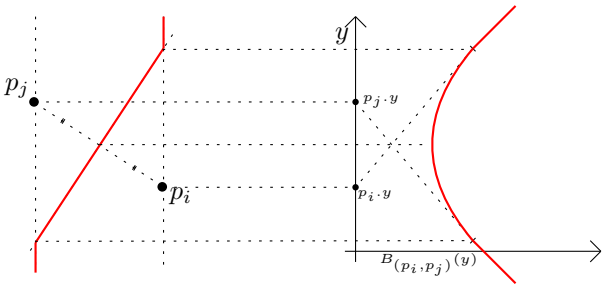


Figure 1: For each y-coordinate, Left: the point with minimum backward pair distance, Right: the minimum backward pair distance.

In order to apply parametric search, we are required to construct a set of critical values (which we will describe in detail at a later stage) so that an optimal solution is guaranteed to be contained within this set. Since this set of critical values is often large, we need to avoid computing the set explicitly, but instead design a decision algorithm that efficiently searches the set implicitly. Megiddo's parametric search [10] states that if:

- the set of critical values has polynomial size, and
- the Fréchet distance is convex with respect to the set of critical values, and
- a comparison-based decision algorithm decides if a given critical value is equal to, to the left of, or to the right of the optimum,

then there is an efficient algorithm to compute the optimal Fréchet distance in $\mathcal{O}(PT_p + T_pT_s \log P)$ time, where P is the number of processors of the (parallel) algorithm, T_p is the parallel running time and T_s is the serial running time of the decision algorithm. For our purposes, $P = 1$ since we run our queries serially, and $T_p = T_s = \mathcal{O}(\text{polylog}\, n)$ for the decision versions of our query algorithms.

3 Computing the Fréchet Distance

The first problem we apply parametric search to is the following. Given any horizontal query segment Q in the plane and any two points u, v on π (not necessarily vertices of π), determine the Fréchet distance between Q and the subtrajectory $\pi[u, v]$.

Let p_u be the first vertex of π along $\pi[u, v]$ and let p_v be the last vertex of π along $\pi[u, v]$, as illustrated

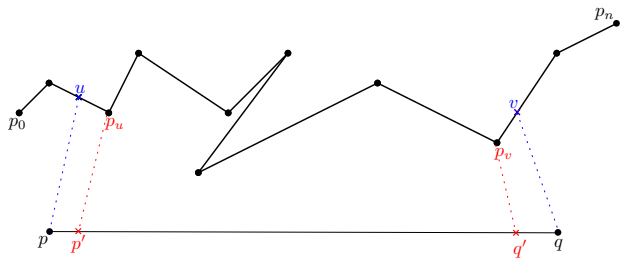


Figure 2: The points p' and q' that are mapped to the vertices p_u and p_v of the trajectory.

in Figure 2. If p_u and p_v do not exist then $\pi[u,v]$ is a single segment so the Fréchet distance between $\pi[u,v]$ and Q can be computed in constant time. Otherwise, our goal is to build a Fréchet mapping $\mu : \pi[u,v] \to Q$ which attains the optimal Fréchet distance. We build this mapping μ in several steps. Our first step is to compute points p' and q' on the horizontal segment pq so that $p' = \mu(p_u)$ and $q' = \mu(p_v)$.

If the point p' is computed correctly, then the mapping $p' \to p_u$ allows us to subdivide the Fréchet computation into two parts without affecting the overall value of the Fréchet distance. In other words, we obtain the following formula:

$$\delta_F(\pi[u,v], pq) = \max\{\delta_F(up_u, pp'),\ \delta_F(\pi[p_u,v], p'q)\} \tag{1}$$

We now apply the same argument to p_v. We compute q' optimally on the horizontal segment $p'q$ optimally so that the mapping $p_v \to q'$ does not increase the Fréchet distance between the subtrajectory $\pi[p_u,v]$ and the truncated segment $p'q$. In other words, we have:

$$\delta_F(\pi[u,v], pq) = \max\{\delta_F(up_u, pp'),\ \delta_F(\pi[p_u,p_v], p'q'),\ \delta_F(p_vv, q'q)\} \tag{2}$$

Now that p_u and p_v are vertices of π, De Berg *et al.* [6] provide an efficient data structure for computing the middle term $\delta_F(\pi[p_u,p_v], p'q')$. The first and last terms have constant complexity and can be handled in constant time. All that remains is to compute the points p' and q' efficiently.

Theorem 1 *Given a trajectory π with n vertices in the plane. There is a data structure that uses $\mathcal{O}(n^2 \log^2 n)$ space and preprocessing time, such that for any two points u and v on π (not necessarily vertices of π) and any horizontal query segment Q in the plane, one can determine the exact Fréchet distance between Q and the subtrajectory from u to v in $\mathcal{O}(\log^8 n)$ time.*

Proof. Decision Algorithm. Let S be the set of critical values (defined later in this proof), let s be the current candidate for the point p', and let $F(s) = \max(\delta_F(ps, up_u), \delta_F(sq, \pi[p_u,v]))$ be the Fréchet distance between pq and $\pi[u,v]$ subject to p_u being mapped to s. Our aim is to design a decision algorithm running in $\mathcal{O}(\log^4 n)$ time that decides whether the optimal p' is equal to s, to the left of s or to the right of s. This is equivalent to proving that all points to one side of s cannot be the optimal p' and may be discarded.

We use the Fréchet distance formula from Section 2 to rewrite $F(s) = \max(\|up\|, \|vq\|, \|p_us\|, \delta_{\overrightarrow{h}}(\pi[p_u,v], sq), B(\pi[p_u,v], y))$. Then we check several cases depending on which of these five terms attains the maximum value $F(s)$, and in each case we either deduce that $p' = s$ or all critical values to one side of s may be discarded.

- If $F(s) = \max(\|up\|, \|vq\|, B(\pi[p_u,v], y))$, then $p' = s$. We observe that none of the three terms on the right hand side of the equation depend on the position of s. Hence, $F(s) = \max(\|up\|, \|vq\|, B(\pi[p_u,v], y)) \le F(p')$, and since $F(p')$ is the minimum possible value, $F(s) = F(p')$. We have found a valid candidate for p' and can discard all other candidates in the set S.

- If $F(s) = \|p_us\|$ and p_u is to the right (left) of s, then p' is to the right (left) of s. We will argue this for when p_u is to the right of s, but an analogous argument can be used when p_u is to the left. We observe that all points t to the left of s will now have $\|p_ut\| > \|p_us\|$. Hence, $F(s) = \|p_us\| < \|p_ut\| \le F(t)$ for all points t to the left of s, therefore all points to the left of s may be discarded.

- If $F(s) = \delta_{\overrightarrow{h}}(\pi[p_u,v], sq)$, then p' is to the left of s. The directed Hausdorff distance maps every point in $\pi[p_u,v]$ to their closest point on sq, so by shortening sq to tq for some point t on sq to the right of s, the directed Hausdorff distance cannot decrease. Hence, $F(s) \le F(t)$ for all t to the right of s, so all points to the right of s may be discarded.

To determine q' for a fixed candidate s for p', we treat the problem in a similar way. We consider the subtrajectory $\pi[p_u,v]$ and the horizontal line segment sq. Defining a function $G(t)$ representing the Fréchet distance when p_v is mapped to t, we obtain a similar decision algorithm. The most notable difference is that since we now consider the end of the subtrajectory, the decisions for moving t left and right are reversed.

Convexity. We will prove that $F(s)$ is convex, and it will follow similarly that $G(t)$ is convex. It suffices to show that $F(s)$ is the maximum of convex functions, since the maximum of convex functions is itself convex. The three terms $\|up\|$, $\|vq\|$, $B(\pi[p_u,v], y)$ are constant. The term $\|p_us\|$ is an upward hyperbola and is convex. If suffices to show that $\delta_{\overrightarrow{h}}(\pi[p_u,v], sq)$ is convex.

We observe that the Hausdorff distance $\delta_{\overrightarrow{h}}(\pi[p_u,v], sq)$ must be attained at a vertex p_i of $\pi[p_u,v]$, and that each $\delta_{\overrightarrow{h}}(p_i, sq)$ as a function of s

is a constant function between p and the orthogonal projection of vertex p_i onto the horizontal segment pq, and a hyperbolic function between the orthogonal projection of vertex p_i onto the horizontal segment pq and q. Thus, the function for each p_i is convex, so the overall Hausdorff distance function is also convex.

Critical Values. A critical value is a value c which could feasibly attain the minimum value $F(c) = F(p')$. We represent $F(s)$ as the minimum of n simple functions and then argue that the minimum of F can only occur at the minimum of one of these functions, or at the intersection of a pair of these functions.

First, $\|up\|, \|vq\|, B(\pi[p_u, v], y)$ are constant functions in terms of s. Next, $\|p_u s\|$ is a hyperbolic function. Finally, $\delta_{\overrightarrow{h}}(\pi[p_u, v], sq)$ is not itself simple, but it can be rewritten as the combination of n simple functions as described in the above section.

Hence, $F(s)$ is the combination (maximum) of n simple functions, and these functions are simple in that they are piecewise constant or hyperbolic. Hence, $F(s)$ attains its minimum either at the minimum of one of these n functions, or at a point where two of these functions intersect. Therefore, there are at most $\mathcal{O}(n^2)$ critical values for $F(s)$.

Query Complexity. Computing q' for a given candidate s for p' takes $\mathcal{O}(\log^4 n)$ time: We can compute the terms $\|up\|$, $\|p_u s\|$, $\|vq\|$, and $\|p_v q'\|$ in constant time. The terms $B(\pi[p_u, p_v], y)$ and $\delta_{\overrightarrow{h}}(\pi[p_u, p_v], sq')$ can be computed in $\mathcal{O}(\log^2 n)$ time using the existing data structure by De Berg *et al.* [6]. We need to determine the time complexity of the sequential algorithm T_s, parallel algorithm T_p, and the number of the processors P. To find q', the decision algorithm takes $T_s = \mathcal{O}(\log^2 n)$. The parallel form runs on one processor in $T_p = \mathcal{O}(\log^2 n)$. Substituting these values in the running time of the parametric search, $\mathcal{O}(PT_p + T_p T_s \log P)$, leads to $\mathcal{O}(\log^4 n)$ time.

The above analysis implies that p' itself can be computed in $\mathcal{O}(\log^8 n)$ time: For a given s, the decision algorithm runs in $T_s = \mathcal{O}(\log^4 n)$ as mentioned above. The parallel form of the decision algorithm runs on one processor in $T_p = \mathcal{O}(\log^4 n)$. Substituting these values in the running time of the parametric search leads to $\mathcal{O}(\log^8 n)$ time.

Preprocessing and Space. To compute the second term of Formula 2, we use the data structure by De Berg *et al.* [6] which uses $\mathcal{O}(n^2 \log^2 n)$ space and preprocessing time and supports $\mathcal{O}(\log^2 n)$ query time. □

4 Concluding remarks

In the full version of the paper we also study the problem of preprocessing π into a data structure such that given any length L and two points u, v on π (not necessarily vertices of π), one can efficiently determine the placement of the horizontal segment Q of length L in the plane that minimises the Fréchet distance to the subtrajectory $\pi[u, v]$. We show how to preprocess π in $\mathcal{O}(n^2 \log^2 n)$ time and space, such that the above query can be answered in $\mathcal{O}(\text{polylog}\, n)$ time.

References

[1] H. Alt and M. Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(2):75–91, 1995.

[2] K. Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science*, pages 661–670, 2014.

[3] K. Bringmann and M. Künnemann. Improved approximation for Fréchet distance on c-packed curves matching conditional lower bounds. *International Journal of Computational Geometry & Applications*, 27(1-2):85–120, 2017.

[4] K. Buchin, M. Buchin, W. Meulemans, and W. Mulzer. Four Soviets walk the dog: Improved bounds for computing the Fréchet distance. *Discrete & Computational Geometry*, 58(1):180–216, 2017.

[5] M. de Berg, A. F. Cook, and J. Gudmundsson. Fast Fréchet queries. *Computational Geometry*, 46(6):747–755, 2013.

[6] M. De Berg, A. D. Mehrabi, and T. Ophelders. Data structures for Fréchet queries in trajectory data. In *Proceedings of the 29th Canadian Conference on Computational Geometry*, 2017.

[7] A. Driemel and S. Har-Peled. Jaywalking your dog: computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42(5):1830–1866, 2013.

[8] J. Gudmundsson and M. Smid. Fast algorithms for approximate Fréchet matching queries in geometric trees. *Computational Geometry*, 48(6):479–494, 2015.

[9] P. Laube. *Computational Movement Analysis*. Springer Briefs in Computer Science. Springer, 2014.

[10] N. Megiddo. Applying parallel computation algorithms in the design of serial algorithms. In *22nd Annual Symposium on Foundations of Computer Science (sfcs 1981)*, pages 399–408. IEEE, 1981.

[11] E. Sriraghavendra, K. Karthik, and C. Bhattacharyya. Fréchet distance based approach for searching online handwritten documents. In *Proceedings of the 9th International Conference on Document Analysis and Recognition*, volume 1, pages 461–465, 2007.

[12] T. Wylie and B. Zhu. Protein chain pair simplification under the discrete Fréchet distance. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 10(6):1372–1383, 2013.

Path Planning with Objectives Minimizing Length and Maximizing Clearance

Mansoor Davoodi* Maryam Sanisales*

Abstract

In this paper, we study the problem of bi-objective path planning with the objectives minimizing the length and maximizing the *clearance* of the path, that is, maximizing the minimum distance between the path and obstacles. We consider a set of vertical segments as the obstacles and propose an efficient algorithm for finding all intervals of Pareto optimal solutions when the fist objective is evaluated with Euclidean metric and the second one is evaluated by Manhattan metric. Finally, we show that the algorithm results in finding $(\sqrt{2}, 1)$-approximation Pareto optimal solutions when both objectives are evaluated with Euclidean metric.

1 Introduction

Path Planning (PP) is one of the challenging problems in the field of robotics. The goal is to find optimal path(s) for two given start and destination points among a set of obstacles. However, usually minimizing the length of the path is considered as the optimality criterion. The application of the problem's other objectives such as smoothness and clearance [2], that is maximizing the distance between the path and obstacles, have been also considered in the literature [2]. For example, in many applications, the robot needs to move around in order to perform its task properly. The need to move around the environment led to the question of what path a robot can take to accomplish its task, in addition to being safe. In this paper, we define the *optimal path* regarding to two objectives minimizing the length of the path and maximizing the minimum distance between the path and obstacles.

A classical approach to compute the minimum length path is computing the visibility graph of obstacles and convert the problem to a graph search problem. For a set of n obstacle vertices, the visibility graph can be computed in $O(n^2 \log n)$ time using a tree structure and ray technique [4]. This approach is one of the best-known algorithms to obtain the shortest path where the distance between the path and the obstacle is equal to zero– a path with clearance zero. Also, Hershberger et al. [5], proposed an efficient planar structure for the PP problem in $O(n \log n)$ time.

*Department of Computer Sciences and Information Technology, Institute for Advances Studies in Basic Sciences, Zanjan, Iran. mdmonfared@iasbs.ac.ir ; maryamsan@iasbs.ac.ir

Wein et al. [7], by using a combination of the Voronoi diagram and the visibility graph, introduced a new type of visibility structure called the *Voronoi Visibility Diagram* to find the shortest path for a predefined value λ of clearance. He considered the PP problem in the setting of single objective optimization with the objective minimizing the length subject to minimum clearance λ. Geraerts [3] proposed a new data structure called the *Explicit Road Map* that creates the shortest possible path with the maximum possible clearance. The introduced structure is useful for computing the path in corridor spaces. Davoodi [1] studied the problem of bi-objective PP in a grid with the two objectives of minimizing path length and maximizing *clearance* and then showed Pareto optimal solutions to the two-objective problem in the grid workspace. He also studied the problem in the continuous space under Manhattan metric, and proposed an $O(n^3)$ time algorithm where the obstacles are n vertical segments. The problem under Euclidean metric remained as an open problem.

We study the problem of bi-objectives PP in continuous space with the objectives minimizing the length of the path and maximizing its minimum distance from obstacles. The goal is computing Pareto optimal solutions, that are, the solutions which cannot be shortened if and only if their clearance is minimized– Since this problem is a bi-objective optimization problem in the continuous space, there is an infinite number of Pareto optimal solutions. So, it is impossible to provide a polynomial algorithm to compute all the solutions. To smooth this issue, we focus on different Pareto optimal solutions, the optimal paths with different middle points –called *Distinct Pareto Optimal* paths. We consider a PP search space with n vertical segments as obstacles and propose an $O(n^3)$ time algorithm to compute all distinct Pareto optimal solutions where the length of the paths is evaluated with Euclidean and the clearance is evaluated with Manhattan Metric. Then, we show that the solutions are efficient approximation solutions of the problem where the both objectives are evaluated under Euclidean metric.

The next section briefly introduces the bi-objective problem. Section 3 proposes an exact algorithm for computing distinct Pareto optimal solutions when the length and clearance objectives are evaluated under Euclidean and Manhattan metric, respectively. Section 4 extends the results to the case both the objectives are evaluated under Euclidean distance.

2 Bi-objective Path Planning Problem

Let $O = \{s_1, s_2, ..., s_n\}$ be a set of left to right sorted vertical segments as the obstacles, and s and t be the start and destination points in the plane. Assume, w.l.o.g., that s and t lie on the left and right side of the obstacles, respectively. Let the shortest path from s to t be denote by *s-t-path*.

For a collision-free path $P = \{s = v_0, v_1, \cdots, v_m, v_{m+1} = t\}$ with m *breakpoints* $v_1, v_2, \cdots, v_m$, define $L(P)$ as the length of P under Euclidean metric. Also, define $C_1(P)$ and $C_2(P)$ as the *clearance* of P under Manhattan and Euclidean metrics, respectively, that is minimum distance between P and the obstacles. We denote the clearance of P with $C(P)$ in general. The objectives are minimizing $L(P)$ and maximizing $C(P)$ in the bi-objective path planning problem.

For two collision-free paths P and P', we say that P *dominates* P' and denote it by $P \preceq P'$, if $L(P) < L(P')$ and $C(P) \geq C(P')$, or if $L(P) \leq L(P')$ and $C(P) > C(P')$. For any pair of paths P and P', three cases may happen, (i) $P \preceq P'$, (ii) $P' \preceq P$ and (iii) none of them dominates. In the third case we say that P and P' are *non-dominated*. That means, there is no strict preference between P and P'.

Given the definition of dominance, paths such as P which is not dominated by other collision-free paths, are called *Pareto optimal paths*. Any improvement of P in its length or clearance comes from sacrificing it in the other objective. Let Π^* be the set of all Pareto optimal paths. Pareto-optimal front(s) consists of the projection of Π^* in the *objective space*, That is, the two- dimensional space with the objective values length and clearance of each path $P \in \Pi^*$. For any small clearance value λ, there is some shortest path in the workspace [1]. Therefore, the projection of Pareto front of the problem on *C-space* is one component, while it is possible the projection of Pareto front of the problem on *L-space* are several components.

Since the workspace of bi-objective path planning is continuous, Pareto front is infinite set in general. Thus, there is no algorithm to construct path is Π^* one-by-one. Two approaches are proposed to handle this issue in this paper:

- Finding all different (or *Distinct*) Pareto optimal paths, the path with different breakpoints in the workspace. In other words, the Pareto front is a set of discrete components. We can compute the *extreme* solution of each component.
- Finding a set of finite and polynomial size of solutions which are an approximation set of all Pareto fronts.

The first approach will result in finding different intervals $I_1, I_2, ..., I_k$, for some k, of clearance. We will compute the lower and upper bound of the intervals and map it with a set of shortest paths with *almost* same breakpoints. We use this approach and solve the problem of bi-objective path planning for solving the problem when clearance of the path is measured with Manhattan metric. Also, we show these computed paths are approximate solutions when the case length and clearance of the paths are measured with Euclidean metric.

3 Bi-Objective Path Planning with Euclidean Length and Manhattan Clearance

Let us explain our approach to solve the bi-objective path planning algorithm roughly at first. We construct a shortest path map –called *SPM*– with clearance $\lambda = 0$ by using the idea proposed by Lee and Preparata [6]. The idea is using a *sweepline* from the start point s to the destination point t algorithm based on monotonicity of the shortest. While the sweep-line moves from left to right, the left side vertices of the segments construct a tree with the root s and the parent of each vertex p is the vertex p' such that p' is the last breakpoint in the shortest path between s and p. Also, in each step the right side of the sweep line is decomposed to a set of regions with the property that the points in a region have the same parent in the left side of the sweepline. When the sweepline reaches the destination point t, the shortest path (with clearance $C_1(P) = \lambda = 0$) can be easily computed from s to t by a simple backward manner. After computing such a general map –called SPM(0)– we increase value of λ from 0 to ∞ to compute the Pareto front intervals. To this end, we compute all critical λ may change the shortest path tree, particularly the breakpoints of *s-t-paths*. Since, the obstacles of the workspace, $O = \{s_1, s_2, ..., s_n\}$, are vertical segments, the following observation is clear.

Observation 1 *The shortest path from s to the endpoints of the obstacles (and also to t) is an $x-$monotone path.*

For finding the shortest path with clearance λ, we need to expand (or fatten) the obstacles with size λ. Define $O(\lambda) = \{s_1(\lambda), s_2(\lambda), \cdots, s_n(\lambda)\}$, where $s_i(\lambda)$ is the segment s_i after fattening with size λ under Manhattan distance. That is, if $\oplus$ shows the Minkowski sum of two objects, $s_i(\lambda) = s_i \oplus Sq(\lambda)$, where $Sq(\lambda)$ is a square with the diameter size 2λ which is rotated by an angle of $\pi/4$. Indeed, the boundary of $Sq(\lambda)$ is the set of all points in the plane whose Manhattan distance from the center of $Sq(\lambda)$ is exactly λ.

Observation 2 *The breakpoints of any s-t-path with clearance λ belong to the set $O(\lambda)$.*

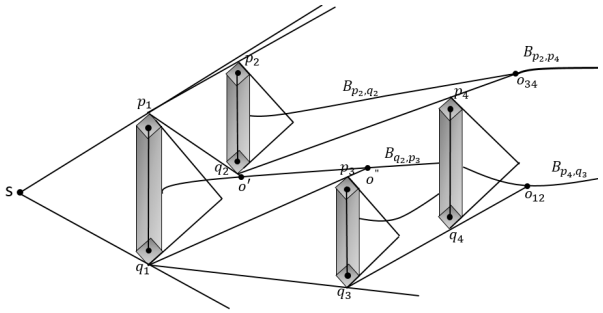


Figure 1: The SPM map for four obstacles with visibility edges, SPM-point and bisectors.

Based on Observation 2, our strategy in finding the Pareto optimal intervals is first computing SPM for $O(\lambda = 0)$. Then by increasing λ, we handle the events may change SPM and the shortest path from s to t. To this end, we construct a data structure that can handle the events and create the shortest possible path and report the paths that are distinct. The tree defines the shortest path on the set $O(\lambda) \cup \{s, t\}$ with root s. We will explain below important features of the tree [6].

3.1 SPM(0), the Shortest Path Map for $O(\lambda = 0)$

SPM(λ =0) is an incremental constructed tree at root s which is obtained by a sweepline strategy and contains the shortest path from s to any obstacle's vertices. Suppose SPM(0) is available for the obstacles $s_1 = \overline{p_1q_1}, \cdots, s_{i-1} = \overline{p_{i-1}q_{i-1}}$. Each node p_j (or q_j), for $j < i$ has a particular $weight(p_j)$ that shows the length of s-p_j-path. The right halfplane of s_{i-1} is decomposed to a set of regions corresponding with a node in SPM(0) tree as its parent.

When the sweepline meets obstacle $s_i = \overline{p_iq_i}$, first, the regions which p_i and q_i lies are founded, and then they inserted as new leaves into SPM(0) with the parents corresponding with the regions. Also, their weights is computed using the weights of their parents. Finally, the decomposition of the halfplane of s_i is updated using bisector of p_i and q_i and the new visibility edges. The bisector of p_i and q_i, denoted by B_{p_i,q_i}, is the intersection of regions corresponding with p_i and q_i. That is, any points p on the right side of s_i which length of s-p_i-p-*path* and s-q_i-p-*path* are the same (see Fig.1). The points of SPM are generally of three types; the intersection points between a pair of bisectors, between bisectors, and obstacles and between bisectors and the visibility edges of the obstacles.

Theorem 1 *For a set of n vertical segments, the size of the SPM containing all points and bisectors is linear and the SPM can be constructed in $O(n \log n)$ time [6].*

3.2 SPM(λ), the Shortest Path Map for $O(\lambda > 0)$

After computing SPM(0), by fatting the obstacles with size λ, we able to compute and report a Pareto optimal solution with clearance λ as shown in Fig.1. To find all Pareto optimal intervals, we need to consider all the distinct paths that are the endpoints of the fronts. When an increase in λ changes the path, some breakpoints may change, in which case an event occurs.

$s_i(\lambda) = s_i \oplus Sq(\lambda)$ has six vertices and they can be easily computed as linear functions respect to parameter λ, e.g., if p_i and q_i are the top and bottom endpoints of s_i, the highest and the lowest vertices of $s_i(\lambda)$ can be shown $p_i(\lambda) = y_{pi} + \lambda$ and $q_i(\lambda) = y_{qi} - \lambda$, where y_{pi} and y_{qi} denotes the y-coordination of p_i and q_i, respectively. Note that, the other four vertices of s_i local changes in the map, can be computed without any increasing in the time or space of the algorithms' order. We explain the details of this issue in the Appendix. When λ increases, it is possible some shortest path change. We call such values of λ the *critical* λs that can be obtained by considering all events may change the structure of the SPM. Three types of events may occur:

1) A function $p_i(\lambda)$ (or $q_i(\lambda)$) intersects with $B_{p_i,p_k}(\lambda)$ for some i and k.

2) A function $p_i(\lambda)$ (or $q_i(\lambda)$) intersects with some visibility edges of the obstacles. When visibility edges change. There are two types of such edges; The edge between $\overline{p_ip_j}$ or $\overline{q_iq_j}$, and the between $\overline{p_iq_j}$ or $\overline{q_ip_j}$. In the first type of edges the slop remains unchanged. However, in the second the slop changes linearly respect to λ.

3) Two obstacles are joined while they are fatting.

To handle the last type, we simply consider the two joined neighbor obstacles as one obstacle with the topmost, bottommost and bisector functions they have. It is possible to handle the differences between their x-coordinates. The size of such type of events is $O(n)$. We can compute all critical λs in the beginning and handle each event in the worst case $O(n)$.
The first type of events occurs when a function $p_i(\lambda)$ (or $q_i(\lambda)$) intersects with the bisector $B_{p_j,p_k}(\lambda)$ for some j and k. We need to handle this case if $p_i(\lambda)$ lies on the corresponding region of one of $p_i(\lambda)$ or $q_i(\lambda)$, otherwise no change is needed. So, we update SPM by inserting an edge (q_k, p_i) and update the weights of the sub tree with root q_k (See Fig. 2).
The second type of events occurs when a functions $p_i(\lambda)$ (or $q_i(\lambda)$) intersects with some $O(n^2)$ visibility edges. So, we update SPM by removing the edge (q_j, p_k) and inserting an edge (p_i, p_k) (See Fig. 3).

Using a heap structure, the mentioned events and critical λs can be handled efficiently. After each iteration of the above processes, we update the length of

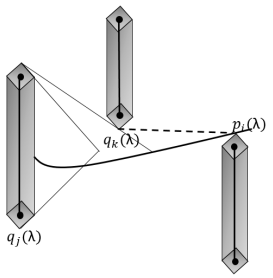


Figure 2: Illustration of the first type of events.

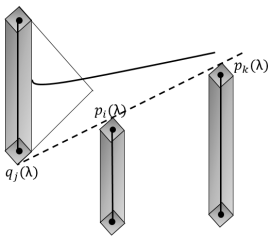


Figure 3: Illustration of the second type of events.

all shortest path from s to the endpoints of the obstacles and update the bisectors one-by-one from left to right as well as updating the critical λs in the heap to extract the minimum critical λ.

Let m be the number of breakpoints of *s-t-path*. So, using SPM(λ) *s-t-path* can be reported in $O(m)$ time for any value of λ. Also, all the *s-t-path* are represented with at most $O(n^2 + R)$ breakpoints in $O(mn^2 + nR)$ time, where R is the number of insertion and deletions to the heap structure, which is $O(n^2)$ in the worst case. Since $m = O(n)$, the time complexity of the algorithm is $O(n^3)$.

4 Bi-Objective Path Planning with Euclidean Length and Clearance

To solve the problem of bi-objective path planning when the both objectives are evaluated with Euclidean distance, we need to fat the obstacles using a disk with radius λ, denoted by $D(\lambda)$, instead of $Sq(\lambda)$. However, the structure of the proposed algorithm remain unchanged for computing the expanded obstacles, it difficult to update the length of the shortest path. In fact, we need to present the shortest paths as a function respect to parameter λ. When $D(\lambda)$ is used to Minkowski sum, the shortest path is obtained by tangent lines of some growing disks which is a high order function based on λ. So, in the following, we show that the proposed algorithm in the previous section provides approximation solutions under the following definition when the both objectives are evaluated under Euclidean distance.

Definition. Let Π be a bi-objective minimization problem with the objectives f_1 and f_2. A solution X is an (α, β)-*approximation* Pareto optimal solution for Π, if there is no solution Y such that $f_1(X) \geq \alpha f_1(Y)$ and $f_2(X) > \beta f_2(Y)$, or $f_1(X) > \alpha f_1(Y)$ and $f_2(X) \geq \beta f_2(Y)$.

Theorem 2 . *For the problem of bi-objective path planning under Euclidean metric, there is a* $(\sqrt{2}, 1)$- *approximation of Pareto optimal solutions.*

Proof. The proof is a straightforward result of the fact that $D(\lambda)$ can be approximated using the cocentric square $Sq(\lambda)$. □

5 Conclusion

In this paper, we considered a bi-objective path planning problem with objectives minimizing the length and maximizing the clearance– That is maximizing the minimum distance between the path and objectives. We assumed the workspace contains a set of n vertical segments, and propose an efficient $O(n^3)$ time algorithm for finding Pareto optimal solutions of the problem where the length and clearance are evaluated by Euclidean and Manhattan metrics, respectively. Also, we show that such path are good approximation solutions when the both objectives are evaluated by Euclidean metric. So, an open problem remains here is proposing an efficient algorithm to find the Pareto optimal solutions for this case of the problem.

References

[1] M. Davoodi, Bi-objective path planning using deterministic algorithms. *Journal of Robotics and Autonomous Systems*, 93:105-115, 2017.

[2] M. Davoodi, F. Panahi, A. Mohades, S. N. Hashemi Clear and smooth path planning, Applies Soft Computing, 32, pp. 568–579, 2015.

[3] R. Geraerts, Planning short paths with clearance using explicit corridors, *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 19972004, 2010.

[4] S. K. Ghosh and D. M. Mount, An output sensitive algorithm for computing visibility graphs, *in 28th Annual Symp. on Found. of Computer Science*, 1987.

[5] J. Hershberger and S. Suri, An Optimal Algorithm for Euclidean Shortest Paths, 28(6), pp. 2215-2256, 1999.

[6] D. T. Lee and F. P. Preparata, Euclidean shortest paths in the presence of rectilinear barriers, Networks, 14(3), pp. 393-410, 1984.

[7] R. Wein, J. P. van den Berg, and D. Halperin, The Visibility Voronoi Complex and Its Applications, Proc. twenty-first Annu. Symp. Comput. Geom. - SCG 05, 39250, pp. 63, 2005.

On the expected weight of the theta graph on uncertain points

Behnam Iranfar* Mohammad Farshi* Amir Mesrikhani*

Abstract

Given a point set $S \subset \mathbb{R}^d$, the θ-graph of S is as follows: for each point $s \in S$, draw cones with apex at s and angle θ and connect s to the point in each cone such that the projection of the point on the bisector of the cone is the closest to s. One can define the θ- graph on an uncertain point set, i.e. a point set where each point s_i exists with an independent probability $\pi_i \in (0, 1]$. In this paper, we propose an algorithm that computes the expected weight of the θ-graph on a given uncertain point set. The proposed algorithm takes $O(n^2\alpha(n^2, n)^{2d})$ time and $O(n^2)$ space, where n is the number of points, d and θ are constants, and α is the inverse of the Ackermann's function.

1 Introduction

In many applications, such as sensor databases, mobile object tracking, computer vision or location-based services, the existence or location of the data is uncertain, but we can use statistical information. There are several models for uncertain data. Some of them assign an area to each point which represents the area that the point resides, but the exact position of the point in its corresponding area is unknown. In the tuple model of uncertain data, each input point has a fixed location but it only exists probabilistically. The input is a pair (S, Π) such that $S = \{s_1, s_2, \ldots, s_n\}$ is a set of n points (sites) in $\mathbb{R}^d$, and $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ is a probability vector with the interpretation of that sites.

The geometric structures on uncertain data are also interesting because they have wide application in solving problems. For example, some geometric graphs have application in protein visualization, wireless networking, motion planning, and real-time animation [8, 10, 12, 19]. The properties on a geometric graph influence the time and space complexity of the problem that uses the structure. For example, if one wants to construct a network on a set of points, then the total cost of the network, i.e. the sum of all edge weights of the graph, is an important parameter. For uncertain data, the properties of the structures on the data are not deterministic, but one can compute the expected properties of the structures on uncertain data, see for example [13, 17].

In this paper, we study the problem of computing the expected weight of a well-known geometric graph known as the θ-graph [10, 14, 15]. For building the θ-graph for $S \subset \mathbb{R}^d$, divide the space around each point $p \in S$ into a set of cones $\mathcal{C}$ of maximum angle θ, and then for each cone $c \in \mathcal{C}$, connect p to a point $q \in S$ that lies in cone c and minimizes the Euclidean distance between p and the projection of q into the bisector l_c of c. For details see [18, Chapter 4].

1.1 Related work.

Uncertainty has been studied in some articles in computational geometry. In 2013, Suri *et al.* [20] have studied the most likely convex hull under the uncertain points and showed that the most likely convex hull under the point model (tuple model) can be computed in $O(n^3)$ time in $d = 2$ dimension, but it is NP-hard for $d \geq 3$. Agarwal *et al.* [4], in 2014, have studied the problem of computing probability of query point lying inside the convex hull, and their results have included both approximation and exact algorithm for given uncertain points. In 2015, Zhang [22] have formulated two different nearest neighbors on uncertain points: the expected nearest neighbor, where the expected distance between each input point and a query point has been considered, and the probabilistic nearest neighbor. There also has several papers that consider range searching, indexing, the uncertain data and community direction [1, 2, 3, 5, 6, 7, 11].

2 The expected weight of the θ-graph

In this section, we will describe an algorithm for computing the expected weight of the θ-graph (EWTG) of n points in $\mathbb{R}^d$ under uncertainty. The algorithm is similar to the algorithm of building the (deterministic) θ-graph.

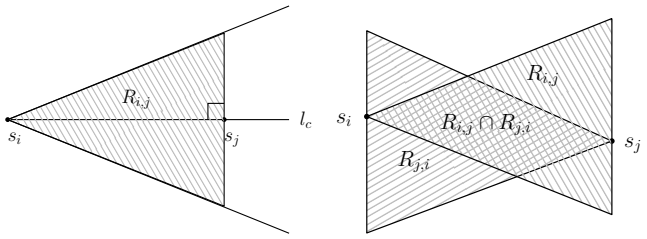
Let (S, Π) denote the uncertain points in d-dimensional space. For computing EWTG, we must calculate probability of existing each edge and multiply it to its length. In other words,

$$\mathrm{EWTG}(S) = \sum_{s_i, s_j \in S,\ i<j} |s_i s_j| \times \pi_{i,j}, \qquad (1)$$

where $\pi_{i,j}$, for all $s_i, s_j \in S$, is the probability of having the edge (s_i, s_j) in the θ-graph.

*Combinatorial and Geometric Algorithms Lab., Department of Mathematics Science, Yazd University, biranfar@stu.yazd.ac.ir, mfarshi@yazd.ac.ir, mesrikhani@stu.yazd.ac.ir

Consider two points $s_i, s_j \in S$, $i \neq j$ in $\mathbb{R}^d$. Let c be the cone with apex s_i that include s_j. We add a half plane to c, where this half plane determine by s_j and orthogonal vector $(-1) \times l_c$, we denote this region by $R_{i,j}$ (see Figure 1(a)).



(a) (b)

Figure 1: Illustration of $R_{i,j}$ and $R_{i,j} \cap R_{j,i}$ in the plane.

Observation 1 *The edge between two points s_i and s_j exists if and only if*

1. *s_i and s_j exist.*
2. *There is no point in $R_{i,j}$ or $R_{j,i}$ (see Figure 1(b)).*

More formally,

$$\pi_{i,j} = \pi_i \times \pi_j \times (\mathbf{A} + \mathbf{B} - \mathbf{C}), \tag{2}$$

where $\overline{\pi}_m = (1 - \pi_m)$, $\mathbf{A} = \prod_{s_m \in R_{i,j}} \overline{\pi}_m$, $\mathbf{B} = \prod_{s_m \in R_{j,i}} \overline{\pi}_m$ and $\mathbf{C} = \prod_{s_m \in R_{i,j} \cap R_{j,i}} \overline{\pi}_m$. Note that the last sentence subtracted because the points in $R_{i,j} \cap R_{j,i}$ considered twice in the previous terms of the equation.

By Equations (1) and (2), we have

$$\mathrm{EWTG}(S) = \sum_{\substack{s_i, s_j \in S, \\ i < j}} (|s_i s_j| \times \pi_i \times \pi_j \times (\mathbf{A} + \mathbf{B} - \mathbf{C})). \tag{3}$$

First, we describe an algorithm for computing

$$\sum_{s_i, s_j \in S,\ i<j} [|s_i s_j| \times \pi_i \times \pi_j \times [\mathbf{A} + \mathbf{B}]]. \tag{4}$$

For each cone $c \in \mathcal{C}_\kappa$, points are sorted based on the orthogonal projection onto l_c. Obviously, for each two points $s_i, s_j \in S$, if $i \geq j$, then $R_{i,j}$ is empty, so we only need to compute $R_{i,j}$, for $i < j$. Consider a cone $c_i \in \mathcal{C}_\kappa$. Let c_{i,s_j} be the cone c_i transfered to a point $s_j \in S$.

Lemma 1 *Algorithm 1 computes the first part of Equation (3) in $O(\kappa n^2)$ time and $O(n)$ space.*

Now, we describe an algorithm for computing the last part of Equation (3), i.e.

$$\sum_{s_i, s_j \in S,\ i<j} [|s_i s_j| \times \pi_i \times \pi_j \times \mathbf{C}]. \tag{5}$$

Algorithm 1:

Input : Uncertain point set (S, Π)
Output: $\sum_{\substack{s_i, s_j \in S, \\ i < j}} [|s_i s_j| \times \pi_i \times \pi_j \times [\mathbf{A} + \mathbf{B}]]$

```
1  Sum = 0;
2  for any cones c_i, 1 ≤ i ≤ κ do
3      Project all points to l_{c_i};
4      Sort the points based on its position on l_{c_i};
5      for j = 1 to n − 1 do
6          μ = π_j;
7          for k = j + 1 to n do
8              if s_k ∈ c_{s_j} then
9                  Sum = Sum + μ × π_k × |s_j s_k|;
10                 μ = μ × π̄_k;
11 return Sum;
```

Partial sum query: Given a d-dimensional array A with n entries from a semigroup, a partial sum query problem is the problem of given a d-dimensional query rectangle $\gamma = [a_1, b_1] \times \cdots \times [a_d, b_d]$, it computes

$$\sigma(A, \gamma) = \sum_{(x_1, x_2, \ldots, x_d) \in \gamma} A[x_1, x_2, \ldots, x_d].$$

The partial-sum query problem is a special case of the classic orthogonal range searching problem.

To convert computing $\prod_{s_m \in R_{i,j} \cap R_{j,i}} \overline{\pi}_m$ to a partial-sum query problem, we do the following:

1. Any region $R_{i,j} \cap R_{j,i}$ is converted to a d-dimensional rectangle.
2. The formula $\prod_{s_m \in R_{i,j} \cap R_{j,i}} \overline{\pi}_m$ converts to an operator in a semigroup.

Each cone of $\mathcal{C}_\kappa$ have $f = 2^{d-1}$ faces. Let d_i^j be the line passing through the origin that is orthogonal to the j-th face of the cone c_i. We define $\mathcal{D} = \{d_i^j : for\ 1 \leq i \leq \kappa\ and\ 1 \leq j \leq f\}$. These lines define a coordinate system that can be used to compute $\prod_{s_m \in R_{i,j} \cap R_{j,i}} \overline{\pi}_m$.

For converting each region $R_{i,j} \cap R_{j,i}$, for $1 \leq i, j \leq n$ and $i \neq j$, to a rectangle, we only need to project all of point in S onto lines in $\mathcal{D}$. New coordinates have at most

$$f \times \kappa = 2^{d-1} \times 2dm^{d-1} = O(2^d d^{(d+1)/2} (\pi/\theta)^{d-1})$$

dimensions.

We assign the value $\overline{\pi}_i$ to $A[s_i]$, for all $s_i \in S$. This partial-sum query can be answered by choosing the semigroup to be $(\mathbb{R}, \times)$, where $\times$ denotes the standard real number multiplication.

Algorithm 2:

Input : Uncertain point set (S, Π)
Output: $\sum_{\substack{s_i, s_j \in S, \\ i<j}} [|s_i s_j| \times \pi_i \times \pi_j \times \mathbf{C}]$

```
Sum = 0;
for i = 1 to n do
    Transform point s_i to the new coordinate system;
for i = 1 to n − 1 do
    for j = i + 1 to n do
        σ(A, γ) = the partial-sum for γ = R_{i,j} ∩ R_{j,i};
        Sum = Sum + |s_i s_j| × π_i × π_j × σ(A, γ) ;
return Sum;
```

Theorem 2 *[9] Given a semigroup of n variables in $\mathbb{R}^d$ and $k \geq 14^d$, there is a scheme that computes any d-dimensional rectangle query in $O(\alpha(kn, n)^d)$ time. Preprocessing this scheme is used to k cells per variable and can be constructed in time proportional to its size.*

The function $\alpha(.,.)$ is the inverse of Ackermann's function defined by Tarjan [21].

Lemma 3 *Algorithm 2 computes Equation (5) in $O(n^2\alpha(n^2, n)^{2^d d^{(d+1)/2}(\pi/\theta)^{d-1}})$ time and $O(n^2)$ space.*

Theorem 4 *Let S be a set of n uncertain points in $\mathbb{R}^d$. The expected weight of the θ-graph on S can be computed in $O(n^2\alpha(n^2, n)^{2^d d^{(d+1)/2}(\pi/\theta)^{d-1}})$ time using $O(n^2)$ space.*

3 Improve the running time of the algorithm

In Section 2, for building set $\mathcal{C}$ of cones, we consider a hypercube $H = [-1, 1]^d$ with $2d$ faces. The each face of H is partitioned to $(d-1)$-dimensional hypercubes with side length $\frac{2}{m}$, where m is $\left\lceil \sqrt{\frac{2(d-1)}{1-\cos\theta}} \right\rceil$. Each partition is called a subhypercube. These cones are not *simplicial* because of subhypercubes were having $2d-1$ vertices define them.

In this section, we want to decrease the number of dimensional partial-sum query and improve the running time complexity of Algorithm 2. First, we partition the cones to simplex-cones such that each cone has d faces. Then, we put the each region generated by the simplex-cones to a group represented by a simplex-cone.

For example, if we partition the cone c_{s_i} that contains point s_j to two cones, then the region $R_{i,j}$ is partitioned to two region $R^1_{i,j}$ and $R^2_{i,j}$.

Simply, it is observed that

$$\begin{aligned} R_{i,j} \cap R_{j,i} &= (R^1_{i,j} \cap R^1_{j,i}) \cup (R^1_{i,j} \cap R^2_{j,i}) \\ &\cup (R^2_{i,j} \cap R^1_{j,i}) \cup (R^2_{i,j} \cap R^2_{j,i}) \end{aligned}$$

and thus

$$\begin{aligned} \prod_{s_m \in R_{i,j} \cap R_{j,i}} \overline{\pi}_m = &\left[\prod_{s_m \in R^1_{i,j} \cap R^1_{j,i}} \overline{\pi}_m \right] \times \\ &\left[\prod_{s_m \in R^1_{i,j} \cap R^2_{j,i}} \overline{\pi}_m \right] \times \left[\prod_{s_m \in R^2_{i,j} \cap R^1_{j,i}} \overline{\pi}_m \right] \times \\ &\left[\prod_{s_m \in R^2_{i,j} \cap R^2_{j,i}} \overline{\pi}_m \right]. \end{aligned}$$

Generally, a d-dimensional hypercube can be triangulation into $d!$ d-simplices with disjoint interiors [16].

Let $V = \{v^0, v^1, \ldots, v^\beta\}$, $0 \leq \beta \leq d$, be a set of $\beta+1$ points in $\mathbb{R}^d$. If the vectors $v^i - v^0$, $1 \leq i \leq \beta$, are linearly independent, then the convex hull of V is called a β-simplex.

Consider the collection $\mathcal{C}_\kappa = \{c_1, c_2, \ldots, c_k\}$ of cones in Section 2. Each cone c_i in $\mathcal{C}_\kappa$ generated by V_i, where V_i is the vertex set of a $(d-1)$-dimensional hypercube that is contained in one of the $2d$ hypercube $x_1 = 1$, $x_1 = -1$, $x_2 = 1$, $x_2 = -1$, ..., $x_d = 1$, $x_d = -1$. This hypercube can be triangulated into $(d-1)!$ many $(d-1)$-simplices $\Delta^1_i, \Delta^2_i, \ldots, \Delta^{(d-1)!}_i$, that are all contained in the same hyperplane as V_i. We define

$$\mathcal{C}_{\kappa_s} = \{c^j_i : for\ 1 \leq i \leq \kappa, 1 \leq j \leq (d-1)!\},$$

where $\kappa_s = 2d! \lceil \sqrt{2(d-1)/(1-\cos\theta)} \rceil^{d-1}$. Since $\kappa = d^{(d+1)/2}(\pi/\theta)^{d-1}$, we have

$$\kappa_s \leq \kappa d^{d-1} = d^{(3d-1)/2}(\pi/\theta)^{d-1}.$$

The collection $\mathcal{C}_{\kappa_s}$ consist κ_s simplicial cones that cover $\mathbb{R}^d$, and that all have their apex at the origin. If d is a constant, then $\mathcal{C}_\kappa$ can be constructed in $O(1/\theta^{d-1})$ time and consists of $\kappa = O(1/\theta^{d-1})$ cones with disjoint interiors [18].

Since the collection $\mathcal{C}_{\kappa_s}$ has at most $d^{(3d-1)/2}(\pi/\theta)^{d-1}$ cones, if we only consider the collection of the vector perpendicular to every face, then we have at most $(d^{(3d-1)/2}(\pi/\theta)^{d-1})^2$ different group regions $R^k_{i,j} \cap R^l_{j,i}$, for all $s_i, s_j \in S, i < j$ and $1 \leq k, l \leq (d-1)!$.

Observation 2 *Let s_i and s_j be two points in S. We have*

$$\prod_{s_m \in R_{i.j} \cap R_{j,i}} \overline{\pi}_m = \prod_{k=1}^{(d-1)!} \left[\prod_{l=1}^{(d-1)!} \prod_{s_m \in R^k_{i.j} \cap R^l_{j,i}} \overline{\pi}_m \right].$$

Similar to Section 2, we will convert $\prod_{s_m \in R^k_{i,j} \cap R^l_{j,i}} \overline{\pi}_m$ to a partial-sum query. First, all regions $R^k_{i,j} \cap R^l_{j,i}$, for all $s_i, s_j \in S$, $i < j$ and $1 \leq k, l \leq (d-1)!$, are grouped base of lines through the origin that are orthogonal to

its face, we denote the i-th group of regions by G_i and the number of groups by q. Next, for each region, let $D_1, D_2, \ldots, D_f$ be the lines through the origin that are orthogonal to the face of this region, where f is equal to $2d$. These lines define the coordinate system that can be used to compute $\prod_{s_m \in R_{i,j} \cap R_{j,i}} \overline{\pi}_m$.

We define the function $Source()$ such that

$$Source(R^k_{i,j} \cap R^l_{j,i}) = R_{i,j} \cap R_{j,i},$$

for any $1 \leq i < j \leq n$ and $1 \leq k, l \leq (d-1)!$. We preprocess every group for computing partial-sum query.

Lemma 5 *One can compute Equation (5) in $O(n^2\alpha(n^2,n)^{2d})$ time and $O(n^2)$ space.*

From Lemma 1 and Lemma 5, the following theorem is obtained.

Theorem 6 *Let S be a set of n uncertain points in $\mathbb{R}^d$. The expected weight of the θ-graph can be computed in $O(n^2\alpha(n^2,n)^{2^d})$ time and $O(n^2)$ space.*

4 Conclusion

In this paper, we studied the θ-graph of a set of n points in uncertain points in tuple model. We proposed an algorithm to compute the expected weight of θ-graph in $O(n^2\alpha(n^2,n)^{2d})$ time using $O(n^2)$ space, where θ and d are constants. There are some interesting problems to be pursued. One of them is computing the expected weight of other spanners on uncertain points, since the θ-graph is a t-spanner for a t which is a function of θ.

References

[1] P. Afshani, P. K. Agarwal, L. Arge, K. G. Larsen, and J. M. Phillips. (approximate) uncertain skylines. *Theory of Computing Systems*, 52(3):342–366, 2013.

[2] P. K. Agarwal, S.-W. Cheng, Y. Tao, and K. Yi. Indexing uncertain data. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 137–146. ACM, 2009.

[3] P. K. Agarwal, S.-W. Cheng, and K. Yi. Range searching on uncertain data. *ACM Transactions on Algorithms (TALG)*, 8(4):43, 2012.

[4] P. K. Agarwal, S. Har-Peled, S. Suri, H. Yıldız, and W. Zhang. Convex hulls under uncertainty. In *European Symposium on Algorithms*, pages 37–48. Springer, 2014.

[5] P. K. Agarwal, N. Kumar, S. Sintos, and S. Suri. Range-max queries on uncertain data. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 465–476. ACM, 2016.

[6] C. C. Aggarwal. *Managing and Mining Uncertain Data*. Springer Publishing Company, Incorporated, 2009.

[7] C. C. Aggarwal and S. Y. Philip. A survey of uncertain data algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering*, 21(5):609–623, 2009.

[8] K. M. Alzoubi, X.-Y. Li, Y. Wang, P.-J. Wan, and O. Frieder. Geometric spanners for wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):408–421, 2003.

[9] B. Chazelle and B. Rosenberg. Computing partial sums in multidimensional arrays. In *Proceedings of the fifth annual symposium on Computational geometry*, pages 131–139. ACM, 1989.

[10] K. L. Clarkson. Approximation algorithms for shortest path motion planning. In *STOC'87: Proceedings of the 19th Annual ACM Symposium on Theory of Computing*, pages 56–65, 1987.

[11] G. Cormode and A. McGregor. Approximation algorithms for clustering uncertain data. In *Proceedings of the twenty-seventh ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 191–200. ACM, 2008.

[12] M. Fischer, T. Lukovszki, and M. Ziegler. Geometric searching in walkthrough animations with weak spanners in real time. In *Algorithms — ESA' 98*, pages 163–174. Springer, 1998.

[13] O. Goldreich and D. Ron. Approximating average parameters of graphs. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 363–374. Springer, 2006.

[14] J. M. Keil. Approximating the complete Euclidean graph. In *SWAT'88: Proceedings of the 1st Scandinavian Workshop on Algorithm Theory*, volume 318 of *Lecture Notes in Computer Science*, pages 208–213. Springer-Verlag, 1988.

[15] J. M. Keil and C. A. Gutwin. Classes of graphs which approximate the complete Euclidean graph. *Discrete and Computational Geometry*, 7:13–28, 1992.

[16] H. W. Kuhn. Some combinatorial lemmas in topology. *IBM Journal of research and development*, 4(5):518–524, 1960.

[17] P. Morin and S. Verdonschot. On the average number of edges in theta graphs. In *Proceedings of the Meeting on Analytic Algorithmics and Combinatorics*, page 121–132, USA, 2014.

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

[19] D. Russel and L. J. Guibas. Exploring protein folding trajectories using geometric spanners. *Pacific Symposium on Biocomputing*, pages 40–51, 2005.

[20] S. Suri, K. Verbeek, and H. Yıldız. On the most likely convex hull of uncertain points. In *European Symposium on Algorithms*, pages 791–802. Springer, 2013.

[21] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM (JACM)*, 22(2):215–225, 1975.

[22] W. Zhang. *Geometric computing over uncertain data*. PhD thesis, Duke University, 2015.

Surrounded k-Center and Applications in MapReduce

Sepideh Aghamolaei* Mohammad Ghodsi†

Abstract

We are given a set of points P and a boundary B in the Euclidean plane. The goal of the *surrounded k-center* problem is to find a subset of k input points as centers C such that the maximum distance from an input point (in P) to its nearest center or its nearest boundary point/segment (in C or B) is minimized.

By a simple reduction from k-center, this problem is NP-hard and there is a lower bound 1.822 on its approximation factor. We give several 2-approximation algorithms for this problem. Based on surrounded k-center, we define the persistent clustering as a version of k-center, where points have timestamps and each point can be assigned to centers with smaller or equal timestamps. For this problem, we give a 2-approximation sequential algorithm and a 4-approximation MapReduce algorithm.

1 Introduction

k-center is a classic computational geometry problem. The goal of metric k-center is to choose a subset of size k called C from a set of input points P such that the maximum distance from each point to its nearest center is minimized. Metric k-center is NP-hard and it has tight 2-approximation algorithms [13]. The lower bound on the approximation factor of Euclidean k-center is 1.822 [6]. A variation of k-center is k-line center, where the goal is to find k lines (or cylinders in higher dimensions) as centers such that the distance from each point to its nearest line is minimized [1].

MapReduce is a distributed and parallel framework for data processing. In MapReduce, data is distributed among a set of machines. Each machine processes its data independently during each parallel round, and the machines communicate after each round. Among famous theoretical MapReduce models are MRC [10] and MPC [3]. MRC limits the memory of each machine m and the number of machines L to be sublinear in the input size n and has a poly-logarithmic number of rounds. MPC adds the following restrictions to MRC: $mL = O(n)$ and the round complexity must be $O(1)$. Semi-group operations take $O(\log_m n)$ MPC rounds [8].

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

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

Metric k-center can be 4-approximated in $O(1)$ rounds in MPC [11]. Euclidean k-center has $(2+\epsilon)$-approximation algorithms with $O(1)$ rounds in MPC [4, 2]. In [9], the radius of remote-edge diversity with $(k+1)$ vertices is also the radius of metric k-center, so the 3-approximation algorithm works for both problems. However, the algorithm is existential and does not give the set of k centers corresponding to that radius.

Problem Definition

We define two new variations of k-center.

Given a set of points P and a boundary B, the *surrounded k-center* problem aims to find a subset C of k points (centers) from P, such that the distance from each point of P to its nearest center in C or to the boundary B is minimized. We assume B is a set of $|B|$ curves of constant complexity, and the distance from a point to each curve of the boundary can be computed in $O(1)$ time. An example of 2-center is shown in Figure 1.

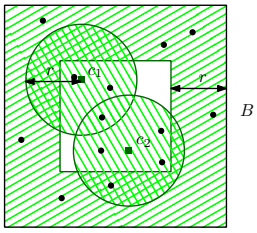


Figure 1: An example of surrounded 2-center. The shaded region is the area covered by the centers $C = \{c_1, c_2\}$ and the boundary B, with radius r.

Given a time-stamped point set P, $T : P \to \mathbb{N}$, *persistent k-center* finds k centers $C \subset P$ such that the maximum distance from each point of P to its nearest center in C with a smaller or equal time-stamp is minimized. Formally, the objective function is:

$$\min_{\substack{C\subset P,\\ |C|=k}} \max_{p\in P} \min_{\substack{c\in C,\\ T(c)\le T(p)}} d(p,c).$$

Existing algorithms for timestamped data do not guarantee persistency. Incremental clustering [5] allows the existing points to be assigned to new centers, and is therefore different. Also, incremental clustering uses competitive ratio instead of approximation factor. Recomputing the solution after a set of points has been

added might lead to a new center for an existing point, which contradicts the persistency of the clustering.

Contributions

We design a set of algorithms for surrounded k-center. The time complexities and the approximation factors of these algorithms are summarized in Table 1.

Approx.	Time	Reference
≥ 2	$O(poly(n))$	Theorem 1
2	$O(n^2 \log n)$	Algorithm 1
2	$O(n^2)$	Algorithm 2
$2^{\dagger}$	$O(nk \log n)^*$	Theorem 6
$2+\epsilon$	$O(\frac{n^2}{\epsilon})$	Algorithms 1 and 2 using [2]

Table 1: Algorithms for surrounded k-center. Here, n denotes the input size. The time complexity marked with a * is reported assuming $k = o(\sqrt{n}), |P| = O(n)$, and $|B| = O(n)$. The algorithm marked with † only works for polygonal boundaries.

Also, for the case where there are constraints for assigning a point to a center or the boundary, we prove our parametric pruning algorithm (Algorithm 1) still works.

We give a 2-approximation for persistent clustering and extend it to MPC, where we get a 4-approximation in $O(1)$ rounds.

2 Hardness

The surrounded k-center problem can be seen as an instance of the k-center problem where the distance function is the minimum of the distance from each input point to its nearest boundary point and the distance to its nearest center. We define this new distance function d' for any pair of points x and y as follows:

$$d'(x, y) = \min(d(x, y), d(x, B)),$$

where d is the Euclidean distance and B denotes the boundary.

When $d(x, B) \neq d(y, B)$, the distance function d' is non-symmetric. Therefore, the proof of metric k-center (including Euclidean k-center) no longer applies to this case and we need to prove the approximation factor.

Reduction from k-Center

Here, we give an approximation factor preserving reduction from k-center to surrounded k-center.

An instance of k-center can be converted into an instance of the surrounded k-center by computing the minimum enclosing ball of the points and multiplying its radius by 4. If the boundary is restricted to be polygonal, the bounding box of this ball can be used. By definition, any solution of the k-center problem is also a solution of the surrounded k-center problem.

The distance between any pair of points inside the minimum enclosing ball is at most $2R$ and the distance between each boundary point to an input point is at least $3R$. Therefore, no point is assigned to the boundary in the optimal solution of the surrounded k-center, which means its solution is a valid solution of k-center and an optimal solution because it has the minimum cost among all solutions.

This reduction and the previous results on Euclidean k-center [6] give us the following corollary:

Corollary 1 *Surrounded k-center is NP-hard, and the lower bound on its approximation factor is* 1.822.

3 Approximate Surrounded k-Center

3.1 A Parametric Pruning Algorithm

In this section, we propose an approximation algorithm based on the parametric pruning for k-center [13].

Algorithm 1 Surrounded k-Center

Input: A point set P, a boundary B, an integer k
Output: A set of at most k centers

```
1: D = {d(p, B) | ∀p ∈ P}
2: D = D ∪ {d(p, q) | p, q ∈ P}
3: Sort D in ascending order
4: for r ∈ D do
5:     P_r = P \ {p | d(p, B) ≤ 2r}
6:     E_r = {(p, q) | d(p, q) ≤ 2r, p, q ∈ P_r}
7:     Build the graph G_r = (P_r, E_r)
8:     I_r = a maximal independent set of G_r
9:     if |I_r| ≤ k then
10:        return I_r
```

Theorem 2 *Algorithm 1 has approximation factor* 2 *for surrounded k-center.*

Proof. The optimal cost is the distance between a point of P to its closest point from B or P. Removing the points near the boundary adds a factor 2 to the optimal radius. I_r has size at most as much as the dominating set of the graph with edges of length at most r on P, since all the points inside a cluster centered at a vertex of the dominating set have distance at most $2r$ from each other. □

Theorem 3 *The time complexity of Algorithm 1 is $O(n^2 \log n)$, assuming $|P| + |B| = n$.*

Proof. Computing the distances between all the points and the boundary takes $|P|$ times the time required for computing the distance from a point to the boundary ($O(|B|)$. So, it takes $O(|P||B|) = O(n^2)$ time.

Sorting $\Theta(n^2)$ distances takes $O(n^2 \log n)$ time. Building G_r by pruning away the edges of the complete graph on P with length more than $2r$ takes $O(n^2)$ time. Computing a maximal independent set using a greedy algorithm takes linear time in the number of vertices, so, it takes $O(n^2)$ time. The overall time complexity of the algorithm is, therefore, $O(n^2 \log n)$. □

3.2 A Greedy Algorithm

We start by finding the point with maximum distance from its nearest boundary point, inside the boundary. Then, at each step, we add the farthest point to the previously chosen points and the boundary, until we have k centers. The algorithm in this section is based on the algorithm of [7] for k-center.

Algorithm 2 Surrounded k-Center (Greedy)

Input: A point set P, a boundary B, an integer k
Output: A set of at most k centers
1: $c_1 = \arg\max_{p \in P} \min_{b \in B} d(b, p)$
2: **for** $i = 2, \cdots, k$ **do**
3: find the farthest point in P to $\{c_j\}_{j=1}^{i-1}$ and B
4: **return** $\{c_j\}_{j=1}^{k}$

Theorem 4 *The time complexity of Algorithm 2 is $O(kn^2)$, where $n = |P| + |B|$.*

Proof. In Line 1, for each of the input points, we compute their distance to every other point and boundary edge. This takes $O(|P||B|)$ time. Finding the point with maximum distance to the previously chosen centers and the boundary takes $O(|P|(|B| + k))$ time. So, the overall time complexity is $O(|P|k(|B| + k)) = O(k|P||B|) = O(kn^2)$. □

Theorem 5 *Algorithm 2 gives a 2-approximation for surrounded k-center.*

Proof. The greedy algorithm like the parametric pruning algorithm (Algorithm 1) computes an independent set, except that it chooses the centers from the points with maximum distance the previously chosen points and the boundary. Since any independent set will find the solution, so does the one computed by this greedy strategy. □

3.3 The Case of Polygonal Boundaries

For polygonal boundaries, using the segments Voronoi diagram [12] of the boundary segments, we can find the nearest segment to each point in $O(\log n)$ time.

Theorem 6 *The time complexity of Algorithm 2 using a points Voronoi diagram on centers and a segments Voronoi diagram on the boundary is $O(nk \log |B| + k^2 \log k + |P| \log k)$.*

Proof. Constructing a segments Voronoi diagram takes $O(|B| \log |B|)$ time. Querying the segments Voronoi diagram for each point for choosing each center takes $O(k|P| \log(|B|)$ time. Finding the nearest center to each point takes $O(\log k)$ time, using a points Voronoi diagram which requires $O(k \log k)$ time. Since we build a new Voronoi diagram after adding each center, the overall time complexity of building points Voronoi diagrams is $O(k^2 \log k)$. Using this Voronoi diagram takes $O(|P| \log k)$ time. The sum of these time complexities gives the bound in the theorem's statement. □

3.4 Constrained Surrounded k-Center

The constrained version of the surrounded k-center puts restrictions on the assignment of a point to a center or the boundary. We assume a graph defining these constraints is given as a part of the input.

In Figure 2, we have visibility constraints, i.e. there must be a direct line of sight between a center or a point of the boundary and the point that it is covering.

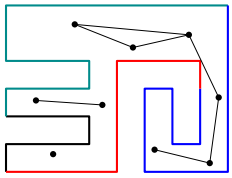


Figure 2: Surrounded k-center with constraints.

Algorithm 1 can be modified to solve this problem by only considering the valid edges.

4 Persistent Clustering

We give an algorithm for persistent k-center (Algorithm 3) based on the parametric pruning for k-center [13], and compute the ratio between the cost of persistent k-center and k-center.

Algorithm 3 updates the set of centers, assuming a new batch of points P is given, the previous radius was r_0 and the previous set of centers was C. Reducing the number of centers is done by computing a persistent k-center of centers (C). Since there are more than k centers, r_0 is less than the optimal radius, so the approximation factor still holds.

Theorem 7 *Algorithm 3 is a 2-approximation.*

Proof. Since D contains all the pairwise distances, then the optimal radius is a subset of D.

Consider an optimal cluster. If the center of this cluster is in C, then we assign the points of the cluster either in Line 4 or Line 7. If it is assigned to a cluster, then the distance from this point to its nearest center is at most r. Since we use $2r$ in Line 8, the resulting cluster covers all the points of that optimal cluster. □

Algorithm 3 Persistent k-Center (Adding Centers)

Input: A point set P, a set of centers C, an integer k, a constant r_0
Output: A set of at most k centers

```
B = the set of disks of radii r_0 centered at C
D = {d(p, B)|∀p ∈ P}
D = D ∪ {d(p, q)|p, q ∈ P}
Remove any point of P with distance ≤ r_0 from C
Sort D in ascending order
for r ∈ D do
    P_r = P \ {p|d(p, B) ≤ 2(r − r_0)}
    E_r = {(p, q)|d(p, q) ≤ 2r, p, q ∈ P_r}
    Define the graph G_r = (P_r, E_r)
    I_r = a maximal independent set of G_r
    if |I_r| ≤ k then
        return I_r
```

Theorem 8 *Algorithm 3 takes $O(n^3)$ time.*

Proof. Set D has size $O(n^2)$, so sorting D takes $O(n^2 \log n)$ time. Computing a maximal independent set of size k, takes $O(n)$ time. Since this computation is repeated D times, the overall time complexity of the algorithm is $O(n^3)$. □

4.1 Persistent Euclidean k-Center in MapReduce

Here, we give a $O(1)$ round algorithm for this problem.

Algorithm 4 Persistent k-Center

Input: A point set P, a set of centers C, an integer k, a constant r_0
Output: A set of at most k centers

```
send C and r_0 to all machines
compute a persistent k-center in each machine (using Algorithm 3)
send the centers from each machine to the first machine
compute a persistent k-center in the first machine
return the centers from the previous step
```

Theorem 9 *Algorithm 4 computes a 4-approximation persistent k-center in MPC.*

Proof. Assume c is the center covering a point p, and t is the center chosen in its local machine. Then, using triangle inequality: $d(p, c) \leq d(p, t) + d(t, c) \leq 2r + 2r = 4r$. For any point assigned to the boundary, its distance is at most $2r$, based on Theorem 7. □

Theorem 10 *Algorithm 4 takes $O(\log_m n)$ rounds.*

Proof. Sending data of size $|C|$ to all machines takes $O(\log_m |C|)$ rounds, using semi-group operations. Computing a solution in each machine takes 1 round. Sending the solutions to the first machine takes $O(\log_m(kL))$ rounds, since the union of the solutions must be computed. Computing the solution in the first machine takes 1 round. □

References

[1] P. K. Agarwal, C. M. Procopiuc, and K. R. Varadarajan. Approximation algorithms for k-line center. In *Annu. European Sympos. Algorithms*, pages 54–63. Springer, 2002.

[2] S. Aghamolaei and M. Ghodsi. A composable coreset for k-center in doubling metrics. *arXiv preprint arXiv:1902.01896*, 2019.

[3] P. Beame, P. Koutris, and D. Suciu. Communication steps for parallel query processing. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI Sympos. Princ. Database Syst.*, pages 273–284. ACM, 2013.

[4] M. Ceccarello, A. Pietracaprina, and G. Pucci. Solving k-center clustering (with outliers) in mapreduce and streaming, almost as accurately as sequentially. *Proceedings of the VLDB Endowment*, 12(7):766–778, 2019.

[5] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004.

[6] T. Feder and D. Greene. Optimal algorithms for approximate clustering. In *Proceedings of the 20th Annu. ACM Sympos. Theory Comput.*, pages 434–444. ACM, 1988.

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

[8] M. T. Goodrich, N. Sitchinava, and Q. Zhang. Sorting, searching, and simulation in the mapreduce framework. In *Annu. Internat. Sympos. Algorithms Comput.*, pages 374–383. Springer, 2011.

[9] P. Indyk, S. Mahabadi, M. Mahdian, and V. S. Mirrokni. Composable core-sets for diversity and coverage maximization. In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGAI Sympos. Princ. Database Syst.*, pages 100–108. ACM, 2014.

[10] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for mapreduce. In *Proceedings of the 21st ACM-SIAM Sympos. Discrete Algorithms*, pages 938–948. SIAM, 2010.

[11] G. Malkomes, M. J. Kusner, W. Chen, K. Q. Weinberger, and B. Moseley. Fast distributed k-center clustering with outliers on massive data. In *Advances in Neural Information Processing Systems*, pages 1063–1071, 2015.

[12] C. D. Toth, J. O'Rourke, and J. E. Goodman. *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 2017.

[13] V. V. Vazirani. *Approximation algorithms*. Springer Science & Business Media, 2013.