February 19, 2019
Sharif University of Technology
Tehran, Iran

2nd Iranian Conference on

Computational Geometry

ICCG 2019

Department of Mathematical Sciences
Sharif University of Technology
Azadi Street
Tehran, Iran, 14588-89694

iccg@sharif.ir

+98 (21) 66005217

iccg.math.sharif.ir

Contents

1 Foreword **3**

2 Conference Committees **5**
- General Chair . . . 5
- Program Committee . . . 5
- Invited Speaker . . . 6
- Local Organizers . . . 6
- Executive Committee . . . 6

3 Conference Program **7**
- Tuesday February 19 . . . 7
  - Invited lecture: External Labeling: Models, Applications, and Geometric Algorithms . . . 7
- Session 1 . . . 8
  - The p-center problem under Uncertainty . . . 9
  - A Plane Region-Fault Tolerant 2.095-Spanner for Points in Convex Position . . . 13
  - Transmission radius games on wireless networks . . . 17
  - On the Maximum Triangle Problem . . . 21
- Session 2 . . . 24
  - Recognizing Visibility Graphs of TINs . . . 25
  - Counting Closed Billiard Paths . . . 32
  - Solving the convex hull problem progressively in the external memory model . . . 36
  - Some Results on the Fewest Net of Simplicial Polyhedra . . . 41
  - Local Geometric Spanners . . . 45
- Session 3 . . . 49
  - Approximate Curve-Restricted Simplification of Polygonal Curves . . . 50
  - On Maximum-Weight Minimum Spanning Tree Color-Spanning Set . . . 54
  - Competitive Strategy for Walking in Streets for an Empowered Simple Robot . . . 59
  - Approximate Discontinuous Trajectory Hotspots . . . 63

Foreword

The second Iranian Conference on Computational Geometry was held on February 19, 2019 at the Mathematical Sciences Department of Sharif 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 thirteen refereed papers that were presented during the conference, in three sections. I would like to thank pc chairs Mohammad Ali Abam and Mohammad Farshi, all the pc members, and members of the local organizing committee. I also want to thank the sponsors: Sharif University of Technology for financial supports and Islamic World Science Citation Center (ISC) for indexing the conference (#ISC 97190-21804).

Alireza Zarei

Conference Committees

General Chair

- Alireza Zarei - Sharif University of Technology

Program Committee

- Mohammad Ali Abam, Sharif University of Technology (co-chair)
- Ahmad Biniaz, University of Waterloo, Canada
- Mohammad Farshi, Yazd University (co-chair)
- Mohammad Ghodsi, Sharif University of Technology, Iran
- Joachim Gudmundsson, The university of Sydney, Australia
- Shahin Kamali, University of Manitoba, Canada
- Philipp Kindermann, University of Waterloo, Canada
- Matias Korman, Tufts university, USA
- Maarten Löffler, Utrecht university, Netherlands
- Mehran Mehr, Eindhoven University of Technology, Netherlands
- Saeed Mehrabi, Carleton university, Canada
- Ali Mohades Khorasani, Amirkabir University of Technology, Iran
- Morteza Monemizadeh, Amazon, USA
- Martin Nöllenburg, Technische Universität Wien, Austria
- Jeff Philips, University of Utah, USA
- Zahed Rahmati, Amirkabir University of Technology, Iran
- Alireza Zarei, Sharif University of Technology, Iran
- Hamid Zarrabi-Zadeh, Sharif University of Technology, Iran

Invited Speaker

- Martin Nöllenburg, Technische Universität Wien, Austria

Local Organizers

- Alireza Zarei - Sharif University of Technology
- Hossein Boomari - Sharif University of Technology

Executive Committee

- Hossein Alimadad (Website) - Sharif University of Technology
- Nima Behrang - Sharif University of Technology
- Amirkasra Jalaldoost - Sharif University of Technology
- Ehsan Karimi Ara - Sharif University of Technology
- Zahra Kyali - Sharif University of Technology
- Hossein Mahdavi Pour - Sharif University of Technology
- Mojtaba Ostovari - Sharif University of Technology

Conference Program

Tuesday February 19

Invited lecture: External Labeling: Models, Applications, and Geometric Algorithms

Session 1

The P-center problem under Uncertainty

Homa Ataei Kachooei* Mansoor Davoodi† Dena Tayebi ‡

Abstract

The problem of P-center asks finding the location of P facilities among a set of n demand points such that the maximum distance between any demand point and its nearest facility is minimized. In this paper, we study the P-center problem under uncertainty, that is, the demand set is given as a set of regions, e.g., n disks. We focus on *Max-P-center* and *Min-P-center* problems as the natural extensions of this problem in uncertainty context. In these problems, we are interested in computing the upper and lower bounds on the P-center solution for the regions. Precisely, we wish to place a point in each region such that the solution of P-center problem for that *placement* is maximized or minimized. We present a $\frac{1}{2}$-approximation and a parameterized approximation algorithm for the Max-P-center and a parameterized approximation algorithm for the Min-P-center problem.

keywords: Facility location, P-center, Uncertainty, *Min-P-center*, *Max-P-center*.

1 Introduction

The P-center problem is a classical facility location problem; given n demand points (customers), the goal is to place P facilities (*centers*) among them such that the maximum distance between any demand point and its nearest center is minimized. It is known that the P-center problem is NP-hard for both the Euclidean and Manhattan metrics [11]. Some special cases of the P-center problem are solvable in polynomial time such as the smallest enclosing circle and its weighted demand set variations [4, 8, 10], the two-center problem [1], the rectilinear three-center problem [6], the P-center problem on trees [12, 2] and the P-center problem in one dimension [13]. The P-center problem has been also studied under uncertainty; the location of the demand points and the weight of demands may be considered as the uncertainty sources. Further, in the graph variation of the problem, the location and the weight of vertices and the length of edges can be considered uncertain. There are different approaches for modeling uncertainty.

Uncertainty can be modeled by continuous or discrete sets. In continuous model, uncertainty is modeled by some regions or intervals, while in the discrete model, it is modeled by some discrete sets. Foul [5] studied the Euclidean 1-center problem under uncertainty in which each demand has a uniform distribution in a given rectangle in the plane. The P-center problem was studied in one dimension such that the location of each demand is uncertain [14]. Uncertainty is modeled using m possible locations with a probability distributed function for each demand point. For this problem an $O(mn \log mn + n \log p \log n)$ time algorithm was presented. Also, the 1-center problem in one dimension, the 1-center problem on a tree and the rectilinear 1-center problem in the plane were studied under this model of uncertainty [15, 17, 16]. Löffler and van Kreveld [9] presented efficient algorithms for 1-center problem when the uncertainty regions are modeled by squares or disks. The goal is finding a point from each region such that the *Smallest Enclosing Circle* (SEC) of them is minimized or maximized.

The problems of P-center under uncertainty can be formally defined as follows. Let $D = \{d_1, d_2, ..., d_n\}$ be a set of n disks in the plane and $I = \{p_1, p_2, ..., p_n\}$ be a *placement*, where $p_i \in d_i$ for $i = 1, 2, ..., n$. Now, I is an input or *instance* of the (certain) P-center problem. Let $p-center(I)$ be the optimal solution of the P-center problem for I. That is, if $C = \{c_1, c_2, ..., c_p\}$ is a solution (set of P centers) for P-center problem, then

$$p - center(I) = \min_{C} \max_{p_i \in I} dis(p_i, C),$$

where $dis(p_i, C)$ is the distance between p_i and the nearest center in C. Therefor, Max-P-*center* and Min-P-*center* are the problems of finding the crucial instances I^{max} and I_{min} such that

$$I^{max} : \max_{I} p - center(I).$$
$$I^{min} : \min_{I} p - center(I).$$

In this paper, we consider both Max-P-center and Min-P-center problems. In Section 2, we present a simple $\frac{1}{2}$-approximation algorithm for the Max-P-center problem when the regions are disjoint disks or a set of discrete points. Also, we present a $1 - \frac{2}{k+4}$-approximation algorithm when the regions are k-*separable*. In Section 3, we consider the Min-P-center problem and present a $1 + \frac{2}{k}$-approximation algorithm when the regions of uncertainty are k-separable disks or discrete points.

*ataei.homa@gmail.com

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

‡denatayebi@yahoo.com

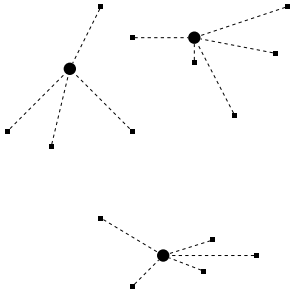


Figure 1: Clustering for three facilities

2 Max-P-center Problem

In this Section, we focus on the *Max*-P-*center* problem when the regions of uncertainty are modeled as disjoint disks or discrete sets. We present a $\frac{1}{2}$-approximation algorithm and a parameterized algorithm for the special case of the Max-P-center problem when the regions are *well-separable*.

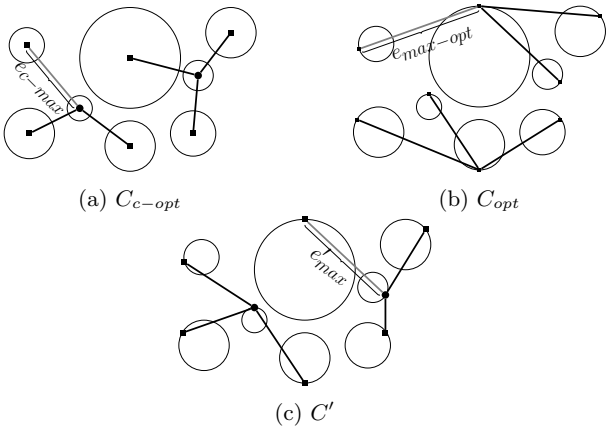
Through this paper, we point out an assignment of the demand points to the facility centers as a *clustering*. An example of a clustering with three clusters is shown in Figure 1. Two clustering have the same *topology* if the assignments of demand points to the centers are the same.

Theorem 1 *Let D be a set of disjoint disks as the input of the Max-P-center problem. The algorithm that chooses disks centers has an approximation ratio of at most $\frac{1}{2}$.*

Proof. We use the same approach of Theorem 1 in [3] for proving the correctness of the theorem. As such, We consider three clusters C_{opt}, C_{c-opt} and C'. C_{opt} is the solution of Max-P-center problem, e.g., I^{max}, C_{c-opt} is the solution of (certain) P-center problem for the center of disks, and C' is the cluster which have the same topology with C_{c-opt} and the same location with I^{max}. We compare C_{c-opt} and C_{opt} using C'. Since, in the P-center problem the goal is minimizing the maximum length edge in the cluster. Let $e_{max-opt}$ be the maximum distance between any demand point and its assigned center in C_{opt}. Actually $e_{max-opt}$ is the longest edge in the cluster C_{opt}. Similarly, let e_{c-max} and e'_{max} be the logtest edge in C_{c-opt} and C', respectively. Figure 2 illustrates clusters C_{c-opt}, C_{opt} and C'. Due to the location of demand points in C' and C_{opt} are the same, thus

$$e_{max-opt} \leq e'_{max}. \tag{1}$$

Since C_{c-opt} and C' have the same topology, if the location of the demand points change anywhere on disks, the length of each edge, increases at most the amount of sum of the radius of two (disjoint) disks. So,



(a) C_{c-opt} (b) C_{opt}

(c) C'

Figure 2: Three kinds of clusters for the Max-P-center problem

$$e'_{max} \leq 2e_{c-max}. \tag{2}$$

According to inequalities 1 and 2, we have

$$e_{max-opt} \leq 2e_{c-max}. \tag{3}$$

We compare the corresponding edges in C_{c-opt} and C'. Note that, the largest edges in these two clusters may not be the same in their clustering. It means, the largest edge in C' is between disks D_i and D_j, however, it is between two other different disks in C_{c-opt}. We claim that inequality 2 is holds even for this case. Suppose that in C_{c-opt}, e is corresponding edge with e'_{max} in C'. So,

$$e'_{max} \leq 2e. \tag{4}$$

e_{c-max} is the largest edge and e is an edge in C_{c-opt}. Thus,

$$e \leq e_{c-max}. \tag{5}$$

According to inequalitys 4 and 5

$$e'_{max} \leq 2e_{c-max}. \tag{6}$$

Therefor the inequality 3 is established even for this case. Consequently, the proof is complete. □

Theorem 1 states that the set of center of disks is $\frac{1}{2}-$approximation for I^{max} when the disks are disjoint. In the following, we show that there is a nice relationship between the approximation ratio of such a solution and *separability factor* of the disks by proposing a parametrized approximation ratio.

Let r_{max} be the radius of the largest disk. A set of disks D is called *k-separable*, if the minimum distance between any pair of disks in D is at least $k.r_{max}$. For an input such as D, *separability* is the maximum k such that D is k-separable.

Theorem 2 *Let D be a set of k-separable disks as the input of the Max-P-center problem. The algorithm that places the center of disks, has $1 - \frac{2}{k+4}-$ approximation ratio.*

Proof. This proof is similar to the proof of Theorem 1. We consider C_{c-opt}, C' and C_{opt} as before. Suppose e' is an arbitrary edge in C', and d_i and d_j are two disks connecting with e'. Let r_i and r_j be the radius of d_i and d_j, respectively, and l be the distance between d_i and d_j. Also, let e be the corresponding edge with e' in C_{c-opt} whose weight is $l+r_i+r_j$. The weight of e' is at most $l + 2r_i + 2r_j$. So, the weight of an edge in C_{c-opt} to the weight of its corresponding edge in C' is at least:

$$\frac{e}{e'} = \frac{l + r_i + r_j}{l + 2r_i + 2r_j} \geq \frac{k.r_{max} + r_i + r_j}{k.r_{max} + 2r_i + 2r_j}$$

$$\geq \frac{k.r_{max} + r_{max} + r_{max}}{k.2r_{max} + 2r_{max} + r_{max}} = \frac{k+2}{k+4}. \quad (7)$$

This inequality holds for any edge in C_{c-opt}. So, regarding the inequality 7,

$$e_{c-max} \geq \frac{k+2}{k+4} e'_{max}, \quad (8)$$

where e_{c-max} is the edge with maximum weight in C_{c-opt} and e'_{max} is the edge with maximum weight in C'. C_{opt} and C' have the same demand points, so,

$$e_{max-opt} \leq e'_{max}, \quad (9)$$

where $e_{max-opt}$ is the edge with maximum weight in C_{opt}. According to inequalities 8 and 9:

$$e_{c-max} \geq \frac{k+2}{k+4} e_{max-opt}. \quad (10)$$

Therefor, the set of center of the disks is $\frac{k+2}{k+4} = 1 - \frac{2}{k+4}-$ approximation solution. □

Finally, we show that the idea behind of the parametrized approximation algorithm can be applied for the uncertain demand regions modeled by discrete sets with preserving the approximation ratio . We are given a set of points for each uncertainty region, e.g., $S = \{S_1, S_2, ..., S_n\}$, e.g., $I = \{s_1, s_2, ..., s_n\}$, where $s_i \in S_i$ is an *instance* of S_i, for $i = 1, 2, ..., n$, and the goal is choosing a point from each set, such that $p - center(I)$ is maximized. Similarly, it is called, S is *k-separable* if the minimum distances between any pair of uncertainty regions is not less than k times of the maximum distance between different instances of any uncertain region.

Theorem 3 *Let $S = \{S_1, S_2, ..., S_n\}$ be a set of k-separable uncertainty regions modeled by set of discrete points. The problem of Max-P-center can be solved with $\frac{k+2}{k+4} = 1 - \frac{2}{k+4}$ approximation ratio.*

Proof. The proof is similar to the proof of Theorem 2. It is sufficient to choose the solution of the 1-center problem for each set S_i, $1 \leq i \leq n$, instead of placing the center of the disks as the solution. Since the solution of 1-center for each set S_i is a point of S_i whose maximum distance from the other points of S_i is minimized, it satisfies the necessary conditions for the proof. □

2.1 Min-P-center Problem

As explained, the goal of Min-P-center problem is finding an instance I^{min} that minimizes $p-center(I)$ among the all possible instances I of the uncertainty regions. We show that again the idea of placing center of the uncertainty regions resulted in good approximation of I^{min}, e.g., a $1 + \frac{2}{k}-$approximation solution when the regions are k-separable.

Theorem 4 *Let D be a set of k-separable disks as the input of the Min-P-center problem. The algorithm that places the center of disks is a $1 + \frac{2}{k}-$ approximation algorithm.*

Proof. This proof is similar to the proof of Theorem 2, however, the definition of the clusters is different. Let C_{opt} be the solution of Min-P-center problem, C_{c-opt} be the solution of P-center for the center of the disks and C' be the cluster which have the same topology with C_{opt} and the same location of demand points with C_{c-opt}. Since both C_{c-opt} and C' are the clusters on the center of disks and C_{c-opt} is the optimal solution of the P-center problem, we have

$$e_{c-max} \leq e'_{max} \quad (11)$$

where e_{c-max} is the edge with maximum weight in C_{c-opt} and e'_{max} is the edge with maximum weight in C'.

We consider an arbitrary edge $e' \in C'$. Suppose d_i and d_j are two connecting disks by e'. Let r_i and r_j be the radius of d_i and d_j, respectively, and l be the maximum distance between d_i and d_j. In C_{opt}, d_i and d_j connect to each other by an edge e which its weight is at least l. The weight of e' is at most $l + r_i + r_j$. So, the weight of an edge in C_{opt} to the weight of its corresponding edge in C' is at least

$$\frac{e}{e'} = \frac{l}{l + r_i + r_j} \geq \frac{k.r_{max}}{k.r_{max} + r_i + r_j}$$

$$\geq \frac{k.r_{max}}{k.r_{max} + r_{max} + r_{max}} = \frac{k}{k+2} \quad (12)$$

This is established for any edge in C_{opt} and its corresponding edge in C'. So,

$$e_{max-opt} \geq \frac{k}{k+2} e'_{max}, \quad (13)$$

where $e_{max-opt}$ is the edge in C_{opt} with maximum length. According to inequalities 11 and 13:

$$e_{max-opt} \geq \frac{k}{k+2} e_{c-max}. \quad (14)$$

Thus, the proof is complete. □

Theorem 5 *The problem Min-P-center for a set of $k-$separable discrete sets can be solved with the $\frac{k+2}{k} = 1 + \frac{2}{k}$ approximation ratio.*

Proof. Similar to the proof of Theorem 3 and Theorem 4, it is sufficient to choose the solution of discrete 1-center problem as the instance of Min-P-center problem, and follow the proof of Theorem 4. □

3 Conclusion and Future Work

In this paper, we defined two variations of the problem of P-center in the uncertainty context, the *Max-P-center* problem and the *Min-P-center* problem. In fact, these problems are the natural extension of P-center under uncertainty. In these problems, a set of regions, called *uncertainty regions*, are as given and the goal is placing a point in each region such that the worst and the best case happen for P-center problem, i.e., the instances resulted in maximizing or minimizing the objective value of the P-center problem. We considered two cases for the uncertainty regions, disjoint disks and discrete set of points. We presented a $\frac{1}{2}$-approximation algorithm and a parameterized approximation algorithm for the Max-P-center problem and a parameterized approximation algorithm for the Min-P-center problem.

In addition to the extension of the P-center problem under uncertainty defined in this paper, there is another extension called *Max-Regret* [7]. The regret is defined as the difference between the cost of a given solution and the cost of the optimal solution for a particular placement of the uncertain points. The worst case of regret between all possible placement of the uncertain points is called Max-Regret. So, a potential direction for future work include consideration of *Max-Regret P-center problem*.

References

[1] T. M. Chan. More planar two-center algorithms. *Computational Geometry*, 13(3):189–198, 1999.

[2] R. Chandrasekaran and A. Tamir. Polynomially bounded algorithms for locatingp-centers on a tree. *Mathematical Programming*, 22(1):304–315, 1982.

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

[4] M. E. Dyer. On a multidimensional search technique and its application to the euclidean one-centre problem. *SIAM Journal on Computing*, 15(3):725–738, 1986.

[5] A. Foul. A 1-center problem on the plane with uniformly distributed demand points. *Operations Research Letters*, 34(3):264–268, 2006.

[6] M. Hoffmann. A simple linear algorithm for computing rectilinear 3-centers. *Computational Geometry*, 31(3):150–165, 2005.

[7] P. Kouvelis, G. L. Vairaktarakis, and G. Yu. *Robust 1-median location on a tree in the presence of demand and transportation cost uncertainty.* Department of Industrial & Systems Engineering, University of Florida, 1993.

[8] D. Lee and Y.-F. Wu. Geometric complexity of some location problems. *Algorithmica*, 1(1-4):193, 1986.

[9] M. Löffler and M. van Kreveld. Largest bounding box, smallest diameter, and related problems on imprecise points. *Computational Geometry*, 43(4):419–433, 2010.

[10] N. Megiddo. Linear-time algorithms for linear programming in r^3 and related problems. *SIAM journal on computing*, 12(4):759–776, 1983.

[11] N. Megiddo and K. J. Supowit. On the complexity of some common geometric location problems. *SIAM journal on computing*, 13(1):182–196, 1984.

[12] N. Megiddo and A. Tamir. New results on the complexity of p-centre problems. *SIAM Journal on Computing*, 12(4):751–758, 1983.

[13] N. Megiddo, A. Tamir, E. Zemel, and R. Chandrasekaran. An $o(n \log^2 n)$ algorithm for the k th longest path in a tree with applications to location problems. *SIAM Journal on Computing*, 10(2):328–337, 1981.

[14] H. Wang and J. Zhang. One-dimensional k-center on uncertain data. *Theoretical Computer Science*, 602:114–124, 2015.

[15] H. Wang and J. Zhang. A note on computing the center of uncertain data on the real line. *Operations Research Letters*, 44(3):370–373, 2016.

[16] H. Wang and J. Zhang. Computing the center of uncertain points on tree networks. *Algorithmica*, 78(1):232–254, 2017.

[17] H. Wang and J. Zhang. Computing the rectilinear center of uncertain points in the plane. *International Journal of Computational Geometry and Applications*, 28(03):271–288, 2018.

A Plane Region-Fault Tolerant 2.095-Spanner for Points in Convex Position

Davood Bakhshesh* Mohammad Farshi†

Abstract

Let S be a set of points in the plane and $t > 1$ is a real number. A *region-fault tolerant t-spanner* G for S with respect to an arbitrary region $C \subset \mathbb{R}^2$ is a t-spanner of S such that for any two vertices p and q in the graph $G \ominus C$ -the resulting graph after removing vertices and edges of G intersecting C- there is a path between p and q in $G \ominus C$ of length at most t times the length of the shortest path in the graph $K_S \ominus C$, where K_S is the complete graph of S. In this paper, we show that for any set S of n points in the plane that are in convex position, there is a plane region-fault tolerant 2.095-spanner T with respect to any convex region in the plane. To the best of our knowledge, the known algorithms that constructs a region-fault tolerant t-spanner for S does not guarantee that the output graph is plane. Moreover, we show that the graph T can be constructed in $O(n)$ time, assuming that the points of S are given in sorted order along the boundary of its convex hull. Previously, the time complexity of the best known algorithm to construct a region-fault tolerant t-spanner for S with respect to the convex regions is $O(n \log n)$.

1 Introduction

Let S be a set of n points in the plane and G is a weighted graph with vertex set S. G is called *geometric*, if each edge (p, q) in G is the straight-line between p and q and the weight of (p, q) is the Euclidean distance between p and q, denoted by $|pq|$. Let $t > 1$ be a real number. The geometric graph G is called a *t-spanner* of S, if for any two points $p, q \in S$, there is a path between p and q in G of length at most $t|pq|$. The minimum value of t that G is a t-spanner is called the *stretch factor (*or *dilation)* of G. To see an overview of algorithms for constructing t-spanners, see the book [5] by Narsimhan and Smid.

Before summarizing our results, we introduce some other definitions. Let $\mathcal{R}$ be a family of regions in the plane. A geometric graph G is called an *$\mathcal{R}$-fault tolerant t-spanner* of S (see [1]), if for any region $R \in \mathcal{R}$ and for any two vertices p and q in the graph $G \ominus R$ (the resulting graph after removing vertices and edges of G intersecting R), there is a path between p and q in $G \ominus R$ of length at most t times the length of the shortest path between p and q in $K_S \ominus R$, where K_S is the complete graph on S. Abam et al. in [1] considered the problem of constructing $\mathcal{C}$-fault tolerant t-spanners of S, where $\mathcal{C}$ is the family of all convex regions in the plane. They proposed an algorithm that constructs a $\mathcal{C}$-fault tolerant t-spanner of S with $O(n \log n)$ edges in $O(n \log^2 n)$ time. In the case where the points of S placed in convex position, Abam et al. [1] provided an algorithm that constructs a $\mathcal{C}$-fault tolerant t-spanner of S in $O(n \log n)$ time containing $O(n)$ edges.

Throughout the paper we assume that the points of S placed in convex position. In 2016, Biniaz et al. [3] introduced a triangulation T of S with the stretch factor at most 1.88. The construction of T is as follows. At first the algorithm adds the edges of the convex hull $CH(S)$ of S to T. Then, it recursively does the following operation. It finds the farthest pair (p, q) of points in S and adds (u, v) to T, where u and v are the two neighbors of p on $CH(S)$. Then, the algorithm applies the above operation on the point set $S \backslash \{p\}$ that is a point set in convex position. The above operations are continued until the point set contains less than four points.

In this paper, we show that the triangulation T is a $\mathcal{C}$-fault tolerant 2.095-spanner. Note that the proposed algorithm by Abam et al. [1] can construct a $\mathcal{C}$-fault tolerant 2.095-spanner for S but it is not necessarily plane.

Finally, we show that if the points of S are given in sorted order along $CH(S)$, then we can construct a $\mathcal{C}$-fault tolerant 2.095-spanner for S in $O(n)$ time.

2 Plane $\mathcal{C}$-Fault Tolerant 2.095-spanner

Here, we use the notations and the terminologies in [3]. Let p and q be two points in S. The clockwise and counter-clockwise paths from p to q along $CH(S)$ are denoted by $\delta^{cw}_{CH(S)}(p, q)$ and $\delta^{ccw}_{CH(S)}(p, q)$, respectively. Let $\delta^{*}_{CH(S)}(p, q)$ be the shorter path of the two paths $\delta^{cw}_{CH(S)}(p, q)$ and $\delta^{ccw}_{CH(S)}(p, q)$. The point p is called a *t-good* point for the point q, if $|\delta^{*}_{CH(S)}(p, q)| \leq t|pq|$. The point p is called a *t-good* point for S, if p is a t-good point for all points in S. A farthest pair (p, q) of points in S is called *diametral pair*, and the points p and q are called *diametral points* and $|pq|$ is called the *diameter* of

*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

S. Biniaz et al. [3] proved that any diametral point is a 1.88-good for S. Hence, they proposed the following simple algorithm (Algorithm 1) that constructs a triangulation T of stretch factor at most 1.88 for S (for more details see [3]).

Algorithm 1: PLANESPANNER(S)

input: A finite set S of points in the plane in convex position.
output: A plane 1.88-spanner T.

1 E :=the edge set of $CH(S)$;
2 $B := S$;
3 **while** $|B| \geq 4$ **do**
4 $\quad p :=$ a diametral point in B;
5 $\quad q, r :=$the two neighbors of p on $CH(S)$;
6 $\quad E := E \cup \{q, r\}$;
7 $\quad B := B \backslash \{p\}$;
8 **end**
9 **return** $T = (S, E)$

For any two points p and q, let $L(p, q)$ be the *lune of p and q* that is the intersection of two closed disks of radius $|pq|$ with centers p and q.

Observation 1 *If (p, q) be a diametral pair for the point set S, then $L(p, q)$ contains all points of S.*

In the following, we provide some lemmas that are needed later.

Lemma 1 ([4]) *Let a, b, and c be three points in the plane, and let $\beta = \angle abc$. Then,*

$$\frac{|ab| + |bc|}{|ac|} \leq \frac{1}{\sin(\beta/2)}.$$

Lemma 2 ([4]) *Let C be a convex chain with endpoints p and q. If C is in $L(p, q)$, then the stretch factor of C is at most $\frac{2\pi}{3}$.*

Let $\mathcal{H}$ be the family of all half-planes in the plane.

Lemma 3 ([1]) *A geometric graph G on a set S of points in the plane is a $\mathcal{C}$-fault tolerant t-spanner if and only if it is an $\mathcal{H}$-fault tolerant t-spanner.*

In the following, we prove the fault-tolerancy of the triangulation T.

Theorem 4 *The graph T generated by algorithm* PLANESPANNER *is a plane $\mathcal{C}$-fault tolerant $\frac{2\pi}{3}$-spanner of S.*

Proof. Since T is a triangulation of S, clearly T is plane. Let $|S|$ be the cardinality of S. By induction on $|S|$, we prove that for any choice of diametral point in Line 4, the graph T is a $\mathcal{C}$-fault tolerant $\frac{2\pi}{3}$-spanner of S. By Lemma 3, it is sufficient to prove that T is an $\mathcal{H}$-fault tolerant $\frac{2\pi}{3}$-spanner of S. Let $t = \frac{2\pi}{3}$.

Base step: it is clear that the graph T for $|S| \leq 3$, is a complete graph. Hence, obviously T is an $\mathcal{H}$-fault tolerant t-spanner.

Induction hypothesis: we suppose that for any set S of points with $|S| < n$, for any choice of diametral point in Line 4, the generated graph by algorithm PLANESPANNER on the point set S is an $\mathcal{H}$-fault tolerant t-spanner.

Induction step: suppose that $|S| = n$ and $n \geq 4$. Let p be the diametral point which is selected at the first iteration of Line 4 of Algorithm 1. Let T' be the subgraph of T which is obtained from T by removing the point p and its incident edges. Let B be the vertex set of T'. Consider the output of the algorithm PLANESPANNER when the input is B. It is easy to see that one can select the diametral points in Line 4 such that PLANESPANNER(B) be the same as T'. Hence, by applying the induction hypothesis to the set B, T' is an $\mathcal{H}$-fault tolerant t-spanner. Now, let h be an arbitrary half-plane. To show that T is an $\mathcal{H}$-fault tolerant t-spanner, it suffices to prove that for any two vertices x and y in $T \ominus h$, there is a path in $T \ominus h$ of length at most $t|pq|$. If both x and y are two vertices of $T' \ominus h$, then since T' is an $\mathcal{H}$-fault tolerant t-spanner, there is a path between x and y of length at most $t|xy|$. Now, suppose that one of the points x and y is not a vertex of $T' \ominus h$. Suppose, without loss of generality, that x is not in $T' \ominus h$. Clearly, $x = p$. Since T contains the edge set of $CH(S)$, there is a convex path P between p and y using only the edges of $CH(S)$. By Observation 1 and Lemma 2 and since p is a diametral point, it is not hard to see that the length of P is at most $t|xy|$. This completes the proof. □

Now, assume that the algorithm in Line 4 always select the diametral point p which has the smallest x-coordinate. In the following, with this assumption, we show that the stretch factor $\frac{2\pi}{3}$ in Theorem 4 is tight. Let p and q be two points in the plane such that the segment pq is horizontal. Let r be an intersection of two disks of radius $|pq|$ that are centered at p and q (see Figure 1). Let $S = \{p, q\} \cup S'$, where S' is the set of some points in the plane which are uniformly distributed on the union of two arcs $\widehat{pr}$ and $\widehat{rq}$. Note that we suppose that the points of S' are sufficiently large. Now, if we run the algorithm PLANESPANNER on S, we obtain a triangulation as depicted in Figure 1.

Now, let $u \in S'$ be the closest point to q and h is a half-plane that contains the point q and does not contain u as depicted in Figure 2. Note that we assume that the point u is sufficiently close to q. Consider the graph $T \ominus h$. Since all points in S are connected to q by an edge, there is only one path P between p and u in $T \ominus h$ which only uses the edges in $CH(S)$. Hence, clearly we have $\frac{|P|}{|pu|} \approx \frac{2\pi}{3}$.

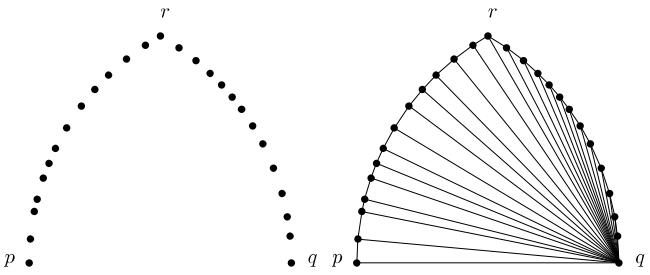


Figure 1: The points set $S = \{p, q\} \cup S'$ (left) and the triangulation T generated by PlaneSpanner(S) (right).

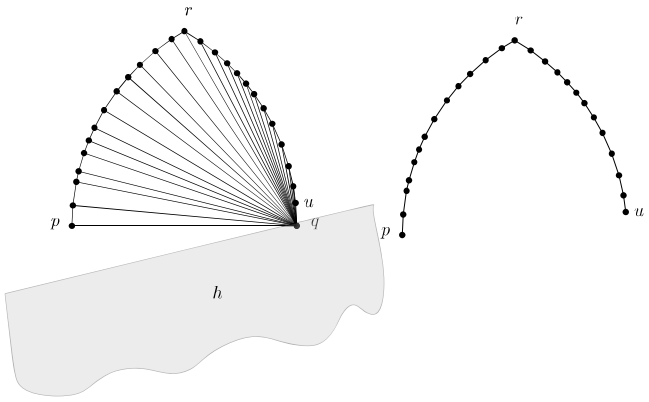


Figure 2: The half-plane h and the point u (left). The graph $T \ominus h$ (right).

Note that if we omit our assumption on the way of selecting the diametral point p, the above point set does not show that $\frac{2\pi}{3}$ is tight.

3 Algorithms

Let S be a set of n points in the plane that are in convex position. A naive implementation of algorithm PlaneSpanner has a running time of $O(n^2)$. In [3], by modifying algorithm PlaneSpanner, Biniaz et al. provided a linear time algorithm that constructs a plane 1.88-spanner for S, assuming that we already know the points of S are in sorted order along $CH(S)$. We call this new algorithm by ModifiedPlaneSpanner. The idea behind the algorithm ModifiedPlaneSpanner is that they replace any diametral pair by a pair of points that approximates the diametral pair. In this section, we show that the graph generated by ModifiedPlaneSpanner is a plane $\mathcal{C}$-fault tolerant 2.095-spanner. Hence, there is a linear time algorithm to construct a plane $\mathcal{C}$-fault tolerant 2.095-spanner.

In the following, we first present some lemmas and theorem that are needed.

Lemma 5 ([3]) *For any set S of points in the plane that are in convex position, there is a pair (p_S, q_S) of points in S such that $D < 1.0001|p_S q_S|$, where D is the diameter of S. Moreover, both p_S and q_S are 1.88-good for S.*

Algorithm ModifiedPlaneSpanner due to Biniaz et al. is similar to the algorithm PlaneSpanner just it needs to replace Line 4 of the Algorithm 1 by the instruction "$p := p_S$ or q_S ", where p_S and q_S were introduced in Lemma 5. The generated graph by algorithm ModifiedPlaneSpanner is a triangulation T_{mod} of S that is 1.88-spanner (see [3]).

Proving that T_{mod} is a $\mathcal{C}$-fault tolerant 2.095-spanner for S is similar to the proof of Theorem 4 just we need to prove a result similar to Lemma 2 (key lemma in the proof of Theorem 4) for the pair (p_S, q_S) that is shown in the following.

Theorem 6 ([2]) *If R_1 and R_2 are convex polygonal regions with $R_1 \subseteq R_2$, then the length of the boundary of R_1 is at most the length of the boundary of R_2.*

Let S be a set of points in the plane that are in convex position. In the following, we suppose, without loss of generality, that the diameter D of S is one, p_S is to the left of q_S and $p_S q_S$ is horizontal. Let C_1 and C_2 be two closed disks with radius one that are centered at p_S and q_S, respectively (see Figure 4). Let f be the intersection of the disk C_1 and the horizontal ray which is emanated from p_S and goes infinitely in the direction of positive x axis. Analogously, the point g is defined (see Figure 4). Let r be the intersection point of C_1 and C_2 which is above the segment $p_S q_S$. We denote the region bounded by gf, $\widehat{gr}$ and $\widehat{rf}$ by $R(p_S, q_S)$. Let $d = |p_S q_S|$. By Lemma 5, we have $\frac{1}{1.0001} < d \leq 1$.

Lemma 7 *Let c be a point that is on the boundary or in the interior of $R(p_S, q_S)$ such that $\angle c p_S q_S$ and $\angle c q_S p_S$ are not obtuse. Then, $\angle p_S c q_S \geq \pi - 2\arccos(d/2)$.*

Proof. Let $\beta = \angle p_S c q_S$ and c' be the orthogonal projection of c on $p_S q_S$. Let $\beta_1 = \angle p_S c c'$ and $\beta_2 = \angle q_S c c'$ (see Figure 3). Clearly, $\beta_1 + \beta_2 = \beta$. Now, we have $|p_S c'| = |p_S c| \sin\beta_1$ and $|q_S c'| = |q_S c| \sin\beta_2$. Since $|p_S c'| + |q_S c'| = |p_S q_S| = d$ and $|p_S c|, |q_S c| \leq 1$, we have

$$d = |p_S c| \sin\beta_1 + |q_S c| \sin\beta_2 \leq \sin\beta_1 + \sin\beta_2.$$

It is not hard to see that if $\beta_1 + \beta_2 = \beta$, then $\sin\beta_1 + \sin\beta_2 \leq 2\sin(\beta/2)$. Hence,

$$d \leq \sin\beta_1 + \sin\beta_2 \leq 2\sin(\beta/2).$$

Using the fact $\sin(\beta/2) = \cos(\pi/2 - \beta/2)$, we can easily conclude that $\beta \geq \pi - 2\arccos(d/2)$. This proves the lemma. □

Lemma 8 *For any set S of points in the plane that are in convex position, the stretch factor of the both paths $\delta^{cw}_{CH(S)}(p_S, q_S)$ and $\delta^{ccw}_{CH(S)}(p_S, q_S)$ are at most 2.095.*

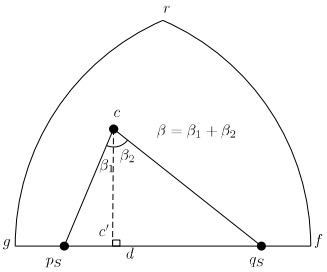


Figure 3: Illustrating proof of Lemma 7.

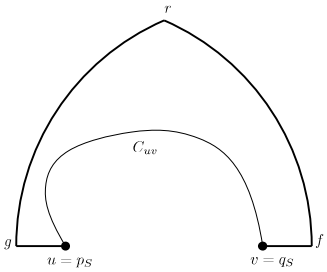


Figure 4: Illustrating proof of Lemma 8: case $u = p_S$ and $v = q_S$.

Proof. We prove the lemma for $\delta^{cw}_{CH(S)}(p_S, q_S)$. Analogously, we can prove the lemma for $\delta^{ccw}_{CH(S)}(p_S, q_S)$. We suppose, without loss of generality, that the segment p_Sq_S is horizontal and p_S is to the left of q_S. For the sake of simplicity, let $C := \delta^{cw}_{CH(S)}(p_S, q_S)$ and C is above the segment p_Sq_S. For any two points u and v of C, let C_{uv} be the path between u and v in which C_{uv} is a subpath of C. To prove the lemma, it suffices to show that for any two points u and v of C, the length of C_{uv} is at most $2.095|uv|$. Proof is almost similar to the proof of Lemma 2, but somewhere in the proof we need to change the equations and formulas.

Suppose, without loss of generality, that when we are traversing from p_S to q_S on C, the point u is seen before v. Let D be the diameter of S and $d = |p_Sq_S|$. We may assume that $D = 1$. So, $d \leq 1$. Two cases are considered: 1) $u = p_S$ or $v = q_S$, 2) $u \neq p_S$ and $v \neq q_S$.

- $u = p_S$ **or** $v = q_S$**.** Suppose, without loss of generality, that $u = p_S$. Because of the convexity, the path C_{uv} is inside the region that is bounded by (u, g), $\widehat{gr}$, $\widehat{rf}$ and (f, v) (see Figure 4). Hence, by Theorem 6, we have:

$$|C_{uv}| \leq |ug| + |\widehat{gr}| + |\widehat{rf}| + |fv|.$$

Since $|ug| = |fv| = D - d = 1 - d$ and $|\widehat{gr}| = |\widehat{rf}| = \arccos(d/2)$, $|C_{uv}| \leq 2 - 2d + 2\arccos(d/2)$. Hence, we have

$$\frac{|C_{uv}|}{|uv|} \leq \frac{2 - 2d + 2\arccos(d/2)}{d}. \qquad (1)$$

Note that $\angle rvg = \angle ruf = \arccos\frac{d}{2}$. By Lemma 5, we have $\frac{1}{1.0001} \leq d \leq 1$. Therefore, it is not hard to see that the maximum value of inequality (1) is obtained by $d = \frac{1}{1.0001}$. By substituting, we have $\frac{|C_{uv}|}{|uv|} \leq 2.095$.

Now, suppose that $v \neq q_S$. Let $\angle p_Sq_Sv = \alpha$ and $x = |q_Sv|$. We have two cases: $\alpha \leq \arccos\frac{d}{2}$ or $\alpha > \arccos\frac{d}{2}$. Since there is a page limitation, we moved the proof of these cases to the appendix.

- $u \neq p_S$ **and** $v \neq q_S$**.** The proof of this case is similar to the last case that was considered in the proof of Lemma 2 due to Biniaz et al. [4]. For more details, see the appendix.

□

According to the above mentioned arguments, we proved the following theorem.

Theorem 9 *Let S be a set of n points in the plane that are in convex position and suppose that the points of S are given in sorted order along the boundary of $CH(S)$. There is a plane $\mathcal{C}$-fault tolerant 2.095-spanner for S that can be constructed in $O(n)$ time.*

4 Conclusion and remarks

In this paper, we have shown that for any set S of points in the plane that is in convex position, there is a $\mathcal{C}$-fault tolerant 2.095-spanner for S that is computed in linear time, assuming that the points of S are given in sorted order along the boundary of $CH(S)$. We close the paper with the following open problems:

1. Is it possible to improve the bound 2.095 for constructing a plane $\mathcal{C}$-fault tolerant t-spanner for points in convex position?
2. Are the Delaunay triangulation and the greedy triangulation a $\mathcal{C}$-fault tolerant t-spanner for a constant t?
3. For any set of points in the plane, is there a plane $\mathcal{C}$-fault tolerant t-spanner for a constant t?

References

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

[2] R. Benson. *Euclidean geometry and convexity*. McGraw-Hill, 1966.

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

[4] A. Biniaz, P. Bose, J.-L. De Carufel, C. Gavoille, A. Maheshwari, and M. Smid. Towards plane spanners of degree 3. *Journal of Computational Geometry*, 8(1):11–31, 2017.

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

Transmission radius games on wireless networks

Mohammad Ali Abam* Mark de Berg† Mehran Mehr‡ Mahnaz Sadat Qafari§

Abstract

We study two new games on wireless networks, in which players corresponds to nodes in the network. The players' goal is to minimize their transmission radius under the condition that their hop distance to all other players is at most a given number k. We study the existence of Nash equilibria in both games, and compare their outcome with the best network when a central authority creates the network.

1 Introduction

Motivation. Recent advances in wireless communication make the formation of large-scale ad-hoc networks possible. The nodes in such wireless networks are small, low-cost devices with simple processing and networking capabilities. Each node p_i has a transmission radius r_i that is determined by the power with which the node decides to transmit. Thus node p_i can send information to all other nodes that are within distance r_i. Since a node's power consumption depends on the transmission radius it wants to achieve, each node prefers to keep its transmission radius small. On the other hand, each node would still like to be able to communicate with all other nodes, either directly or indirectly. In the latter case it may be desirable if every other node can be reached via only a few hops, that is, via only a few intermediate nodes. Sometimes there is a central authority determining the transmission radii of each of the nodes, so that overall power consumption is minimized. In ad-hoc networks this is typically not the case, however, and the nodes may act as selfish agents who only care about their own power consumption and their own capability to be able to communicate to all other nodes. This leads to several interesting game-theoretic questions, which we address in this paper.

Game Theory. A ***strategic game*** is a triple

$$G := (\mathcal{N}, (S_{p_i})_{p_i \in \mathcal{N}}, (c_{p_i})_{p_i \in \mathcal{N}})$$

which consists of a set $\mathcal{N}$ of players, a non-empty set S_{p_i} of ***strategies*** and a ***cost function*** $c_{p_i} : S_{p_1} \times \cdots \times S_{p_n} \to \mathbb{R}$ for each player $p_i \in \mathcal{N}$. Each element $\mathbf{s} \in S_{p_1} \times \cdots \times S_{p_n}$ is called a ***strategy profile***. When focusing on a single player, we can write a strategy profile $\mathbf{s}$ as (s_{p_i}, s_{-p_i}), where s_{p_i} is the strategy of player p_i and s_{-p_i} is the strategy profile of the other players. Given a strategy profile $\mathbf{s} := (s_{p_i}, s_{-p_i})$, a player p_i can deviate profitably from $\mathbf{s}$ if there is $s'_{p_i} \in S_{p_i}$ such that $c_{p_i}(s'_{p_i}, s_{-p_i}) < c_{p_i}(s_{p_i}, s_{-p_i})$. A strategy profile $\mathbf{s}$ is called a ***Nash equilibrium*** (NE, for short) if no one can deviate profitably from $\mathbf{s}$.

The ***social cost*** of a given strategy profile $\mathbf{s}$ is defined as $\mathbf{sc}(\mathbf{s}) := \sum_{i \in \mathcal{N}} c_{p_i}(\mathbf{s})$, and the strategy with the minimum social cost is called the ***social optimum***. The ratio between the highest resp. lowest social cost of a Nash equilibrium and the social cost of the social optimum is called the ***price of anarchy (PoA)*** resp. the ***price of stability (PoS)***.

Related work. Network creation games, where the players correspond to nodes in the network, were introduced by Fabrikant et al. [11]. In their games each player aims to minimize their own cost, which is defined as the sum of the shortest-path distances to all other vertices (the *usage cost*) plus the price for creating connections to other players (the *creation cost*). Later various other network creation games were proposed as well [1, 2, 6, 8, 14]. Also, several games have been proposed in which the players' goal is to minimize just the usage cost [4, 9], or where the cost of creating edges is non-uniform [5, 7, 16].

By now there is a large amount of literature on the application of game theory in wireless networks; see [3, 11, 13, 17] for some examples. The work by Eidenbenz *et al.* [10] is most closely related to our work. They define several games, considering settings where the underlying communication network is wireless, wireless with directional antenna, or wireline. In particular, they introduce the *strong connectivity game* where all nodes need to be connected to all other nodes, and the *connectivity game* where each player needs to be connected to a given set of nodes. One of our games is equivalent to the strong connectivity game; see below for details.

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

†Department of Computer Science, Eindhoven University of Technology (TU/e), m.t.d.berg@tue.nl

‡Department of Computer Science, Eindhoven University of Technology (TU/e), m.mehr@tue.nl

§Department of Computer Science, Institute for Research in Fundamental Sciences (IPM), m.s.qafari@ipm.ir

2 Problem statements and notations

Suppose a set of players $\mathcal{N} \coloneqq \{p_1, \ldots, p_n\}$ is given. Each player is a wireless node represented as a point $p_i \in \mathbb{R}^d$. Each player determines its own transmission radius $r(p_i)$ (or simply r_i), and therefore it can send messages to those players which are inside the ball with radius r_i centered at p_i. Each player prefers to have a k ***hop distance*** to every other player in the network, i.e., any message sent by the player reaches the destination by passing through at most k hubs (players). The parameter k is predefined and equal for all players. For each strategy profile $\mathbf{r} = (r_1, r_2, \cdots, r_n)$, p_i's cost function is defined to be:

$$c_i \coloneqq \begin{cases} r_i & if \quad \text{hop}_{G_\mathbf{r}}(p_i, p_j) \leq k, \quad \forall p_j \in \mathcal{N} \\ \infty & o.w. \end{cases}$$

where $G_\mathbf{r}$ is the network defined by the strategy $\mathbf{r}$ and $\text{hop}_{G_\mathbf{r}}(p_i, p_j)$ is the hop distance of p_i and p_j in $G_\mathbf{r}$. The graph $G_\mathbf{r}$ depends on a communication policy which can be either directed or bidrected. In the bidirected policy, $G_\mathbf{r} = (\mathcal{N}, \{\{p_i, p_j\} : |p_i p_j| \leq r_i \text{ and } |p_i p_j| \leq r_j \text{ where } 1 \leq i, j \leq n\})$, and in the directed policy $G_\mathbf{r} = (\mathcal{N}, \{(p_i, p_j) : |p_i p_j| \leq r_i \text{ where } 1 \leq i, j \leq n\})$ where $|p_i p_j|$ denotes the Euclidean distance of p_i and p_j. Therefore, the BiDirected Transmission Radius (BDTR) game and the Directed Transmission Radius (DTR) game can be respectively defined. In both games, the social cost of $\mathbf{r}$, $\mathbf{sc}(\mathbf{r})$, is equal to $\sum r_i$.

For ease of writing, we say p_i ***sees*** p_j if $|p_i p_j| \leq r_i$, denoted by $p_i \to p_j$, and we say p_i does not see p_j otherwise, denoted by $p_i \not\to p_j$. So, in the BDTR game there is an undirected edge between p_i and p_j in $G_\mathbf{r}$ if and only if $p_i \to p_j$ and $p_j \to p_i$, and in the DTR game there is an directed edge from p_i to p_j in $G_\mathbf{r}$ if and only if $p_i \to p_j$.

When $k = 0$, in both games each player needs direct access to every other player, then it is obvious that both games have just one NE with the desirable property in which $r_i \coloneqq |p_i p_j|$ where p_j is the furthest player from p_i. We next tackle two interesting cases: $k = 1$ and $k = n - 2$. It is worth mentioning that in the case $k = n - 2$, each player only tries to have connection to any other player, and the number of hops is not important anymore. We study both games in $\mathbb{R}^d$ $(d = 1, 2)$, and denote the x and y coordinates of a point p with $x(p)$ and $y(p)$.

3 DTR Game

We recall that the underlying communication graph $G_\mathbf{r}$ in the DTR game is directed, and each player aims at having a k hop-distance connection to every other player while minimizing his transmission radius.

3.1 Case $k = 1$

The existence of a stable network: Suppose $\mathcal{N} = \{p_1, \cdots, p_n\}$ where $p_i \in \mathbb{R}^1$, and $x(p_i) < x(p_j)$ for any $i < j$. Here, we propose an algorithm which creates an NE. The sketch of the algorithm is to find two players (not necessarily distinct) p_q and $p_{q'}$ in $\mathcal{N}$ and a strategy profile $\mathbf{r}$ such that every two players can connect each other through either p_q or $p_{q'}$, and $\mathbf{r}$ is an NE. Indeed, the algoirthm finds p_q and $p_{q'}$ by setting p_q ($p_{q'}$) to the leftmost (rightmost) point and moving p_q to right and $p_{q'}$ to the left. Next, we explain the algorithm in detail.

Let $\vec{q} = \langle 1, q, q', n \rangle$ be a hub vector of players' indices, and let $q \leq q'$. We say the strategy $\mathbf{r}$ ***respects*** $\vec{q}$ if it satisfies the following conditions:

$$C_1 \coloneqq x(p_i) \leq x(p_q) : p_i \to p_1, \tag{1}$$
$$C_2 \coloneqq x(p_i) > x(p_q) : p_i \not\to p_1, \tag{2}$$
$$C_3 \coloneqq x(p_i) \geq x(p_{q'}) : p_i \to p_n, \tag{3}$$
$$C_4 \coloneqq x(p_i) < x(p_{q'}) : p_i \not\to p_n. \tag{4}$$

C_1 means that p_q and all the player on his left side can see p_1, while C_2 means no player on his right side can see p_1. C_3 and C_4 are similarly defined with respect to $p_{q'}$ and p_n.

Let $\mathbf{r}(\mathcal{N}, \vec{q})$ be the ***induced strategy profile*** by a hub vector $\vec{q}$ as defined below:

$$r_i = \begin{cases} \max\{|p_1 p_q|, |p_q p_{q'}|\} & \text{if } p_i = p_q, \\ \max\{|p_{q'} p_n|, |p_q p_{q'}|\} & \text{if } p_i = p_{q'}, \\ \max\{|p_i p_q|, |p_i p_{q'}|\} & \text{otherwise.} \end{cases}$$

It is simple to see if $\mathbf{r}(\mathcal{N}, \vec{q})$ respects $\vec{q}$, then the network defined by $\mathbf{r}(\mathcal{N}, \vec{q})$ is an NE. Algorithm 1 helps us to find such players p_q and $p_{q'}$.

Theorem 1 *The DTR game admits an NE when players are in one dimension and $k = 1$.*

Social optimum: It is easy to show there are instances of n players such that the the social optimum network, the one created by a central authority (not players) to minimize the sum of the radii, is not necessarily an NE. As the PoA dependes on the social optimum network, we first provide some lower bounds and upper bounds on the social cost of the social optimum network. Indeed, we provide an approximation algorithm to compute a network whose social cost is at most $(6 + \epsilon)$ times the social cost of the social optimum network for any $\epsilon > 0$.

Let $\mathcal{N}' \subseteq \mathcal{N}$ be a subset of the players. We define the strategy profile $\mathbf{r} = (r_1, \ldots, r_n)$ induced by $\mathcal{N}'$ on $\mathcal{N}$ and denote it by $\mathbf{r}(\mathcal{N}, \mathcal{N}')$ as follows,

- $r_i \coloneqq \text{dist}(p_i, \mathcal{N}')$ if $p_i \notin \mathcal{N}'$, and
- $r_i \coloneqq \text{diam}(\mathcal{N})$ if $p_i \in \mathcal{N}'$,

Algorithm 1:

$\vec{q} \coloneqq \langle 1, 2, n-1, n \rangle$
$\mathbf{r} = \mathbf{r}(\mathcal{N}, \vec{q})$
$j \coloneqq 1$
while $j < 2$ **do**
  if $|p_1 p_q| \geq |p_q p_n|$
    $q' \coloneqq q$
    $j = j + 1$
  else if $|p_{q'} p_n| \geq |p_1 p_{q'}|$
    $q \coloneqq q'$
    $j = j + 1$
  else if $p_{q+1} \not\to p_1 \wedge p_{q'-1} \to p_n$
    $q' \coloneqq q' - 1$
  else if $p_{q'-1} \not\to p_n \wedge p_{q+1} \to p_1$
    $q \coloneqq q + 1$
  else if $p_{q'-1} \to p_n \wedge p_{q+1} \to p_1$
    if $|p_1 p_{q+1}| \leq |p_{q'-1} p_n|$
      $q \coloneqq q + 1$
    else
      $q' \coloneqq q' - 1$
  else
    $j = j + 1$
  $\mathbf{r} = \mathbf{r}(\mathcal{N}, \vec{q})$

where $\text{dist}(p, S)$ denotes the Euclidean distance of a point p to the nearest point in a pointset S and $\text{diam}(S)$ is the diameter of the pointset S, i.e. the Euclidean distance between the two farthest points in S.

We use a near-linear algorithm by Kolliopoulos and Rao [15] to solve the discrete version of the k-median problem: for a set of points S, the goal is to select a subset $S_k \subset S$ of size k to minimize $\sum_{p \in S} \text{dist}(p, S_k)$.

Let $\mathcal{N}_i$, $1 \leq i \leq n$, be a solution to the discrete i-median problem on $\mathcal{N}$, and let t be a number such that $\mathbf{sc}(\mathbf{r}(\mathcal{N}, \mathcal{N}_t))$ is minimum. We show that $\mathbf{sc}(\mathbf{r}(\mathcal{N}, \mathcal{N}_t))$ is at most six times the cost of the social optimum.

Let $\mathbf{r}^*$ be the strategy profile creating the social optimum network. We modify $\mathbf{r}^*$ to get a strategy profile $\mathbf{r}'$ with $\mathbf{sc}(\mathbf{r}(\mathcal{N}, \mathcal{N}_t)) \leq \mathbf{sc}(\mathbf{r}') \leq 6 \cdot \mathbf{sc}(\mathbf{r}^*)$ which proves the claim. We iterate on players increasingly on their radii. For each player p_i,

1. if p_i sees all other players, set $r_i \coloneqq \text{diam}(\mathcal{N})$,
2. otherwise, let p_j be the player with the largest radius seen by p_i, set $r_j \coloneqq \text{diam}(\mathcal{N})$.

Obviously, in $\mathbf{r}'$ each player can see any other player through at most one hub as we just increase some radii. It is not hard to see that $\mathbf{sc}(\mathbf{r}')$ is more than $\mathbf{sc}(\mathbf{r}(\mathcal{N}, \mathcal{N}_t))$ as its structure is exactly similar to an induced strategy, and $\mathbf{r}(\mathcal{N}, \mathcal{N}_t)$ has the minimum social cost by definition.

We next sketch why $\mathbf{sc}(\mathbf{r}') \leq 6 \cdot \mathbf{sc}(\mathbf{r}^*)$. Let r_j increase when we process p_i. Since the hop distance is at most 1, r_j increases at most by $2r_i$ as $|p_j p_\ell| \leq |p_j p_i| + |p_i p_\ell| \leq r_i + (r_i + r_j)$ for any player p_ℓ. Indeed, $2r_i + r_j \geq \text{radius}(\mathcal{N})$ (the radius of $\mathcal{N}$). First consider the $+2r_i$ increments which results in a three fold increase in the total cost. Using the fact that $\text{diam}(\mathcal{N}) \leq 2 \cdot \text{radius}(\mathcal{N})$, we increase the total cost by another two fold which proves the claim. Therefore, we get the following lemma,

Theorem 2 *Let $\mathcal{N}$ be a set of n players and $\epsilon > 0$ a fixed real number. There is a polynomial-time algorithm to compute a $(6+\epsilon)$-approximate to the social optimum.*

The PoA: It is known that, regardless of what k is, the price paid by each player is at most equal to his maximum distance to other players which is at most equal to $w(\text{MST}(\mathcal{N}))$, where $w(G)$ denotes the weight of the graph G and $\text{MST}(P)$ is the minimum spanning tree of a point set P. So the PoA is $O(n)$. Also it is possible to choose $\mathcal{N}$ such that the $\mathbf{r} = \mathbf{r}(\mathcal{N}, \vec{q})$ returned by algorithm 1 be much heavier than the weight of the social optimum network. Consider the set of points $\mathcal{N}$ in which $p_1 = 0$, $p_2 = 1 + \epsilon$ and $p_3, \ldots, p_n \in [2 - \epsilon, 2]$. The cost of the strategy profile $\mathbf{r} = \mathbf{r}(\mathcal{N}, \vec{q})$ returned by algorithm 1 on $\mathcal{N}$ is almost $n + 2$ while the cost of its social optimum is almost 2. This example shows that the PoA is $\Theta(n)$.

Theorem 3 *The PoA is $\Theta(n)$ in DTR game for $k = 1$.*

3.2 Case $k = n - 2$

When $k = n - 2$, the game is identical to the strong connectivity game introduced in [10]. There, it is shown that this game always has an NE, and the PoA is $\Theta(n)$.

4 BDTR Game

We remind that the graph $G_{\mathbf{r}}$ in the BDTR game is undirected, and each player aims at having a k hop-distance connection to every other player while minimizing his transmission radius. Obviously, the strategy profile $(0, \ldots, 0)$ is a trivial NE of the BDTR game for any k as no one benefits from increasing his strategy. It is a worthless equilibrium as there is not a k hop path between every pair of players. Similar to the DTR game, we next consider this game when $k = 1$ and $k = n - 2$.

4.1 Case $k = 1$

The existence of a stable network: When $k = 1$ and $\mathcal{N} \subseteq \mathbb{R}^1$, we show that there always exist a stable network by determining the structure of the NE networks for every possible configuration of player locations. So we have:

Theorem 4 *In one dimension, when $k = 1$, BDTR game admits a stable network for every set of players.*

Social optimum: Here, we present a 3-approximation algorithm for computing the social optimum network. This is done by introducing a non-trivial lower bound, and a strategy profile $\mathbf{r}$ whose cost is close to the lower bound.

Theorem 5 *Let $\mathcal{N}$ be a set of n players. There is a polynomial-time 3-approximation algorithm for computing the social optimum network in the BDTR game.*

The PoA: Using an example and an argument similar to the one used in the DTR game, we can show that the PoA is $\Theta(n)$.

Theorem 6 *In the BDTR game, the PoA is $\Theta(n)$ when $k = 1$.*

4.2 Case $k = n - 2$

Our results in this section are about players in $\mathcal{N} \subseteq \mathbb{R}^2$. We start with the existance of an NE. Consider M as a minimum spanning tree on $\mathcal{N}$. To find a stable strategy profile, initially, set r_i equal to his maximum distance to his neighbors in M. The network defined by this strategy profile is composed of one connected component. Then players, according to an arbitrary order, decrease their radii as much as the resulting network remains connected. It is easy to show that this gives us a strategy profile which is an NE. Moreover, the social cost of this strategy profile is at most twice $w(M)$ which means that the weight of the social optimum network for every set of points $\mathcal{N}$ is at most $2w(\text{MST}(\mathcal{N}))$. Finally, we provide a tight bound on the PoA.

Theorem 7 *In the BDTR game, the PoA is $\Theta(\log(n))$ when $k = n - 2$.*

References

[1] S. Albers, S. Eilts, E. Even-Dar, Y. Mansour, and L. Roditty. On nash equilibria for a network creation game. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 89–98. Society for Industrial and Applied Mathematics, 2006.

[2] N. Andelman, M. Feldman, and Y. Mansour. Strong price of anarchy. *Games and Economic Behavior*, 65(2):289–317, 2009.

[3] V. Behzadan and B. Rekabdar. A game-theoretic model for analysis and design of self-organization mechanisms in iot. In *International Conference on Game Theory for Networks*, pages 74–85. Springer, 2017.

[4] D. Bilò, L. Gualà, and G. Proietti. Bounded-distance network creation games. *ACM Transactions on Economics and Computation*, 3(3):16, 2015.

[5] A. Chauhan, P. Lenzner, A. Melnichenko, and L. Molitor. Selfish network creation with non-uniform edge cost. In *International Symposium on Algorithmic Game Theory*, pages 160–172. Springer, 2017.

[6] J. Corbo and D. Parkes. The price of selfish behavior in bilateral network formation. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 99–107. ACM, 2005.

[7] A. Cord-Landwehr, A. Mäcker, and F. M. auf der Heide. Quality of service in network creation games. In *International Conference on Web and Internet Economics*, pages 423–428. Springer, 2014.

[8] E. D. Demaine, M. Hajiaghayi, H. Mahini, and M. Zadimoghaddam. The price of anarchy in network creation games. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pages 292–298. ACM, 2007.

[9] S. Ehsani, S. S. Fadaee, M. Fazli, A. Mehrabian, S. S. Sadeghabad, M. Safari, and M. Saghafian. A bounded budget network creation game. *ACM Transactions on Algorithms (TALG)*, 11(4):34, 2015.

[10] S. Eidenbenz, V. A. Kumar, and S. Zust. Equilibria in topology control games for ad hoc networks. *Mobile Networks and Applications*, 11(2):143–159, 2006.

[11] A. Fabrikant, A. Luthra, E. Maneva, C. H. Papadimitriou, and S. Shenker. On a network creation game. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 347–351. ACM, 2003.

[12] M. Flammini, V. Gallotti, G. Melideo, G. Monaco, and L. Moscardelli. Network movement games. *Theoretical Computer Science*, 667:101–118, 2017.

[13] J. Gui, L. Hui, and N. N. Xiong. A game-based localized multi-objective topology control scheme in heterogeneous wireless networks. *IEEE Access*, 5:2396–2416, 2017.

[14] Y. Halevi and Y. Mansour. A network creation game with nonuniform interests. In *International Workshop on Web and Internet Economics*, pages 287–292. Springer, 2007.

[15] S. G. Kolliopoulos and S. Rao. A nearly linear-time approximation scheme for the euclidean k-median problem. *SIAM Journal on Computing*, 37(3):757–782, 2007.

[16] E. A. Meirom, S. Mannor, and A. Orda. Network formation games with heterogeneous players and the internet structure. In *Proceedings of the fifteenth ACM conference on Economics and computation*, pages 735–752. ACM, 2014.

[17] A. Nahir, A. Orda, and A. Freund. Topology design and control: A game-theoretic perspective. In *INFOCOM 2009, IEEE*, pages 1620–1628. IEEE, 2009.

On the Maximum Triangle Problem

Afrouz Jabalameli* Hamid Zarrabi-Zadeh†

Abstract

Given a set P of n points in the plane, the maximum triangle problem asks for finding a triangle with three vertices on P enclosing a maximum number of points of P. While the problem is easily solvable in $O(n^3)$ time, it has been open whether a subcubic solution is possible. In this paper, we show that the problem can be solved in $o(n^3)$ time, settling this open problem. We also improve the runtime of some of the previous approximation algorithms available for the problem.

1 Introduction

Let P be a set of n points in the plane. In the maximum triangle problem, the objective is to find a triangle with three vertices on P, so that the number of points of P enclosed by the triangle is maximum (see Figure 1 for an illustration). Eppstein *et al.* [4] showed that the problem can be solved in $O(n^3)$ time. They indeed solved a more general problem of finding a convex k-gon enclosing a maximum (or minimum) number of points in $O(kn^3)$ time. They left this question open whether the problem can be solved faster.

Douïeb *et al.* [3] revisited the maximum triangle problem, and presented several subcubic approximation algorithms for it. They again posed finding an $o(n^3)$-time exact algorithm as an open problem.

In this paper, we settle this open problem in affirmative by showing that an $o(n^3)$-time exact algorithm is indeed possible, using a reduction to the min-plus matrix multiplication, for which slightly subcubic algorithms are already known [1, 2, 5, 6]. The min-plus matrix multiplication (also known as distance product) has recently attracted considerable attention due to its connection to several fundamental problems such as all-pairs shortest paths, minimum cycles, replacement paths, metricity, etc. [7]. The current best time complexity for computing the min-plus product is $n^3/2^{\Omega(\sqrt{\log n})}$ [2, 6].

We also consider approximation algorithms for the maximum triangle problem, and improve the runtime of several algorithms proposed by Douïeb *et al.* [3] for the problem. Table 1 shows a summary of our results. In this table, h denotes the size of the convex hull of P.

*IDSIA Institute, University of Lugano, afrouz@idsia.ch.

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

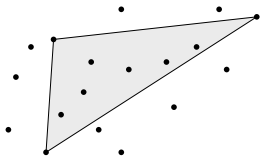


Figure 1: An example of a maximum triangle.

Algorithm	Runtime	
	Previous	New
Exact	$O(n^3)$	$n^3/2^{\Omega(\sqrt{\log n})}$
3-approx	$O(nh^2 \log n)$	$O(nh \log n + nh^2)$
4-approx	$O(nh^2 \log h)$	$O(nh \log h + h^3)$
4-approx	$O(n \log^2 n)$	$O(n \log n \log h)$

Table 1: Summary of the results.

2 Preliminaries

Let P be a set of n points in the plane. Throughout this paper, we assume that the points are in general position, i.e., no three points are co-linear, and no two points have the same x-coordinates.

Given three points $p, q, r \in P$, we call $\triangle pqr$ a triangle in P, and denote by $|\triangle pqr|$ the number of points of P enclosed by $\triangle pqr$. A triangle $\triangle pqr$ with maximum $|\triangle pqr|$ is called a *maximum triangle* of P, or in short, an *optimal triangle*.

3 A Subcubic Exact Algorithm

In this section, we show how the maximum triangle problem can be solved in $o(n^3)$ time, using matrix multiplication over the $(min, +)$-semiring, for which slightly subcubic algorithms are available. Recall that the *min-plus* product of two $n \times n$ matrices A and B is defined as

$$(A \oplus B)_{i,j} = \min_{1 \le k \le n} \{A_{i,k} + B_{k,j}\}.$$

Theorem 1 *Let P be a set of n points in the plane. A maximum triangle of P can be found in $O(T(n))$ time, where $T(n)$ is the time needed for computing the min-sum product of two $n \times n$ matrices, the best current algorithm for which has $n^3/2^{\Omega(\sqrt{\log n})}$ runtime.*

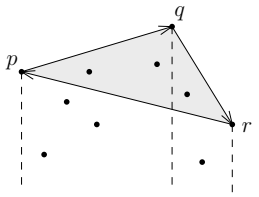


Figure 2: Points inside the triangle $\triangle pqr$.

Proof. For each pair of points $p, q \in P$, we denote by $n_{\overline{pq}}$ the number of points of P in the vertical slab below the line segment $\overline{pq}$. The value of $n_{\overline{pq}}$ for all pairs $p, q \in P$ can be computed in $O(n^2)$ time [4]. For any two points $p, q \in P$, we set $n_{\overrightarrow{pq}} = n_{\overline{pq}}$ if the vector $\overrightarrow{pq}$ is directed from left to right, and set $n_{\overrightarrow{pq}} = -n_{\overline{pq}}$ otherwise.

Now, for any three points $p, q, r \in P$ in clockwise order, the number of points in the triangle $\triangle pqr$ can be written as:

$$|\triangle pqr| = n_{\overrightarrow{pq}} + n_{\overrightarrow{qr}} + n_{\overrightarrow{rp}}$$

(see Figure 2 for an illustration). For points in counterclockwise order, we have $|\triangle pqr| = -(n_{\overrightarrow{pq}} + n_{\overrightarrow{qr}} + n_{\overrightarrow{rp}})$.

Let A be a $n \times n$ matrix with $A_{p,q} = n_{\overrightarrow{pq}}$, and let $B = A \oplus (A \oplus A)$. By the definition of the min-plus product, we have

$$B_{p,p} = \min_{q,r \in P} \{A_{p,q} + A_{q,r} + A_{r,p}\},$$

for all $p \in P$. Therefore, to obtain a maximum triangle, we just need to check the n values on the main diagonal of the matrix B for the smallest (negative) number, whose absolute value corresponds to the number of points in a maximum triangle. The optimal triangle itself can be easily found in $O(n^2)$ time by enumerating all $O(n^2)$ triangles with one vertex on the point realizing the smallest value in the diagonal. The whole runtime of the algorithm is therefore bounded by that of computing the min-plus product. □

4 Improved Approximation Algorithms

Douïeb *et al.* [3] proposed several subcubic approximation algorithms for the maximum triangle problem. The main idea behind their algorithms is to reduce the number of triangles enumerated by fixing 1, 2, or 3 vertices of the optimal triangle on the convex hull of the points. They also used this observation that if the surface of an optimal triangle is covered by c triangles (for a constant $c \geq 1$), then one of these triangles is a c-approximation of the optimal triangle.

In this section, we improve the runtime of the approximation algorithms proposed by Douïeb *et al.* [3], using faster methods for counting the number of points in the enumerated triangles.

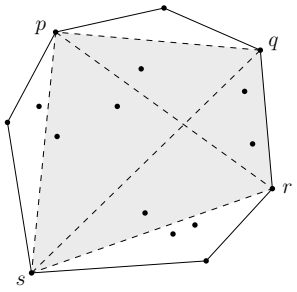


Figure 3: Triangles formed by four points on convex hull.

In the remaining of this section, we assume that P is a set of n points in the plane, H is the convex hull of P, and $h = |H|$. We will use the following two auxiliary results from Douïeb *et al.* [3].

Lemma 2 ([3]) *Among all triangles in P with k vertices on the convex hull ($1 \leq k \leq 3$), there exists a triangle that $(k+1)$-approximates an optimal triangle.*

Lemma 3 ([3]) *Given two points $p, q \in H$, the value of $|\triangle pqr|$ for all $r \in P$ can be computed in $O(n \log n)$ time. Furthermore, $|\triangle pqr|$ for all $r \in H$ can be computed in $O(n \log h)$ time.*

The following is a direct corollary of Lemma 3.

Lemma 4 *Given a point $p \in H$, the value of $|\triangle pqr|$ for all $q, r \in H$ can be computed in $O(nh \log h)$ time. Furthermore, $|\triangle pqr|$ for all $q \in P$ and $r \in H$ can be computed in $O(nh \log n)$ time.*

Proof. Fix a point q on H. By Lemma 3, $|\triangle pqr|$ for all $r \in H$ can be computed in $O(n \log h)$ time. Since there are $h - 1$ option for choosing q, computing $|\triangle pqr|$ for all $q, r \in H$ takes $O(nh \log h)$ time in total. Similarly, if we fix $q \in P$, the algorithm takes $O(nh \log n)$ time by Lemma 3. □

Now, we prove two lemmas which are the main ingredients of our improved algorithms.

Lemma 5 *The value of $|\triangle pqr|$ for all $p, q, r \in H$ can be computed in $O(nh \log h + h^3)$ time.*

Proof. Let p, q, r, s be four points on H in clockwise order. The value of $|\triangle pqr|$ can be written as $|\triangle spq| + |\triangle sqr| - |\triangle spr|$ (see Figure 3). By Lemma 4 we can compute the number of points enclosed by all triangles on H whose one vertex is fixed on s in $O(nh \log n)$ time. Therefore, after this preprocess step, we can compute the value of $|\triangle pqr|$ for each $p, q, r \in H$ in $O(1)$ time. Since there are $O(h^3)$ such triangles, the whole process takes $O(nh \log h + h^3)$ time in total. □

Lemma 6 *For all $p, q \in H$ and $r \in P$, the value of $|\triangle pqr|$ can be computed in $O(nh \log n + nh^2)$ total time.*

Proof. For a fixed point s on H, we compute the number of points enclosed by all triangles with one vertex on s, and the other two vertices freely chosen one from P and the other from H in $O(nh \log n)$ time using Lemma 4. Now, for any triangle $\triangle pqr$ with $p, q \in H$ and $r \in P$, we compute $|\triangle pqr|$ as follows.

(i) If r lies inside $\triangle pqs$, then $|\triangle pqr| = |\triangle pqs| - |\triangle prs| - |\triangle qrs|$.

(ii) If $\overline{rp}$ crosses $\overline{sq}$, then $|\triangle pqr| = |\triangle pqs| + |\triangle qrs| - |\triangle prs|$.

(iii) If $\overline{rq}$ crosses $\overline{sp}$, then $|\triangle pqr| = |\triangle pqs| + |\triangle prs| - |\triangle qrs|$.

(iv) If $\overline{rs}$ crosses $\overline{pq}$, then $|\triangle pqr| = |\triangle prs| + |\triangle qrs| - |\triangle pqs|$.

In any of the above cases, $|\triangle pqr|$ can be computed in $O(1)$ time. Since there are $O(nh^2)$ different triangles $\triangle pqr$ with $p, q \in H$ and $r \in P$, we can compute $|\triangle pqr|$ for all such triangles in $O(nh \log n + nh^2)$ total time. □

Now, Lemmas 5 and 6 together with Lemma 2 yield the following theorem.

Theorem 7 *A 3-approximation of an optimal triangle can be found in $O(nh \log n + nh^2)$ time. Furthermore, a 4-approximation of an optimal triangle can be found in $O(nh \log h + h^3)$ time.*

Remark. Eppstein *et al.* [4] proved that P can be preprocessed in $O(n^2)$ time, so that for any query triangle $\triangle pqr$ in P, $|\triangle pqr|$ can be reported in $O(1)$ time. Using this as an alternative way for counting the number of points in the enumerated triangles, we can rewrite the time bounds in Theorem 1 as $O(\min(n^2 + nh^2, nh \log n + nh^2))$ for the 3-approximation, and $O(\min(n^2 + h^3, nh \log h + h^3))$ for the 4-approximation algorithm.

In the following theorem, we present an alternative 4-approximation algorithm for the problem.

Theorem 8 *A 4-approximation of an optimal triangle can be found in $O(n \log n \log h)$ time.*

Proof. Let $t_1, t_2, \ldots, t_h$ be the vertices of H in clockwise order, and let $m = \lfloor h/2 \rfloor + 1$. We partition H into two convex polygons $H_1 = t_1, t_2, \ldots, t_m$ and $H_2 = t_m, \ldots, t_h, t_1$. Let P_1 and P_2 be the points of P enclosed by H_1 and H_2, respectively. We use Lemma 3 to compute $|\triangle t_1 t_m p|$ for all $p \in P$ in $O(n \log n)$ time. We then recurse on P_1 and P_2, and return a triangle found containing a maximum number of points.

To prove correctness, we first recall that there exists a triangle $\triangle t_1 pq$ with $p, q \in P$ that 2-approximates an optimal triangle [3]. If $t_1 t_m$ crosses pq, then the two triangles $\triangle t_1 t_m p$ and $\triangle t_1 t_m q$ cover $\triangle t_1 pq$, and hence, one of them is a 2-approximation of $\triangle t_1 pq$, which is in turn, a 4-approximation of an optimal triangle. On the other hand, if pq lies in one side of $t_1 t_m$, the recursive call on that side returns a 2-approximation.

Let $T(n, h)$ be the time required by the algorithm on a point set of size n whose convex hull has size h. Then, $T(n, h) = T(n_1, h_1) + T(n_2, h_2) + O(n \log n)$, where $n_1 + n_2 = n + 2$, $h_1 = \lfloor h/2 \rfloor + 1$, and $h_2 = \lceil h/2 \rceil + 1$. The recurrence tree for this relation has height $O(\log h)$, and yields $T(n, h) = O(n \log n \log h)$. □

5 Conclusions

In this paper, we presented a slightly subcubic algorithm for the maximum triangle problem, and improved the runtime of several approximation algorithms available for the problem. A main question that remains open is whether a truly subcubic algorithm with $O(n^{3-\varepsilon})$ time is possible for the problem. It is also interesting to study the generalized maximum k-gon problem, for $k \geq 4$.

Acknowledgments The authors would like to thank Mohammad-Reza Maleki, Hamed Valizadeh, and Hamed Saleh for their helpful discussions during the early stages of this work.

References

[1] T. M. Chan. More algorithms for all-pairs shortest paths in weighted graphs. *SIAM J. Comput.*, 39(5):2075–2089, 2010.

[2] T. M. Chan and R. Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. In *Proc. 27th ACM-SIAM Sympos. Discrete Algorithms*, pages 1246–1255, 2016.

[3] K. Douïeb, M. Eastman, A. Maheshwari, and M. Smid. Approximation algorithms for a triangle enclosure problem. In *Proc. 23rd Canad. Conf. Computat. Geom.*, pages 105–110, 2011.

[4] D. Eppstein, M. Overmars, G. Rote, and G. Woeginger. Finding minimum area k-gons. *Discrete Comput. Geom.*, 7(1):45–58, 1992.

[5] M. L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5(1):83–89, 1976.

[6] R. Williams. Faster all-pairs shortest paths via circuit complexity. In *Proc. 46th Annu. ACM Sympos. Theory Comput.*, pages 664–673, 2014.

[7] V. V. Williams and R. Williams. Subcubic equivalences between path, matrix, and triangle problems. *J. ACM*, 65(5):27, 2018.

Session 2

Recognizing Visibility Graphs of TINs

Hossein Boomari* Mojtaba Ostovari† Alireza Zarei‡

Abstract

A *Triangulated Irregular Network*(*TIN*) is a data structure that is usually used for representing and storing monotone geographic surfaces, approximately. In this representation, the surface is approximated by a set of triangular faces whose projection on the XY-plane is a triangulation. The visibility graph of a *TIN* is a graph whose vertices correspond to the vertices of the *TIN* and there is an edge between two vertices if their corresponding vertices on *TIN* see each other, i.e. the segment that connects these vertices completely lies above the *TIN*.

Computing the visibility graph of a *TIN* and its properties have been considered thoroughly in the literature. In this paper, we consider this problem in reverse: Given a graph G, is there a *TIN* with the same visibility graph as G? We show that this problem is $\exists\mathbb{R} - Complete$.

1 Introduction

A monotone surface is a surface with at most one intersection point with any vertical line. Such a surface can be represented as a function $S : \mathbb{R}^2 \to \mathbb{R}$. As a linear estimation, such surfaces are represented by *Triangulated Irregular Network* (*TIN*) which are composed of a set of triangular faces. Projection of a *TIN* on the XY-plane is therefore a triangulation in the plane (See Fig. 1).

The visibility relations between vertices of a TIN can be represented by a graph called its *visibility graph*. In this graph, each vertex corresponds to a TIN's vertex and there is an edge between a pair of vertices, if and only if the segment between their corresponding vertices on the TIN lies completely above the TIN. The visibility graph has applications in many *Computational Geometry* problems such as motion planning and ray tracing, *Computer Graphics*, robotics and other fields which consider the geometry of surfaces like *GIS*[1] and *Geology* [8]. Therefore, the problem of computing the visibility graph has been considered thoroughly and there are several polynomial time algorithms for computing these graphs on a TIN [7, 4]. The visibility graph is also defined for some other classes of geometric shapes like polygons, monotone curves and points in the plane.

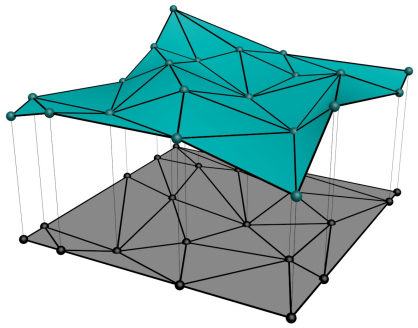


Figure 1: A Triangulated Irregular Network(TIN) and its planar projection.

Considering the problem of computing the visibility graph in reverse is another interesting theoretical problem in computational geometry[3, 5]. Restricting this problem for TINs, the goal is to determine whether there exists a TIN whose visibility graph is isomorphic to a given graph. This problem is known as *recognizing visibility graphs* and mainly is focused on defining neccesary and sufficient conditions on a graph to be a visibility graph. If the answer of the recognizing problem for a given graph is "yes", the next question is to build such a TIN which is known as *reconstructing visibility graph* problem. These problems have also been defined for other geometric shapes like simple polygons in which the target geometric object, is a simple polygon in the plane.

Despite many progresses in computing the visibility graphs, the computational complexity of the recognition and reconstruction problems are still open for almost all general shapes, including simple polygons and TINs. The most precise result states that these problems belong to *PSPACE*, and more precisely, to the *Existential Theory of The Reals* ($\exists\mathbb{R}$) complexity class.

The existential theory of the reals ($\exists\mathbb{R}$) is a complexity class introduced in 1989 [1] and formally defined by Shor later in 1991 [10]. This is the complexity class of problems which can be reduced to the problem of deciding whether there is a solution for a Boolean formula $\phi : \{True, False\}^n \to \{True, False\}$ in propo-

*Department of Mathematical Sciences, Sharif University of Technology, Tehran, Iran, h.boomari1@student.sharif.ir

†Department of Mathematical Sciences, Sharif University of Technology, Tehran, Iran, mojtaba.ostovari@alum.sharif.edu

‡Department of Mathematical Sciences, Sharif University of Technology, Tehran, Iran, zarei@sharif.ir

[1]Geographic information system

sitional logic, in the form $\phi(F_1, F_2, ..., F_n)$, where each $F_i : \mathbb{R}^N \to \{True, False\}$ consists of a polynomial function $G_i : \mathbb{R}^N \to \mathbb{R}$ on some real variables, compared to 0 with one of the comparison operators in $\{<, \leq, =, >, \geq\}$ (for example $G_i(X_1, X_2) = X_1^3X_2^2 - X_1X_2^3$ and $F_i(X_1, X_2) \equiv G_i(X_1, X_2) < 0$). Clearly, satisfiability of quantifier free Boolean formula belongs to $\exists\mathbb{R}$. Therefore, $\exists\mathbb{R}$ includes all NP problems. In addition, $\exists\mathbb{R}$ strictly belongs to $PSPACE$ [9] and we have $NP \subseteq \exists\mathbb{R} \subset PSPACE$. Although this class is inherently an Algebraic complexity class, but has gotten attention from geometers because some of the main arguments in its literature rely on geometric properties. Another reason is that some geometric problems are *complete* for this class. For example Recognizing *LineArrangement*, *Stretchability*, *Simple Order Type*, *Intersection Graph of Segments*, and *Intersection Graph of Unit Disks* in the Plane are complete for $\exists\mathbb{R}$ or simply $\exists\mathbb{R} - Complete$ [2]. As the most related result to this paper, in 2017 Cardinal *et al.* showed that recognizing visibility graph of a point set is $\exists\mathbb{R} - Complete$ [3]. The visibility graph of a set of 3D-points is a graph with these points as its vertices and there is an edge between two vertices if their connecting segment does not pass through any other point.

Because of their application in our proofs, we discuss Recognizing *LineArrangement* and *Stretchability* problems with more details in Section 2. In Section 3, we consider the recognition problem on *TIN*s and prove that this problem is also $\exists\mathbb{R} - Complete$. For this problem, we are given a pair of a graph and a triangulation which are respectively the visibility graph and the triangulation of a target *TIN* on the plane. Then, the question is whether there is a 3D *TIN* realization for this triangulation whose visibility graph is the same as the given graph. Note that the input triangulation does not contain position of the vertices on the plane and it only defines the faces and their adjacency relations on the target *TIN*.

2 Preliminaries and Definitions

2.1 LineArrangement and Stretchability

Combinatorial description of a geometric shape in the plane is an interesting problem in both theory and applications of computational geometry. *LineArrangement* is an example of such descriptions for a set of lines in the plane. *LineArrangement* describes the leftmost vertical order of the lines in the plane (initial order) and the left to right order of intersections of each line by the other lines (this ordering, for a line L_i, is denoted by $Seq(L_i)$). This arrangement is well defined for an arbitrary set of lines in the plane, especially when the lines are in *general position*[2] as we assumed in this paper. Recognizing *LineArrangement* for a given line arrangement instance, is the problem of deciding whether there is a set of lines in the plane with the same arrangement as the input. This problem is $\exists\mathbb{R} - Complete$ [2].

A set of pseudo-lines in the plane is a set of monotone curves where each pair intersect exactly once. Based on this property, *PseudoLineArrangement* (and its recognition problem), is defined analogously. In contrast to *LineArragement*, recognizing a *PseudoLineArragement* can be solved efficiently in polynomial time and the computational complexity of this problem belongs to P [6]. This difference introduces another problem called *Stretchability*, which is stated as: "Is it possible to stretch a pseudo-line realization of a *PseudoLineArrangement* and make them a set of lines without changing their arrangement?". *PseudoLineArrangement* belongs to P and recognizing *LineArrangement* is $\exists\mathbb{R} - Complete$; proving that the stretchability problem is $\exists\mathbb{R} - Complete$.

2.2 From PseudoLineArrangement to Triangulation

Similar to recognizing *PseudoLineArrangement*, its reconstruction[3] problem also belongs to P and can be computed efficiently [6] (See Fig. 2-a and Fig. 2-b). According to the given implementation of a reconstruction algorithm depicted in Algorithm 1 in Appendix section, we can obtain a special reconstruction of the pseudo-lines in which each pseudo-line is composed of a sequence of segments and the joint points of these segments correspond to the intersection points of the arrangement (See Fig. 2-c). This special reconstruction of a *PseudoLineArrangement* partitions the plane into convex regions. The regions are convex because any of the four segments connected to a break-point p lies in a different quadrant around p. To convert this subdivision to a triangulation, we put a new vertex (splitting vertices) on each segment, and split each edge into two edges with slightly different slopes (See Fig. 3-b). In addition, we add a vertex (middle vertices) inside each convex region. Then, each middle vertex is connected to all vertices on the boundary of its region (See Fig. 3-b). The resulting subdivision is a triangulation.

For an instance $\mathcal{A}$ of *PseudoLineArrangement*, this special reconstruction is denoted by $\mathcal{S}_\mathcal{A}$; the corresponding triangulation is denoted by $\mathcal{T}_\mathcal{A}$; $\mathcal{S}_\mathcal{A}(P)$ and $\mathcal{T}_\mathcal{A}(P)$ are respectively the sequences of segments of a pseudo-line P in $\mathcal{S}_\mathcal{A}$ and $\mathcal{T}_\mathcal{A}$; the first end-point of P in $\mathcal{S}_\mathcal{A}$ and $\mathcal{T}_\mathcal{A}$ are respectively denoted by $\mathcal{S}_\mathcal{A}^{first}(P)$ and $\mathcal{T}_\mathcal{A}^{first}(P)$, and the last ones are respectively denoted by $\mathcal{S}_\mathcal{A}^{last}(P)$ and $\mathcal{T}_\mathcal{A}^{last}(P)$. The corresponding graph of $\mathcal{T}_\mathcal{A}$, whose

[2] An arrangement which has no pair of parallel lines and no tripple of lines intersecting at the same point

[3] Computing a set of planar pseudo-lines with the given arrangement

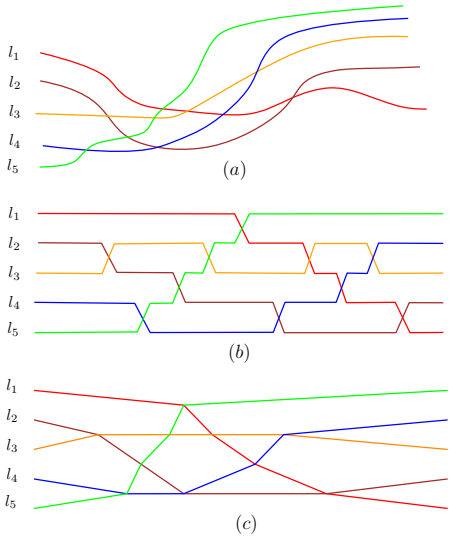


Figure 2: The reconstruction algorithm for *PseodoLineArrangement*. Algorithm 1 in Appendix section contains a pseudo code implementation of this algorithm.

vertices and edges respectively correspond to the vertices and edges of the triangulation, is called the triangulation graph of $\mathcal{T}_\mathcal{A}$.

Lemma 1 *For each pair of adjacent triangles t_1 and t_2 in $\mathcal{T}_\mathcal{A}$ with vertex sets $\{a,b,c\}$ and $\{a,b,d\}$ in $\mathcal{T}_\mathcal{A}$, the edge (c,d) does not exist in $\mathcal{T}_\mathcal{A}$. The proof of this lemma is given in Appendix 5.2*

It is simple to obtain a realization for an instance $\mathcal{A}$ of *PseudoLineArrangement* from a realization of its corresponding $\mathcal{T}_\mathcal{A}$. This can be done by removing the added middle and splitting vertices and their adjacent edges from $\mathcal{T}_\mathcal{A}$ and adding segments to connect each pair of vertices separated by a splitting vertex. Therefore, *Stretchability* can be reduced to the problem of whether there is a triangulation $\mathcal{T}_\mathcal{A}$ in the plane in which each pseudo-line in $\mathcal{T}_\mathcal{A}$ lies along a single line. We call this problem as *StretchableTriangulation*. We can summarize the above discussion as the following theorem.

Theorem 2 *Stretchability can be reduced to StretchableTriangulation in polynomial time and therefore, StretchableTriangulation is $\exists\mathbb{R}-Complete$.*

3 From StretchableTriangulation to Recognizing Visibility Graphs of TINs

We prove that deciding whether a triangulation $\mathcal{T}_\mathcal{A}$ is stretchable, can be reduced to an instance of the problem of recognizing visibility graphs of *TIN*s. For this purpose, for an instance $\mathcal{T}_\mathcal{A}$ of *StretchableTriangulation*, we build an instance $\langle G,T\rangle$ of recognizing visibility

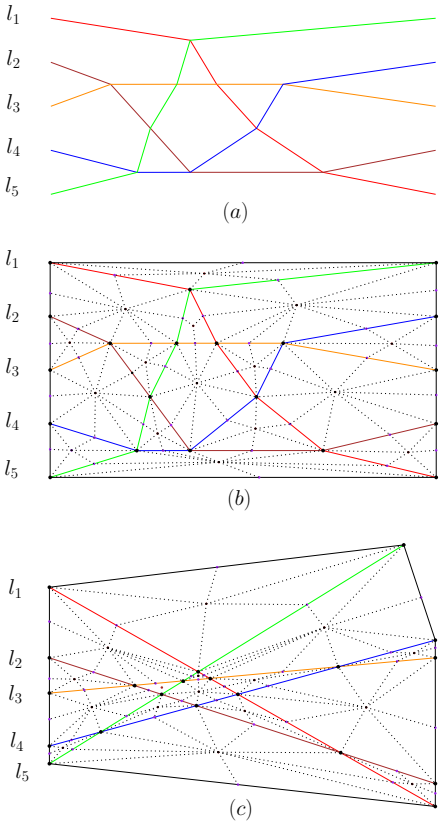


Figure 3: (a) The *PseudoLineArrangement* realization for an instance with initial order $\langle L_1,L_2,L_3,L_4,L_5\rangle$, $Seq(L_1)=\langle L_5,L_3,L_4,L_2\rangle$, $Seq(L_2)=\langle L_3,L_5,L_4,L_1\rangle$, $Seq(L_3)=\langle L_2,L_5,L_1,L_4\rangle$, $Seq(L_4)=\langle L_5,L_2,L_1,L_3\rangle$ and $Seq(L_5)=\langle L_4,L_2,L_3,L_1\rangle$, (b) The corresponding triangulation for the *PseudoLineArrangement* realization as an instance of *StretchableTriangulation* (c) A realization for the instance of *StretchableTriangulation* which is also a realization for the initial *LineArrangement* instance.

graphs of *TIN*s problem where $\mathcal{T}_\mathcal{A}$ is realizable if and only if $\langle G,T\rangle$ is recognizable. Remember that in $\langle G,T\rangle$ instance, G is the visibility graph of the vertices of the target *TIN* and T is the corresponding triangulation of the planar projection of this *TIN*. To construct the $\langle G,T\rangle$ instance, T is initially set to be the triangulation $\mathcal{T}_\mathcal{A}$ and G is initialized by exactly the vertex set and edges of this triangulation. Trivially, for a realizable instance of $\mathcal{T}_\mathcal{A}$, this initial $\langle G,T\rangle$ instance of the recognition problem has at least one realization which is exactly the realization of $\mathcal{T}_\mathcal{A}$ in the plane (which is a flat *TIN*). The next theorem states that all realizations of $\langle G,T\rangle$ are concave *TIN*s, i.e. the angle between each pair of adjacent faces is concave, seeing the TIN from above.

Lemma 3 *Any realization of the above $\langle G,T\rangle$ instance of recognizing visibility graphs of TINs is concave.*

Proof. We leave the proof to Appendix 5.3 □

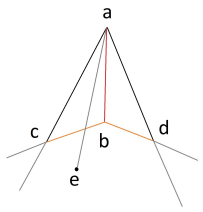


Figure 4: There can be no adjacent faces with a convex angle in any realization of $\langle G, T\rangle$.

To complete the reduction, we need a mechanism to enforce that in all $\mathcal{T}_{\mathcal{A}}$ realizations, each pseudo-line P_i lies along a single line. For this purpose, we attach gadgets, called alignment gadgets, to both corresponding vertices of $\mathcal{T}_{\mathcal{A}}^{first}(P_i)$ and $\mathcal{T}_{\mathcal{A}}^{last}(P_i)$ in G and T. The structures of this pair of gadgets for any pseudo-line P_i, restricts P_i, in any realization of $\langle G, T\rangle$, to lie in an arbitrary narrow convex volume. As shown in Fig. 5, the start and end gadgets of P_i that are connected to the first and last vertices a_i and b_i of a pseudo-line P_i are respectively $\{s_i, r_i, o_i\}$ and $\{s'_i, r'_i, o'_i\}$ vertices, where all three vertices in each gadget are connected to their corresponding endpoint in triangulation T. Moreover, to be a valid triangulation, there are edges from o_i (resp. o'_i) to both vertices s_i and r_i (resp. s'_i and r'_i) in T. As shown in Fig. 6.a in Appendix 5.5, this triangulation still needs some extensions to be a polished complete triangulation. This is done by adding some extra vertices and edges as shown in Fig. 6.b. This refinement is formally obtained by applying the following changes (assume that $\langle\{s_1, r_1, o_1\}, ..., \{s_n, r_n, o_n\}, \{s'_1, r'_1, o'_1\}, ...\{s'_n, r'_n, o'_n\}\rangle$ is the order of the gadgets around the outer boundary of $\mathcal{T}_{\mathcal{A}}$):

T-1. Add edges (r_i, s_{i+1}) and (s'_i, r'_{i+1}) for $1 \leq i \leq n-1$ and the two edges (r_n, r'_1) and (s_1, s'_n)

T-2. Add a vertex in each region $(r_i, s_{i+1}, a_{i+1}, a_i)$ and $(s'_i, r'_{i+1}, b_{i+1}, b_i)$ for $1 \geq i \geq n-1$ and two vertices in regions (a_n, r_n, r'_1, b_1) and (s'_n, s_1, a_1, b_4) and connect any one of these new vertices to the four vertices on its region boundary.

Note that T is always a subgraph of G which means that, till now, all vertices and edges of this refined triangulation exists in G as well. In order to enforce the stretchability of any pseudo-line P_i in any realization of the $\langle G, T\rangle$ instance of the recognition problem, the following visibility edges are added to G:

G-1. For each pair of vertices p and q of G where none is o_i or o'_i for the same $i : 0 < i \leq n$, the visibility edge (p, q) is added to G.

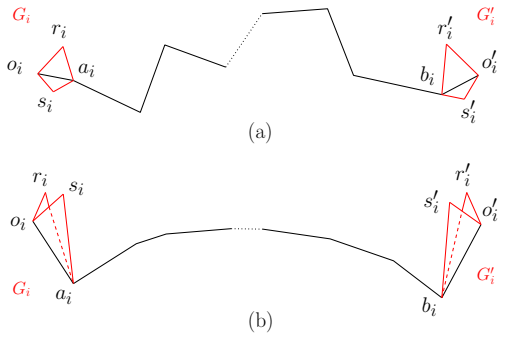


Figure 5: Alignment gadgets from a) top view b) side view.

G-2. For each pseudo-line P_i, the visibility edges from o_i and o'_i to all vertices of $G(P_i) \cup \{r_i, s_i, r'_i, s'_i\}$ as well as the edge (o_i, o'_i) are added to G.

In Theorem 4, we state that this structure forces the stretchability of P_i in any realization of the $\langle G, T\rangle$ instance and leave the proof to Appendix 5.4.

Theorem 4 *StretchableTriangulation is reducible to recognizing visibility graphs of TINs in polynomial time.*

Combining these results we have the following theorem about the complexity of the recognizing problem:

Theorem 5 *Recognizing visibility graphs of TINs is* $\exists\mathbb{R} - Complete$.

Proof. It is easy to show that recognizing visibility graphs of *TIN*s belongs to $\exists\mathbb{R}$ which is done by simply modeling an instance of this problem by an instance of satisfiability of a formula in $\exists\mathbb{R}$.

On the other hand, Theorem 2 states that *Stretchability* is reducible to *StretchableTriangulation* and Theorem 4 states that *StretchableTriangulation* is reducible to recognizing visibility graphs of *TIN*s, both in polynomial time. As *Stretchability* is $\exists\mathbb{R} - Complete$, recognizing visibility graphs of *TIN*s is $\exists\mathbb{R} - Complete$ as well. □

4 Conclusion

In this paper, we showed that recognizing the visibility graphs of *TIN*s problem is $\exists\mathbb{R} - Complete$. We proved it by making a reduction from the *stretchability* problem. As a possible future work, this result and technique may be useful in determining the computational complexity of other visibility graph recognition problems, particularly, recognizing the visibility graphs of monotone curves in two dimentions, whose computational complexity is still open.

References

[1] L. Blum, M. Shub, and S. Smale. On a theory of computation and complexity over the real numbers: Np-completeness, recursive functions and universal machines. *Bulletin of the American Mathematical Society*, 21(1):1–46, 1989.

[2] J. Canny. Some algebraic and geometric computations in pspace. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 460–467. ACM, 1988.

[3] J. Cardinal and U. Hoffmann. Recognition and complexity of point visibility graphs. *Discrete & Computational Geometry*, 57(1):164–178, 2017.

[4] L. Floriani and P. Magillo. Algorithms for visibility computation on terrains: a survey. *Environment and Planning B: Planning and Design*, 30(5):709–728, 2003.

[5] S. K. Ghosh. On recognizing and characterizing visibility graphs of simple polygons. In *Scandinavian Workshop on Algorithm Theory*, volume LNCS 318, pages 96–104. Springer, 1988.

[6] J. E. Goodman and R. Pollack. Allowable sequences and order types in discrete and computational geometry. In *New trends in discrete and computational geometry*, pages 103–134. Springer, 1993.

[7] G. Nagy. Terrain visibility. *Computers & graphics*, 18(6):763–773, 1994.

[8] D. O'Sullivan and A. Turner. Visibility graphs and landscape visibility analysis. *International Journal of Geographical Information Science*, 15(3):221–237, 2001.

[9] J. Richter-Gebert. Mnëv's universality theorem revisited. *Séminaire Lotaringien de Combinatorie*, 1995.

[10] P. Shor. Stretchability of pseudolines is np-hard. *Applied Geometry and Discrete Mathematics-The Victor Klee Festschrift*, 1991.

5 Appendix

5.1 Pseudo code for Algorithm 1

Algorithm 1 Recognizing and Reconstruction algorithm for *PseudoLineArrangement*

function FINDPSEUDOLINEARRANGEMENT(Initial vertical order $\mathcal{L} = \langle l_1, l_2, ..., l_n \rangle$, A queue for each l_i, $S_i = \langle l_{a(i,1)}, l_{a(i,2)}, ..., l_{a(i,n-1)} \rangle$, that contains the left to right order of the other pseudo lines intersections by l_i)
  if Any of S_i's has duplicate members **then**
return Arrangement Rejected
  end if
  Let $L_1, L_2, ..., L_n$ be n empty queues of points
  $i \leftarrow 1$
  while $i <= n$ **do**
    enqueue $(0, i)$ into L_i
    $i \leftarrow i + 1$
  end while
  $j \leftarrow 0$
  while $j < (n * (n-1))/2$ **do**
    Let (p, q) be a pair such that $p < q$ and $S_p.First() = l_q$ and $S_q.First() = l_p$
    if There is no such pair (p, q) **then**
      return Arrangement Rejected
    end if
    Dequeue S_p
    Dequeue S_q
    $j \leftarrow j + 1$
    Enqueue $(j, L_p.Top().Y)$ into L_p
    Enqueue $(j, L_q.Top().Y)$ into L_q
    Enqueue $(j - 0.5, L_p.Top().Y + 0.5)$ into L_q
    Enqueue $(j - 0.5, L_q.Top().Y - 0.5)$ into L_p
    $j \leftarrow j + 1$
    Enqueue $(j, L_p.Top().Y)$ into L_q
    Enqueue $(j, L_q.Top().Y)$ into L_p
  end while
  return $(L_1, L_2, ..., L_n)$
end function

5.2 Proof of Lemma 1

Proof. The common edge of t_1 and t_2 is either a part of a pseudo-line, or a segment that connects a vertex of a pseudo-line to a newly added vertex inside a region (the middle vertices) of the *PseudoLineArrangement* realization $\mathcal{S}_\mathcal{A}$. In the former case, the non-common vertices c and d are necessarily two newly added vertices inside different regions, and, in the latter case, these vertices are two non-adjacent vertices on the pseudo-lines. In both cases c and d are not adjacent in $\mathcal{T}_\mathcal{A}$. □

5.3 Proof of Lemma 3

Any realization of the above $\langle G, T \rangle$ instance of recognizing visibility graphs of *TIN*s is concave.

Proof. For the sake of a contradiction, assume that there is a non-concave realization $\mathcal{R}$ for an instance $\langle G, T \rangle$ of *StretchableTriangulation*. There must be at least two adjacent faces A and B in $\mathcal{R}$ whose interior angle is convex. Let $\{a, b, c\}$ and $\{a, b, d\}$ be respectively

the vertices of faces A and B (see Fig. 4). Lemma 1 states that c and d are neither adjacent in T, and hence not visible in G. Therefore, their visibility must be blocked by some part of the *TIN* that lies above the plane through vertices b, c and d. Because of the monotonicity of the *TIN*, there must be at least one vertex, like e, in this part of the *TIN* which is not adjacent to but visible from a. The reason of non-adjacency of a and e is that otherwise ae must either break the monotonicity of the *TIN* or intersect cb or bd which is in contradiction with the existance of the faces A and B. Consequently, a and e must have a visibility edge in T which is impossible according to its structure. □

5.4 Proof of Theorem 4

StretchableTriangulation is reducible to recognizing visibility graphs of *TIN*s in polynomial time.

Proof. Clearly, the above construction of the $\langle G, T\rangle$ instance of recognizing visibility graphs of *TIN*s (the visibility graph and the triangulation) can be done in polynomial time. Therefore, we need to prove that there is a realization for the instance $\langle G, T\rangle$ if and only if there is a realization for the instance of *StretchableTriangulation* $\mathcal{T}_{\mathcal{A}}$.

The visibility graph forces the vertices r_i, s_i, r'_i and s'_i as well as the vertices added in $T-2$ step to be high enough to see all other vertices except o_j and o'_j which are only visible to the vertices of their pseudo-line and its attached allignment gadgets. Moreover, the construction forces vertices s_i, r_i, s'_i and r'_i to be on the outer boundary of the triangulation. Therefore, the visibility and triangulation constraints in $\langle G, T\rangle$ imply that the new vertices must be located as the ridge points on the boundary of the triangulation, except o_i and o'_i vertices which are located as drain points of this boundary.

If there is a realization for the instance $\mathcal{T}_{\mathcal{A}}$ of the *StretchableTriangulation*, there is a realization for the instance $\langle G, T\rangle$ of the visibility graphs of *TIN*s. This instance can be constructed by locating the realization of $\mathcal{T}_{\mathcal{A}}$ on the XY-plane, locating the vertices r_i, s_i, r'_i and s'_i and the vertices in $T-2$ step on a plane U above the XY-plane in such a way that follow the triangulation faces and these vertices see each other and see all vertices of $\mathcal{T}_{\mathcal{A}}$ as well, and finally, putting o_i (resp. o'_i) on a segment $s'_i r'_i$ where s'_i and r'_i are respectively projections of s_i and r_i on a plane M between XY-plane and U. This guarantees support of the triangulation faces. It is simple to show that this realization is valid for the $\langle G, T\rangle$ instance.

For the other side of the proof, assume that the instance of $\langle G, T\rangle$ has a TIN realization. The projection of this realization on the XY-plane is a triangulation. Let's append a superscript star (*) to the name of each vertex in G, to denote its corresponding projected vertex on the plane. All vertices of $G_{\mathcal{A}}(P_i)$ (corresponding to a pseudo-line P_i) are visible from both o_i and o'_i. This implies that in this projected triangulation, these vertices must lie inside the intersection of the wedges defined by $\angle r_i^* o_i^* s_i^*$ and $\angle r_i^{*\prime} o_i^{*\prime} s_i^{*\prime}$. On the other hand, all other projected vertices must be outside this quadrilateral, because otherwise, they will be visible to either o_i or o'_i in G which is impossible (See Fig. 7-a in Appendix 5.5). Then, we can draw a straight line through o_i^* and $o_i^{*\prime}$ inside this convex region and move the projection of the vertices of $G(P_i)$ to this line without changing their order. This is done without changing the combinatorial structure of the triangulation T. In this manipulated triangulation, the projected vertices corresponding to vertices of $G(P_i)$ are along the line through $o*_i$ and $o*'_i$ (See Fig. 7-b). Indeed, the induced triangulation on this projected vertices is a realization for the instance $\mathcal{T}_{\mathcal{A}}$ of the *StretchableTriangulation* problem. This completes the proof by stating that, if the instance $\langle G, T\rangle$ of recognizing the visibility graphs of *TIN*s is realizable there is a realization for the corresponding instance $\mathcal{T}_{\mathcal{A}}$ of *StretchableTriangulation* problem and vice versa. □

5.5 Complementary Figures

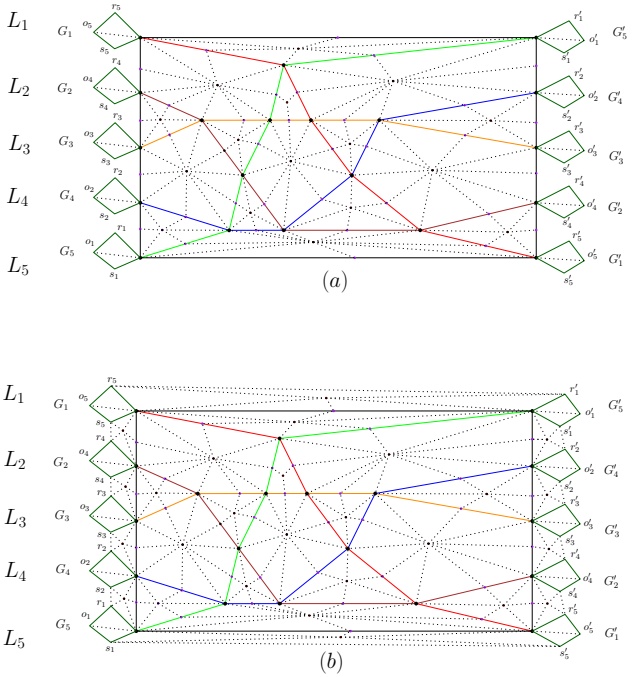


Figure 6: Reducing $\mathcal{T}_{\mathcal{A}}$ to $\langle G, T\rangle$ (a) adding alignment gadgets (b) refining the triangulation

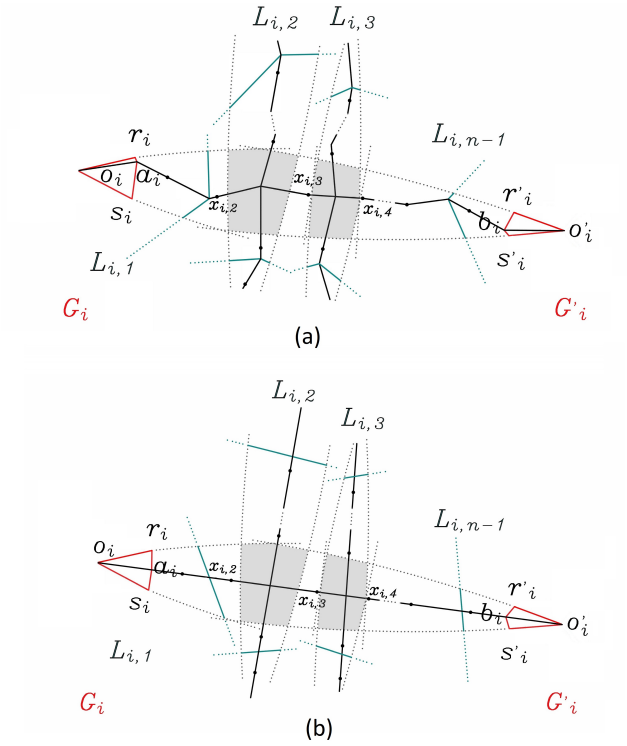


Figure 7: Stretching a pseudo-line P_i in a realization of the $\langle G, T \rangle$ instance.

Counting Closed Billiard Paths

Zahed Rahmati* Sina Farahzad† Shervin Naseri‡

1 Introduction

This paper addresses a problem that was presented at the open problem session of the 29th Canadian Conference on Computational Geometry. O'Rourke defines the problem as follows [1]. Consider a collection of axis-aligned rectangles, which all are enclosed in one outer rectangle; the outer rectangle is the enclosing boundary (billiard table) and other rectangles are inner boundaries (obstacles). A *simple, closed billiard path* is a path that is closed, non-self-intersecting forming a simple polygon, that do not touch a rectangle corner, where all reflections are mirror reflection. The *signature* of a billiard path is defined by the labels of edges which reflect the billiard path, where the repeated edge reflections $(ab)^k$ are reduced to ab. In Figure 1, the signature of the billiard path $\langle 4(56)^2 1292(37)^2 348\rangle$ is 456129237348.

For a set of rectangles with a total of n edges, O'Rourke posed the following open questions on simple, closed billiard paths.

1. What is the maximum length of such a signature?
2. What is the largest number of distinct signatures achievable for one fixed reflection angle (*e.g.,* 45° in Figure 1).
3. What is the largest number of distinct signatures achievable for paths at arbitrary reflection angles?

In this paper, in Section 2, we prove an upper bound of $4.5n-9$ for the length of a signature, and we show that there exists an arrangement of rectangles with signature length $1.25n + 2$. Also, in Section 3, we prove that the number of distinct signatures for fixed reflection angle 45° is bounded by $1.5n - 6$.

2 Maximum Length of a Signature

Consider a billiard path P and its signature S. We define an alternative signature S' from S, where any pattern aba in S is replaced by ab; this results in $|S| \leq 1.5 \cdot |S'|$. In Figure 1, the alternative signature of the billiard path with signature $S = 456129237348$ is $S' = 4561293748$.

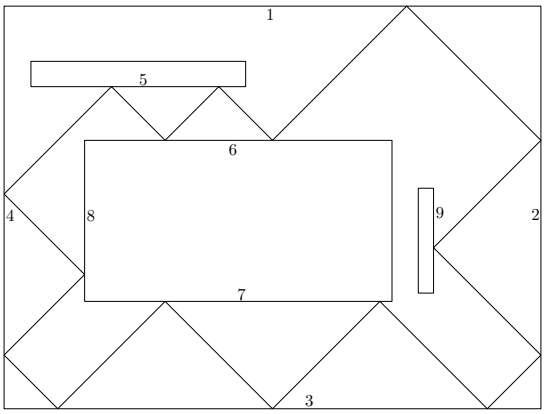


Figure 1: A billiard path of a set of four rectangles.

To compute the maximum length of a signature, we construct a planar graph $G(V, E)$ from a billiard path $P = \langle \dots p_{k-1}p_kp_{k+1} \dots \rangle$ such that $|E| = |S'|$, where S' is the alternative signature of P. The set V of nodes of G are defined as follows: Corresponding to each rectangle edge e_i we place a node v_i at the middle of e_i (see Figure 2(b)). We also draw the set E of edges (v_i, v_j) of G using the following rules.

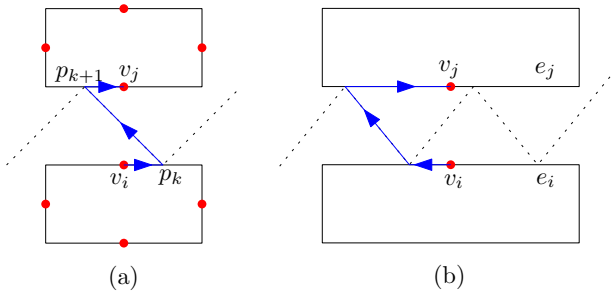


Figure 2: Consecutive segments between e_i and e_j; (a) only one segment. (b) more than one segment.

(a) If there is only one segment $\overline{p_kp_{k+1}}$ in P between rectangle edges e_i and e_j, where $p_k \in e_i$ and $p_{k+1} \in e_j$, then we draw the edge (v_i, v_j) with two bends at p_k and p_{k+1}, *i.e.,* $(v_i, v_j) = \langle \overline{v_ip_k}, \overline{p_kp_{k+1}}, \overline{p_{k+1}v_j}\rangle$. See Figure 2(a).

(b) If there exists a sequence of consecutive segments $\overline{p_kp_{k+1}}, \overline{p_{k+1}p_{k+2}}, \dots$ in P between two parallel

*Department of Mathematics and Computer Science, Amirkabir University of Technology, zrahmati@aut.ac.ir
†Department of Mathematics and Computer Science, Amirkabir University of Technology, sina-farahzad@aut.ac.ir
‡Department of Mathematics and Computer Science, Amirkabir University of Technology, shervin.naseri@aut.ac.ir

rectangle edges e_i and e_j, we ignore all but the first one ($\overline{p_k p_{k+1}}$) and draw an edge (v_i, v_j) in the graph G, similar to case (a).

(c) If there are two *non-consecutive* segments $\overline{p_k p_{k+1}}$ and $\overline{q_k q_{k+1}}$ in P between two *horizontal* rectangle edges e_i and e_j, where $p_k, q_k \in e_i$ and $p_{k+1}, q_{k+1} \in e_j$ and $|p_k|_x < |q_k|_x$, then there exists at least one rectangle inside the quadrilateral $p_k, p_{k+1}, q_{k+1}, q_k$. Here, $|r|_x$ denotes the x-coordinate of r. Take the leftmost rectangle inside the quadrilateral and place a node v_l on its left edge e_l; see Figure 3.

Note that there is no segment in path P touching e_l, otherwise it would cause a self-intersection which means that P is not a valid path. Also note that there can be no two segments in P both above and below e_l for the same reason; following this observation, w.l.o.g., assume there is no segment of P between e_l and e_j.

Corresponding to $\overline{p_k p_{k+1}}$, we draw an edge $(v_i, v_j) = \langle \overline{v_i p_k}, \overline{p_k p_{k+1}}, \overline{p_{k+1} v_j} \rangle$ with two bends at p_k and p_{k+1}, and also corresponding to $\overline{q_k q_{k+1}}$ we draw an edge (v_l, v_j) with one bend at b, the intersection of e_j and the vertical line through v_l; *i.e.*, $(v_l, v_j) = \langle \overline{v_l b}, \overline{b v_j} \rangle$.

Our method can easily be extended if there exist more than two, say three, non-consecutive segments (denote by $p_k p_{k+1}$, $q_k q_{k+1}$, and $r_k r_{k+1}$ in order from left to right) in P between two horizontal rectangle edges e_i and e_j: all we need to do is adding a new edge (v'_l, v_j) in G corresponding to the next segment $r_k r_{k+1}$, where v'_l is on the left edge e'_l of the leftmost rectangle in the quadrilateral $q_k q_{k+1} r_{k+1} r_k$.

A similar approach works if the *non-consecutive* segments are between two *vertical* rectangle edges e_i and e_j. Note that if there are two segments in P between a vertical rectangle edge and a horizontal rectangle edge, as shown in Figure 4, this leads to self-intersection in P and is thus obsolete.

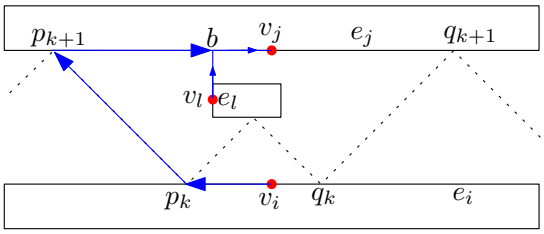


Figure 3: Two non-consecutive segments $\overline{p_k p_{k+1}}$ and $\overline{q_k q_{k+1}}$ between e_i and e_j.

By the above construction, there could be the case that some segments s_i and s_l (like $s_l = \overline{b v_j}$ and $s_i =$

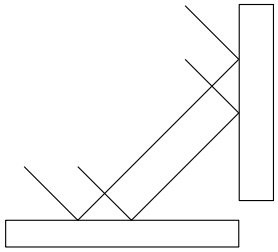


Figure 4: Two segments in path between a vertical and a horizontal rectangle edge lead to self-intersection.

$\overline{p_{k+1} v_j}$ in Figure 3) of edges (v_i, v_j) and (v_l, v_j) overlap on an edge e_j of a rectangle. We can define a valid ordering between the edges of G whose segments overlap on some rectangle edge e_j: Consider a line L_j parallel to e_j which intersects all the edges of G overlapping on e_j; see Figure 5. The distances from the intersection points to v_j define an ordering for the corresponding edges (and hence an ordering for their segments which overlap on e_j; denote the segments in order by $s_{j,1}, s_{j,2}, \ldots, s_{j,k}$). Since we have an ordering for segments $s_{j,1}, s_{j,2}, \ldots, s_{j,k}$, it is easy to slightly move the segments in order (by adding some new bends) inside the rectangle of e_j in such a way that no two edges of G overlap. Thus we can obtain a planar drawing for G. Figure 5 depicts a drawing of three edges inside a rectangle with no overlappings of their segments.

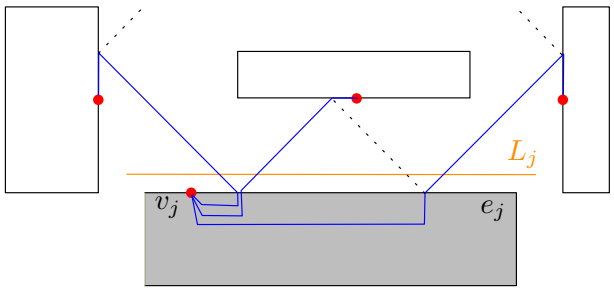


Figure 5: Drawing the edges with no overlappings.

By Eulers formula [2], it is proven that for any planar graph G, the number of edges is at most $3n - 6$. Thus $|S'| \leq 3n - 6$. Since the length of a signature S of a billiard path P is at most 1.5 times the length of the alternative signature S' of P, the following obtains.

Theorem 1 *For a collection of axis-aligned rectangles, all enclosed in one rectangle, with a total of n edges, the maximum length of a signature is at most* $4.5n - 9$.

Remark. In Theorem 1 there is no dependency on the reflection angle. Therefore, such a bound holds for arbitrary reflection angles.

Lower bound. Consider a set of $4k+2$ rectangles with a total of n edges. Let R be the outer rectangle, where we place the bottom-left corner of R at the origin and its top-right corner at $(4k+3, 8)$.

Let $R_1, \ldots, R_{k+1}$ be a collection of rectangles, each with width 1 and height 7; place the rectangle R_i with its center at $(4i - 2.5, 4)$. Let $R'_1, , R'_k$ be a collection of rectangles, each with width 2 and height 2; place the rectangle R'_i with its center at $(4i - 0.5, 4)$. Let $R''_1, \ldots, R''_{2k}$ be a collection of rectangles, each with width 1 and height 2; place the rectangle R''_{2i+1} with its center at $(4i - 0.5, 1.5)$ and the rectangle R''_{2i} with its center at $(4i - 0.5, 6.5)$.

If we shoot a ray from $(1, 0)$ with reflection angle $45°$ to the right, length of the signature of the resulting billiard path would be $1.25n + 2$. As seen in Figure 6, for $k = 2$, the number of edges is equal to $n = 40$ and the length of the signature is 52. It is obvious to see that, by increasing k by one unit (*i.e.*, adding four new inner rectangles) the signature length increases by 20.

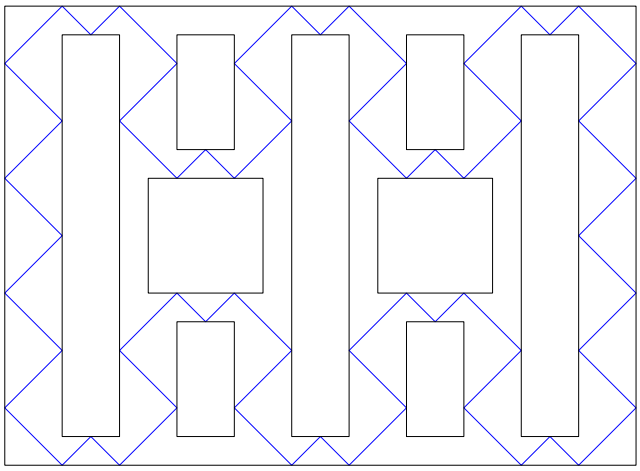


Figure 6: The signature length of ten rectangles is 50.

3 Number of Distinct Signatures

To count the number of distinct signatures, we charge each signature to a corner of a rectangle.

For each biliard path P, we define a starting point p_s residing on some upper edge of a rectangle [1]. Now, you can trace the path by following a ray of light shooting from p_s, where the edges of rectangles are perfect mirrors[2]. Since we have a fixed reflection angle $45°$, the billiard path P can now be represented by the *shooting point* p_s residing on some upper edge of a rectangle.

[1] If there is no upper edge involved in the path P, rotate the whole collection of rectangles by $90°$ until you find such upper edge; if you still don't find such an upper edge, let p_s be some reflection point on the bottom edge of the bounding rectangle.

[2] The ray shooting is the problem of determining the first intersection of a ray with a set of obstacles [2].

Lemma 2 *Let P be a billiard path with signature S and shooting point p_s . If p_s is translated to the right by the amount α such that the corresponding rays do not touch any corner over translation, the new point p'_s (which is of distance α to p_s) is a shooting point for a billiard path P' with the same signature $S' = S$.*

Proof. Let $P = \langle p_1 \ldots p_k \rangle$ and $P' = \langle p'_1 \ldots p'_k \rangle$, where $p_1 = p_s$ and $p'_1 = p'_s$. We claim, for $1 \leqslant i \leqslant k$, that the points p_i and p'_i touch the same edge (resulting in the same signature) and are within distance α.

We know that p_s and p'_s are at distance α. Also, p_s and p'_s reside on the same edge, as otherwise there would be a point from p_s at distance $\beta < \alpha$ such that it would touch a corner (the contradiction).

Assume that for all p_1 to p_i and p'_1 to p'_i, our claim is true, and p_i and p'_i reside on some upper edge (if not, rotate the whole collection of rectangles until it satisfies). Let $p_j = p_{i+1}$ and $p'_j = p'_{i+1}$. It is easy to see that $\|p_j - p'_j\|_\infty = \|p_i - p'_i\|_\infty = \alpha$ (see Figure 7): If p_j and p'_j both reside on a horizontal edge, then we have $d(p_j, p'_j) = \alpha$ (because $p_j p'_j$ is parallel to $p_i p'_i$). If p_j and p'_j both reside on a vertical edge, then $d(p_j, p'_j) = \alpha \cdot \tan(45°) = \alpha$. Thus we have proven our claim. □

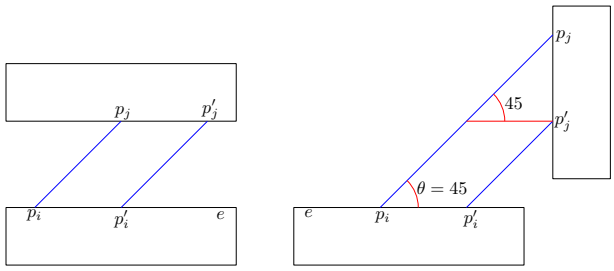


Figure 7: Shooting rays to the right.

Note that (as shown in Figure 8) there are three cases if a ray touches the corner of some rectangle.

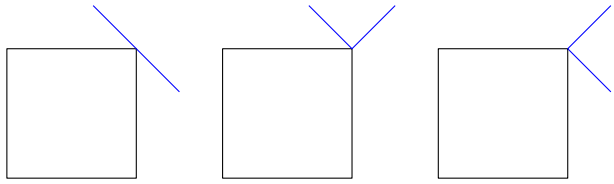


Figure 8: Three cases when a ray touching the corner of some rectangle.

Lemma 3 *For each signature, there exists a path with the same signature touching a corner of a rectangle.*

Proof. By translating the shooting point p_s of a path P slightly to *right*, unless we touch a corner of a rectangle, our signature remains the same (from Lemma 2).

Over translation, if a corner touches, which is one of the three cases in Figure 8, we still can have the same signature as before. If Case 2 (resp. Case 3 and Case 1) occurred when touching the corner, and we move the path more to the right (resp. up and left/down), then the corresponding signature changes. Therefore, we can charge any path P and its signature to (one of the three cases of) some corner. In Figure 9, the translated path can touch three corners p_1 (Case 2), p_2 (Case 1), and p_3 (Case 3) with the same signature as before.

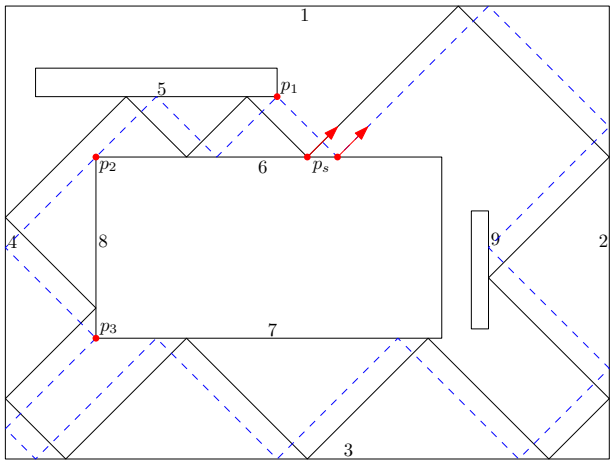


Figure 9: The dashed blue path is the translation of the billiard path.

Note that it not possible that a translated path touches a corner of the outer rectangle *before* a corner of an inner rectangle. If there exists such a path (see the blue dashed path in Figure 10), then a bit before touching the corner of the outer rectangle the path must lead to self-intersection, which is not a valid billiard path.

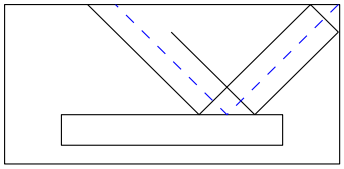


Figure 10: A path touches a corner of the outer rectangle before a corner of an inner rectangle.

If the translated path touches a corner of the outer rectangle, where at the same time it touches a corner of an inner rectangle, then the path reflects to itself (a degenerate case); we interpret this case which will have the same signature as before. Consider the billiard path in Figure 11 (with signature $S = \dots 13124 \dots$). We can say that the translated path first reflects to the rectangle side 3 (Case 2), next touches both rectangle sides 1 and 2, respectively, at the cornet of the outer rectangle, and finally reflects to the rectangle side 4 (Case 3). This implies that both Case 2 and Case 3 occur at the same time (and we say that the signature of the translated path is still $S = \dots 13124 \dots$). □

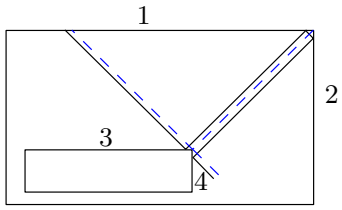


Figure 11: A translated path touches both a corner of the outer rectangle and a corner of an inner rectangle.

From above lemma, the following results:

Corollary 4 *There exists a rectangle corner from which a shooting ray can produce the maximum-length signature.*

Theorem 5 *For a collection of axis-aligned rectangles, all enclosed in one rectangle, with a total of n edges, there are at most $1.5n - 6$ distinct signatures.*

Proof. From Lemma 3, over translation of a path to the *right*, each signature S can be charged to a corner (one of the three cases, in Figure 8) of a rectangle. Thus we have at most $3(n-4)$ distinct signatures.

Note that by translating the path in the opposite direction (*i.e.,* translating to the *left*), without changing the signature S, the path touches another corner. This implies that each signature is counted twice. Thus the number of distinct signatures is at most $3(n-4)/2$. □

4 Discussion

For a set of axis-aligned rectangles with a total of n edges, we gave an upper bound $4.5n - 9$ for the maximum length of a signature, showed an arrangement of rectangles with signature length $1.25n + 2$, and proved that the number of distinct signatures is at most $1.5n - 6$. It would be interesting to improve these bounds. Also, another interesting question is to provide an efficient algorithm to find the maximum-length signature.

Acknowledgement We would like to thank the anonymous reviewers for their suggestions and comments.

References

[1] Joseph O'Rourke. Open Problems from CCCG 2017. In Proc. of the 30th Canadian Conference on Computational Geometry, pages 149-154, 2018.

[2] Mark de Berg, Otfried Cheong, Marc van Kreveld, Mark Overmars. Computational Geometry: Algorithms and Applications. Springer-Verlag TELOS, Santa Clara, CA, 2008.

Solving the convex hull problem progressively in the external memory model

Amir Mesrikhani* Mohammad Farshi*†

Abstract

Executing an algorithm with a massive input data may cause a high waiting time to see the output of the algorithm. To decrease the waiting time, progressive algorithms produce partial solutions which approximate the final output during executing of the algorithm. In this paper, we propose a progressive algorithm for computing the convex hull of the set of N points in the plane in the external memory model. This algorithm consists of $O(\log_{M/B} N/M)$ steps, each step takes $O(N/B)$ I/O operations. The upper bound of the partial solution error in step r is $\frac{D^2}{(M/B)^r}$ where M is the size of main memory, and B is the block size, and D is the diameter of the input points set.

1 Introduction

In Random Access Memory (RAM) model, we always assume that all input data is maintained in the main memory. Therefore, the performance of an algorithm in this model is usually measured by the time or space complexity. Many modern applications process and maintain massive data which are not fit into the main memory. Therefore, massive data must be stored in the secondary memory like disks. In this issue, the number of Input/Output operations (denoted by I/O in the rest of the paper) between the main memory and disk is the bottleneck of the performance, because I/O in the secondary memory is much slower than the computation speed. So, in this case, if one reduces the number of I/O interactions of an algorithm, even if it adds some extra computations, it will improve the running time of the algorithm. Adding the progressiveness to the algorithm also makes the algorithm more practical, because it gives the partial solutions during the execution of the algorithm.

To clarify the importance of designing progressive algorithm, consider an algorithm that faces with the large input data. The user of this algorithm must wait until all steps of the algorithm are executed to see the last output of the algorithm. Because of the large input data, the waiting time might be very high. One way to decrease this time is producing partial solutions in the particular steps of the algorithm. Based on the error of these solutions, the user can stop running the algorithm. Algorithms that solve a problem step by step and generate partial solutions with an explicit upper bound on the error value are called progressive algorithms.

Modeling memory system exactly with a main memory and a disk is a complex task. The main property of the disk that must be notified is the high random access time to the data compared to the main memory. So instead of transferring data one by one between the main memory and the disk, continuous blocks of data should transfer between them. In this paper, we use a standard external memory model that contains one main memory and one disk with the following parameters:

- N: size of the input data.
- M: size of the main memory.
- B: size of the block.

1.1 Progressive algorithms

Progressive algorithms report a partial solution to the user in some middle steps of the algorithm.

The quality of a progressive algorithm is measured according to the two following parameters:

- The quality of the partial solution which is measured according to the error function *err*. The function *err* takes a partial solution and outputs a nonnegative value as the error.
- The speed of convergence to the final solution which is measured by the convergence function f_{conv}. This function takes the step number and specifies the upper bound for the error value of the partial solution.

In 2014, Alewijnse et.al [1] proposed the following formal definition for progressive algorithms.

Definition 1 (progressive algorithm) *A k-step progressive algorithm with convergence function f_{conv} and the error function err is an algorithm that outputs the partial solution s_r in step r such that $err(s_r) \leq f_{conv}(r)$.*

They also clarified the relation between the progressive algorithms and other well-known algorithms like anytime and approximation algorithms [11]. In the context

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

†corresponding author

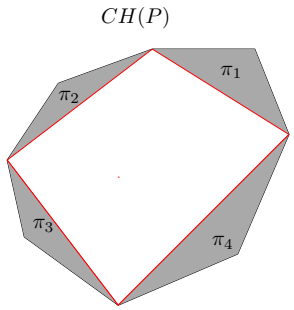


Figure 1: The red polygon is $poly_r$ and $\{\pi_1, \pi_2, \pi_3, \pi_4\}$ are the portions of $CH(P)$ that do not belong to $poly_r$.

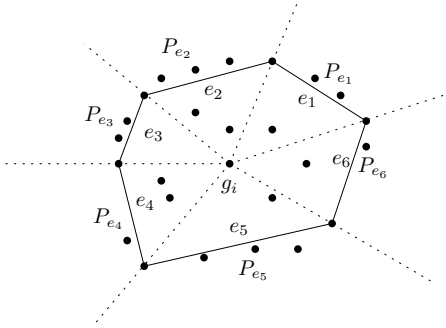


Figure 2: Edges of a polygon and the corresponding cones.

of the progressive algorithms, some geometric problems like finding k-popular places in a set of trajectories, the convex hull problem, Euclidean minimum spanning tree and the group Steiner tree search were studied in the RAM model [3, 4, 7, 10]. In the external memory model, Mesrikhani et.al [9] designed a progressive algorithm for sorting a set of N elements. Their algorithm consists of $O(\log_{M/B} N/B)$ steps and takes $O(N/B)$ I/O in each step. Also, the convergence function of their algorithm is $\frac{N}{(M/B)^{r/2}}$. Goodrich et.al [5] designed a non-progressive algorithm for the convex hull problem in the external memory model.

In this paper, we design a progressive algorithm for computing the convex hull of a set of N points in the plane in the external memory model firstly. Our progressive algorithm contains $O(\log_{M/B} N/M)$ steps, such that each step takes $O(N/B)$ I/O operations. The convergence function of this algorithm is $\frac{D^2}{(M/B)^r}$ where D is the diameter of the input points.

2 The progressive algorithm for the convex hull problem

To design the algorithm, first, we define the partial solution and the error function for the convex hull problem. Let $CH(P)$ be the exact convex hull of P. In step r of the algorithm, the polygon $poly_r$ will be reported to the user, such that $poly_r \subseteq CH(P)$. There exist various criteria to measure the similarity of two polygons like area, width, Hausdorff distance between them, etc. Let $\pi_i, \ldots, \pi_k$ denote the portions of $CH(P)$ that are not contain in $poly_r$ (see Figure 1). In this paper, we use the following error function

$$err_{area}(poly_r) = \max_{i=1,\ldots,k} |\pi_i|, \qquad (1)$$

where $|\pi_i|$ denotes the area of π_i. The vertices of the $CH(P)$ are the extreme points in some direction. So in each step of the progressive algorithm, some vertices of the $CH(P)$ are computed. Consequently, if the algorithm consists of k steps, we have:

$$poly_1 \subseteq poly_2 \subseteq \cdots \subseteq poly_k = CH(P).$$

The idea of the progressive algorithm is as follows. We know that the vertices of the $CH(P)$ are extreme points in some direction, so we use this property in our algorithm. More precisely to compute some vertices of $CH(P)$ in step r, consider the set of equally spaced direction $D(r) = \{i \times (2\pi/(M/B)^r) : 0 \leq i < (M/B)^r\}$. We use the set of directions D in each step of the progressive algorithm to find some vertices of $CH(P)$. If we only consider D, the progressive algorithm may require so many steps to detect all vertices of $CH(P)$ and this is exhausting for the users of the algorithm since they must wait a lot of time to see the $CH(P)$. To reduce the number of steps, we need to prune some points that are placed inside the $CH(P)$. Given a direction d, we can use a linear programming with points of P as the constraints to compute an edge of $CH(P)$ intersected by a ray with direction d in $O(N/B)$ I/O operations [3, 8]. So we use the linear programming and finding extreme points in $D(r)$ to reduce the number of steps of the progressive algorithm.

Let g_i be the geometric centroid of the vertices of $poly_i$. The point g_i is inside $poly_i$, since $poly_i$ is a convex polygon. Consider the lines through g_i and each vertex of $poly_i$. These lines assign a cone to each edge of $poly_i$. For any edge e in $poly_i$, let $P_e \subseteq P$ be the points that are inside the cone corresponding to e but outside $poly_i$, see 2.

In the first step of the algorithm, consider M/B directions in $D(1)$. We scan the point set P in blocks of size B. For each block, we compute the extreme point in each direction $D(1)$ and save it into the main memory. If the extreme points is better than the extreme points we have comuputed so far, then we update them. So by scanning all points of P, the extreme points in each direction are computed in $O(N/B)$ I/O operations. Finally, since we have all extreme points in the main memory we can compute the convex hull of the

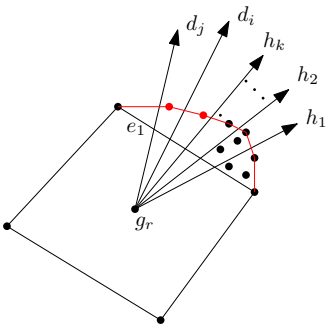


Figure 3: e_1 is replaced with the red chain. The red points are the extreme points in $d_i, d_j \in D(r)$ and the black points are the endpoints of the edges of $CH(P)$.

extreme points without any I/O cost and report it as the first partial solution.

In generic step $r > 1$, we do the following operations:

1. Distribute each new direction in $D(r)$ into the cone of the edges of $poly_{r-1}$. By definition of $D(r)$, in each cone, we have M/B new directions. To compute the extreme point in each new direction, P_e is being scanned block by block like the first step. So for each edge e of $poly_{r-1}$, we can obtain all extreme points of P_e in M/B directions in $O(\#P_e/B)$ I/O operations where $\#P_e$ denotes the number of points in P_e.

2. For each new direction in $D(r)$ and associated edge e of $poly_{r-1}$, consider the set P_e and do the followings:

   (a) If $\#P_e$ is at most M, then compute the convex chain of P_e together with the endpoints of e and replace e by it.

   (b) Otherwise, let L be a horizontal line through g_{r-1}. Suppose s_i be the line through g_{r-1} and $p_i \in P_e$. For each s_i, $i = 1, 2, \ldots, \#P_e$, compute the angle between s_i and L, and denotes it by θ_i (see Figure 4). Distribute P_e into subsets of equal size by computing splitting rays $h_1, \ldots, h_k$ emanating from g_{r-1} according to θ_i, $i = 1, \ldots, n$, for $k = \lceil\sqrt{M/B}\rceil$. Denote these subsets by $P_{e_1}, \ldots, P_{e_k}$. Then determine the edge e_i of $CH(P)$ that intersected by h_i, for $i = 1, \ldots, k$, using a linear programming that the points of P_{e_i} induces the constraints [8] (see Figure 3).

   (c) In the worst case, we have M/B extreme points from Operation 1 and at most $2 \times \sqrt{M/B}$ endpoints from the computed edges of Operation 2. So they fit into the main memory and we replace e by the convex chain of them (See Figure 3).

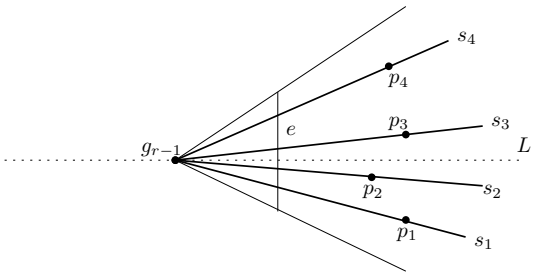


Figure 4: Lines correspond to each point of P_e.

To compute the set $P_{e'}$ for each new edge e', we do the following. In step r of the algorithm, each edge of $poly_{r-1}$ is replaced with at most $M/B + \sqrt{M/B} - 1$ new edges. For each edge $e \in poly_{r-1}$, we scan P_e and assign its member to the new edges. This could be done in $O(\#P_e/B)$ I/O operations. Consequently, this work takes $O(\sum_e \#P_e/B) = O(N/B)$ I/O in total.

To obtain the convergence ratio, first, we should show that in any step of the algorithm, $poly_r$ is a convex polygon.

Lemma 1 *If $poly_r$ is the partial solution of the algorithm in step r then $poly_r$ is a convex polygon.*

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

2.1 Finding $\sqrt{M/B}$ splitting elements for a set of N real number

Let $R = \{s_1, \ldots, s_N\}$ be a set of N real numbers. The goal is partitioning R into $k = \lceil\sqrt{M/B}\rceil$ roughly equal size subsets $R_1, \ldots, R_k$ such that all members of R_i is less than or equal to all members of R_{i+1}, for all $1 \le i < k$. We use Aggarwal et.al [2] algorithm for it. The idea of the algorithm is based on sampling from R. To do so, N/M sorted lists are created from R that fit into the main memory. Then, from the sorted lists, every $\frac{1}{4}\sqrt{M/B}$'th element is chosen. By this, we have a sample list with size $\frac{4N}{\sqrt{M/B}}$. The k'th smallest element algorithm is calling over the sample list $\sqrt{M/B}$ times and every $\frac{4N}{M/B}$'th element is chosen as splitting elements. Finally, by scanning R, we can distribute the elements into the corresponding subsets. Since the size of the sample list is $\frac{4N}{\sqrt{M/B}}$ and each calling the k'th smallest element algorithm over a list with C elements takes $O(C/B)$ I/O operations, the I/O complexity of this algorithm is:

$$\sqrt{M/B} \times O(\frac{N}{\sqrt{M/B} \times B}) = O(N/B).$$

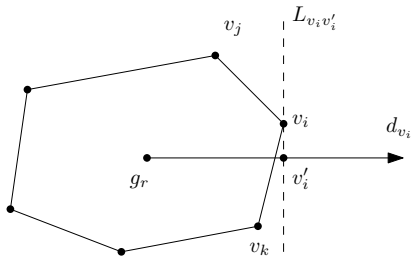


Figure 5: Illustration of the proof of Lemma 1.

2.2 The convergence function

To analyze the error of the partail solution in any steps, we introduce the uncertainty triangles of the partial solution. Then we will obtain an upper bound of the error using these triangles. Let $poly_r$ be the partial solution in step r with k vertices. Consider two consecutive vertices p_i and p_{i+1} of $poly_r$ in the counter clockwise order. Assume p_i and p_{i+1} be the extreme points in the direction d_i and d_{i+1}. The supporting lines of p_i and p_{i+1} are the lines passing p_i and p_{i+1} that are perpendicular to d_i and d_{i+1}. The triangle bound by $\overline{p_ip_{i+1}}$ and supporting lines of p_i and p_{i+1} is the uncertainty triangle of $\overline{p_ip_{i+1}}$. In similar way, all edges of $poly_r$ define an uncertainty triangle. Hershberger et.al [6] showed that the height of the uncertainty triangles of any edges is $O(D/k)$, where D is the diameter of the points set. Trivially, all vertices of the final convex hull contained in the uncertainty triangles.

The following theorem specifies the convergence function of the progressive algorithm.

Lemma 2 *Let $poly_r$ be the partial solution in step r then $err_{area}(poly_r) = O(\frac{D^2}{(M/B)^r})$, where D is the diameter of the input points set.*

Proof. By Lemma 1, the polygon $poly_r$ is a convex polygon and has $O((M/B)^r)$ edges in step r. So the number of the uncertainty triangles of $poly_r$ is $O((M/B)^r)$. The area of the uncertainty triangle T is $\frac{D}{2(M/B)^r} \times b$ where b is the base of T. The value b is not bigger than D. So the upper bound of the area of T is $O(\frac{D^2}{(M/B)^r})$. Consequently, the upper bound of $err_{area}(poly_r)$ is:

$$err_{area}(poly_r) \leq \frac{D^2}{(M/B)^r}$$

and the proof is complete. □

Theorem 3 *There exists an external memory progressive algorithm for computing convex hull of set of N points in the plane that consists of $O(\log_{\frac{M}{B}} \frac{N}{M})$ steps. The convergence function is $O(\frac{D^2}{(M/B)^r})$ with respect to err_{area} and the algorithm spends $O(N/B)$ I/O in each step.*

Proof. In operation 1 of the algorithm, by traversing the edges of the $poly_{r-1}$, we can determine the cone that contains each new direction. So it takes $O((M/B)^r)$ I/O. We know that $CH(P)$ has N edges in worst-case; therefore traversing and extracting the edges of $poly_{r-1}$ never takes more than $O(N/B)$ I/O. Finding the extreme points in all cones of the edges of $poly_{r-1}$ takes $O\left(\sum_e \#P_e/B\right) = O(N/B)$ I/O. In operation 2, for each cone corresponds to the edges of $poly_{r-1}$, we compute the intersected edges by $k = \sqrt{M/B}$ rays by Megiddo's algorithm. He proposed an algorithm for solving linear programming in the fixed dimension by extending the algorithm for finding the median of a set of real numbers [8]. The median of a set of N numbers in the external memory model takes $O(N/B)$ I/O operations [2]. Therefore, computing the intersected edge in each cone of the edge $e \in poly_{r-1}$ takes $O(\sum_{i=1}^{k} \#P_{e_i}/B) = O(\#P_e/B)$. So the total I/O operations for this work for all edges of $poly_{r-1}$ is $O(N/B)$. Also in operation 2, finding splitting rays could be done in $O(N/B)$ I/O by using the algorithm of Section 2.1.

In step r, we know that $\#P_e \leq \frac{N}{(\sqrt{M/B})^r}$. If $\#P_e = M$ then the convex hull of P_e will be computed directly. Therefore, the splitting stops in step r, if $\frac{N}{(\sqrt{M/B})^r} = M$, so the number of steps is:

$$r = \log_{\sqrt{M/B}} N/M = O(\log_{M/B} \frac{N}{M}). \qquad \square$$

3 Conclusion

In this paper, an external memory progressive algorithm is proposed for finding the convex hull of a set of N points in the plane. This algorithm consists of $O(\log_{\frac{M}{B}} \frac{N}{M})$ steps and takes $O(N/B)$ I/O in each step. The upper bound on the error of the partial solution in step r is $O(\frac{D^2}{(M/B)^r})$, when D is the diameter of the input points set.

References

[1] A. Aggarwal, B. Alpern, A. Chandra, and M. Snir. A model for hierarchical memory. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 305–314. ACM, 1987.

[2] A. Aggarwal and J. Vitter. The Input/Output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[3] S. P. A. Alewijnse, T. M. Bagautdinov, M. de Berg, Q. W. Bouts, A. P. ten Brink, K. Buchin, and M. A. Westenberg. Progressive geometric algorithms. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, SOCG'14, pages 50–59, New York, USA, 2014. ACM.

[4] M. Benkert, B. Djordjevic, J. Gudmundsson, and T. Wolle. Finding popular places. *International Journal of Computational Geometry & Applications*, 20(01):19–42, 2010.

[5] M. T. Goodrich, J.-J. Tsay, D. E. Vengroff, and J. S. Vitter. External-memory computational geometry. In *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*, pages 714–723. IEEE, 1993.

[6] J. Hershberger and S. Suri. Adaptive sampling for geometric problems over data streams. *Computational Geometry*, 39(3):191–208, 2008.

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

[8] N. Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM (JACM)*, 31(1):114–127, 1984.

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

[10] A. Mesrikhani, M. Farshi, and M. Davoodi. Progressive algorithm for euclidean minimum spanning tree. In *Proceedings of the 1st Iranian Conference on Computational Geometry*, February 27, 2018.

[11] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI magazine*, 17(3):73, 1996.

Appendix

We prove Lemma 1 in this part.

Lemma 1. *If $poly_r$ is the partial solution of the algorithm in step r then $poly_r$ is a convex polygon.*

Proof. Let v_k, v_i, v_j be three consecutive vertices of $poly_r$, in the partial solution at step r. By the progressive algorithm, v_i is an extreme point in a direction of $D(r)$ or an endpoint of an edge of $CH(P)$. Let v_i be an extreme point in a direction d_{v_i} (If v_i is an endpoint of an edge of $CH(P)$, then $d_{v_i} \notin D(r)$). Assume v_i' be the projection of v_i onto a ray with direction d_{v_i} emanating from g_r. Assume $L_{v_i v_i'}$ be a line passing through v_i and v_i' (see Figure 5). All vertices of $poly_r$ are lying on same side with v_k and v_j, since v_i is an extreme point in d_{v_i}. So $\measuredangle v_k v_i v_j$ is less than 180. Consequently, $poly_r$ is a convex polygon. □

Some Results on the Fewest Net of Simplicial Polyhedra

Mohammad Asgaripour* Ali Mohades†

Abstract

Given polyhedron, consisting of n vertices and F faces, what is the smallest number k such that the polyhedron can be divided to k parts by cutting its edges so each part can be unfolded to a simple polygon without overlapping? Although this question is open, there is a conjecture that claims the answer is 1 for any convex polyhedron. Indeed the fewest net problem is about finding an upper bound of this number of parts as a function of F and/or n. For the general polyhedra the best achieved result is $3F/8$. Also for simplicial polyhedra this result is $4F/11$ up to now. We will show that every simplicial polyhedra can be unfolded in at most $F/4$ parts.

1 Introduction

This classic problem: "Is there edge unfolding for any convex polyhedron or not?" is still open. Though mentioned by Albert Durer in 16th century [1], this problem was for the first time stated formally in 1975 by Shepherd:

Conjecture 1 *Every convex polyhedron can be cut along edges and unfolded into a non-overlapping net. [2]*

This conjecture is incorrect for non-convex polyhedra and there exist some non-convex polyhedra that cannot be unfolded in this way.
If we relax the condition of unfolding in one part, we will have the fewest net problem that was stated by O'Rourke.
The Fewest Net Problem Given a convex polyhedron with n vertices and F faces, what is the fewest number of pieces, each of which unfolds to a simple polygon, into which it may be cut by slices along edges? [3] Our objective is to find an upper bound as a function of n or F.

Conjecture 1 states that this number is always 1; however the truth of this conjecture is still an open problem. On the other hand the trivial upper bound for this problem can be reached by cutting all the edges of polyhedra, and this upper bound is obviously F. If the discussing polyhedron was a simplicial one, that is all of its faces were triangles, the dual graph of it would be a cubic, bridge-less graph. Due to the Peterson theorem, there would be a perfect matching for this kind of graph. It means that we can separate all the vertices of these graphs two by two so that every pair of vertices become adjacent in the graph. It is clear that every pair of faces, related to one of these pairs, can be unfolded without any overlap. So a simplicial polyhedron can be unfolded in $\lceil F/2 \rceil$ distinct pieces.
The best result for this problem is obtained at 2007, by Val Pinciu. Using graph domination, he showed that every convex polyhedron can be unfolded in not more than $3F/8$ non-overlapping net. Also he has shown that for simplicial polyhedra, $4F/11$ is an upper bound for the number of nets to unfold without overlap. [4]

2 Definitions

We define a polyhedron as a connected set of closed planar polygons called faces, in the 3-dimmentional space such that: (1) the intersection of two faces is a set of vertices and edges common to both polygons (2) every edge is shared by at most two faces. All polygons in this paper are considered closed i.e. every edge is shared by exactly two faces. We will denote the faces of the polyhedron by $A, B, C, ...$, and the vertices by $u, v, w, ...$ .The edge ended with two vertices u and v will be denoted by $\overline{uv}$. For any face A of a polyhedron, π_A is the plane that contains the polygon A.
Notice that a polyhedron is convex if its interior is a convex set, that is, for any two points u and v interior to the polyhedron, the segment $\overline{uv}$ is interior. If a polyhedron is convex, then for any distinct faces A and B, all the points of A that are not in B would be on the same side of the plane π_B. For any face A of a polyhedron we will denote by $N[A]$ the subset of faces of the polyhedron, induced by A and the faces adjacent to it.
We define $N'[A]$ to be the subset of faces of the polyhedron, such that each member of it (1) either is A, (2) or has an edge in common with A (3) or has a vertex in common with A. We also define $N^2[A]$ in polygon P as a subset of its faces, such that the distance between each member of this subset and A, in the dual graph

*Laboratory of Algorithm and Computational Geometry, Department of Mathematics and Computer Science, Amirkabir University of Technology., asgaripour@aut.ac.ir

†Laboratory of Algorithm and Computational Geometry, Department of Mathematics and Computer Science, Amirkabir University of Technology., mohades@aut.ac.ir

of the faces of P, is less than or equal by 2. Due to the properties of simplicial complexes, we can observe that $N^2[A]$ is a subset of $N'[A]$ in this kind of polyhedra.

3 Volcano Unfolding and improving the result for simplicial polyhedra

$N[A]$ can be unfolded by cutting along its edges, except the edges bounding A. This unfolding blows out all the faces adjacent to A around it. It is called Volcano Unfolding. Val Pinciu proved following theorem in 2007 [4]:

Theorem 1 *Let A be any face of a convex polyhedron. Then the volcano unfolding of $N[A]$ is non-overlapping.*

Now we try to improve the result by showing more powerful unfolding for simplicial polyhedra.

Conjecture 2 *For each face A of a simplicial polyhedron, $N'[A]$ can be unfolded in plane without any overlap by cutting its edges.*

Indeed the conjecture 2 will neither be proved nor disproved here. It is here just because of its inspiration for next results, the results that cover part of this conjecture, but this part is the one that we need for the main proof.
In volcano unfolding a face of polyhedron and all of its neighbors unfold in a plane without any overlap during doing it. Lemma 2 is a generalization of this related to $N'[A]$.

Lemma 2 *For each face A of a convex polyhedron P, $N'[A]$ can be unfolded in a plane without any overlap, by cutting its edges in a way that in the final result, the faces having an edge in common with A will have the same edge in common and also the face that have a vertex in common with A have the same vertex in common.*

Proof. At first we put the face A of P on plane. After that we intersect plane π, parallel to A and with height of ε form it. We choose ε small enough, such that π has intersection with just the edges that have one endpoint on A. This is always possible. It is sufficient to compute the height of every vertex from the plane that P is put on it, and choose ε something smaller than all of them. We define P' as a part of P that occurred above π, and also A' as a part of π that make a face of P'. The outcome polyhedron P' is convex, because it is intersection of two convex regions.
Due to the property of ε, each vertex of A' will have a degree of three. Therefore $N[A']$ and $N'[A']$ would be equal in P'. On the other hand because of theorem 1 we know that $N[A']$ can be unfolded without any overlap in a plane by cutting its edges.
Consider Volcano Unfolding of $N[A']$. If we make ε enough close to zero, members of $N[A']$ will be close to members of $N'[A]$. The only event that happens when ε becomes zero is that the edges of A that are in common with faces of $N[A']$ and not $N[A]$ would change to vertex. And because of that the result will not be a simple polygon. □

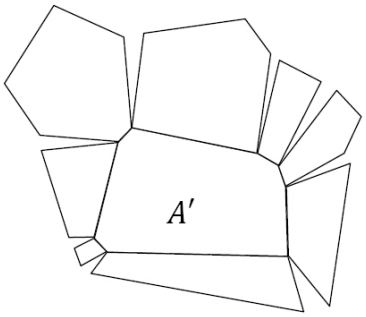


Figure 1: Unfolding of $N[A']$

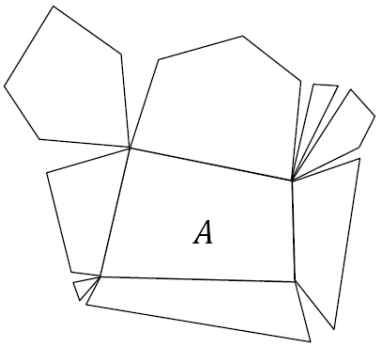


Figure 2: Unfolding of $N'[A]$

To be more intuitive Figure 1 and Figure 2 show the unfolding of $N[A']$ and $N'[A]$ respectively.

Simplicial polyhedra are the ones that all of their faces are triangles and because of that they have a following property; Considering dual graph of faces of a simplicial polyhedron, faces related to two nodes in the distance of two from each other sure have a vertex in common. Figure 3 illustrate this property.

Due to the aforementioned property of simplicial polyhedra we specify the conjecture 2 and state the theorem 3.

Theorem 3 *For each face A of a simplicial polyhedron, $N^2[A]$ can be unfolded properly by cutting its edges.*

Proof. If there is an overlap it cannot be out of three cases. Other cases are either equivalence to these three or cannot be happened because of contradiction with

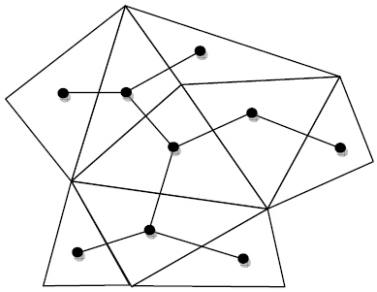


Figure 3: Each face with distance two from the central triangle have a vertex in common with it

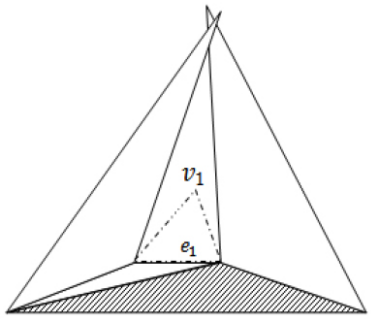


Figure 4: Collision of two triangles with distance of one and two from A. A is striped triangle.

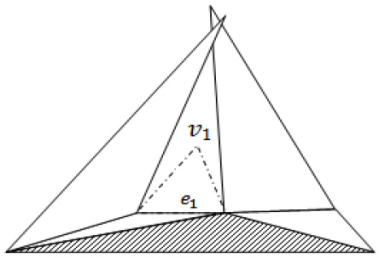


Figure 5: Collision of two triangles with distance of two from A. A is striped triangle.

theorem 1. The first case is depicted in Figure 4. In this case in dual graph of faces, one of two collided face has distance of 1 from A and the other has distance of 2. In the second and third cases each of two collided faces has distance of 2 from A. These two cases are depicted in Figure 5 and Figure 6 respectively.

In the first and second cases, there exists an edge like e_1 that its endpoints belong to two collided triangles. Each edge of polyhedron is belonged to two faces, and so there should be another triangle that contains e_1. This triangle should be created by adding a vertex like v_1 and it is depicted by dashed line in Figure 4 and Figure 5. We call this triangle, t. v_1 could not

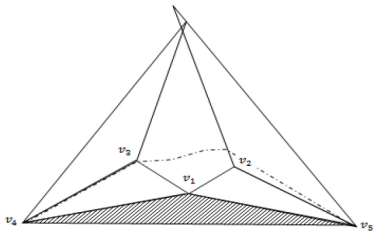


Figure 6: Collision of two triangles with distance of two from A. A is striped triangle.

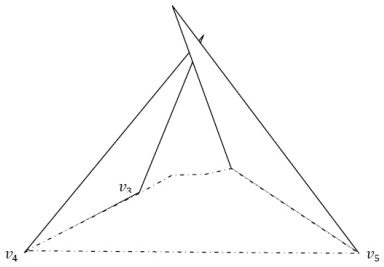


Figure 7: Unfolding of cut polyhedron. Collision of two triangles that are adjacent of a face.

be out of the region between two collided triangles, because in that case sum of the angles of faces in one of the endpoints of e_1 would become more than 2π and it is a contradiction with the assumption of convexity of polyhedron.

In both of the Figure 4 and Figure 5, every triangle has an edge or a vertex in common with t. Considering the arrangement of triangles, we can see that two collided triangles cannot be rotated to avoid collision when they are still connected by their vertices to t. It is contradiction with lemma 2 that states that for each face A of every simplicial polyhedron, $N^2[A]$ can always be unfolded without any overlap and with conserving vertex and edge connections. As a result, these two cases cannot be happened.

In the third case, the non-adjacent vertex of A with two collided triangle, -like v_1 in Figure 6-, cannot have the degree of 3, because it means that v_2 and v_3 should occur on each other in the polyhedron and sum of their angles must be more than 2π. It means that the curvature of this vertex would be negative and it is against the assumption of convexity.

If the degree of v_1 is 4, the other triangle on it, would have a vertex in common with both collided triangles. So they cannot be taken place in plane without any overlap and with conserving their connectivity. This case also cannot be happened.

Finally if the degree of v_1 is greater than 4; we consider two planes. First, the plane contains v_4, v_5, and v_2 and second, the plane contains v_4, v_5, and v_3. These

two planes have the edge $\overline{v_4v_5}$ in common. Among these two, we choose the one that is above other one in vertices v_2, and v_3. Without loss of generality, we consider the one that passes v_3 as the proper one. With cutting the polyhedron by this plane, a new face will be appeared that is adjacent with two collided triangles. Dashed lines in Figure 6 shows the intersections of the new face with existing faces. The resultant polyhedron is convex.
If we put the new face and two collided triangles in a plane, the Figure 7 will be appeared. In this case the distance between vertices v_4 and v_5 will not change and the angels shown in the Figure at these two vertices will be decreased. So, we can observe that if two triangles are collided in the primary case, they also will be collided here. Note that both of them have an edge in common with the new face, and these adjacencies are contradictory with convexity of polyhedron and theorem 1. So this case also cannot be happened. □

Using achieved results we state the following theorem:

Theorem 4 *Every simplicial polyhedron, consist of F faces, can be unfolded in $\lceil F/3 \rceil$ distinct pieces.*

Proof. The dual graph of a simplicial polyhedron is 3-regular graph. Each spanning tree of this graph has the maximum degree of 3. And it is clear that each of these trees has some vertices with degree of 1. Consider a spanning tree of the dual graph. We choose a vertex with degree of 1 as root and make directed binary tree. Finding the deepest vertex of this tree we choose its grand ancestor, in two lever upper, as a central vertex and omit it with all the vertices below it; because this vertex has the distance of at most 2 with all of its children and grandchildren. We also check upside of this central vertex to distance 2 from it, if it did not make the tree disconnected we would also omit them. In this way in each level we choose at least three faces and unfold them in a piece.
In the last step if any number of vertices of tree less than six remains, we can choose all of them and unfold them in a piece. So every simplicial polyhedron, consisting of F faces, can be unfolded in $\lceil F/3 \rceil$ distinct pieces. □

4 More Improvement

For a polyhedron P, we define the set of all faces that include the vertex with the highest height of polyhedron, L_1. Omitting L_1 from polyhedron some of its edges will be in one side and in fact each of them will be adjacent with just one of the remaining faces. These edges and related vertices make a simple closed path. We call this closed path R_1. Also after omitting L_1, we call the rest set of P's faces, P_1.
We define as L_2, the set of all faces of P_1 that have vertex or edge at R_1. In the same way and recursively, we define as R_{i+1}, the subset of P_i, that is achieved with omitting L_{i+1} from it. We define as R_i, the path that after omitting L_i is adjacent with P_i in just one side. And we define as L_{i+1}, the set of faces of P_i that have some vertices or edges in common with R_i.
One can observe that except of the last step, all R_i is consist of at least three edges, otherwise polyhedron cannot be continued. Also the number of faces of each layer L_i, unless the first and the last, is equal to the sum of R_i's and R_{i+1}'s edge numbers. This is because of the fact that each face of L_i has an edge either on R_i or on R_{i+1}. Therefore except of the first and the last layer each L_i has at least six faces.
Every two consequent layer L_i and L_{i+1} have intersection in the edges of R_i and on the other hand for each L_i there is closed path that is subset of the dual graph of P's faces and the vertices of that path are mapped with faces of L_i. So we can conclude that:

Lemma 5 *For each two consequent layer L_i and L_{i+1} of the polyhedron P, there exist a path in the dual graph of P's faces such that its vertices can be mapped onto faces of these two layers. It is clear that the number of vertices of each of these paths is at least 12.*

Theorem 6 *Every simplicial polyhedron can be unfolded in $[F/4 + C]$ distinct pieces.*

Proof. Due to theorem 3, every chain of 5 connected faces can be unfolded properly. Due to the lemma 5, every two consequent intermediate layers of polyhedron's faces, make a connected chain with length of at least 12. Due to the mentioned note, every chain with length of 12 can be unfolded properly in 3 pieces. And any greater number for length of chains would either make this ratio better or not change it -having no change is just for the length of 16-. □

References

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

[2] G.C. Shephard. Convex Polytopes with Convex Nets. *Mathematical Proceedings of the Cambridge Philosophical Society*, 78(1975), no. 3, 389-403.

[3] E. Demaine and J. O'Rourke. Open Problems from CCCG 2003. *Proc. 16th Canad. Conf. Comput. Geom.*, Concordia(2007), 209-211.

[4] V. Pinciu. On the Fewest Nets Problem for Convex Polyhedra. *Proceedings of the 19th Canadian Conference on Computational Geometry (CCCG2007)*, pages 21-24, 2007.

Local Geometric Spanners

M. A. Abam* M. S. Borouny* A. Mousavi*

Abstract

We introduce the concept of local spanners for planar point sets with respect to a family of regions, and prove the existence of local spanners of small size for some families. For a geometric graph G on a point set P and a region R belonging to a family $\Re$, we define $G \cap R$ to be the part of the graph G that is inside R. A local t-spanner with respect to $\Re$ is a geometric graph G on P such that for any region R in $\Re$, the graph $G \cap R$ is a t-spanner with respect to $G_c(P) \cap R$, where $G_c(P)$ is the complete geometric graph on P. We prove that any set P of n points for any constant $\varepsilon > 0$ admits a local $(4+\varepsilon)$-spanner of size $O(n \log^2 n)$ and a local $(1+\varepsilon)$-spanner of size $O(n \log n)$ with respect to squares and vertical slabs, respectively. If adding Steiner points is allowed, then local $(1+\varepsilon)$-spanners with $O(n)$ edges can be obtained for squares. Moreover, if adding Steiner points is allowed and regions are disks, then we can obtain a local $(1+\varepsilon)$-spanner of size $O(n \log^2 n)$ using $O(n)$ Steiner points.

1 Introduction

A geometric graph G on a set P of n points on the plane is a t-spanner (for the given $t > 1$) if for any two vertices p, q of G we have $d_G(p, q) \leq t \cdot |pq|$ where $d_G(p, q)$ and $|pq|$ denote the shortest distance of p and q on G, and the Euclidean distance of p and q, respectively. G is called a local t-spanner with respect to a family $\Re$ of regions if for any region $R \in \Re$, the graph $G \cap R$ is a t-spanner for $G_c(P) \cap R$, where $G_c(P)$ is the complete geometric graph on P. For some families of regions, G must have $O(n^2)$ edges to be local. The family of rectangles is such a family. As depicted in Fig 1, there are $O(n^2)$ different rectangles containing just two points; therefore, these pairs of points must be connected in G. In this paper, we focus on some specific families, namely vertical slabs, squares and disks, and present local t-spanners with near-linear sizes.

*Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.

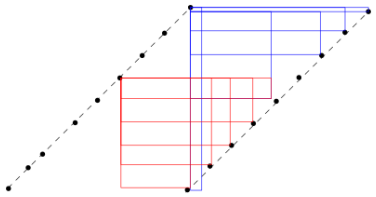


Figure 1: An example showing if regions are rectangle, the $O(n^2)$ edges are required.

We shall also consider the case where we are allowed to add Steiner points to the graph. In other words, instead of constructing a geometric network for P, we are allowed to construct a network for $P \cup Q$ for some set Q of Steiner points. In this case, we only require short connections between the points in P. Thus, we say that a graph G on $P \cup Q$ is a local Steiner t-spanner with respect to $\Re$ for P if, for any $R \in \Re$ and any two points $u, v \in P \cap R$, the distance between u and v in $G \cap F$ is at most t times their distance in $G_c(P) \cap R$.

As a close work, Abam *et al.* [2] presented a region-fault $(1+\varepsilon)$-spanner of size $O(n \log n)$ with respect to half-planes. This is indeed a local spanner with respect to half planes.

In Section 2, we present a general method to convert a well-separated pairs decomposition (WSPD) [4] for P into a local t-spanner with respect to $\Re$ for P. In section 3, we construct a local $(1+\varepsilon)$-spanner of size $O(n \log n)$ when regions are vertical slabs. In section 4, we consider square regions, and show it is possible to construct a local $(4+\varepsilon)$-spanner of size $O(n \log^2 n)$. Moreover, we show if Steiner points are allowed, we can reduce the number of edges to be $O(n)$. We dedicate Section 4 to disk regions, and show by adding $O(n)$ Steiner points, we can construct a local $(1+\varepsilon)$-spanner of size $O(n \log^2 n)$.

2 Constructing local spanners using WSPD

In this section, we show a general method to obtain a local spanner from a well-separated-pair decomposition of a point set P defined below.

Definition 1 *[4] Let P be a set of n points in the plane and let $s > 0$ be a real number. Two*

point sets A and B in the plane are well-separated with respect to s, if there are two disjoint disks of the same radius that each of them cover each point set and their distance are at least s times of their radius, A well-separated pair decomposition (WSPD) for P with respect to s is a collection $W := (A_1, B_1), \ldots, (A_m, B_m)$ of pairs of non-empty subsets of P such that (i) A_i and B_i are well-separated w.r.t. s, for all $i = 1, \cdots, m$. (ii) for any two distinct points p and q of P, there is exactly one pair (A_i, B_i) in the collection, such that either $p \in A_i$ and $q \in B_i$, or $q \in A_i$ and $p \in B_i$. The number of pairs, m, is called the size of the WSPD. Callahan and Kosaraju show that any set P admits a WSPD of size $m = O(s^2 n)$.

It is known by setting $s := 4 + (8/\epsilon)$, and selecting an edge per pair (A_i, B_i), we can construct $(1+\varepsilon)$-spanner of size $O(n)$. Unfortunately this construction might not be local spanner. Indeed, for a region $R \in \Re$, $R \cap A_i$ and $R \cap B_i$ might not be empty but the selected edge for pair (A_i, B_i) might not exist in R. To get a local spanner, for each pair (p, q) where $p \in A_i$ and $q \in B_i$, we add $\{p, q\}$ to the spanner provided that there is a region $R \in \Re$ just containing p, q among points in $A_i \cup B_i$. It is easy to show for the families of squares and disks, the number of edges added to the spanner is $\sum |A_i| + |B_i|$ over all pairs of WSPD. Therefore, we get the following theorem.

Theorem 1 *For a set P of n points on the plane and families of square and disks, we can construct local $(1+\varepsilon)$-spanners of size $\sum |A_i| + |B_i|$ where (A_i, B_i) are the pairs of a $(4 + (8/\epsilon))$-WSPD.*

3 Local spanners with respect to vertical slabs

For arbitrary slabs (not necessarily vertical), there is an example showing that we need $\Omega(n^2)$ edges to have a local spanner (see Fig. 2). Hence, we only focus on vertical slabs. Our method to construct local $(1+\varepsilon)$-spanner with respect to vertical slab is similar to the general method given in the previous section: for each pair (p, q) where $p \in A_i$ and $q \in B_i$, we add $\{p, q\}$ to the spanner provided that there is a vertical slab just containing p, q among points in $A_i \cup B_i$. We show that the number of the edges of the constructed graph is $O(n \log n)$.

We use the fact that there exists a WSPD such that $\sum \min(|A_i|, |B_i|)$ over all pairs (A_i, B_i) of the WSPD is $O(n \log n)$ [6]. Assume (A, B) is a pair of such a WSPD and $|A| \leq |B|$. It is easy to show that in the above construction, each point of A is connected to at most two points of B. Hence, the number of the edge added to the spanner is less than $2 \sum \min(|A_i|, |B_i|)$.

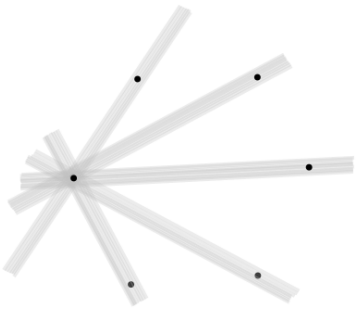


Figure 2: For points on convex position, any local t-spanner with respect to arbitrary slabs must be complete.

Theorem 2 *For a set P of n points on the plane and any $\varepsilon > 0$, there exists a local $(1+\varepsilon)$-spanner of size $O(n \log n)$ with respect to vertical slabs.*

4 Local spanners with respect to squares

We start with the case where we are allowed to use Steiner points. We use the WSPD technique explained above, and add a set Q of Steiner points to P such that $|E(A, B)| = O(1)$ for any pair (A, B) in the WSPD where $E(A, B)$ is the set of edges added to the spanner for pair (A, B).

Our method is based on the WSPD construction by Fisher and Har-Peled [5]. Their construction uses a compressed quad tree. The set Q of Steiner points that we use is defined as follows. Let $T(P)$ be a compressed quad tree for P. For each internal node v of $T(P)$, we add the four corner points of the associated region with v to Q. The resulting set Q has size at most $4(n-1)$. Combining the WSPD technique and the above Steiner points, we get the following theorem.

Theorem 3 *For any set P of n points in the plane, and any $\varepsilon > 1$, one can construct a local Steiner $(1+\varepsilon)-$spanner of size $O(n)$ with respect to squares by adding at most $4(n-1)$ Steiner points.*

Our local spanner construction without using Steiner points is based on the concept of CSPD [1], Cone Separated Pair Decomposition, defined next.

Let α-cone be a cone such that the angle of any two rays emanating at the cone apex and being inside the cone is at most α. Let C be a collection interior-disjoint α-cone whose apices are the origin and also they together cover the whole plane.

Let P be a set of points in the plane. A pair (A, B) $(A, B \subset P)$ is said to be cone separated if there exists a translated cone δ of cones in C such that all points of A are inside δ and all points of B are inside the reflection of δ about its apex.

A CSPD of P with respect to α is a collection $\psi_\alpha := \{(A_1, B_1), \cdots, (A_m, B_m)\}$ of pairs of subsets for P such that:

- A_i and B_i are cone separated.
- For any two distinct points $p, q \in P$, there exists precisely one pair in $(A_i, B_i) \in \psi_\alpha$ such that $p \in A_i$ and $q \in B_i$ or vice versa.

In [1] it has been shown that any set P of n points and any angle α admit a CSPD $\psi_\alpha = \{(A_1, B_1), \cdots, (A_m, B_m)\}$ such that $\sum |A_i|+|B_i| = O(n \log^2 n)$.

The sketch of construction is to construct a CSPD for $\alpha = \pi/2$, and then for each pair (A_i, B_i), we add the Delaunay edges $A_i \cup B_i$ with respect to squares. To this work, our spanner needs more edges and we prove with $O(|A_i| + |B_i|)$ extra edges we can get a local spanner with respect to squares.

Theorem 4 *For any set P of n points in the plane and any $\varepsilon > 1$, one can construct a local $(4+\varepsilon)$-spanner of size $O(n \log^2 n)$.*

5 Local spanners with respect to disks

Constructing local spanners with respect to disks is more challenging. Here, we just explain how to construct a local spanner using Steiner points. We start with some definitions and a lemma given in [3].

Definition 2 *Let $b(p, r)$ be the set of all points of P whose distance to $p \in P$ is at most r. Also, let $ring(p, r, r')$ is the set of all points of P whose distance to $p \in P$ is at most r' and at least r.*

Lemma 5 *([3]) Let P be a set of n points in $\mathbb{R}^d$, $t > 0$ be a parameter, and let c be a sufficiently large constant. Then, one can compute in linear time a ball $b = b(p, r)$, such that (i) $|b \cap P| \geq n/c$, (ii) $ring(p, r, r(1 + 1/n))$ is empty, and (iii) $|P \backslash b(p, 2r)| \geq n/2$.*

Based on Lemma 5, let $P_{\text{in}} = b(p, r)$, $P_{\text{out}} = b(p, 2r) \backslash b(p, r)$ and $P_{\text{outer}} = P \backslash b(p, 2r)$. We define an angle $\theta = 2\pi/k$ depending on the given ε—note that k is constant depending on ε. We select k points on the circle centered at p and radius $\frac{3}{2}r$ such that for any two consecutive such points q and q', the angle qpq' is θ. We look at all these points as Steiner points and add them all to P_{out}. It is easy to see any disk D that $P_{\text{in}} \cap D \neq \emptyset$ and $P_{\text{outer}} \cap D \neq \emptyset$ contains at least one Steiner point defined above.

In the first step of our construction, we connect each Steiner point to all other points in our spanner. Since the number of Steiner points is constant, then the number of edges added to the spanner is $O(n)$. Moreover, it is easy to show that there is always a $(1 + \varepsilon)$-path from a point in P_{in} to a point in P_{outer} even we cut the spanner by a disk D. Next we explain how to guarantee existing a $(1+\varepsilon)$-path from a point in P_{in} to a point in P_{out} by adding $O(n \log n)$ edges. If we succeed to do that, then we can recursively continue our construction over P_{in} and $P_{\text{out}} \cup P_{\text{outer}}$ and totally get a spanner of size $O(n \log^2 n)$. Note that the number of Steiner points is $O(n)$ as we add a constant number of Steiner points at each node of the recursion tree.

To construct a spanner over $(P_{\text{in}}, P_{\text{out}})$, we use the general method given in Section 2. We construct an s-WSPD (s depends on ε) over $(P_{\text{in}}, P_{\text{out}})$ using a compressed quadtree. As $\text{ring}(p, r, r(1 + 1/n))$ is empty, the depth of the quadtree becomes $O(\log n)$—note that we just want to seperate P_{in} from P_{out} by WSPD. In each level of the quadtree, we only add $O(n)$ edges to the spanner. As the depth is $O(\log n)$, the total edges added to the spanner is $O(n \log n)$. Putting all this togehter, we get the following theorem.

Theorem 6 *Suppose ε is a parameter and P is a arbitrary point set on the plane. One can construct a local $(1+\varepsilon)$-spanner with respect to disks that uses $O(n)$ Steiner points and has $O(n \log^2 n)$ edges.*

References

[1] M. A. Abam and M. de Berg Kinetic Spanners in $\mathbb{R}^d$. *Discrete Computational Geometry*, 45(4):723–736, 2011.

[2] M. A. Abam, M.de Berg, M. Farshi and J. Gudmundsson Region-Fault Tolerant Geometric Spanners. *Discrete Computational Geometry*, 41(4):556–582, 2009.

[3] M. A. Abam and S. Har-Peled New constructions of SSPDs and their applications. *Comput. Geom*, 45(5-6):200–214, 2012.

[4] P. B. Callahan and S. R. Kosaraju Faster algorithms for some geometric graph problems in higher dimensions. *In the Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pages 291–300, 1993.

[5] J. Fischer and S. Har-Peled Dynamic well-separated pair decomposition made easy. *In the*

Proceedings of the Canadian Conference on Computational Geometry, pages 235–238, 2005.

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

Session 3

Approximate Curve-Restricted Simplification of Polygonal Curves

Ali Gholami Rudi*

Abstract

The goal in the curve simplification problem is to reduce the number of the vertices of a polygonal curve without changing its shape significantly. In this paper we study curve-restricted min-# simplification of polygonal curves, in which the vertices of the simplified curve can be placed on any point of the input curve, provided that they respect the order along that curve. For local directed Hausdorff distance from the input to the simplified curve in $\mathbb{R}^2$, we present an approximation algorithm that computes a curve whose number of vertices is at most twice the number of the vertices of the curve-restricted simplification with the minimum number of vertices.

Keywords: Curve simplification, geometric algorithms, computational geometry.

2010 Mathematics subject classification: 68U05.

1 Introduction

The goal of the classical curve simplification problem is to reduce the number of the vertices of a polygonal curve, without changing its shape significantly. There are several applications in which curve simplification plays an important role. In trajectory analysis, for instance, there are two important reasons for this reduction. First, it reduces the storage and bandwidth requirements for storing and transferring huge and growing collections of trajectory data. Second, and probably more importantly, the complexity of most trajectory analysis algorithms depends on the number of the vertices of the input curves, and simplifying trajectories can greatly reduce the running time of these algorithms.

Curve simplification is usually studied in two settings. In the min-ϵ setting the maximum number of the vertices of the simplified curve is specified and the amount of distance between the original and simplified curves is minimised, and in the min-# setting the maximum distance between the curves is specified while the number of the vertices is minimised. The distance between the original and simplified curves is either *global* and computed for the curves as a whole, or is *local* and computed as the maximum distance of the corresponding sub-curves. The distance between curves is computed using measures such as Fréchet or Hausdorff [1].

The simplified curve may be vertex-restricted, i.e. its vertices should coincide with the vertices of the input curve, it may be curve-restricted, i.e. its vertices can be placed on any point of the input curve, or it may be unrestricted with no limitation on the placement of its vertices. In the first two cases, the vertices of the simplified curve should appear in order on the input curve. There are numerous results on vertex-restricted curve simplification in the min-# setting, only some of which provide a guarantee on the number of the vertices of the simplification. Zhang et al. [2] surveyed some of these results, concentrating especially on heuristic algorithms, such as [3, 4].

The well-known algorithm presented by Douglas and Peucker [5] does not minimise the number of the vertices of the simplified curve, but is both simple and effective. It assumes local directed Hausdorff distance from the input curve to the simplified curve. For simplifying curve $P = \langle p_1, p_2, ..., p_n \rangle$ with the maximum distance ϵ, it finds the most distant vertex p_k from segment p_1p_n; if their distance is at most ϵ, this segment is a link of the simplification. Otherwise, the algorithm recursively simplifies $\langle p_1, ..., p_k \rangle$ and $\langle p_k, ..., p_n \rangle$. Hershberger and Snoeyink [6] improved the running time of this algorithm to $O(n \log n)$.

Among optimal algorithms, the one presented by Imai and Iri [7] is probably the most popular for local Hausdorff distance. It creates a shortcut graph, the vertices of which represent the vertices of the input curve. An edge p_ip_j shows that the distance between link p_ip_j and sub-curve $\langle p_i, p_{i+1}, ..., p_j \rangle$ is not greater than ϵ. A shortest path algorithm on this graph, finds the simplification with the minimum number of vertices. Chan and Chin [8] improved the running time of this algorithm to $O(n^2)$.

There are many other results on vertex-restricted simplification that consider different distance measures, such as global Hausdorff or Fréchet [9], different distance metrics, such as uniform [1], or under different assumptions, such as streaming input [10, 11, 12, 13]. Despite the number of results on vertex-restricted curve simplification, curve-restricted simplification, which has attracted less attention, can yield a curve with much fewer vertices, as in Figure 1: a curve-restricted simplification with only four vertices is demonstrated for a curve

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

Figure 1: An example showing that curve-restricted simplifications can have far fewer vertices compared to vertex-restricted simplifications; the dashed links are a curve-restricted simplification of the curve with solid edges.

whose vertex-restricted simplification is the same as the input curve. For global directed Hausdorff distance, van de Kerkhof et al. [14] showed that curve-restricted simplification is NP-hard and provided an $O(n)$ algorithm for global Fréchet distance in $\mathbb{R}^1$.

In this paper, we study the min-# curve-restricted simplification problem with maximum local Hausdorff distance ϵ from the input curve to the simplified curve. We present an algorithm that computes a simplified curve, the number of the vertices of which is at most twice the minimum possible. This paper is organized as follows: In Section 2 we introduce the notation used in this paper and in Section 3 we present our algorithm. We conclude this paper Section 4.

2 Preliminaries and Notation

A two-dimensional polygonal curve is represented as a sequence of vertices on the plane, with line segments as edges between contiguous vertices. The directed Hausdorff distance between curves P and P', denoted as $H(P, P')$, is defined as the maximum of the distance between any point of P to the curve P'.

A simplification of curve P is a curve P' that starts at the starting vertex of P and ends at P's ending vertex. Given a parameter ϵ, the goal in the min-# simplification is to find a simplified curve with the minimum number of vertices, such that the distance between the original and the simplified curve is at most ϵ. In what follows, we use *links* to distinguish the edges of the simplified curve from the edges of the input curve. The simplification is curve-restricted, if its vertices are on the input curve and appear in order along P.

Let P' be a curve-restricted simplification of curve $P = \langle p_1, p_2, ..., p_n \rangle$. For a link ℓ of P', suppose x and y on P are points corresponding to the start and end points of ℓ and suppose x is on edge $p_i p_{i+1}$ and y is on $p_j p_{j+1}$. Then, ℓ *covers* all edges $p_k p_{k+1}$ for $i \leq k \leq j$. Let P_ℓ be the sub-curve of P corresponding to link ℓ, i.e. the sub-curve of P from point x to y. The local Hausdorff distance from P to P' is the maximum of $H(P_\ell, \ell)$ over all links ℓ of P'. In this paper we assume local Hausdorff distance to measure the distance between the input and simplified curves.

In the next section, we refer to the ϵ-neighbourhood of a vertex of P, which is defined as follows.

Definition 1 *The ϵ-neighbourhood of a point p, denoted as $N(p)$ is a circle with radius ϵ, whose centre is at p. Clearly, the set of points inside $N(p)$ are all points at distance at most ϵ from p.*

3 The Main Result

Lemma 2 *For the curve $P = \langle p_1, p_2, ..., p_n \rangle$, a segment s from point x on edge $p_i p_{i+1}$ to point y on edge $p_j p_{j+1}$ can be a link of a (not necessarily optimal) curve-restricted simplification if and only if it intersects $N(p_k)$ for every index k, where $i < k \leq j$.*

Proof. Let C be the sub-curve P from x to y. If s is a link of a simplification of P, $H(C, s)$ is at most ϵ. This implies that the distance of every point of C to s is at most ϵ. For the vertices of C like p this means that s should include at least one point from $N(p)$.

For the converse, suppose s intersects $p_i p_{i+1}$ at x and $p_j p_{j+1}$ at y, as well as $N(p)$ for every vertex of C, the sub-curve of P from x to y. It is enough to show that $H(C, s) \leq \epsilon$. For each edge, since the distance between its end points and s is at most ϵ, the distance of other points of the edge cannot be greater. This holds for every internal edge of C and implies $H(C, s) \leq \epsilon$ as required. □

Lemma 3 *Suppose ℓ is a link of a curve-restricted simplification of curve $P = \langle p_1, p_2, ..., p_n \rangle$, such that ℓ starts from point x on edge $p_i p_{i+1}$ and ends at point y on edge $p_j p_{j+1}$. There exists another link ℓ' covering the same set of edges such that the line that results from extending ℓ' has the following property for at least two values of k where $i < k \leq j$: either i) it is a tangent to $N(p_k)$, or ii) it passes through one of the end points of $p_i p_{i+1}$ or $p_j p_{j+1}$, or their intersection with $N(p_k)$.*

Proof. Let L be the line resulting from extending the segment ℓ. If none of the mentioned properties hold for any value of k, we move L downwards until one of them holds for some value k, i.e. it becomes tangent to the ϵ-neighbourhood of p_k or passes through the intersection of the ϵ-neighbourhood of p_k and the last or the first edge covered by the s. We then rotate L around p_k for case i, or the intersection of case ii, until one of the conditions holds for another index. Let s be the segment on line L with endpoints on $p_i p_{i+1}$ and $p_j p_{j+1}$; such a segment surely exist, since the movement or rotation stops at the endpoints of these edges.

Clearly L cannot leave $N(p_k)$ for any possible index k for both the downward movement and the rotation; just before leaving $N(p_k)$, L becomes its tangent. The only problem may be that although $N(p_k)$, for some k where $i < k \leq j$, is intersected by both ℓ and L, s may be too short to intersect $N(p_k)$; this is demonstrated in Figure 2. However, since the rotation stops at the

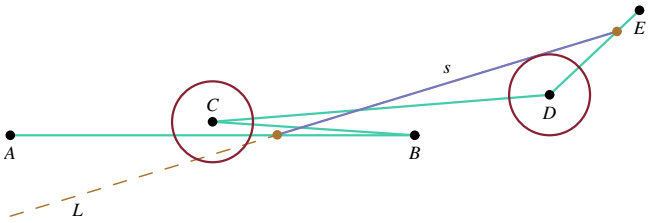


Figure 2: Rotating line L around $N(D)$ counterclockwise; s no longer intersects $N(C)$.

intersection the first or the last edge and $N(p_k)$, this case never happens. □

Lemma 4 *A link of a curve-restricted simplification of $P = \langle p_1, p_2, ..., p_n \rangle$, from a point on edge p_ip_{i+1} to a point on edge p_jp_{j+1} can be found with the time complexity $O(m^3)$ where $m = j - i + 1$, provided such a link exists.*

Proof. We find a line for which the condition mentioned in Lemma 3 holds. To do so, we find three parallel lines at distance ϵ on the plane, L_1, L_2, and L_3, such that a link can be found on line L_2. We consider possible placements of these lines according to Lemma 3 and check for which of them the condition of Lemma 2 holds for a segment on L_2. If L_2 is a tangent to $N(p_k)$ for some value of k where $i < k \leq j$, then either L_1 or L_3 should pass through p_k. We therefore try different placements of these three lines such that the following property holds for two values of k for $i < k \leq j$: either i) L_1 or L_3 passes through p_k, or ii) L_2 passes through the intersection $N(p_k)$ and one of $p_{i-1}p_i$ or p_jp_{j+1} or the endpoints of these edges. Since there are $O(m)$ choices for the first and the second conditions, the number of total cases to consider is $O(m^2)$.

For each of $O(n^2)$ possible placements of these lines, we have to verify if there exists a segment s on L_2 such that $H(C, s)$ is at most ϵ. Let x be the intersection of L_2 and p_ip_{i+1} and let y be the intersection of L_2 and p_jp_{j+1}; if x or y do not exist, L_2 cannot contain a link. Based on Lemma 2, if the segment xy intersects $N(p_k)$ for every $i < k \leq j$, it is a valid link. This can be checked with the time complexity $O(m)$. □

To force the link to start from p_i, instead of any point on edge p_ip_{i+1} in Lemma 4, we can fix this point on L_2 and try condition mentioned in the proof of the lemma for only one value of k.

Definition 5 *For a polygonal curve $C = \langle p_1, p_2, ..., p_m \rangle$, the* left-most link point *from edge p_1p_2 to edge $p_{m-1}p_m$, denoted as $\text{llp}(C)$ is the first point on $p_{m-1}p_m$, such that there is a link from p_1p_2 to this point. $\text{llp}'(C)$ is defined similarly, except that the links should start from p_1.*

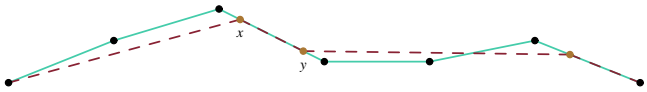


Figure 3: Merging links in Theorem 7.

Since the line containing a link can be moved or rotated to obtain a new link, unless the conditions mentioned in Lemma 3 holds for it, Lemma 4 yields the following corollary.

Corollary 6 *For a polygonal curve $C = \langle p_1, p_2, ..., p_m \rangle$, $\text{llp}(C)$ (and similarly $\text{llp}'(C)$) and its corresponding link can be computed with the time complexity $O(m^3)$.*

In Theorem 7 we present an algorithm for computing a curve-restricted simplification of an input curve P with maximum local directed Hausdorff distance ϵ.

Theorem 7 *A curve-restricted simplification of $P = \langle p_1, p_2, ..., p_n \rangle$ can be computed in $O(n^6)$.*

Proof. We use dynamic programming to fill the tables $D[i][d]$ and $L[i][d]$, where $1 \leq i, d < n$. Each entry of D holds a point on P and each entry of L is a link. For points u and v on P, $u < v$ holds if u appears before v on P. We fill the tables as follows.

1. $D[i][1]$ is initialized as $\text{llp}'(\langle p_1, p_2, ..., p_i \rangle)$ and $L[i][1]$ as the link corresponding to $D[i][1]$.

2. For d from 2 to n, $D[i][d]$ and $L[i][d]$ for $1 \leq i \leq n$ is filled as follows: The value $D[i][d]$ is the minimum value of $\text{llp}(\langle q, p_{j+1}, p_{j+2}, ..., p_i \rangle)$, in which q is $D[j][d-1]$, over all indices of j, where $j < i$. The value of $L[i][d]$ should indicate the link corresponding to $D[i][d]$.

Based on Corollary 6, filling these tables can be done with the time complexity $O(n^6)$.

Let m be the lowest index, such that $D[n][m]$ is filled. By following the links backwards using table L, we obtain a sequence S of links covering all edges from p_1p_2 to $p_{n-1}p_n$. These links can be merged to obtain the simplified curve as follows. Let a link ℓ of S end at point x of edge $p_{i-1}p_i$ and suppose the next link starts from point y on the same edge; because of the way tables D and L are filled, x is surely before y. For the last link, let y be p_m. We merge ℓ with its next link by adding the link xy; this is demonstrated in Figure 3. Since these new links are on the edges of P, the local Hausdorff distance for these links is zero. Therefore, the curve that results from merging all contiguous pairs of links in S is a valid curve-restricted simplification. Note that the vertices of this curve appear in order along P. □

Theorem 8 *The number of the links of a curve-restricted simplification P' obtained from Theorem 7 for curve $P = \langle p_1, p_2, ..., p_n \rangle$ is at most twice the number of the links of an optimal simplification.*

Proof. Let $O = \langle o_1, o_2, ..., o_m \rangle$ be a curve-restricted simplification with the minimum number of links, in which $o_1 = p_1$ and $o_m = p_n$. Suppose the vertex o_i is on the edge $p_j p_{j+1}$ of P; for every such vertex it can be inductively shown that $D[j][i]$ is surely filled with a point before o_i along P. This implies that $L[n][m]$ is also filled. Therefore, the sequence S in the proof of Theorem 7 has at most m entries. Merging the links in Theorem 7 would make the number of the links of the resulting curve at most $2m$, as required. □

4 Concluding Remarks

Although, the min-ϵ curve-restricted simplification of polygonal curves can reduce the number of the vertices of the curves much better than vertex-restricted simplification, the time complexity of the algorithm presented in this paper is not very appealing for real-world applications. A faster approximate or exact algorithm may fill this gap.

References

[1] P. K. Agarwal, S. Har-Peled, N. H. Mustafa, and Y. Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3–4):203–219, 2005.

[2] D. Zhang, M. Ding, D. Yang, Y. Liu, J. Fan, and H. T. Shen. Trajectory simplification - an experimental study and quality analysis. *Proceedings of the VLDB Endowment*, 11(9):934–946, 2018.

[3] M. Chen, M. Xu, and P. Fränti. A fast $o(n)$ multiresolution polygonal approximation algorithm for gps trajectory simplification. *IEEE Transactions on Image Processing*, 21(5):2770–2785, 2012.

[4] H. V. Jagadish C. Long, R. C.-W. Wong. Direction-preserving trajectory simplification. *PVLDB*, 6(10):949–960, 2013.

[5] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10(2):112–122, 1973.

[6] J. Hershberger and J. Snoeyink. An $o(n \log n)$ implementation of the douglas-peucker algorithm for line simplification. In *Annual ACM Symposium on Computational Geometry*, pages 383–384. ACM, 1994.

[7] H. Imai and M. Iri. Polygonal approximations of a curve - formulations and algorithms. In G. T. Toussaint, editor, *Computational Morphology: A computational Geometric Approach to the Analysis of Form*. North-Holland, 1988.

[8] W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *International Journal of Computational Geometry & Applications*, 6(1):59–77, 1996.

[9] M. J. van Kreveld, M. Löffler, and L. Wiratma. On optimal polyline simplification using the hausdorff and frchet distance. In *Symposium on Computational Geometry*, pages 56:1–56:14, 2018.

[10] M. A. Abam, M. de Berg, P. Hachenberger, and A. Zarei. Streaming algorithms for line simplification. *Discrete & Computational Geometry*, 43(3):497–515, 2010.

[11] X. Lin, S. Ma, H. Zhang, T. Wo, and J. Huai. One-pass error bounded trajectory simplification. *PVLDB*, 10(7):841–852, 2017.

[12] W. Cao and Y. Li. Dots - an online and near-optimal trajectory simplification algorithm. *Journal of Systems and Software*, 126:34–44, 2017.

[13] J. Muckell, P. W. Olsen, J.-H. Hwang, C. T. Lawson, and S. S. Ravi. Compression of trajectory data - a comprehensive evaluation and new approach. *GeoInformatica*, 18(3):435–460, 2017.

[14] M. van de Kerkhof, I. Kostitsyna, M. Löffler, M. Mirzanezhad, and C. Wenk. On optimal min-# curve simplification problem. *CoRR*, abs/1809.10269, 2018.

On Maximum-Weight Minimum Spanning Tree Color-Spanning Set

Homa Ataei Kachooei* Mansoor Davoodi† Dena Tayebi ‡

Abstract

Given n points with m colors in the plane, we aim at finding m points with distinct color such that their minimum spanning tree is maximized. In this paper, we study a very simple case of this problem, that is, there are at most two points of each color whose distance is unit and have the same $x-$ or $y-$ coordinate. We prove that this problem even under these restrictions is NP-hard and does not have an FPTAS unless $P = NP$. Also, we present an approximation algorithm for a special case of this problem.

Keywords: Minimum Spanning Tree, Color-spanning Set, Uncertainty, NP-hardness, Approximation Algorithm.

1 Introduction

In addition to the theoretical aspect of multi-colored problems, they have real world applications such as modeling uncertain data, e.g., an uncertain point can be modeled by a discrete set of candidates with a special color. In this setting, one of candidates can be considered as a certain point. For example, for a given set of n points with m colors, one can select m colored points (exactly one point from each colored set) such that the minimum spanning tree obtained from them is maximized. We call this problem Max-MST and show that it is NP-hard even for a simple case such that each colored set contains at most two points with coordinates $p = (x, y)$ and $q = (x, y+1)$ or $q = (x+1, y)$. We prove the NP-hardness of this simple problem, and present an approximation algorithm for a special case of this problem

Related work. For a set of n points with m colors, *Planar Smallest Perimeter Convex Hull Color-spanning Set problem* ($PSPCHCS$) is to find m points with different colors such that their convex hull is minimized. It was proved that the PSPCHCS problem is NP-complete and two efficient constant factor approximation algorithms were proposed to solve it [7]. *Maximum Diameter Color-spanning Set* ($MaxDCS$) problem is finding m distinct colored points which diameter of them is maximized. An $O(n^{1+\epsilon})$ time algorithm was proposed to solve it, where ϵ is an arbitrary small positive constant [7]. Also, an $O(n \log n)$ time algorithm was presented for MaxDCS [3]. The *Largest Closest Pair Color-spanning Set problem* ($LCPCS$) also was studied in [7]. This problem is finding the m different color points such that the distance between the closest pair of them is maximized. It is proved that the LCPCS problem is NP-complete even in one dimension [7]. The *Minimum Diameter Color − spanning Set* ($MinDCS$) problem is finding m points with distinct colors such that the diameter of them is minimized. In [4], it was shown this problem is NP-hard for any L_p metric, except L_1 and L_∞ which admit polynomial time algorithms. In d dimensions, Ghodsi et al. [5] presented a $(1+\epsilon)$-approximation algorithm in $O(2^{\frac{1}{\epsilon^d}}.\epsilon^{-2d}.n^3)$ time for the MinDCS problem. Kazemi et al. [9] presented also a $(1+\epsilon)$-approximation algorithm and improved the running time to $2^{O(\lambda log \lambda) \times O(nlogn)}$. Further, the problem of *Smallest Color-Spanning Ball* ($SCSB$), which is finding the smallest ball containing at least one point of each color, was studied by Khanteimouri et al. [10]. They presented a 3-approximation algorithm and a $(1+\epsilon)$-approximation algorithm for solving SCSB problem.

Two other problems related to the Minimum Spanning Tree (MST) under uncertainty, are *Planar Smallest Minimum Spanning Tree Color spanning Set* ($PSMSTCS$) and *Planar Largest Minimum Spanning Tree Color-spanning Set* ($PLMSTCS$). In these problems, the goal is finding m points with distinct colors such that their MST is minimized in PSMSTCS problem and is maximized in PLMSTCS problem. Both problems are NP-complete [7]. The similar problem to PSMSTCS is *Generalized Minimum Spanning Tree* ($GMST$) problem. The GMST problem has two variations. Let n points be clustered in k clusters. First variation of GMST, is finding an MST consisting of at least one point in each cluster while second variation is finding exactly one point in each cluster. When each cluster contains three points, the first problem was proved NP-complete [6]. However, the later one is the same as PSMSTCS problem. It was proved that this variation of the GMST problem is NP-complete even if every cluster contains two points with equal y coordinate and also did not have an FPTAS unless $P = NP$ [8].

*ataei.homa@gmail.com

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

‡denatayebi@yahoo.com

In addition to the discrete regions for modeling uncertainty, continuous regions have been also applied for this purpose. The *Minimum Spanning Tree with Neighborhood* ($MSTN$) problem was introduced in 2007 [13]. In this problem given a set of regions, the goal is placing a point on each region such that MST obtained from them is minimized. Yang et al. [13] studied the MSTN problem when the regions of uncertainty are a set of disks. They presented two approximation algorithms, two lower bounds and a PTAS for this problem. Löffler and van Kreveld proved that the MSTN problem is NP-hard when the regions are disks or squares [12]. Dorrigive et al. [2] proved the NP-hardness of the problem for disjoint disks. They also introduced the *Max-MSTN* problem which its goal is placing a point on each region such that MST is maximized. They proved that the Max-MSTN problem where neighborhoods are disjoint disks is also NP-hard, and proposed a parameterized approximation algorithm for the Max-MSTN problem.

2 NP-hardness of the PLMSTCS-2 problem

We are given a set of n points with m colors and the goal is selecting m points with distinct colors such that their MST is maximized. In this section, we study this problem where there are at most two points with the same color whose coordinates are $p = (x, y)$ and $q = (x, y + 1)$ or $q = (x + 1, y)$. We denote this problem by *PLMSTCS-2* and prove its NP-hardness by a reduction from planar 3SAT.

Planar 3SAT is a special 3SAT variation whose corresponding graph is planar. This graph is constructed as follows. For each variable and clause in the 3SAT instance, there is a node in this graph. The nodes corresponding with variables are called variable nodes and the nodes corresponding with gadgets are called gadget nodes. There is also an edge between a variable node and a clauses node if the corresponding variable appears in the corresponding clause. Further, all variable nodes are connected by a path. Precisely, for a 3SAT instance, let $C = \{c_1, c_2, ..., c_m\}$ be the set of clauses and $V = \{v_1, v_2, ..., v_n\}$ be the set of variables. The corresponding graph $G = (U, E)$ is constructed as follows:

$$U = C \cup V, \tag{1}$$

and

$$E = E_1 \cup E_2, \tag{2}$$

where E_1 and E_2 are:

$$E_1 = \{(c_i, v_j) \mid v_j \in c_i \ \ or \ \ \overline{v_j} \in c_i\} \tag{3}$$

and

$$E_2 = \{(v_j, v_{j+1}) \mid 1 \leq j < n\} \cup \{(v_n, v_1)\}. \tag{4}$$

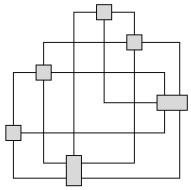


Figure 1: An example of the orthogonally drawing [1]

The set of all edges in E_2 is called *spinal path* [2]. It is proved that the planar 3SAT problem is NP-hard [11].

Theorem 1 *The PLMSTCS-2 problem is NP-hard and it does not admit an FPTAS unless $P = NP$.*

Proof. We prove this theorem by a reduction from the planar 3SAT problem. Let ϕ be an instance of the planar 3SAT problem. We design two types of gadgets –called *variable gadget* and *clause gadget*. We convert ϕ to an instance of PLMSTCS-2 by replacing the nodes with these gadgets. Since the number of colors used in this reduction should be polynomially bounded in the size of ϕ, we use a special drawing graph, called *Orthogonally Drawing* [1], to achieve this bound. In orthogonally drawing, each node is considered as a box and each edge is considered as a sequence of vertical and horizontal line segments. Figure 1 shows an example.

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

In Theorem 2, $deg(v)$ is the degree of node v. The bounds are presented in Theorem 2 are established for planar triconnected graphs [1]. As planar 3SAT graph is at most triconnected, we can convert it to the orthogonally drawing in polynomially time bounded in the size of ϕ.

Now we explain variable and clause gadgets.

2.1 Variable Gadgets

We design a gadget for each variable of ϕ. This gadget is constructed by some points with k colors such that there is exactly two points with the same color. Parameter k is an even number between 4 and $6c - 2$ where c is the number of clauses containing the variable. Suppose that we want to construct a gadget for a variable x_i. We consider a structure shown in Figure 2. This structure is part of the variable gadget and we called it *StructureA*. We prove the following lemma in Appendix.

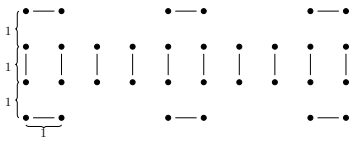


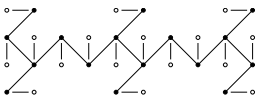
Figure 2: Structure A. The segment between two points indicates they have the same color. That means there is 16 colors in this structure

Lemma 3 *The PLMSTCS-2 with k colors for the structure A admits two optimal solutions with weight of $\sqrt{2}(k-1)$ (See Figure 3). Further, the weight of MST for any other (non-optimal) solution is at least 0.41 less than the optimal solutions.*

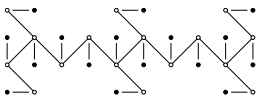
Now we should design some points for connection of spinal path to variable gadgets. These points should be located such that do not affect on the selection of points in the optimal solutions. So, we locate these points in two sides of the gadget such that have equal distance from the nearest top and bottom points with the same color. These points are shown in Figure 4. Also, we should design some points for connection of the variable gadget to the edges that come from the clauses. According to Lemma 3, there are two optimal solutions for structure A. One of the optimal solutions leads to select the solid points and the other one leads to select the hollow points. We called these solutions the *Solid Solution* and the *Hollow Solution*, and consider them to be correspond with the variable x_i is TRUE and FALSE, respectively. For each clause containing x_i, we put a point at the distance of 2 units to one of the hollow points in the direction of y. Also, for each clause which contains $\overline{x}_i$, we set a point at the distance of 2 units to one of the solid points in the direction of y. Figure 4 shows a variable gadget which is correspond with a variable x_i appeared in three clauses such that x_i appears in one clause and $\overline{x}_i$ appears in two clauses.

2.2 Clause Gadgets

A clause gadget is constructed by three sequence of points in the length of the edges which meet in a point.



(a) Solid Solution. An optimal solution of the PLMSTCS-2 problem for structure A.



(b) Hollow Solution. An optimal solution of the PLMSTCS-2 problem for structure A.

Figure 3: Optimal solutions of the structure A. The solid and hollow points which is connected to each other have the same color

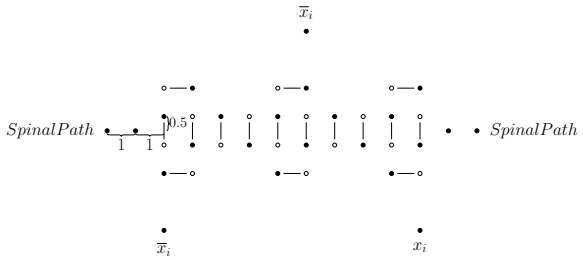


Figure 4: An example of the variable gadget

Figure 5a illustrates a clause node in the orthogonally drawing and figure 5b illustrates the clause gadget replaced with the clause node shown in Figure 5a.

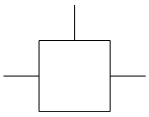
2.3 Reduction

We designed two gadgets correspond with the variables and clauses. If they are replaced with the variables and clauses of the planar 3SAT graph, we have a PLMSTCS-2 instance. We scale up the orthogonally drawing of the planar 3SAT graph with a proper constant factor, e.g., 2, and then replace the graph nodes with the gadgets. The graph edges also should be exchange with a sequence of points which have unit distances along the edges. Maximum size of the variable gadget is $(4(c-1)+3)\times 7$ which is equal to $(4(deg(v)-9))\times 7$ and it is polynomially bounded in the size of the box in the orthogonally drawing. Because of the size of orthogonally drawing is at most $(\frac{m+n}{2} \times \frac{m+n}{2})$, the number of fixed points used in this reduction is polynomially bounded in the size of ϕ.

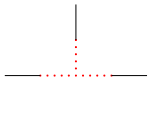
We convert every Planar 3SAT instance to a PLMSTCS-2 instance in polynomially time. Now we show that every PLMSTCS-2 solution determines whether the planar 3SAT problem has a TRUE assignment or not. We have:

$$W_T = W_E + W_G. \tag{5}$$

Where W_T is the total weight of the MST in the optimal solution of the PLMSTCS-2, W_E is the weight of the MST obtained from the fixed points and W_G is the weight of the MST obtained from variable gadgets. The fixed points which is used in this reduction have a MST with a unique constant weight, so W_E has a constant



(a) A clause node



(b) A clause gadget

Figure 5: Replacing a clause node in the orthogonally drawing with a clause gadget

value and it is enough to consider W_G. If we select either solid points or hollow points in all the variable gadgets,

$$W_G = (R - n)\sqrt{2} + 2m, \tag{6}$$

where R is the number of colors, n is the number of variables and m is the number of clauses. If W_G is equal to equation 6, then each variable gadget connects to either the clauses in which variable appears or the clauses in which variable negation appears. This means there exists a TRUE assignment in the planar 3SAT problem. If W_G is less than the equation 6, there exists at least one variable that connects to the clause in which variable appears and a clause in which variable negation appears. This means does not exist a TRUE assignment for the corresponding planar 3SAT instance.

Now we show that the PLMSTCS-2 problem does not admits an FPTAS unless $P = NP$. Suppose that there is an FPTAS for the PLMSTCS-2. Consider an instance of the planar 3SAT problem and construct the corresponding instance of the PLMSTCS-2 and compute W_T. If we set $\epsilon \leq \frac{0.41}{W_T}$, using a $(1-\epsilon)-$solution for the PLMSTCS-2, it is possible to decide whether one can exist a TRUE assignment for the planar 3SAT or not. So, PLMSTCS-2 problem does not have an FPTAS unless $P = NP$. □

3 Approximation Algorithm

In this section we present a $\frac{1}{2}$-approximation algorithm for a special case of the PLMSTCS problem. Given n points with m colors such that the minimum distance between the points with different colors is twice greater than that of the maximum distance between the points with the same color. This means the points of each color are separated from the points of other colors.

Let c_i be the point from color i, whose distance from the furthest point with color i is minimum. Consider $C = \{c_1, c_2, ..., c_m\}$ as a solution for PLMSTCS. We prove the following theorem in Appendix.

Theorem 4 *The solution $C = \{c_1, c_2, ..., c_m\}$ described above is a $\frac{1}{2}-$approximation solution for PLMSTCS problem.*

Proof.

4 Conclusion

In this paper we study the problem of *Planar Largest Minimum Spanning Tree Color-spanning Set* (*PLMSTCS*). We prove that this problem is NP-hard and does not have an FPTAS even if there are two points from each color such that their distance is unit and have the same horizontal or vertical coordinates. We guess this variation of the PLMSTCS problem is the simplest case of this problem which is NP-hard. We also present a $\frac{1}{2}$-approximation algorithm for an special case of PLMSTCS that the points with different colors are well-separable.

References

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

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

[3] C.-L. Fan, J. Luo, W.-C. Wang, F.-R. Zhong, and B. Zhu. On some proximity problems of colored sets. *Journal of Computer Science and Technology*, 29(5):879–886, 2014.

[4] R. Fleischer and X. Xu. Computing minimum diameter color-spanning sets. In *International Workshop on Frontiers in Algorithmics*, pages 285–292. Springer, 2010.

[5] M. Ghodsi, H. Homapour, and M. Seddighin. Approximate minimum diameter. In *International Computing and Combinatorics Conference*, pages 237–249. Springer, 2017.

[6] E. Ihler, G. Reich, and P. Widmayer. On shortest networks for classes of points in the plane. In *Workshop on Computational Geometry*, pages 103–111. Springer, 1991.

[7] W. Ju, C. Fan, J. Luo, B. Zhu, and O. Daescu. On some geometric problems of color-spanning sets. *Journal of Combinatorial Optimization*, 26(2):266–283, 2013.

[8] H. A. Kachooei, M. Davoodi, and D. Tayebi. On the generalized minimum spanning tree in the euclidean plane. *1st Iranian Conference on Computational Geometry*, 2018.

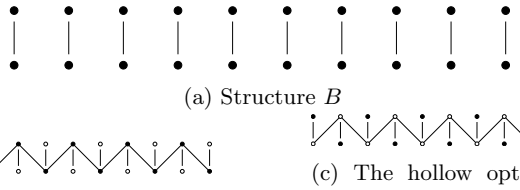
[9] M. R. Kazemi, A. Mohades, and P. Khanteimouri. Approximation algorithms for color spanning diameter. *Information Processing Letters*, 135:53–56, 2018.

[10] P. Khanteimouri, A. Mohades, M. A. Abam, and M. R. Kazemi. Efficiently approximating color-spanning balls. *Theoretical Computer Science*, 634:120–126, 2016.

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

[12] M. Löffler and M. van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235, 2010.

[13] Y. Yang, M. Lin, J. Xu, and Y. Xie. Minimum spanning tree with neighborhoods. In *International Conference on Algorithmic Applications in Management*, pages 306–316. Springer, 2007.



(a) Structure B

(b) The solid optimal solution of the structure B.

(c) The hollow optimal solution of the structure B.

Figure 6: Structure B and its two optimal solution.

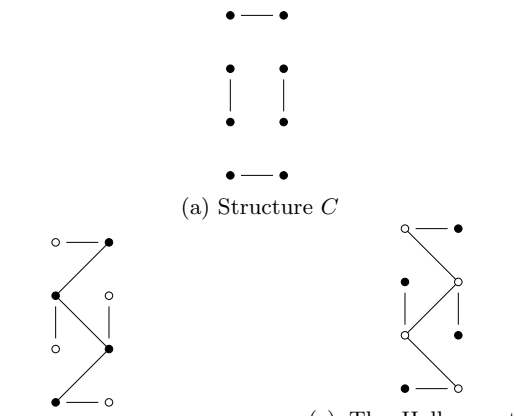
Appendix

Lemma 3 *The PLMSTCS-2 with k colors for the structure A admits two optimal solutions with weight of* $\sqrt{2}(k-1)$ *(See Figure 3). Further, the weight of MST for any other (non-optimal) solution is at least 0.41 less than the optimal solutions.*

Proof. In Figures 3a and 3b, two selections of the points which lead to the optimal solutions are shown. First, we consider the structure B which is shown in Figure 6a. Consider the symbol U which is equivalent to the selection of the top point between two same color points, and the symbol D which is equivalent to the selection of the bottom point between two same color points.

Consider a sequence of U and D symbols for each selection of the points for the structure B. We claim that the solution of the PLMSTCS-2 problem for B is $UDUDUD...UD$ or $DUDUDU...DU$ sequence which is shown in Figure 6b and 6c. The weight of MST in these solutions is $\sqrt{2}(p-1)$, where p is the number of colors in B. If we have UU or DD in the sequence of the optimal solution, the weight of MST is $\sqrt{2}(p-2)+1$ which is $\sqrt{2}-1$ less than $\sqrt{2}(p-1)$. So in the sequence leads to optimal solution we could not have UU or DD.

Now we consider the structure C which is shown in Figure 7a. Structure C is a part of the structure A and repeats $\lceil \frac{k}{6} \rceil$ times. Structure C has two optimal solutions can be obtained by verifying all 2^8 possible solutions. These two optimal solutions of C are shown in Figures 7b and 7c. Clearly, in all repeats of the structure C in A either solid points or hollow points are selected, otherwise, in the solution of PLMSTCS-2 problem for structure B, we have at least one UU or DD which is contradict with the optimality of the solution. So, the PLMSTCS-2 problem for structure A has two optimal solutions which are shown in Figure 3. We call the optimal solution leads to selection of the solid points as *solid solution* and the optimal solution leads to selection of the hollow points as *hollow solution*. The weight of the MST in the solid solution and hollow solution is $\sqrt{2}(k-1)$, where k is the number of colors. Also, the weight of MST in other solutions is $(\sqrt{2}-1) \approx 0.41$ less than the weight of the optimal solutions. □



(a) Structure C

(b) The solid optimal solution of the structure C.

(c) The Hollow optimal solution of the structure C.

Figure 7: Structure C and it's two optimal solutions.

Theorem 4 *The solution* $C = \{c_1, c_2, ..., c_m\}$ *described above is a* $\frac{1}{2}$*–approximation solution for well-separated PLMSTCS problem.*

Proof. We originate three trees $Tree_{opt}$, $Tree_{center}$ and $Tree_m$ from [2]. $Tree_{center}$ is an MST obtained from C, $Tree_{opt}$ is an optimal solution of PLMSTCS problem and $Tree_m$ is a spanning tree whose nodes are vertices of the $Tree_{opt}$ and it's topology is similar to the topology of the $Tree_{center}$, it means that for every edge in $Tree_m$ that connects a point from color i to a point from color j, there exists an edge in $Tree_{center}$ between colors i and j, and vice versa.

Since $Tree_{opt}$ and $Tree_m$ have the same vertices and $Tree_{opt}$ is an MST of them, we have:

$$Weight(Tree_{opt}) \leq Weight(Tree_m). \tag{7}$$

Since, any edge of $Tree_{center}$ connects two centers and $Tree_m$ and $Tree_{center}$ have the same topology,

$$Weight(Tree_m) \leq 2Weight(Tree_{center}). \tag{8}$$

According to inequalities 7 and 8 :

$$Weight(Tree_{opt}) \leq 2Weight(Tree_{center}). \tag{9}$$

Consequently, C is a $\frac{1}{2}$-approximation solution for the well-separated PLMSTCS problem. □

Competitive Strategy for Walking in Streets for an Empowered Simple Robot

Azadeh Tabatabaei* Mohammad Ghodsi† Fardin Shapouri‡

Abstract

We consider the problem of walking in an unknown street, for a robot that has a minimal sensing capability. The basic robot is equipped with a sensor that only detects the discontinuities in depth information (gaps). In the former recent researches some competitive strategies for walking the robot in street polygons have been presented. In this research we have empowered the robot by adding a compass to reach the target t along a shorter route starting from s, in street polygons. We present an online strategy that generates a search path for the empowered robot in streets such that the competitive ratio of our strategy is $3\sqrt{2}$.

1 Introduction

Exploring an unknown environment is a fundamental problem characterized by researcher in robotics, computational geometry, game theory and online algorithm [4]. An autonomous mobile robot without access to the geometry of the scene depending the information collected through its sensor moves to reach a goal. Variants of robot models, and problems have been studied in this context [2, 6, 10]. We are interested in using a minimalist robot model system for walking in unknown scene.

Our basic robot is a simple point robot with the sensing model of gap sensor. At each point the robot locates the depth discontinuities (gaps) of its visibility region in a circularly ordered, (Figure 1). All times the robot can track the gaps and detects each topological changes of the gaps. These changes are the appearance, disappearance, merging, or splitting of gaps which are called *critical events*. While the robot traverses an environment, it can change its direction as often as each of the critical events arises, or a target point enters in its visibility region.

In order to measure the performance of an online search strategy, the notation of competitive analysis is used. The competitive ratio is the worst case ratio of the path travelled by the robot in the unknown environment to the shortest path. Tabatabaei and Ghodsi designed an online strategy for a simple robot to walk in streets. By the strategy the robot explores a street from a vertex s to a vertex t such that the travelled distance by the robot is at most 9 times longer than the shortest path [11]. A street polygon is characterized by the feature that the two boundary chains from s to t (L_{chain} and R_{chain}) are mutually weakly visible. In other words each point on the left chain is visible from at least one point on the right chain and vice versa, see Figure 1(a).

In this research, our goal is equipping the simple robot, with a smallest set of additional capabilities, in order to reach the goal along a shorter route. So, we consider the following extension of the simple robot. The robot carries a compass that denotes to it the north, west, south and east directions. It can moves toward the directions, in additional to the gap tracking (Figure 1). We present an online search strategy for exploring the street environment, from a vertex s to a vertex t, for the empowered robot with the competitive ratio of $3\sqrt{2}$. The ratio is almost half of the competitive ratio of 9, presented in the previous research for the simple robot, not equipped with a compass [11].

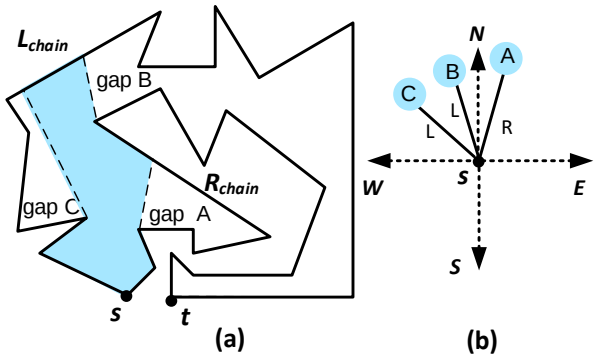


Figure 1: (a) A street polygon. The colored region is the visibility polygon of the point robot at the start point s. (b) The position of discontinuities in the depth information (gaps) reported by the sensor and directions of the compass at the start point s.

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 [7]. Also, Icking, et al. presented an optimal search strategy for the problem with the competitive factor of $\sqrt{2}$ [5].

*Department of Computer Engineering, University of Science and Culture, Tehran, Iran, a.tabatabaei@usc.ac.ir
†Sharif University of Technology and Institute for Research in Fundamental Sciences (IPM), Tehran, Iran, ghodsi@sharif.edu
‡Department of Computer Engineering, Qazvin Branch, Azad University, Qazvin, Iran, shapouri@qiau.ac.ir

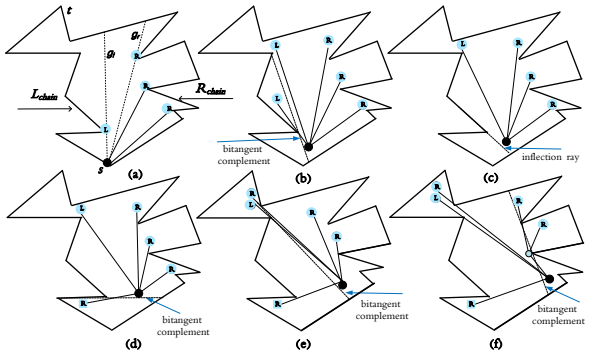


Figure 2: Street polygons, and the dynamically changes of the gaps as the robot walks towards a gap in street polygon. The dark circle is the location of the robot, and squares and other circles denote primitive and non-primitive gaps respectively. (a) Existing gaps at the start point. (b) A split event. (c) A disappearance event. (d) An appearance event. (e) Another split event. (f) A merge event.

Many online strategies for patrolling unknown environment such as street, generalized street, and star polygon are presented in [4, 8].

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 [9, 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 using 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, that allowing use of many pebbles reduces the factor to 9 [1, 11, 12].

Another minimal sensing model was presented by Suri, et al. [10]. 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 of 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 [3].

2 Preliminaries

2.1 The Sensing Model and Motion Primitives

The basic robot has an abstract sensor that reports a cyclically ordered of discontinuities in the depth information (gaps) in its visibility region. 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 2.

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. By equipping the robot with a compass, the robot is empowered; such that it can detects and tracks the north, west, south and east directions to desired number of steps, see Figure 1. Also the robot moves towards the target as it enters in 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 2.

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 in 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 2(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 *[5, 11] 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 of the gap. Thus, there is no ambiguity and the robot moves towards the gap, see Figure 3(a). When both of g_r and g_l exist, a funnel case arises, the angle between g_r and g_l that is always smaller than π is called the opening angle [5], see Figure 3(b). At each funnel case, usually, a detour from the shortest path is unavoidable.

2.3 Essential Information

All we maintain during the search strategy is 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, (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, (Figure 2(e)).
3. When g_l or g_r disappears, the robot may achieve a critical point in which a funnel situation ends, (point 2 in Figure 3(b)).

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

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 are maintained and dynamically updated as explained in the previous section.

3.1 Main 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, see Figure 3(a). The robot moves towards the gap until the target is achieved or a funnel situation arises, (point 2 in Figure 3(a)).
- If both of g_r and g_l exist, a funnel case arises. The possible locations of the compass directions are as follows:
  1. One of the directions of the compass lies in the opening angle (start point s in Figure 3(b)). By our strategy the robot moves along the direction.
  2. Two directions of the compass lie in the opening angle (point 1 in Figure 3(b)). In order to bound the detour, the robot moves one step towards one of the compass direction, and moves one step towards the other, alternatively.
  3. None of the compass directions lies in the opening angle (point 3 in Figure 3(b)). The robot moves towards g_r or g_l.

  The robot continues to move along the selected path until the position of the gaps changes (point 1 in Figure 3(b)), or the critical point of the funnel achieved (point 2 in Figure 3(b)).

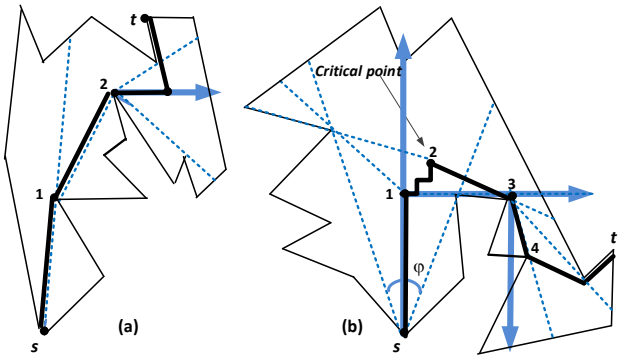


Figure 3: Bold path is the robot search path. (a) There is only g_r. (b) g_r and g_l are the two most advanced gaps at the start point s. The angle between the gaps, φ, is the opening angle at the start point.

3.2 Correctness and Analysis

Throughout the search, the robot path coincides with the shortest path unless a funnel case arises. Then, in order to prove the competitive ratio of our strategy, we compare length of the path and the shortest path in a funnel case. We use the concept of opening angle to calculate the competitive ratio.

Lemma 2 *In the alternative strategy that used in case two, the detour from shortest path for a small opening angle is shorter than detour for a large opening angle [13].*

Theorem 3 *Our deterministic strategy guarantees a path at most $3\sqrt{2}$ times longer than the shortest path, in the street from s to t.*

Proof. According to the Lemma 2, when there are two directions of the compass in the opening angle, the greatest deviation from the shortest arises. We show

the directions with X and Y. Without loss of generality, assume that moving towards g_r coincides with the shortest path. The robot alternately moves one step towards X and moves one step towards Y until it reaches critical point (point $p(x', y')$ in Figure 4). At the critical point, the robot moves towards the only existing advanced gap (point $q(x, y)$). If we compare length of the robot search path with L_1-shortest path, moving along one of the directions (for example X) is correct and moving along the other is deviation from the L_1-shortest. The robot traverses a maximum length of $|x'|+|y'|+|y'|+|x|+|y|$ to reach point q, where $|x'| = |y'| \leq |x|$. So the robot's path length is less than $3(|x|+|y|)$, 3 times longer than the L_1-shortest path. Then, the competitive ratio of our strategy is $3\sqrt{2}$, in the L_2-metric. □

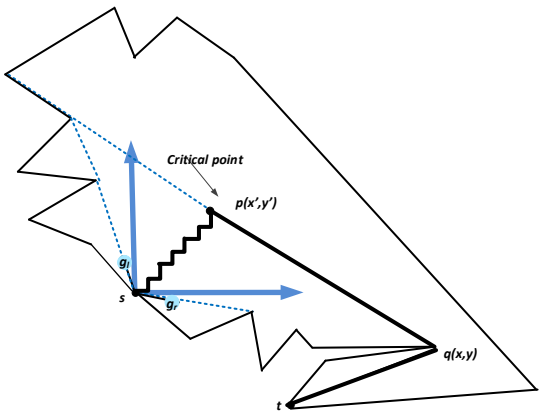


Figure 4: Bold path is the robot search path.

4 Conclusions

In this paper, we considered the problem of walking in street environment for a simple point robot. The basic robot has a minimal sensing model that can only detect the gaps and the target in the street. We have empowered the robot by adding a compass. We presented an online competitive strategy that generates a search path for the empowered robot to reach the target. Length of the path is at most $3\sqrt{2}$ times longer than length of the shortest path. The length of this path is almost half the presented path in the previous research.

References

[1] Abouei Mehrizi, M., Ghodsi, M., Tabatabaei, A. Robots Cooperation for Finding a Target in Streets. *International Conference on Topics in Theoretical Computer Science*, 30–43, 2015.

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

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

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

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

[6] 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.

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

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

[9] 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.

[10] 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.

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

[12] Tabatabaei, A., Ghodsi, M., Shapouri, F. A Competitive Strategy for Walking in Generalized Streets for a Simple Robot. *CCCG*, 75–79, 2016.

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

[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. An optimal on-line strategy for walking in streets with minimal sensing. *ICCC*, 896-899, 2016.

Approximate Discontinuous Trajectory Hotspots

Ali Gholami Rudi*

Abstract

A hotspot is an axis-aligned square of fixed side length s, the duration of the presence of an entity moving in the plane in which is maximised. An exact hotspot of a polygonal trajectory with n edges can be found with the time complexity $O(n^2)$. Defining a c-approximate hotspot as an axis-aligned square of side length cs, in which the duration of the entity's presence is no less than that of an exact hotspot, in this paper we present an algorithm to find a $(1 + \epsilon)$-approximate hotspot of a polygonal trajectory with the time complexity $O(\frac{n\phi}{\epsilon} \log \frac{n\phi}{\epsilon})$, where ϕ is the ratio of average trajectory edge length to s.

Keywords: Computational geometry, geometric algorithms, trajectory analysis, trajectory hotspots.

2010 Mathematics subject classification: 68U05.

1 Introduction

Many objects on earth move and huge collections of trajectory data have been collected by tracking some of them with technologies like GPS devices. In the analysis of these trajectories, many interesting geometric problems arise, such as simplification [1], segmentation [2], grouping [3], classification [4], and finding the interesting regions like where objects spend a significant amount of time [5, 6, 7, 8].

Few results have been published to present exact geometric algorithms for the identification of regions that are frequently visited, called *hotspots* in the rest of this paper (several heuristic algorithms have been published though, such as [7]). The movement of an entity (its trajectory) is commonly represented as a polygonal curve. A set of vertices show the location of the entity at specific points in time, and line segments (as edges) connect contiguous vertices. For multiple entities, Benkert et al. [5] defined a hotspot as an axis-aligned square, which is visited by the maximum number of distinct entities. They presented an $O(n \log n)$ time sweep-line algorithm, where n is the number of trajectory vertices, when only the inclusion of a trajectory vertex is considered a visit. For the case where the inclusion of any portion of a trajectory edge is a visit, they presented an $O(n^2)$ algorithm, which subdivides the plane to find a hotspots. They showed in both cases that their algorithm is optimal.

In a more recent paper, Gudmundsson et al. [6] examined different definitions of trajectory hotspots, in which the duration of the entity's presence is significant. In this paper, we focus on one of their definitions, as follows: a hotspot is an axis-aligned square of some pre-specified side length, in which the entity (or entities) spends the maximum possible duration and the presence of the entity in the region can be discontinuous. For this problem and for a trajectory with n edges, they presented an exact $O(n^2)$ algorithm, which subdivides the plane based on the breakpoints of the function that maps the location of a square of the specified side length to the duration of the presence of the entities in that square.

When s is the side length of exact hotspots, a c-approximate hotspot, where $c > 1$, is an axis-aligned square of side length cs, in which the duration of the entity's presence is no less than that of an exact hotspot. In this paper we present an algorithm to find $(1 + \epsilon)$-approximate hotspots of a trajectory in the plane. The algorithm first subdivides each edge to small segments and then finds the square that contains the maximum number of such segments. The time complexity of the algorithm, which shall be presented in the rest of this paper, is $O(\frac{n\phi}{\epsilon} \log \frac{n\phi}{\epsilon})$, where ϕ is the ratio of average length of trajectory edges to s. We then use this algorithm to find a duration-approximate hotspot of a trajectory T with approximation ratio $1/4$, i.e. a square of side length s, in which the entity is present at least $1/4$ of the time it is present in the exact hotspot.

This paper is organized as follows. The algorithm and its analysis are presented in Section 3, after introducing the notation and defining some of the concepts discussed in this paper in Section 2.

2 Preliminaries

A trajectory describes the movement of an entity and is represented as a sequence of vertices in the plane with timestamps that specify the location of the entity at different points in time. The entity is assumed to move from one vertex to the next in a straight line and with constant speed.

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

Definition 1 *The weight of a square r with respect to a trajectory T, denoted as $w(r)$ is the total duration in which the entity spends inside it. Also, $w(u)$ for any sub-trajectory (or edge) u of T, indicates the duration of u (the difference between the timestamps of its end-points).*

The input to the problem studied in this paper is a trajectory T and the value of s. The goal is to find a hotspot of T (Definition 2). Unless explicitly mentioned otherwise, every square discussed in this paper is axis-aligned and has side length s.

Definition 2 *A hotspot of trajectory T in R^2 is a placement of a square of side length s in the plane with the maximum weight (the duration of the presence the entity in the square is maximised).*

A hotspot of a trajectory with n edges can be found with the time complexity $O(n^2)$ [6]. Our goal in this paper is finding approximate hotspots of a trajectory more efficiently (Definition 3).

Definition 3 *A c-size-approximate (or c-approximate for brevity) hotspot is a square whose weight is at least the weight of an exact hotspot and its side length is c times s.*

In Definition 3, hotspots are enlarged. A more natural definition may be squares with the same size as the exact ones, but with smaller weights (Definition 4).

Definition 4 *A c-duration-approximate hotspot is a square of side length s, whose weight is at least c times the weight of an exact hotspot.*

3 The Approximation Algorithm

For a trajectory T in R^2 and some constant ϵ, where $\epsilon > 0$, in this section we present an algorithm to find a $(1+\epsilon)$-approximate hotspot and use it to find a $1/4$-duration-approximate hotspot.

We first subdivide each edge of the trajectory into segments of height and width at most ϵs; we do so by covering each edge by non-overlapping axis-aligned squares of side length $\epsilon s/2$. For each such tile, we add a point at its centre. Let the weight of this point be the duration of the portion of the edge that is inside its tile. For every resulting point p, we use p_t to denote its corresponding tile and p_g to denote the corresponding segment (Figure 1). Note that the tiles of different edges may overlap.

Definition 5 *The point-weight of a square r with respect to a trajectory T, denoted as $w'(r)$, is the total weight of the points inside r. Also, $w'(p)$ for point p denotes the weight of point p.*

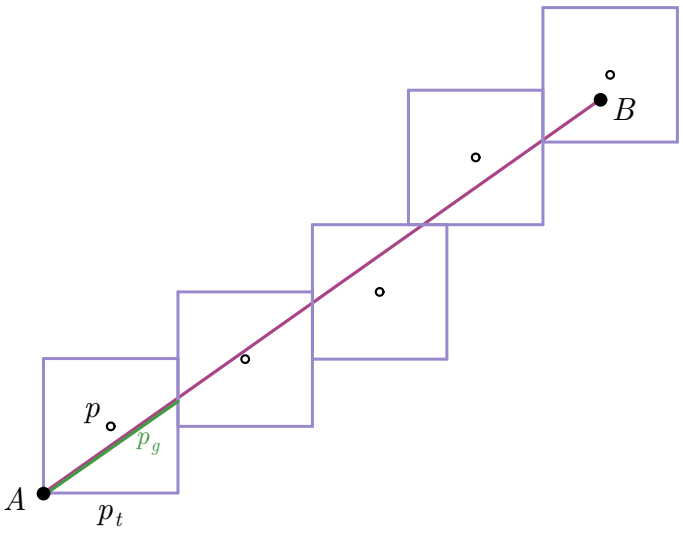


Figure 1: Tiling an edge AB and adding a point at the centre of each tile

Lemma 6 *Let r be a square of side length s and let r' be a square of side length $s+\epsilon s/2$, with the same centre of gravity. We have $w(r) \leq w'(r')$.*

Proof. Every point in the sub-trajectory inside r belongs to some segment p_g corresponding to tile p_t and point p. Let P be the set of all points, whose segments are intersected by r. Since r contains all or some part of any segment corresponding to these points, $w(r) \leq \sum_{p\in P} w(p)$. Since r' is $\epsilon s/4$ longer than r at each side, when r intersects p_t, the centre of p_t, p, is contained in r'. Therefore, $\sum_{p\in P} w(p) \leq w'(r')$, which implies that $w(r) \leq w'(r')$. □

Lemma 7 *Let r be a square of side length $s+\epsilon s/2$ and let r' be a square of side length $s+\epsilon s$, with the same centre of gravity. We have $w'(r) \leq w(r')$.*

Proof. Let P denote the set of points inside r; the point-weight of square r is the sum of the weights of the points in P. Suppose p is a member of P. Since p is the centre of p_t and p is inside r, the whole of p_t is contained in r', because r' is $\epsilon s/4$ longer than r at each side. This implies that the whole of p_g is inside r'. Therefore, $\sum_{p\in P} w'(p) = \sum_{p\in P} w(p_g) \leq w(r')$, as required. □

In Theorem 8, we use a data structure for storing m numbers that supports obtaining their maximum in $O(1)$ and increasing the numbers in any continguous interval of the numbers by a value in $O(\log m)$. This can be implemented using a balanced binary tree that stores the numbers at the leaves, for each internal node maintains the maximum value of its subtree, and stores changes to the leaves of subtrees at internal nodes, instead of updating the value of every node in that subtree

(alternatively a range minimum query data structure, like the Fenwick tree [9], can be augmented).

Theorem 8 *Given a trajectory T in R^2 and the value of s, after tiling, a square of side length $s + \epsilon s/2$ and with the maximum point-weight can be found with the time complexity $O(m \log m)$, where m is the number of points.*

Proof. To find a square with the maximum weight, it suffices to search among the squares that have a point on each of their lower and left sides (any square with the maximum weight can be moved up and right without changing its point-weight, until their lower and left sides meet a point).

Let σ be the sequence of points in P, ordered by their y-coordinate. We sweep the plane horizontally using two parallel sweep lines with distance $s+\epsilon s/2$ as follows. During the sweep line algorithm, we maintain the point-weight of m squares in the data structure W, such that the i-th number in W denotes the point-weight of the square whose lower side has the same height as the i-th point and its left and right sides are on the sweep lines; we use r_i to refer to this square.

In the sweep line algorithm, we process the following events: when the left or the right sweep line intersects a point p. We process an event for point p as follows. Let p be the i-th item of σ and let j be the index of the lowest point in σ such that the difference between the height of p and the j-th point of σ is at most $s+\epsilon s/2$; the value of j can be found using binary search on σ. When p meets the right sweep line, we increase the weight of every square r_k such that $j \leq k \leq i$ by $w'(p)$, because every such square contains p. Similarly, when p meets the left sweep line, we decrease the weight of every square r_k such that $j \leq k \leq i$ by $w'(p)$. During the sweep line algorithm, we record the square with the maximum weight in W. At the end of the algorithm, it denotes a square with the maximum point-weight among all squares with a vertex on their lower and left sides.

The complexity of sorting m points based on their y-coordinate and handling m events, each with complexity $O(\log m)$ is $O(m \log m)$. □

In Theorem 9, we present and analyse the main algorithm.

Theorem 9 *Given a trajectory T in R^2 and the value of s, there is an algorithm that finds a $(1 + \epsilon)$-approximate hotspot of trajectory T with the time complexity $O(\frac{n\phi}{\epsilon} \log \frac{n\phi}{\epsilon})$, in which ϕ is the ratio of average length of trajectory edges to s.*

Proof. After tiling, as described in the beginning of this section, an edge of length d is subdivided into at most $\lceil \frac{d}{\epsilon s} \rceil$ segments. Therefore, if the total length of the edges of T is a, the number of resulting segments is at most $\frac{a}{\epsilon s} + n$, which is equal to $O(\frac{n\phi}{\epsilon})$ asymptotically. Theorem 8 shows how the square with the maximum point-weight, r, can be found in $O(m \log m)$. Let r' be the square with the same centre of gravity as r but of side length $s + \epsilon s$. Also, let h denote the weight of an exact hotspot of T. We show that r' is a $(1 + \epsilon)$-approximate hotspot.

Lemma 6 implies that there is at least one square with side length $s + \epsilon s/2$ whose point-weight is equal to h, the weight of an exact hotspot of T (of side length s). Since, r is the square with the maximum point-weight among squares of side length $s + \epsilon s/2$, its point-weight is at least h. Furthermore, Lemma 7 shows that the weight of r' is at least the point-weight of r (at least h). Therefore, the algorithm finds a square of side length $s + \epsilon s$ and with weight at least h; a $(1 + \epsilon)$-approximate hotspot by Definition 3. □

We now use the algorithm presented in Theorem 9 to find a duration-approximate hotspot of a trajectory in the plane. For that, we need Observation 1, which can be shown by placing the smaller squares at the corners of a hotspot.

Observation 1 *Let h be the weight of an exact hotspot of a trajectory T in the plane. There exists a square of side length cs and weight at least $h/4$, provided that $c \geq 1/2$.*

Corollary 10 *Given a trajectory T in R^2 and the value of s, there is an algorithm that finds a $1/4$-duration-approximate hotspot of trajectory T with the time complexity $O(n\phi \log n\phi)$, in which ϕ is the ratio of average length of trajectory edges to s.*

Proof. Theorem 9 for hotspot side length $s' = s/2$ and $\epsilon = 1$ yields a square r of weight h and side length s. Since h is the maximum weight of the squares with side length $s/2$, Observation 1 implies that the weight of an exact hotspot of side length s of T cannot be greater than $4h$. Therefore, r is a $1/4$-duration-approximate hotspot of T. □

References

[1] M. J. van Kreveld, M. Löffler, and L. Wiratma. On optimal polyline simplification using the hausdorff and fréchet distance. In *International Symposium on Computational Geometry*, pages 56:1–56:14, 2018.

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

[3] Kevin Buchin, Maike Buchin, Marc J. van Kreveld, Bettina Speckmann, and Frank Staals. Trajectory grouping structure. *JoCG*, 6(1):75–98, 2015.

[4] 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.

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

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

[7] M. L. Damiani, H. I., 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 (SIGSPATIAL)*, pages 253–262, 2014.

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

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