5th Iranian Conference on

Computational Geometry

(ICCG 2022)

Proceedings

Sharif University of Technology
Tehran, Iran, February 24, 2022

Foreword

We are pleased to bring you this collection of the accepted papers from the fifth Iranian Conference on Computational Geometry (ICCG 2022), held at Sharif University of Technology, Tehran, Iran, on February 24, 2022. As ICCG does not have formal proceedings, the papers presented here could also be submitted to peer-reviewed conferences and journals.

ICCG intends to bring together students and researchers from academia and industry to promote research in the fields of Combinatorial and Computational Geometry. All submissions to ICCG 2022 were carefully reviewed by the members of an international Program Committee comprising 19 leading scientists, and at least three review reports were provided for each paper. Besides paper presentations, the program included one invited talk by Dr. Prosenjit Bose, a renowned computer scientist from Carleton University, Canada, who gave a presentation in the memory of Dr. Saeed Mehrabi. Due to the COVID-19 pandemic, ICCG 2022 was forced to run virtually through the excellent facilities provided by Sharif University. of Technology.

We would like to thank the authors and the invited speaker who contributed to the success of ICCG 2022. We are also grateful to all PC members for their professional work in providing expert review reports. Last but not least, our gratitude extends to Sharif University of Technology for generously supporting the conference.

Shahin Kamali and Alireza Zarei (Program Committee Chairs)

Invited Speaker

Prosenjit Bose Carleton University, Canada

General Chair

Shahin Kamali University of Manitoba
Alireza Zarei Sharif University of Technology

Program Committee

Mohammad Ali Abam	Sharif University of Technology
Sharareh Alipour	IPM
Alireza Bagheri	Amirkabir University of Technology
Yeganeh Bahoo	Ryerson University
Sergey Bereg	University of Texas at Dallas
Ahmad Biniaz	University of Windsor
Mansoor Davoodi Monfared	Institute for Advanced Studies in Basic
Amin Gheibi	Amirkabir University of Technology
Ali Gholami Rudi	Babol Noshirvani University of Technology
Mahdieh Hasheminezhad	Yazd University
Akitoshi Kawamura	Kyoto University
Philipp Kindermann	Universität Würzburg
Jayson Lynch	University of Waterloo
Debajyoti Mondal	University of Saskatchewan
Mostafa Nouri Baygi	Ferdowsi University Of Mashhad
Zahed Rahmati	Amirkabir University of Technology
Hamid Zarrabi-Zadeh	Sharif University of Technology

Local Organizers

Mohammad Reza Bahrami Sharif University of Technology
Mohammad Sadegh Borouny Sharif University of Technology

Conference Program

Thursday February 24

Session 1

Session 2

Index of Authors

Geometric Spanner in MPC Model

Mohammad Ali Abam* Mohammad Reza Bahrami† Peyman Jabbarzade Ganje‡

Abstract

The importance of processing large-scale data is growing rapidly in contemporary computation. In order to design and analyze practical distributed algorithms, recently, the MPC model has been introduced as a theoretical framework. In this paper, we present the geometric spanner construction in the Massively Parallel Computing (MPC) model. Constructing θ-graph using the given ϵ, we modified distributed range tree to find the nearest point to the apex of a θ-cone efficiently and form a $(1+\epsilon)$-spanner in $\mathcal{O}(1)$ rounds and $\widetilde{\mathcal{O}}(S)$ time, where S is the memory size of a single machine.

1 Introduction

We consider the problem of finding a geometric spanner for a set of points in $\mathbb{R}^d$ in the MPC model. In this section we talk about both the spanner and MPC model and also we motivate our work. Then the sketch of the proof is presented.

1.1 Spanners

Given an edge-weighted graph $G = (V, E)$, the t-spanner of G is a weighted graph $S = (V, E')$ where $E' \subseteq E$ and for every two vertices u and v, $d_S(u, v) \leq t \cdot d_G(u, v)$, where $d_S(u, v)$ indicates the shortest path distance between u and v in graph S. That is, the spanner of graph G, is a graph which preserve the distances by a t multiplicative factor. For a fixed graph G, the smallest t, which preserve spanner property is called dilation or stretch factor.

Among general graphs, we are focused on Euclidean geometric graphs, which are complete graphs with a Euclidean metric edge weight function. Hence, a geometric spanner is a complete Euclidean graph, in which for any two points we have the weight of edge $(u, v) \leq |uv|$, where $|uv|$ indicates the Euclidean distance between points u and v.

Also, we can view a spanner as compression for the main graph. Assuredly, the fewest edges it has, the more compressed it is considered. However, we look forward to keeping the dilation as low as possible. Assuming Erdös girth conjecture, there is a tight bound for general graphs that there is a $(2k-1)-$*spanner* with $\mathcal{O}(n^{1+1/k})$ edges [19].

The extensiveness of spanners' applications like approximating shortest path queries, routing distributed and parallel algorithms, designing work efficient PRAM algorithms [13], and etc. make researchers come up with many ideas and variants to build them efficiently.

1.2 MPC Model and Geometric Problems

The importance of processing large scale data-set is becoming more crucial every day. Over past years we have faced exponential growth in the speed of generating new data. Recently, many tools have been developed to process these data practically, among them Map-Reduce model [10], Hadoop [21], Spark [22] and Dryad [16]. Due to the diversity of tools and methods, some general theoretical models were built to conceptualise current and forthcoming tools. One of the most realistic and practical abstract models is *Massively Parallel Computing* or **MPC** for short, which was introduced by Karloff et. al [17], as a computation model for Map-Reduce.

In this model [7], [14], [17], it is assumed that there are M machines, each of which has a memory of size S known as local memory and the given input size is N which distributed arbitrarily. Usually, we assume the memory size is polynomially smaller than the input size, i.e. $S = \widetilde{\mathcal{O}}(N^\alpha)$ where $0 < \alpha < 1$. As the whole data must fit into machines, we consider $M \geq \frac{N}{S}$. Also, it is common to add polylog factor to the number of machines, i.e. $M = \widetilde{\mathcal{O}}(\frac{N}{S}) = \mathcal{O}(N/S\ poly(\log N))$. Also, despite K-machine model [18], there is no point to point connection between machines, so there is a $\widetilde{\mathcal{O}}(S)$ communication limit for each machine.

However, while we consider graph problems, given a graph $G = (V, E)$ the total memory N is considered as the number of graph edges E which can be $\mathcal{O}(n^2)$ where $n = |V|$. But we are interested in settings where local memory is small. Thus, there are three studied regimes in the MPC model for graph problems; Strong superlinear where $S \geq n^{1+c}$ for a constant $c > 0$, near-linear where $S = \widetilde{O}(n)$, and strongly sublinear where $S = n^\gamma$ for a constant $\gamma < 1$.

In geometric problems, especially in geometric graphs, as the weight function is defined implicitly without keeping all edge weights on memory, we only con-

*Department of Computer Science, Sharif University of Technology, Tehran, Iran abam@sharif.edu

†Department of Computer Science, Sharif University of Technology, Tehran, Iran mrbahrami@ce.sharif.edu

‡Department of Computer Science, University of Maryland, College Park, MD, US peymanj@cs.umd.edu

sider in the later regime. Henceforth, from now on we consider n points and $M = \widetilde{\mathcal{O}}(n^{1-\gamma})$ machines where each of them has $S = \widetilde{\mathcal{O}}(n^{\gamma})$ local memory for a constant $\gamma < 1$.

1.3 Prior Works

There is an extensive line of research on *spanners* and also *geometric spanners*, especially after Chew [9] introduced them in 1986 but term spanner coined by Peleg and Ullman [20]. In addition to some surveys [11], [15], there is a book completely about geometric spanner [19].

Although there is a classic greedy algorithm for computing geometric spanner [4], there are many works to reduce the running time while preserving useful properties like minimizing maximum degree or not using too many more edges. E.g. using θ-graph [5] a geometric spanner can be built in $\mathcal{O}(n \log n^{d-1})$ time with $\mathcal{O}(n)$ edges.

Assuming a large data set, there are some works on streaming mode like [6], which construct a spanning tree in a single path with $\mathcal{O}(\min(m, kn^{1+1/k}))$ edges. Also there is a recent work for general graph by Ghaffari et.al [8], constructs a $\mathcal{O}(k^{1+o(1)})$-spanner in $\mathcal{O}(\log \log n)$ rounds in strongly sublinear regime with $\mathcal{O}(n^{1+1/k} \cdot \log k)$ many edges. Also Ghodsi et al. provide a constant round algorithm in MapReduce framework to build the geometric spanner in L_1 metric in $\mathbb{R}^2$ using $\mathcal{O}(n/\epsilon^2)$ edges [3].

In recent years, also, there is a growing line of research to solve the geometric problems in the distributed model; such as triangulation algorithms in 2D and 3D [23], or finding the nearest neighbor in K-Machine mode [12].

1.4 Overview of Method

Our algorithm works as follows, for each cone σ we build a distributed data structure (*Next Tree*) to obtain the nearest point inside the cone. Inside each cone, it takes $\mathcal{O}(d)$ rounds to find the point which is the nearest one with respect to its projection on the representative edge. This is constant if we naturally assume d is constant. To obtain constant round complexity for all points, we must re-distribute sub-trees stored on each machine among searching round, while we look for the nearest point image on the representative edge. This leads to an algorithm with constant rounds of computation and $\mathcal{O}(S)$ time.

2 Preliminaries

θ-graph We use an approach based on θ-graph [1] to build our spanner. Given constant ϵ, we choose θ such that $\cos 2\theta - \sin 2\theta \geq 1/(1+\epsilon)$. On the other hand, a θ-cone is the intersection of d half-spaces such that the angle of any two rays emanating at the cone's apex and being inside the cone is at most θ. Hence we define the canonical cone set $\mathcal{C}$, as a collection of $\mathcal{O}(1/\theta^{d-1})$ interior disjoint θ-cones, where share apex at the origin. For a $p \in \mathbb{R}^d$, translating each $\sigma \in \mathcal{C}$ to the point p, constructs our θ-graph. Choosing one of the edges of each $\sigma \in \mathcal{C}$ as representative edge (ℓ_θ) the following lemma from [1] can be stated:

Lemma 1 *Let $\mathcal{C}$ be a collection of θ-cones, where $\cos 2\theta - \sin 2\theta \geq 1/(1+\epsilon)$. Choosing ℓ_σ as representative edge for each θ-cone σ and connecting each point p to $q \in \sigma(p)$ that its projection on ℓ_σ minimize the Euclidean distant to p lead us to a $(1+\epsilon)$-spanner.*

Distributed Range Tree A d-dimensional range tree is a data structure that can answer orthogonal range queries in $\mathcal{O}(\log^{d-1} n)$ time. If $d = 1$, the range tree is a balanced binary search tree. For $d > 1$, the range tree consists of a primary structure in which every node has a range tree on $d-1$ dimension for the point below that node.

Agarwal et al. in [2] show that it is possible to build a range tree in the MPC model efficiently. As the range tree should be stored in a distributed fashion, a primary top tree is built which contains $\mathcal{O}(s^{1/5})$ top nodes of the range tree. Hence for each node, the top of the secondary structure build locally and recursively. The procedure continues by using leaves of the primary structure and build the tree recursively. We will use this structure and the result of the following theorem from [2]:

Theorem 2 *Given a set $P \subset \mathbb{R}^n$, a range tree on P can be built with size $\mathcal{O}(n \log^{d-1} n)$ in constant rounds and $\mathcal{O}(s \log^d s)$ time which can answer orthogonal queries in $\mathcal{O}(1)$ rounds and $\mathcal{O}(\log^{d-1} n)$ time in MPC model.*

3 Next Tree

First, we consider the definition of Next Tree. The Next Tree is an augmented distributed range tree that helps to find the nearest point in a cone to the apex of the query cone. To build this we amend the range tree [2].

As we discussed earlier, we need a range tree to find the subset of points inside the query cone σ. Also, we need to find the nearest point $q \in \sigma(p)$ to connect it to p in terms of q's projection to one of $\sigma(p)$ representative edges or even any arbitrary line at that half-space. Consider a line in one of the mentioned half-spaces as ℓ_σ, We assume that q'_σ the projection of q on ℓ_σ is stored within q.

To build the Next Tree, we add one dimension to each point which contains its rank on ℓ_σ. This rank can be computed easily with a constant round of computation [14]. Afterward, a $(d+1)$-dimension distributed range tree is built open this augmented point set P_σ, just like

the one appeared in [2]. We only alter the query procedure to find the lowest rank point in the canonical subset. This leads us to the following lemma:

Lemma 3 *For a point set $P \subset \mathbb{R}^d$, Next Tree can be built in $\mathcal{O}(d)$ rounds, in $\mathcal{O}(s \log^{d+1} n)$ time with size $\mathcal{O}(n \log^d n)$.*

A query to Next tree is an orthogonal cone $\sigma(p)$. The desired response is a point q where q'_σ, the projection of q on ℓ_σ, is the next projected point on ℓ_σ and $q \in \sigma(p)$.

Lemma 4 *In d-dimension Next Tree, the next query can be answered in $\mathcal{O}(1)$ rounds and $\mathcal{O}(\log^{d+1} n)$ time.*

Proof. Without loss of generality consider a orthogonal cone $\sigma(p) = (x_1, \ldots, x_d)$ which is open to $(+\infty, +\infty)$. Just like distributed range tree for each query, we must go through the data structure. For each x_i, where $i \leq d$, our search in Next Tree is a path with at most $\mathcal{O}(\log n)$ points. Just like lemma 3, we must repeat it for each d level. However, at the d+1 level, we take a $\mathcal{O}(\log n)$ path just to find the minimum ranked point as the $(d+1)$th coordinate of q is rank of q_σ, which takes $\mathcal{O}(1)$ round and $\mathcal{O}(\log^{d+1} n)$ time for a single point. □

4 Spanner Construction

Briefly, the procedure of constructing the spanner is as follows, as algorithm 1 specify, for each canonical cone σ, we transform points and build a Next Tree. Then we find the nearest point to each cone's apex and connect them.

Algorithm 1 Distributed algorithm to build a $(1+\epsilon)-$*spanner* using θ-graph

Require: Point set P, ϵ, n nodes distributed arbitrary

1: Compute θ based on ϵ and canonical θ - cones $\mathcal{C}$
2: **for all** σ in $\mathcal{C}$ **do**
3: `transform_and_order_points(`σ, ℓ_σ`)`
4: T_σ = `build_next_tree(`σ`)`
5: `find_nearest(`P, T_σ`)`
6: **end for**

Recall that for each $1 \leq i \leq n$, $\sigma(p_i)$ is translated of the same canonical cone $\sigma \in C$. Henceforth, we need to project all points into ℓ_σ and also transform all points to form all $\sigma(p_i)$ orthogonal. After that, we must compute the rank of each projected point on ℓ_σ which can be done in a constant rounds for all points [14]. Hence the following proposition is inferred:

Proposition 5 *In algorithm 1, the procedure* `transform_and_order_points` project the points onto ℓ_σ and rank them in $\mathcal{O}(1)$ rounds and $\widetilde{\mathcal{O}}(S)$ time.

Now we must show how `find_nearest` works in parallel for all points. If we want to find the nearest point one by one it takes $\mathcal{O}(n)$ rounds to build the spanner which is unacceptable in our model.

By lemma 4, it is clear that query one point in the Next Tree requires $\mathcal{O}(1)$ rounds of computation. However, we design an algorithm to find the next point, the adjacent vertex in the desired spanner, for all points in constant rounds. To this end, we must be able to initiate the search procedure locally for all machines. Hence, each machine needs to have the top sub-tree or the master node which contains the top $\mathcal{O}(S^{1/5})$ nodes [2]. This can be done on $\log_{S^{1/5}} n = \mathcal{O}(1)$ rounds of computation using the broadcast tree [14]. Afterward, in each machine M_i, s points have been searched and in the next step their search procedure must continue among set $\mathcal{M}_i = \{M_1, \ldots, M_s\}$. As a result, we can imagine some cases in which a machine is overwhelmed by $\Omega(S)$ nodes to resume its search procedure. To resolve this problem, at the end of the first searching round, each machine sends the number of applicant nodes to their destination machine say M_i. As each machine sends a number to M_i, if the sum of these numbers exceeds S, M_i should make a copy of its self and inform nodes about the new destinations. M_i choose these machines randomly. Also it sends the number of required machines, so using the broadcast tree, we can handle the distribution. The following lemma shows us that this procedure (algorithm 2) can be done on constant rounds of computation.

Lemma 6 *Targeted sub-trees to resume the query procedure to find neighbour of the point subject to a cone σ can be redistributed to be able to host all requests at a round, with $\mathcal{O}(1)$ rounds of computation and $\mathcal{O}(S)$ time.*

Proof. Notice that, the total number of points is n and also a machine requires another helping machine for every $\mathcal{O}(S)$ applicant point. As $S \geq s = n/M$, where s is the number of points hosted by each machine, the total number of sub-trees to redistribute is less than M. This is also an upper bound for redistributing a fixed sub-tree. Again using the broadcast tree to distribute the sub-trees. A machine with exceeded query requests should choose uniformly at random $\max(S, r/S)$ machines, where r is the number of arriving requests. Using the broadcast tree this will end up in the constant round. However, if a machine receives more nodes than its memory size, it should drop extra sub-trees. So, we consider X_i the number of guest sub-trees requested to join M_i. As $E[X_i] = 1$ we can use chernoff bound:

$$Pr[X_i \geq \log n] \leq \exp(\frac{-\log^2 n}{3}) \leq \frac{1}{n^2}$$

Using union bound, the probability of no machine re-

ceive more than $\mathcal{O}(\log n)$ nodes is:

$$Pr[X_1 \cup \cdots \cup X_M] \leq \sum_{1}^{M} \frac{1}{n^2} = \frac{M}{n^2} \leq \frac{1}{n}$$

So, with high probability, no machine receives more than $\log n$ nodes. Using the fact that $S = \widetilde{\mathcal{O}}(n^\delta)$, the lemma concludes. □

Algorithm 2 `find_nearest`(P, T_σ)

Require: Point set P and its projection on ℓ_σ within, Pre-computed and stored T_σ, Constant $k = \log_{S^{1/5}} n = \mathcal{O}(1)$

1: Distribute the master node to all machines and set as current searching node M_i on each machine
2: **for** k times **do**
3: Perform the search on M_i locally and find the $\mathcal{M}_i$ machine set to resume the search
4: Inform $\mathcal{M}_i$ members about the coming requests
5: Redistribute the machine M_i if necessary
6: **end for**

Using proposition 5 and lemma 3, we conclude that the first two lines of the for loop in Algorithm 2 requires $\mathcal{O}(1)$ rounds and $\widetilde{\mathcal{O}}(S)$ time. Moreover, by lemmas 4 and 6 the `find_nearest` procedure requires $\mathcal{O}(1)$ rounds and $\mathcal{O}(S)$ time. Assuming d and ϵ are constants leads to the following theorem:

Theorem 7 *In the MPC model, a geometric $(1+\epsilon)$-spanner can be built using $\mathcal{O}(1)$ rounds of computation and $\widetilde{\mathcal{O}}(S)$ time.*

5 Concluding Remarks

We considered constructing a geometric spanner in the MPC model. Benefiting from θ-graph, we designed an algorithm to build this spanner in constant rounds of computation. However, we can consider the same problem to find a spanner with a smaller final degree. Also, we can consider the problem in dynamic or kinetic models if we consider the possibility of temporal changes in our point, with less work.

References

[1] M. A. Abam and M. de Berg, "Kinetic spanners in $\mathbb{R}^d$," in *Proceedings of the Twenty-Fifth Annual Symposium on Computational Geometry*, ser. SCG '09, Aarhus, Denmark: Association for Computing Machinery, 2009, pp. 43–50.

[2] P. K. Agarwal, K. Fox, K. Munagala, and A. Nath, "Parallel algorithms for constructing range and nearest-neighbor searching data structures," in *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, ser. PODS '16, San Francisco, California, USA: Association for Computing Machinery, 2016, pp. 429–440.

[3] S. Aghamolaei, F. Baharifard, and M. Ghodsi, "Geometric spanners in the MapReduce model," in *Lecture Notes in Computer Science*, Springer International Publishing, 2018, pp. 675–687.

[4] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares, "On sparse spanners of weighted graphs," *Discrete Comput. Geom.*, vol. 9, no. 1, pp. 81–100, Dec. 1993.

[5] S. Arya, D. M. Mount, and M. Smid, "Dynamic algorithms for geometric spanners of small diameter: Randomized solutions," *Computational Geometry*, vol. 13, no. 2, pp. 91–107, 1999.

[6] S. Baswana, "Streaming algorithm for graph spanners—single pass and constant processing time per edge," *Information Processing Letters*, vol. 106, no. 3, pp. 110–114, 2008.

[7] P. Beame, P. Koutris, and D. Suciu, "Communication steps for parallel query processing," in *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, ser. PODS '13, New York, New York, USA: Association for Computing Machinery, 2013, pp. 273–284.

[8] A. S. Biswas, M. Dory, M. Ghaffari, S. Mitrović, and Y. Nazari, "Massively parallel algorithms for distance approximation and spanners," in *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '21, Virtual Event, USA: Association for Computing Machinery, 2021, pp. 118–128.

[9] L. Chew, Paul, "There are planar graphs almost as good as the complete graph," *Journal of Computer and System Sciences*, vol. 39, no. 2, pp. 205–219, 1989.

[10] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.

[11] D. Eppstein, "Chapter 9-spanning trees and spanners," *Handbook of Computational Geometry*, pp. 425–461,

[12] R. Fathi, A. R. Molla, and G. Pandurangan, "Efficient distributed algorithms for the k-nearest neighbors problem," in *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, ser. SPAA '20, Virtual Event, USA: Association for Computing Machinery, 2020, pp. 527–529.

[13] S. Friedrichs and C. Lenzen, "Parallel metric tree embedding based on an algebraic view on moore-bellman-ford," *J. ACM*, vol. 65, no. 6, Nov. 2018.

[14] M. T. Goodrich, N. Sitchinava, and Q. Zhang, "Sorting, searching, and simulation in the mapreduce framework," in *Proceedings of the 22nd International Conference on Algorithms and Computation*, ser. ISAAC'11, Yokohama, Japan: Springer-Verlag, 2011, pp. 374–383.

[15] J. Gudmundsson and C. Knauer, "Dilation and detours in geometric networks," *Handbook of Approximation Algorithms and Metaheuristics Chapman & Hall/CRC Computer & Information Science Series*, 2007.

[16] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed data-parallel programs from sequential building blocks," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 3, pp. 59–72, Mar. 2007.

[17] H. Karloff, S. Suri, and S. Vassilvitskii, "A model of computation for mapreduce," in *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '10, Austin, Texas: Society for Industrial and Applied Mathematics, 2010, pp. 938–948.

[18] H. Klauck, D. Nanongkai, G. Pandurangan, and P. Robinson, "Distributed computation of large-scale graph problems," in *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '15, San Diego, California: Society for Industrial and Applied Mathematics, 2015, pp. 391–410.

[19] G. Narasimhan and M. Smid, *Geometric Spanner Networks*. USA: Cambridge University Press, 2007.

[20] D. Peleg and J. D. Ullman, "An optimal synchronizer for the hypercube," in *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing*, 1987, pp. 77–85.

[21] T. White, *Hadoop: The definitive guide.* " O'Reilly Media, Inc.", 2012.

[22] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster computing with working sets," in *Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10, Boston, MA: USENIX Association, 2010, p. 10.

[23] H. Zhou, H. Wu, S. Xia, M. Jin, and N. Ding, "A distributed triangulation algorithm for wireless sensor networks on 2d and 3d surface," in *2011 Proceedings IEEE INFOCOM*, 2011, pp. 1053–1061.

Red Blue Set Cover Problem on Axis-Parallel Hyperplanes and Other Objects

Abidha V P* Pradeesha Ashok

Abstract

In this paper we study the Red Blue Set Cover problem which is a bichromatic variant of the set cover problem. We prove that the Red Blue Set Cover problem is NP-hard even for red and blue points in $\mathbb{R}^2$ and set of axis parallel lines. We then study the parameterized complexity of a generalization of this problem, for points in $\mathbb{R}^d$ and a family of axis-parallel hyperplanes in $\mathbb{R}^d$, under different parameterizations. We further consider the Red Blue Set Cover problem for some special types of rectangles in $\mathbb{R}^2$.

1 Introduction

The Set Cover problem and its variants are well studied in Computer Science [2]. Given a universe $\mathcal{U}$ with n elements and a family $\mathcal{F}$ of subsets of $\mathcal{U}$, the minimum Set Cover problem is to find a subset $\mathcal{F}' \subseteq \mathcal{F}$ that covers all elements in $\mathcal{U}$ and $|\mathcal{F}'|$ is minimized. Here $u \in \mathcal{U}$ is said to be covered by $\mathcal{F}'$, if $\exists F \in \mathcal{F}'$ such that $u \in F$.

In this paper, we study a generalization of the Set Cover problem, where $\mathcal{U}$ is bi-chromatic. Given a universe $\mathcal{U} = R \cup B$ of a finite set R of red elements, and a finite set B of blue elements and a family $\mathcal{F}$ of subsets of $\mathcal{U}$, the Red Blue Set Cover (RBSC) problem is to find a subset $\mathcal{F}'$ of $\mathcal{F}$ that covers all blue points of B and the minimum number of red points from R. Here, R and B are points in $\mathbb{R}^d$. The sets in $\mathcal{F}$ correspond to maximal subset of points contained in geometric objects of a certain type. Note that the number of sets in $\mathcal{F}'$ is not optimized. The minimum Set Cover problem can be considered as a special case of the RBSC problem where every set contains a distinct red element.

The RBSC problem was introduced by Carr et al. [3] in 2000, motivated by applications in data mining. Geometric variants of the RBSC problem are also studied. Approximation algorithms for different geometric RBSC problems are given in [4, 8].

The geometric RBSC problem is also studied in the context of Parameterized complexity. Ashok et al.[1] investigate parameterized algorithms on general lines and hyperplanes under an array of parameters and all possible combinations of those parameters.

1.1 Problems studied and results

We study the decision variant of a geometric version of the RBSC problem where given $R \cup B$, and $\mathcal{F} \subseteq 2^{\mathcal{U}}$ and $k_r \in \mathbb{N}$, our objective is to decide if there exists $\mathcal{F}' \subseteq \mathcal{F}$ that covers all points in B and at most k_r points from R.

Theorem 1 *(*) **The** **RBSC** **problem on axis-parallel lines is NP-complete.***[1]

We study the following variants of the RBSC problem.

AxRBSC-hyperplane: We study the RBSC problem for a generalization of lines to d dimension. We consider the problem where $\mathcal{U}$ is a set of points in $\mathbb{R}^d$ and $\mathcal{F}$ is defined by a set of axis-parallel hyperplanes. This problem is NP-hard by theorem 1. We study the parameterized complexity of AxRBSC-hyperplane under different parameterizations. For each parameter, we consider whether the problem is fixed parameter tractable and if yes, we investigate whether the problem admits polynomial kernels. We show that the RBSC problem admits more positive results when restricted to axis-parallel hyperplanes as compared to hyperplanes of any orientation.

AxRBSC-skylines: Bi-directional strips are either of the form $[a, b] \times (-\infty, \infty)$ or $(-\infty, \infty) \times [a, b]$. Note that axis-parallel lines is a special case of bi-directional strips and hence the RBSC problem on bi-directional strips is NP-Hard. We study the RBSC problem on skylines which are rectangles of the form $[a, b] \times (-\infty, c]$ or $(-\infty, c] \times [a, b]$ (referred to as AxRBSC-skylines) and show that this problem is W[1]-hard and therefore unlikely to admit FPT algorithms.

AxRBSC-quadrants: We further consider quadrants, a special case of skylines. Quadrants are axis-parallel rectangles of the form $[a, b] \times (+\infty, +\infty)$. We show that the RBSC on quadrants is polynomial time solvable.

*International Institute of Information Technology Bangalore, India. abidha.vp@iiitb.ac.in, pradeesha@iiitb.ac.in

[1]The proofs of the results marked with a star are in the full version of the paper.

1.2 Preliminaries

In this paper, we study the parameterized complexity of the problems using different parameters. Here we aim to solve the problems in $f(k) \cdot |n|^{O(1)}$ time, for some computable function f and $k << n$. For detailed reading of parameterized complexity, refer to [5].

Notations: Let $P = R \cup B$ be a point set, where R is a set of r red points and B is a set of b blue points and $|R| + |B| = n$. For any point p, let $x(p)$ and $y(p)$ denote the x and y coordinate values of p respectively. Let $[d]$ be the set, $\{1, 2, \cdots, d\}$.
We use the following reduction rules exhaustively on any instance of the RBSC problem in multiple sections throughout the paper.

Reduction rule 1 If a blue point b is contained in only one set $S \in \mathcal{F}$, then delete S from $\mathcal{F}$, $S \cap B$ from B and $S \cap R$ from R, set $k_r = k_r - |S \cap R|$.
Reduction rule 2 If a set S contains only red points, then delete S.
Reduction rule 3 If a set S contains only blue points, then delete S from $\mathcal{F}$ and $S \cap B$ from B.
Reduction rule 4 If a set S contains more than k_r red points, then delete S from $\mathcal{F}$.

2 RBSC on Axis-parallel hyperplanes

Parameterizing by k_r: In this section, we consider AxRBSC-HYPERPLANE parameterized by k_r. Let $(R \cup B, \mathcal{H}, k_r)$ be an instance of AxRBSC-HYPERPLANE parameterized by k_r.

After applying reduction rules 1, 2 and 3 on $(R \cup B, \mathcal{H}, k_r)$, every hyperplane contains at least one red point and at least one blue point. Every blue point is covered by at most d hyperplanes. There is a branching algorithm that can solve this problem in FPT time. For a blue point, we can branch on the hyperplane that covers it in the solution (if it exists). On every branch, the budget k_r drops by at least one, thus giving us an algorithm that runs in $O(d^{k_r})$ time. The next result improves this running time.

Theorem 2 *The problem* AxRBSC-HYPERPLANE *in* $\mathbb{R}^d$ *can be solved in time* $O({c_d}^{k_r}.n^{O(1)})$ *where* $c_d = \frac{d-1+\sqrt{(d-1)^2+4}}{2}$.

Proof. Here, we consider different cases. One case is when every hyperplane contains exactly one red point. We observe that this case can be reduced to the d-HITTING SET problem parameterized by the solution size. Let $(R \cup B, \mathcal{H}, k_r)$ be an instance of AxRBSC-HYPERPLANE such that every $H \in \mathcal{H}$ contains exactly one red point from R. We construct an instance $(U, \mathcal{F}, k)$ of d-HITTING SET with $U = R, k = k_r$ and $\mathcal{F}$ contains sets corresponding to every blue point, where a set corresponding to a blue point b contains all the red points contained in the at most d hyperplanes that contain b. It is easy to see that $(R \cup B, \mathcal{H}, k_r)$ is a YES instance if and only if $(U, \mathcal{F}, k)$ is a YES instance.

Our branching algorithm considers a blue point b such that at least one hyperplane that contains b has two red points and branches on the hyperplane that covers b in the solution. Thus, k_r drops by at least 2 in at least one branch and drops by at least one in all other branches. This gives us a branching algorithm that runs in $O({c_d}^{k_r}.n^{O(1)})$ time where $c_d = \frac{d-1+\sqrt{(d-1)^2+4}}{2}$. If the instance does not contain such a blue point, then the algorithm converts this to a d-HITTING SET instance. d-HITTING SET admits a branching algorithm that runs in $O({c_d}^{k}.n^{O(1)})$ time [7].

□

Corollary 3 *The RBSC problem on* AxRBSC-HYPERPLANE *in* $\mathbb{R}^d$ *is FPT parameterized by* $k_r + d$.

Polynomial Kernels: We apply the following set of reduction rules for all $j = 1$ to $d - 1$ exhaustively. For $2 \leq j \leq d-1$, we apply the reduction rule corresponding to j only when reduction rule corresponding to $(j-1)$ is not applicable. For a point $p \in \mathbb{R}^d$, $x_i(p)$ denotes the value of the ith coordinate of p, for $1 \leq i \leq d$.

Reduction rule 5 Repeat for all $d' \subset [d]$ such that $|d'| = d - j$.
Let $B' \subseteq B$ be such that for all $p, q \in B'$, $x_i(p) = x_i(q)$ if $i \in d'$. If $|B'| > j!(k_r)^j$ then delete all but $j! \cdot k_r^j + 1$ points of B'.

Lemma 4 *(*) Reduction rule 5 is safe.*

Theorem 5 AxRBSC-HYPERPLANE *admits a kernel* $(R \cup B, \mathcal{H})$ *with* $|B| \leq d!k_r^d$, $|R| \leq d \cdot d! \cdot k_r^{d+1}$, $|\mathcal{H}| \leq d \cdot d!k_r^d$.

Proof. Let $(R \cup B, \mathcal{H}, k_r)$ be an instance of AxRBSC-HYPERPLANE such that none of the reduction rules are applicable. Then, if it is a YES instance, all the points in B can be covered by at most $k_r.d$ hyperplanes. Any hyperplane can contain at most $(d-1)!k_r^{(d-1)}$ blue points, otherwise Reduction Rule 5.$(d-1)$ can be applied. Thus a YES instance can have at most $d!k_r^d$ blue points.

Every hyperplane in $\mathcal{H}$ contains at least one blue point and any blue point is contained in at most d hyperplanes. Therefore, $|\mathcal{H}| \leq d.d!k_r^d$. By Reduction Rule 4, a hyperplane contains at most k_r red points. Thus $|R| \leq d.d!.k_r^{d+1}$.

□

Parameterizing by b: Since every blue point is contained in at most d hyperplanes, the next result is easy to see.

Lemma 6 *The* AxRBSC-hyperplane *problem parameterized by b can be solved in time* $O(d^b.n^{O(1)})$.

Theorem 7 *The* AxRBSC-hyperplane *problem parameterized by b admits a polynomial kernel.*

Proof. Since, we cannot apply reduction rules anymore on $(R \cup B, \mathcal{H}, k_r)$, every hyperplane contains at least one blue point. One blue point is covered by at most d-hyperplanes. Then the number of hyperplanes $\mathcal{H}$ is at most db. To bound the number of red points we reduce our AxRBSC-hyperplane instance to a Weighted AxRBSC-hyperplane instance as follows:

The family of hyperplanes and the set of blue points remain the same as in the reduced instance. Let $R_d \subset R$ be the set of red points that lie in the intersection of d hyperplanes in $\mathcal{H}$. For all $r \in R_d$, assign $w(r) = 1$. Since any d hyperplanes in $\mathbb{R}^d$ intersect at a point, $|R_d| \leq h^d$. For $i = 1$ to $d-1$, we perform the reduction as follows. For all sets of i hyperplanes in $\mathcal{H}$, say, $H_i = \{h_1, h_2, \ldots, h_i\}$, consider the set of red points that is contained in all hyperplanes of H_i and no other hyperplane in $\mathcal{H}$. In the reduced instance, delete all but one of these red points and assign a weight equal to the number of deleted points +1.

It easy to see that AxRBSC-hyperplane has a solution of size k if and only if WEIGHTED AxRBSC-hyperplane has a solution of total weight k.

In the reduced instance, the number of red points that lie in the intersection of exactly i hyperplanes is $O(h^i)$. Therefore the total number of red points in the reduced instance is $O(h^d + h^{d-1} + h^{d-2} + \cdots + h) = O(h^d) = O((db)^d)$. Now we bound the weight of each red point in the reduced instance. By Reduction rule 4, all hyperplanes in $\mathcal{H}$ contain at most k_r red points. Therefore, the weight of any red point is at most k_r. We see that we need at most h bits to encode the weight of a red point. If not, $k_r > 2^h$. Note that there exists a brute force algorithm that solves the problem in $O(2^h)$ time. If $2^h < k_r$, this is a polynomial time algorithm. Hence it follows that the weight can be encoded using $h \leq db$ bits. Hence the size of the reduced instance is $O((db)^{d+1})$.

There exists a polynomial-time many-one reduction from WeightedAxRBSC-hyperplane to AxRBSC-hyperplane[1]. Thus, we obtain a polynomial-size kernel for AxRBSC-hyperplane parameterized by b.

□

Parameterizing by h: We design a parameterized algorithm and a kernel for AxRBSC-hyperplane when parameterized by the number of hyperplanes, h. We enumerate all possible subsets of $\mathcal{H}$ and for each subset, we check in polynomial time whether it covers all blue points and at most k_r red points. The algorithm runs in time $O(2^h(|U| + |F|)$.

To bound the number of blue points, we use the following reduction rule. We apply the following set of reduction rules for all $j = 1$ to $d-1$ exhaustively. For $2 \leq j \leq d-1$, we apply the reduction rule corresponding to δ only when reduction rule corresponding to $(j-1)$ is not applicable. Let $A_j = \sum_{i=0}^{j} h^i$.

Reduction rule 6 Repeat for all $d' \subset [d]$ such that $|d'| = d - j$.
Let $B' \subseteq B$ such that for all $p, q \in B'$, $x_i(p) = x_i(q)$ if $i \in d'$. If $|B'| > A_j$ then delete all but $A_j + 1$ points of B'.

Lemma 8 *(*) Reduction rule 6 is safe.*

Exhaustive application of the reduction rule 6 gives an equivalent instance of the problem. Now the next theorem follows.

Theorem 9 *(*) The* AxRBSC-hyperplane *parameterized by h admits a polynomial kernel.*

3 Red Blue Set Cover on Skylines

Given $P = R \cup B$ in the plane and $\mathcal{K}$, a set of m bidirectional skylines, We prove that, AxRBSC-skylines parameterized by k_r is W[1]-hard by giving a parameterized reduction from the Square Stabbing problem.
Rectangle Stabbing : Given a set $\mathcal{S}$ of n rectangles and a set of axis-parallel lines $\mathcal{L}$, the Rectangle Stabbing problem is to decide whether there exists a subset $\mathcal{L}' \subseteq \mathcal{L}$ with $|\mathcal{L}'| \leq k$ such that every rectangle from $\mathcal{S}$ is intersected (stabbed) by at least one line in $\mathcal{L}'$. If all the rectangles are unit squares it is the Square Stabbing problem. Square Stabbing problem parameterized by k is known to be W[1]-hard [6].

Let $(\mathcal{S}, \mathcal{L})$ be an instance of the Square Stabbing problem. Let $\{l_1, l_2, ..l_n\}$ be the set of vertical lines in $\mathcal{L}$ which are in non-decreasing order of their x-coordinates and let $\{h_1, h_2, ..h_n\}$ be the set of horizontal lines in $\mathcal{L}$ which are in non-increasing order of their y-coordinates. For every square $S \in \mathcal{S}$, add a blue point b_S to P at the top-left corner of S. For every vertical line l_i, we add a unit width vertical skyline S_{l_i} with its right edge coinciding with the given line l_i. The top edge of the leftmost vertical skyline S_{l_1} is unit distance above the topmost point in P. For $i > 1$, the top edge of the skyline S_{l_i} is unit vertical distance away from the top edge of $S_{l_{i-1}}$. Similarly for all horizontal lines, we add a unit width horizontal skyline S_{h_i} whose bottom edge coincides with the line h_i. The right edge of the topmost horizontal skyline S_{h_1} is a unit distance away from the rightmost point in P. The right edge of S_{h_i} is unit horizontal distance away from the $S_{h_{i-1}}$, for $i > 1$. For every horizontal skyline, add a red point to P in the top-right corner of the skyline and for every vertical skyline,

add a red point to P in the top-left corner of the skyline. Thus every skyline has a unique red point.

Lemma 10 *(*) $< \mathcal{S}, \mathcal{L} >$ can be stabbed using k lines if and only if $< P, \mathcal{K} >$ has a solution of size k.*

Theorem 11 *The* RBSC *problem on bidirectional skylines is W[1]-hard parameterized by the solution size.*

This implies that RBSC problem on axis parallel rectangles is W[1]-hard parameterized by the solution size k.

4 Quadrants

In this section, we give a polynomial time algorithm to optimally solve the AxRBSC-QUADRANTS problem. Given a universe $U = R \cup B$ and a family $\mathcal{Q}_1$ of quadrants, find the subfamily $\mathcal{Q}_1'$ that covers all points from B and minimum possible number of red points from R.

***Observation* 12** *Let q_i and q_j be two nested quadrants i.e., $(q_i \cap B) \subseteq (q_j \cap B)$ and $(q_i \cap R) \subseteq (q_j \cap R)$. Then both q_1 and q_2 together are not part of an optimal solution of* AxRBSC-QUADRANTS.

Theorem 13 *For a given bichromatic point set with n points, RBSC using quadrants can be computed in $O(mn^2)$ time.*

Proof. Let $\mathcal{O}$ be the orthogonal convexhull of B. Consider the left-bottom chain, P_{lb}, of $\mathcal{O}$ i.e., the chain of $\mathcal{O}$ that extends from the blue point with the minimum x-coordinate to the blue point with the minimum y-coordinate, in the anti-clockwise direction. Let $B' = \{b_1, b_2, \cdots b_l\}$ be the set of blue points on P_{lb} ordered by increasing x-coordinate. Note that this is also an order of decreasing y-coordinate. Let $R' \subseteq R$ and $B'' \subseteq B \backslash B'$ respectively be the set of red and blue points that lie on the right side of P_{lb}. Any set of quadrants that cover all points in B' will cover all points in B'' and R'. Therefore it is enough to consider the AxRBSC-QUADRANTS problem for $(R = (R \backslash R') \cup (B = B'), \mathcal{Q}_1)$.

For this reduced instance $(R \cup B, \mathcal{Q}_1)$, we introduce the following notations. Let $\pi : R \cup B \longmapsto [n]$ be the bijection corresponding to the ordering of the points in $R \cup B$ by increasing x coordinates and $\sigma : R \cup B \longmapsto [n]$ be the bijection corresponding to the ordering of the points in $R \cup B$ by decreasing y coordinates. For any $Q \in \mathcal{Q}_1$, let $p \in Q \cap (R \cup B)$ be the point with smallest x coordinate in Q and let $q \in Q \cap (R \cup B)$ be the point with the smallest y coordinate in Q. Define $left(Q) = \pi[p]$ and $bottom(Q) = \sigma(q)$. Let $B_i = \{b \in B | \sigma(b) > i\}$, $R_i = \{r \in R | \sigma(r) > i\}$ and $\mathcal{Q}_j = \{Q \in \mathcal{Q}_1 | left(Q) > j\}$.

Now we give a dynamic programming algorithm to solve the AxRBSC-QUADRANTSproblem for this instance.

Let $dp[i, j]$ returns the minimum number of red points from R_i, that is covered by a subset of $\mathcal{Q}_j$ that covers all points in B_i. Then $dp[\ ]$ can be given as follows. Here b' is the blue point with the smallest $\sigma(b')$ value in B_i.

$$dp[i, j] = \min_{Q \in \mathcal{Q}_j, b' \in Q} \left\{ dp[bottom(Q), left(Q)] + |Q \cap R_i| \right\} \tag{1}$$

To see the correctness of the above, observe that the recurrence considers all $Q \in \mathcal{Q}_j$ to cover b'. For the correct guess, the rest of the solution only covers uncovered red points from $R_{bottom(Q)}$ using $Q \in \mathcal{Q}_{left(Q)}$. Otherwise, if a $Q \notin \mathcal{Q}_{left(Q)}$ is part of the solution then it contradicts Observation 12. Similarly an uncovered red point outside $R_{bottom(Q)}$ is not part of an optimal solution.

It is clear that computing the value of one DP table entry takes $O(m)$ time. The number of $dp[*, *]$ values to be computed is clearly $O(n^2)$ time. We can construct and verify the orthogonal path in $O(n \log n)$ time. Therefore, the algorithm runs in $O(mn^2)$ time.

□

References

[1] Pradeesha Ashok, Sudeshna Kolay, and Saket Saurabh. Multivariate complexity analysis of geometric red blue set cover. *Algorithmica*, 79(3):667–697, 2017.

[2] Shuai, Tian-Ping and Hu, Xiao-Dong. Connected set cover problem and its applications. *International Conference on Algorithmic Applications in Management*, 243–254, 2006.

[3] Robert D Carr, Srinivas Doddi, Goran Konjevod, and Madhav Marathe. On the red-blue set cover problem. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 345–353, 2000.

[4] Timothy M. Chan and Nan Hu. Geometric red-blue set cover for unit squares and related problems. *Computational Geometry*, 48:380–385, 2015.

[5] Fomin F.V. Kowalik Ł. Lokshtanov D. Marx D. Pilipczuk M. Pilipczuk M. Saurabh S Cygan, M. Parameterized algorithms. In *Journal of Algorithms*, pages 99–127, 2005.

[6] Michael Dom, Michael R. Fellows, and Frances A. Rosamond. Parameterized complexity of stabbing rectangles and squares in the plane. WALCOM '09, pages 298–309, Berlin, Heidelberg, 2009. Springer-Verlag.

[7] Rolf Niedermeier and Peter Rossmanith. An efficient fixed-parameter algorithm for 3-hitting set. *Journal of Discrete Algorithms*, 1(1):89–102, 2003. Combinatorial Algorithms.

[8] S Pandit RR Madireddy, SC Nandy. On the geometric red-blue set cover problem. *WALCOM*, pages 129–141, 2021.

Angle-Monotonicity of theta-graphs for points in convex position

Davood Bakhshesh* Mohammad Farshi†

Abstract

For a real number $0 < \gamma < 180°$, a geometric path $P = (p_1, \ldots, p_n)$ is called angle-monotone with width γ from p_1 to p_n if there exists a closed wedge of angle γ such that every directed edge $\overrightarrow{p_ip_{i+1}}$ of P lies inside the wedge whose apex is p_i. A geometric graph G is called angle-monotone with width γ if for any two vertices p and q in G, there exists an angle-monotone path with width γ from p to q. In this paper, we show that for any integer $k \geq 1$ and any $i \in \{2, 3, 4, 5\}$, the theta-graph Θ_{4k+i} on a set of points in convex position is angle-monotone with width $90° + \frac{i\theta}{4}$, where $\theta = \frac{360°}{4k+i}$. Moreover, we present two sets of points in the plane, one in convex position and the other in non-convex position, to show that for every $0 < \gamma < 180°$, the graph Θ_4 is not angle-monotone with width γ.

1 Introduction

Let S be a set of points in the plane. For two points $p, q \in S$, the Euclidean distance between p and q is denoted by $|pq|$. A *geometric* graph $G = (S, E)$ is a weighted graph such that any edge (x, y) of G is a straight-line segment between x and y and the weight of (x, y) is $|xy|$. The length of a path $P = (p_1, p_2, \ldots, p_r)$ between p_1 and p_r in G is denoted by $|P|$, and it is defined as $|P| = \sum_{i=1}^{r-1} |p_ip_{i+1}|$. For any two points $p, q \in S$, the *stretch factor* (or *dilation*) between p and q in a geometric graph G is the ratio of the length of a shortest path between p and q in G over $|pq|$. The stretch factor of a geometric graph G is the maximum stretch factor between all pairs of vertices of G.

Let $t > 1$ be a real number. A geometric graph G is called a *t-spanner* if the stretch factor of G is at most t. In Computational Geometry, constructing the geometric graphs with low stretch factor, small number of edges (small size) and low weight is an important problem. We refer the reader to the book [9] to study t-spanners and their algorithms.

Let $\theta > 0$ be a real number. In [6], Dehkordi et al., introduced *θ-paths*. Let W_p^θ be a $90°$ closed wedge delimited by the rays starting at p with the slopes $\theta - 45°$ and $\theta + 45°$. A path $(p_1, p_2, \ldots, p_n)$ is called a *θ-path* if for every integer i with $1 \leq i \leq n-1$, the vector $\overrightarrow{p_ip_{i+1}}$ lies in the wedge $W_{p_i}^\theta$. Using the concept of θ-paths, Bonichon et al. [3] introduced *angle-monotone graphs*. A geometric graph $G = (S, E)$ is called *angle-monotone* if for any two points $u, v \in S$, there is a real number $\theta > 0$ such that G contains a θ-path between u and v. Bonichon et al. [3] generalized the concept of angle-monotone graphs to angle-monotone graphs with width γ. Let γ be a real number with $0 < \gamma < 180°$. A geometric path $P = (p_1, \ldots, p_n)$ is called *angle-monotone with width γ* from p_1 to p_n if for some closed wedge of angle γ, every vector $\overrightarrow{p_ip_{i+1}}$ lies in the wedge whose apex is p_i (see Figure 1).

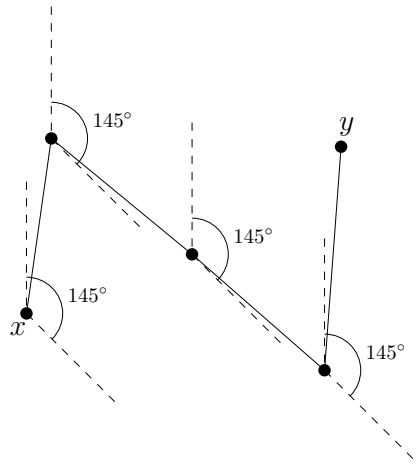


Figure 1: An angle-monotone path between x and y with width $\gamma = 145°$.

A geometric graph G is called *angle-monotone with width γ* if for any vertex p of G, there is an angle-monotone path with width γ from p to all other vertices of G. It is remarkable that if a path is angle-monotone with width γ from x to y, then the path is also angle-monotone with width γ from y to x.

In [6], Dehkordi et al. show that any Gabriel triangulation is an angle-monotone graph with width $90°$. In [8], Lubiw and Mondal show that for any set of points in the plane, there is an angle-monotone graph with width $90°$ with a subquadratic size. Furthermore, they showed that for any angle β with $0 < \beta < 45°$, and for any set of points in the plane, there is an angle-monotone graph with width $(90° + \beta)$ of size $O(\frac{n}{\beta})$. Bakhshesh and Farshi [1] present a point set in the plane such that

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

†Department of Mathematical Sciences, Yazd University, Yazd, Iran. mfarshi@yazd.ac.ir

its Delaunay triangulation is not angle-monotone with width less than 140°. Bakhshesh and Farshi [2] prove that the minimum value of an angle γ such that for any set of points in the plane there is a plane angle-monotone graph with width γ is equal to 120°.

One of the most popular graphs in computational geometry are *theta-graphs* which were introduced by Clarkson [5] and independently by Keil [7]. Informally, for every point set S in the plane and an integer $m \geq 2$, the theta-graph Θ_m is constructed by partitioning the plane into m cones at each point $p \in S$, and joining the *closest* point to p at each cone (in the next section, closest will be defined). Bonichon et al. [3] proved that for any set of points in the plane, *half-Θ_6-graph*, a plane subgraph of Θ_6, whose edges are obtained by selecting every other cone, i.e., alternate cones, is angle-monotone with width 120°. In [6], Dehkordi et al. prove that for every set of n points in the plane that are in convex position, there exists an angle-monotone graph (angle-monotone graph with width 90°) with $O(n \log n)$ edges. To the best of our knowledge, it is unknown if the theta-graphs except Θ_6 are angle-monotone with a constant width.

In this paper, we show that for any set of points in convex position, and any integer $k \geq 1$ and any $i \in \{2, 3, 4, 5\}$, the theta-graph Θ_{4k+i} is angle-monotone with width $90° + \frac{i\theta}{4}$, where $\theta = \frac{360°}{4k+i}$. Moreover, we present two sets of points in the plane, one in convex position and the other in non-convex position, to show that for every $0 < \gamma < 180°$, the graph Θ_4 is not angle-monotone with width γ.

2 Preliminaries

Let $m \geq 3$ be an integer, and let $\theta = \frac{2\pi}{m}$ be a real number. For any integer i with $0 \leq i < m$ and a point p in the plane, let $\mathcal{R}_i^p$ be the ray emanating from p making the angle $\theta \times i = 2\pi i/m$ with the positive x-axis (the angles are considered in counter-clockwise). Let C_i^p be the cone which is constructed by the rays $\mathcal{R}_i^p$ and $\mathcal{R}_{i+1}^p$. Note that we assume that $\mathcal{R}_m^p = \mathcal{R}_0^p$. For a point r and a cone C_i^p, we say C_i^p contains r (or, $r \in C_i^p$) if r lies strictly between $\mathcal{R}_i^p$ and $\mathcal{R}_{i+1}^p$, or lies on $\mathcal{R}_{i+1}^p$. If r lies on $\mathcal{R}_i^p$, then $r \notin C_i^p$. For a point set S, the theta-graph Θ_m is constructed as follows. For each point $p \in S$, we partition the plane into m cones $C_0^p, C_1^p, \ldots, C_{m-1}^p$ (see Figure 2). Then, for each cone C_i^p containing at least one point of S other than p, let $r_i \in C_i^p$ be a point such that $|pr_i'|$ is minimum where r_i' is the perpendicular projection of r_i onto the bisector of C_i^p. Then, we add the edge (p, r_i) to the graph. We assume that a pair (a, b) is a directed edge. We call the point r the *closest* point to p in C_i^p. For a point $q \in C_i^p$, the *canonical triangle* T_{pq} is the isosceles triangle which is constructed by the rays of C_i^p and the line through q perpendicular to the bisector of C_i^p. For more details on theta-graphs, see [9].

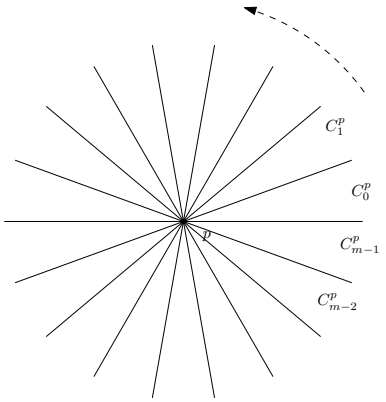


Figure 2: Partition the plane into $m = 18$ cones with apex at p.

Let S be a set of $n \geq 3$ points in the plane that are in convex position. In the following, when we use the notation G, we mean one of the graphs Θ_{4k+2}, Θ_{4k+3}, Θ_{4k+4} and Θ_{4k+5}. Throughout the paper, we assume that p and q are two distinct points in S and suppose, without loss of generality, that $q \in C_0^p$. Let $\mathcal{W}_O$ be the wedge with apex at the origin O that is the union of all cones C_t^O with $\lceil \frac{m-1}{4} \rceil \leq t \leq \lceil \frac{m-2}{2} \rceil$. Let $\mathcal{W}_O'$ be the reflection of $\mathcal{W}_O$ with respect to the point O. Now, let $\mathcal{U}_O$ be a wedge with apex at the origin O such that $\mathcal{U}_O = \mathcal{W}_O' \cup C_0^O$ (see Figure 3).

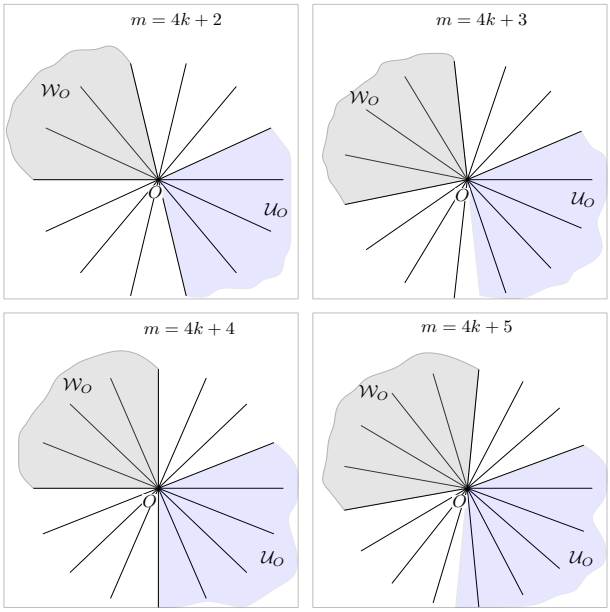


Figure 3: The wedges $\mathcal{W}_O$ and $\mathcal{U}_O$ for the different values of m.

3 Angle-monotonicity of theta-graphs

In this section, we show that for any integer $k \geq 1$ and any $i \in \{2, 3, 4, 5\}$, the theta-graph Θ_{4k+i} is angle-monotone with width $90^\circ + \frac{i\theta}{4}$. To this end, we show that there is an angle-monotone path between p and q in G with width $90^\circ + \frac{i\theta}{4}$. Let $P = (p = v_0, v_1, \ldots, v_l)$ be the directed path in G such that $v_{i+1} \in C_0^{v_i}$ is the closest point to v_i, and v_l is the last vertex of the path P that lies in T_{pq}. Let $\ddot{P}$ be the directed path which is obtained by reversing the direction of all edges of P. If $v_l = q$, then obviously P is an angle-monotone path from p to q with width θ. Then, we are done. Now, in what follows, we assume that $v_l \neq q$. Suppose, without loss of generality, that q is below $P \cup C_0^{v_l}$ (see Figure 4). Let $Q = (q = a_0, a_1 \ldots, a_g = v_l)$ be the

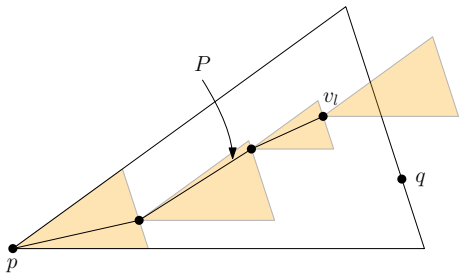


Figure 4: The path P.

path constructed by the algorithm Θ-WALK(q, v_l) (see Algorithm 1). The path Q is a path between q and v_l in G such that for any a_i there exists a cone $C_j^{a_i}$ such that $v_l \in C_j^{a_i}$ and (a_i, a_{i+1}) is an edge of G.

Algorithm 1: Θ-WALK(a, b) (see [9])

output: A path between a and b in theta-graphs

```
a_0 = a;
i := 0;
while a_i ≠ b do
    s := an integer such that b ∈ C_s^{a_i};
    a_{i+1} := a point of C_s^{a_i} ∩ S\{a_i} such that
    (a_i.a_{i+1}) is an edge of Θ_k;
    i := i + 1;
end
return the path (a_0, a_1, ..., a_i);
```

3.1 The graphs Θ_{4k+2} and Θ_{4k+4}

We first prove the following lemma.

Lemma 1 *If $G = \Theta_{4k+2}$, then every edge (a_i, a_{i+1}) of the path Q lies in the wedge $\mathcal{W}_{a_i}$.*

Proof. Let ℓ_1 be the horizontal line passing through v_l, and ℓ_2 be the line passing through v_l that forms an angle θ with the positive x-axis. Let c_1 and c_2 be the intersection of ℓ_1 and ℓ_2 with the sides of the triangle T_{pq} which are incident to p (see Figure 5). Based on

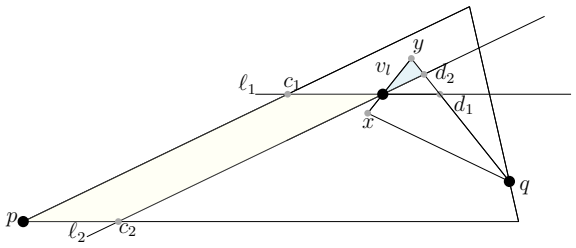


Figure 5: Illustrating the proof of Lemma 1.

the construction of the path P, the vertex v_{l-1} lies in the quadrilateral $pc_1v_lc_2$. Let j be an integer such that $q \in C_j^{v_l}$. Since we assume that q is below $P \cup C_0^{v_l}$, we have $3k + 2 \leq j \leq 4k + 1$. Since $q \in C_j^{v_l}$, we have $v_l \in C_{j-(2k+1)}^{q}$. Consider the triangle T_{qv_l}. Let x and y be the two other vertices of T_{qv_l} as depicted in Figure 5. Let $d_1 \neq v_l$ be the intersection of ℓ_1 and T_{qv_l}, and let $d_2 \neq v_l$ be the intersection of ℓ_2 and T_{qv_l}. It is notable that it is possible that the segment xy completely lies on the line ℓ_2. In this case, we assume that $d_2 = y$. Now, if any vertex u of the path Q lies in the triangle $\triangle v_l y d_2$, since v_{l-1} lies in the quadrilateral $pc_1v_lc_2$, the triangle quv_{l-1} contains the vertex v_l that contradicts the convexity of the points. Hence, no vertices of Q lie in the triangle $\triangle v_l y d_2$. By similar reasons, no vertices of Q lie in the triangle $\triangle q v_l p$. Since $C_0^{v_l} \cap T_{pq}$ does not contain any point of S, the path Q completely lies in the triangle $\triangle q d_1 v_l$. Then, for any edge (a_i, a_{i+1}) of Q, there is an integer t with $j-(2k+1) \leq t \leq 2k$ such that $a_{i+1} \in C_t^{a_i}$. Since $3k+2 \leq j \leq 4k+1$, clearly (a_i, a_{i+1}) lies in the wedge $\mathcal{W}_{a_i}$. □

Now, we have the following lemma.

Lemma 2 *If $G = \Theta_{4k+2}$, then every edge (x, y) of the path $P \cup \ddot{Q}$ lies in the wedge $\mathcal{U}_x$.*

Proof. By Lemma 1, every edge (a, b) of Q lies in the wedge $\mathcal{W}_a$. Therefore, every edge (b, a) of $\ddot{Q}$ lies in the wedge $\mathcal{W}'_b$. On the other hand, every edge (v_i, v_{i+1}) of P lies in the cone $C_0^{v_i}$. Since $\mathcal{U}_O = \mathcal{W}'_O \cup C_0^O$, every edge (x, y) of the path $P \cup \ddot{Q}$ lies in the wedge $\mathcal{U}_x$. □

Theorem 3 *For any set S of points in the plane that are in convex position and for any integer $k \geq 1$, the graph $G = \Theta_{4k+2}$ is angle-monotone with width $90^\circ + \frac{\theta}{2}$.*

Proof. Consider the points p and q. By Lemma 2, every edge (x, y) of the path $P \cup \ddot{Q}$ lies in the wedge $\mathcal{U}_x$. Therefore, the path $P \cup \ddot{Q}$ is an angle-monotone path from p to q in G with width $k\theta + \theta$. Note that for $G = \Theta_{4k+2}$, the angle of the wedge $\mathcal{U}_x$ is $k\theta + \theta$. Since

$\theta = \frac{360^\circ}{4k+2}$, we have $k\theta+\theta = 90^\circ - \frac{\theta}{2} + \theta = 90^\circ + \frac{\theta}{2}$. Hence, $P \cup \ddot{Q}$ is an angle-monotone path with width $90^\circ + \frac{\theta}{2}$. This completes the proof. □

Similar to the proof of Theorem 3, for $G = \Theta_{4k+4}$ with $k \geq 1$, we can prove that the path $P \cup \ddot{Q}$ is an angle-monotone path from p to q with width $(k+1)\theta + \theta = 90^\circ + \theta$. Note that for $G = \Theta_{4k+4}$, the angle of the wedge $\mathcal{U}_x$ is $(k+1)\theta + \theta$. Hence, we have the following theorem.

Theorem 4 *For any set S of points in the plane that are in convex position and for any integer $k \geq 1$, the graph $G = \Theta_{4k+4}$ is angle-monotone with width $90^\circ + \theta$.*

In [3], Bonichon et al., show that any angle-monotone graph with width $\gamma < 180^\circ$ is a t-spanner with $t = 1/\cos\frac{\gamma}{2}$. Hence, we have the following result.

Corollary 1 *For any set of points in the plane that are in convex position and for any integer $k \geq 1$, the graphs Θ_{4k+2} and Θ_{4k+4} have the stretch factor at most $1/\cos\left(\frac{\pi}{4} + \frac{\theta}{4}\right)$ and $1/\cos\left(\frac{\pi}{4} + \frac{\theta}{2}\right)$, respectively.*

3.2 The graphs Θ_{4k+3} and Θ_{4k+5}

We first assume that $G = \Theta_{4k+3}$. Here, we present an algorithm that finds an angle-monotone path $\mathcal{P}$ between p and q in G with a constant width. The algorithm is as follows. It first finds the path $P = (p = v_0, \ldots, v_l)$ which was introduced earlier. If $v_l = q$, then clearly $\mathcal{P} = P$ is an angle-monotone path with width θ, and we are done. Now, in the following we assume that $v_l \neq q$. Let a be the topmost vertex of the triangle T_{pq} and let $b \neq p$ be the other vertex of T_{pq}. Let m be the midpoint of ab. The algorithm considers the following cases.

- **Case 1:** q lies on the segment am. Now, let $Q = (q = a_0, \ldots, v_l)$ be the path constructed by the algorithm Θ-Walk(q, v_l). Then, the algorithm outputs the path $\mathcal{P} = P \cup \ddot{Q}$.

- **Case 2:** q lies on the segment bm. Let $P' = (q = u_0, \ldots, u_s)$ be the path in G such that $u_{i+1} \in C^{u_i}_{2k+1}$ and u_{i+1} is the closest point to u_i, and u_s is the last vertex of the path P' that lies in T_{qp}. Let b' be the topmost vertex of the triangle T_{qp} and let a' be the bottommost vertex of T_{qp}. Let m' be the midpoint of $a'b'$. Since $q \in C^p_0$, it is easy to see that p lies on the segment $a'm'$. Now, there are two cases:

  - **(I):** P and P' have a common vertex w. The algorithm outputs the path R which is formed by the portion of P from v_0 to w followed by the portion of P' from w to q.
  - **(II):** P and P' do not have any common vertex. Now, consider two following cases: **(a):** there is a vertex $g \neq q$ of the path P' below the path P. **(b):** all vertices of P' are above the path P. For the case **(a)**, let u_h be the last vertex of P' below the path P and let Q' be the constructed path by the algorithm Θ-Walk(p, u_h). Then, the algorithm outputs path $\mathcal{P} = P' \cup \ddot{Q}'$. For the case **(b)**, first the path $Q = \Theta$-Walk(q, v_l) is constructed. Then, the algorithm outputs the path $\mathcal{P} = P \cup \ddot{Q}$.

For more details, see Algorithm 2.

Algorithm 2: Angle-Monotone-Path-$\Theta_{4k+3}(p, q)$

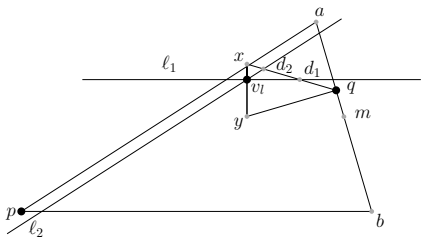
```
output: An angle-monotone path between p and q in Θ_{4k+3}
P := ∅;
Compute the path P = (p = v_0, ..., v_l);
if v_l ≠ q then
    if q lies on the segment "am" then
        Q := Θ-Walk(q, v_l);
        P := P ∪ Q̈;
    end
    else
        Compute the path P' = (q = u_0, ..., u_s);
        if P and P' have a common vertex w then
            R := the path which is formed by the portion of P
                 from v_0 to w followed by the portion of P' from w
                 to q;
            P := R;
        end
        else
            if there is a vertex g ≠ q of the path P' below
               the path P then
                u_h := the last vertex of P' below P;
                Q' := Θ-Walk(p, u_h);
                P := P' ∪ Q̈';
            end
            else
                Q := Θ-Walk(q, v_l);
                P = P ∪ Q̈;
            end
        end
    end
end
else
    P := P;
end
return P;
```

In the following, we show that the path $\mathcal{P}$ returned by Algorithm 2 is an angle-monotone path between p and q with width $90^\circ + \frac{3\theta}{4}$. We first prove the following lemma.

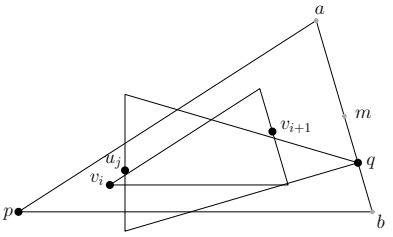
Lemma 5 *If q lies on the segment am, then every edge (a_i, a_{i+1}) of the path $Q = (q = a_0, \ldots, v_l)$ lies in the wedge $\mathcal{W}_{a_i}$.*

Proof. Let j be an integer such that $v_l \in C^q_j$. Since we assumed that q is below $P \cup C^{v_l}_0$, we have $k+1 \leq j \leq 2k+1$. Consider the triangle T_{qv_l}. Let x and y be the two other vertices of T_{qv_l} as depicted in Figure 6(a). It is notable that the line passing through p and m is parallel to the line passing through q and y. Then, since q lies on the segment am, the point p is below the line passing

through q and y. Hence, because of the convexity of the points, no points of Q lie in the triangle $\triangle qv_l y$. Consider the lines ℓ_1 and ℓ_2, and the points d_1 and d_2



(a) Illustrating the proof of Lemma 5.



(b) Illustrating the proof of Lemma 8.

Figure 6: Illustrating the proofs of Lemma 5 and Lemma 8.

as defined in the proof of Lemma 1. By the reasons similar to the proof of Lemma 1, we can prove that the path Q completely lies in the triangle $\triangle qd_1v_l$. Then, for any edge (a_i, a_{i+1}) of Q, there is an integer t with $j \leq t \leq 2k+1$ such that $a_{i+1} \in C_t^{a_i}$. Clearly, this shows that (a_i, a_{i+1}) lies in the wedge $\mathcal{W}_{a_i}$. □

Now, we prove the following lemma.

Lemma 6 *If q lies on the segment bm, then every edge (r_i, r_{i+1}) of the path R lies in the wedge $\mathcal{U}_{r_i}$.*

Proof. According to Algorithm 2, the path R is constructed when the paths $P = (v_1, \ldots, v_l)$ and $P' = (u_1, \ldots, u_s)$ have a common vertex. It is clear that for every edge (v_i, v_{i+1}) of the path P, we have $v_{i+1} \in C_0^{v_i}$, therefore (v_i, v_{i+1}) lies in the wedge $\mathcal{U}_{v_i}$. On the other hand, for every edge (u_i, u_{i+1}) of P', we have $u_{i+1} \in C_{2k+1}^{u_i}$. Therefore, $u_i \in C_{4k+2}^{u_{i+1}}$ or $u_i \in C_0^{u_{i+1}}$. Hence, the edge (u_{i+1}, u_i) lies in the wedge $\mathcal{U}_{u_{i+1}}$. This completes the proof. □

Let $\mathcal{Y}_O$ be a wedge with $\mathcal{Y}_O = \left(\bigcup_{i=3k+2}^{4k+2} C_i^O\right) \cup \left(C_{2k+1}^O\right)'$ ($\left(C_{2k+1}^O\right)'$ is the reflection of C_{2k+1}^O with respect to the origin O). It is clear that the angle of $\mathcal{Y}_O$ is equal to $(k+1)\theta + \theta/2$. Now, we prove the following lemma.

Lemma 7 *If q lies on the segment bm and the paths P and P' do not have any common vertex, and there is a vertex $g \neq q$ of the path P' below the path P, then every edge (c_i, c_{i+1}) of the constructed path $\mathcal{P}$ by Algorithm 2 lies in the wedge $\mathcal{Y}_{c_i}$.*

Proof. Le u_h be the last vertex of P' below P. According Algorithm 2, $\mathcal{P} = P' \cup \ddot{Q}'$ that Q' is the constructed path by Θ-Walk(p, u_h). It is clear that for every edge (u_i, u_{i+1}) of P', we have $u_{i+1} \in C_{2k+1}^{u_i}$, and therefore $u_i \in \left(C_{2k+1}^{u_{i+1}}\right)'$. Hence, (u_{i+1}, u_i) lies in the wedge $\mathcal{Y}_{u_{i+1}}$. Let $Q' = (p = a'_1, a'_2, \ldots, a'_z = u_h)$. We claim that every edge (a'_i, a'_{i+1}) lies in the wedge $\mathcal{Y}_{a'_i}$. Since p lies on the segment $a'm'$, by the arguments similar to the proof of Lemma 5, the claim is proved. These show that if (c_i, c_{i+1}) be an edge of the path $\mathcal{P}$, it lies in the wedge $\mathcal{Y}_{c_i}$. □

Now, we have the following lemma.

Lemma 8 *If q lies on the segment bm and the paths P and P' do not have any common vertex, and there is no vertex $g \neq q$ of the path P' below the path P, then every edge (r_i, r_{i+1}) of the constructed path $\mathcal{P}$ by Algorithm 2 lies in the wedge $\mathcal{U}_{r_i}$.*

Proof. Let u_j be a vertex of P' above the path P. Let v_i be the last vertex of P to the left of u_j (see Figure 6(b)). Since p is to the left of u_j, the vertex v_i always exist. Since there is no vertex $g \neq q$ of the path P' below the path P, we have $u_{j-1} = q$. Now, consider the triangle $T_{v_i v_{i+1}}$. Since P and P' have no common vertex, clearly $u_j \notin T_{v_i v_{i+1}}$. Hence, if $v_i \neq p$, then the triangle $\triangle p u_j v_{i+1}$ contains the vertex v_i which contradicts the convexity of the points. Then, $v_i = p$. On the other hand, since $v_l \neq q$, we must have $v_{i+1} \notin T_{q u_j}$, and therefore $v_l \in C_t^q$ with $k+1 \leq t < 2k+1$. Now, by the arguments similar to the proof of Lemma 5, we can prove that every edge (a_i, a_{i+1}) of the path Q lies in the wedge $\mathcal{W}_{a_i}$. Hence, it is clear that every edge (r_i, r_{i+1}) of the path $\mathcal{P} = P \cup \ddot{Q}$ lies in the wedge $\mathcal{U}_{r_i}$. □

Based on Lemmas 5, 6, 7 and 8, any path constructed by Algorithm 2 is angle-monotone with width $(k+1)\theta + \frac{\theta}{2}$. Since $\theta = \frac{360^\circ}{4k+3}$, we have $(k+1)\theta + \frac{\theta}{2} = 90^\circ + \frac{3\theta}{4}$. Then, the following theorem holds.

Theorem 9 *For any set S of points in the plane that are in convex position and for any integer $k \geq 1$, Θ_{4k+3} is angle-monotone with width $90^\circ + \frac{3\theta}{4}$.*

By the arguments similar to the proof of Theorem 9, for $G = \Theta_{4k+5}$ with $k \geq 1$, we can prove that the path $\mathcal{P}$ is an angle-monotone path from p to q with width $(k+1)\theta + \frac{\theta}{2}$. Since $\theta = \frac{360^\circ}{4k+1}$, we have $(k+1)\theta + \frac{\theta}{2} = 90^\circ + \frac{5\theta}{4}$. Then, the following theorem holds.

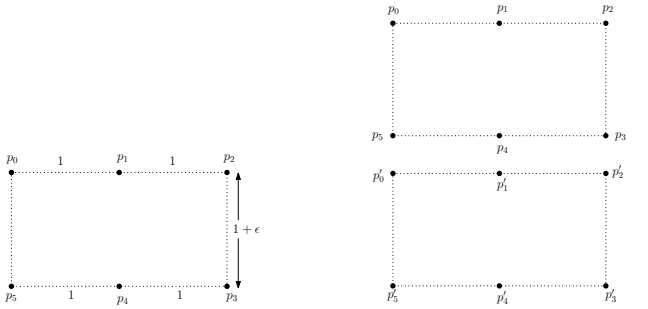
Theorem 10 *For any set S of points in the plane that are in convex position and for any integer $k \geq 1$, Θ_{4k+5} is angle-monotone with width $90^\circ + \frac{5\theta}{4}$.*

We close this section with the following result.

Corollary 2 *For any set of points in the plane that are in convex position, the graphs Θ_{4k+3} and Θ_{4k+5} with $k \geq 1$ have the stretch factor at most $1/\cos\left(\frac{\pi}{4}+\frac{3\theta}{8}\right)$ and $1/\cos\left(\frac{\pi}{4}+\frac{5\theta}{8}\right)$, respectively.*

4 Theta-graph Θ_4

In the following, we present two point sets, one in convex position and the other in non-convex position, to show that the graph Θ_4 of the point set is not angle-monotone for any width $\gamma > 0$. Let p_0, p_2, p_3 and p_5 be the vertices of a rectangle with length 2 and width $1+\epsilon$, where $\epsilon > 0$ is a small real number (see Figure 7(a)). Let p_1 and p_4 be the midpoints of the segments p_0p_2 and p_3p_5, respectively. Now, let $P = \{p_0, p_1, \ldots, p_5\}$.



(a) The point set P.

(b) The point set $V = P \cup P'$.

Figure 7: The point sets P and V.

Consider the theta-graph Θ_4 on P. It is not hard to see that the edge set E of Θ_4 is

$$E = \{(p_0,p_1),(p_1,p_2),(p_2,p_3),(p_3,p_4),(p_4,p_5),(p_5,p_0)\}.$$

Now, since p_0p_2 and p_3p_5 are parallel, it is obvious that for any $0 < \gamma < 180°$, any path between p_1 and p_4 is not angle-monotone with width γ.

Let $P' = \{p'_0, p'_1, \ldots, p'_5\}$ be a copy of point set P such that the points of P' placed below the points of P as depicted in Figure 7(b). Let $V = P \cup P'$. It is easy to see that the edge set F of the theta-graph Θ_4 on the point set V is

$$F = E \cup \{(p'_0,p'_1),(p'_1,p'_2),(p'_2,p'_3),(p'_3,p'_4),(p'_4,p'_5),(p'_5,p'_0)\} \cup \{(p'_0,p_5),(p'_1,p_4),(p'_2,p_3)\}.$$

It is obvious that for any $0 < \gamma < 180°$, any path between p_1 and p_4 is not angle-monotone with width γ. Now, we have the following theorem.

Theorem 11 *For any angle $0 < \gamma < 180°$, the graph Θ_4 is not necessarily angle-monotone with width γ.*

5 Remarks

In Corollaries 1 and 2, we examined the stretch factor of the graphs Θ_{4k+2}, Θ_{4k+3}, Θ_{4k+4} and Θ_{4k+5} when the points placed in convex position. In [4], Bose et al., show that the stretch factor of the graphs Θ_{4k+2}, Θ_{4k+3}, Θ_{4k+4} and Θ_{4k+5} are at most $1 + 2\sin(\theta/2)$, $\cos(\theta/4)/(\cos(\theta/2)-\sin(3\theta/4))$, $1 + 2\sin(\theta/2)/(\cos(\theta/2)-\sin(\theta/2))$ and $\cos(\theta/4)/(\cos(\theta/2)-\sin(3\theta/4))$, respectively.

By comparing the results of Corollaries 1 and 2 with the results in [4], we find that the results of the corollaries do not improve the stretch factors known in [4].

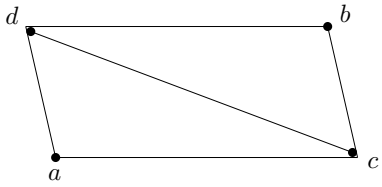


Figure 8: The lower bound for the width of Θ_{4k+2}.

In the following, we indicate whether the bounds on the width presented in Theorems 3, 4, 9 and 10 are tight or not. Consider the graph Θ_{4k+2}. Figure 8 shows that the upper bound on the width presented in Theorems 3 is tight. We place a vertex c close to the lower corner of T_{ab} that is sufficiently far from the vertex b. We also place a vertex d close to the upper corner of T_{ba} that is sufficiently far from the vertex a. Now, the graph Θ_{4k+2} of four points a, b, c and d is as shown in Figure 8. We can easily see that each of the paths acb and adb are angle-monotone with width $90° + \frac{\theta}{2} - \epsilon$, for some real number $\epsilon > 0$ that only depends on the distance between c (d) and the lower corner (upper corner) of T_{ab} (T_{ba}). If ϵ approaches zero, then the width approaches $90° + \frac{\theta}{2}$.

For Theorems 4, 9 and 10, we do not know whether the bounds for the width is tight or not.

6 Conclusion

In this paper, we showed that for any set of points in the plane that are in convex position and for any integer $k \geq 1$ and any $i \in \{2,3,4,5\}$, the theta-graph Θ_{4k+i} is angle-monotone with width $90° + \frac{i\theta}{4}$, where $\theta = \frac{360°}{4k+i}$. Moreover, we presented two sets of points in the plane, one in convex position and the other in non-convex position, to show that for every $0 < \gamma < 180°$, the graph Θ_4 is not angle-monotone with width γ. It is notable that our technique in Section 3.2, does not work for Θ_5 because by the proposed technique, the resulting path $\mathcal{P}$ is angle-monotone with width $90° + \frac{5\theta}{4}$. Since for Θ_5, we have $\theta = \frac{2\pi}{5} \equiv 72°$. Then, $90° + \frac{5\theta}{4} = 180°$. We conjecture for any set of points in convex position, Θ_5 is angle-monotone with a constant width. We tried to

prove our conjecture but we did not succeed. Finally, we present the following conjecture.

Conjecture 1 *For any set of points in the plane that are not convex position, for any integer $k \geq 1$ and any $i \in \{2, 3, 4, 5\}$, the theta-graph Θ_{4k+i} is angle-monotone with width $90^\circ + \frac{i\theta}{4}$, where $\theta = \frac{360^\circ}{4k+i}$.*

References

[1] D. Bakhshesh and M. Farshi. Angle-monotonicity of Delaunay triangulation. *Computational Geometry*, 94:101711, 2021.

[2] D. Bakhshesh and M. Farshi. On the plane angle-monotone graphs. *Computational Geometry*, 100:101818, 2022.

[3] N. Bonichon, P. Bose, P. Carmi, I. Kostitsyna, A. Lubiw, and S. Verdonschot. Gabriel triangulations and angle-monotone graphs: Local routing and recognition. In *Proceedings of the 24th International Symposium on Graph drawing (GD 2016)*, pages 519–531, 2016.

[4] P. Bose, J.-L. D. Carufel, P. Morin, A. van Renssen, and S. Verdonschot. Towards tight bounds on theta-graphs: More is not always better. *Theoretical Computer Science*, 616:70 – 93, 2016.

[5] K. Clarkson. Approximation algorithms for shortest path motion planning. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 56–65, New York, NY, USA, 1987. ACM.

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

[7] J. M. Keil. Approximating the complete euclidean graph. In R. Karlsson and A. Lingas, editors, *1st Scandinavian Workshop on Algorithm Theory*, pages 208–213, Berlin, Heidelberg, 1988.

[8] A. Lubiw and D. Mondal. Construction and local routing for angle-monotone graphs. *Journal of Graph Algorithms and Applications*, 23(2):345–369, 2019.

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

Guarding Weakly-Visible Polygons with Half-Guards

Hannah Miller Hillberg* Erik Krohn† Alex Pahlow‡

Abstract

We consider a variant of the art gallery problem where all guards are limited to seeing to the right inside a weakly-visible polygon. Guards that can only see in one direction are called half-guards. In this paper, we give a polynomial time approximation scheme for vertex guarding a weakly-visible polygon with half-guards. We then show NP-hardness for vertex guarding a weakly-visible polygon with half-guards.

1 Introduction

An instance of the original *art gallery problem* takes as input a simple polygon P. A polygon P is defined by a set of points $V = \{v_1, v_2, \ldots, v_n\}$. There are edges connecting (v_i, v_{i+1}) where $i = 1, 2, \ldots, n-1$. There is also an edge connecting (v_1, v_n). If these edges do not intersect other than at adjacent points in V (or at v_1 and v_n), then P is called a simple polygon. The edges of a simple polygon give us two regions: inside the polygon and outside the polygon. For any two points $p, q \in P$, we say that p sees q if the line segment $\overline{pq}$ does not go outside of P. The art gallery problem seeks to find a guarding set of points $G \subseteq P$ such that every point $p \in P$ is seen by a point in G. In this paper, we study the vertex guarding problem which says that guards are only allowed to be placed at the vertices V. The optimization problem is defined as finding the smallest such G.

1.1 Previous Work

There are many results about guarding art galleries. Several results related to hardness and approximations can be found in [1, 5, 6, 10].

Additional Structure. Due to the inherent difficulty in fully understanding the art gallery problem for simple polygons, there has been some work done guarding polygons with additional structure, see [3, 8] for example. In this paper we consider *weakly-visible polygons* that we will describe below. Motivated by the fact that many cameras/sensors cannot sense in 360°, referred to as *full-guards* in this paper, we study guards that can sense in 180°, referred to as *half-guards*. We restrict the problem even further by only allowing these half-guards to see to the right. Even with these restrictions, the problem is difficult to solve.

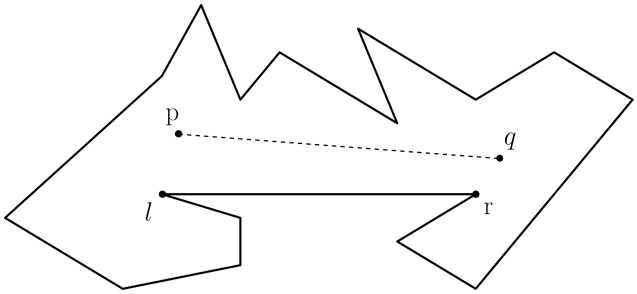


Figure 1: A WV-polygon where p sees q and every point in the polygon is seen by a point on $e = [l, r]$ or sees a point on e.

1.2 Definitions

A weakly-visible polygon (WV-polygon) P contains an edge $e = (l, r)$ such that every point in P sees at least one point on edge e. The usual definition of *sees* says that for any two points p and q inside of the polygon, if the line segment connecting p and q does not go outside of the polygon, then p *sees* q. Let $p.x$ be the x-coordinate for point p. In this paper, for a point p to see a point q, it must be the case that $p.x \leq q.x$, see Figure 1. The definition of a WV-polygon is slightly modified to say that it contains an edge $e = (l, r)$ such that every point in P sees (or is seen by) at least one point on edge e.

1.3 Our Contribution

NP-hardness has been shown for many variants of the art gallery problem. In many of those reductions, guards are allowed to see in all directions. If the problem is restricted enough, it can become polynomially time solvable, for example, see [4, 9]. If the polygon is restricted to be a WV-polygon, restrict guards to be at the vertices and only allow them to see to the right, we show that even with these many restrictions, the problem is still NP-hard.

*Department of Computer Science, University of Wisconsin - Oshkosh, Oshkosh, Wisconsin, USA hillbergh@uwosh.edu

†Department of Computer Science, University of Wisconsin - Oshkosh, Oshkosh, Wisconsin, USA krohne@uwosh.edu

Supported by the Faculty Development Sabbatical Program at the University of Wisconsin - Oshkosh.

‡Department of Computer Science, University of Wisconsin - Oshkosh, Oshkosh, Wisconsin, USA pahloa45@uwosh.edu

Figure 2: A WV-polygon that requires $\Omega(n)$ half-guards that see to the right.

Ashur et al. [2] give a polynomial time approximation scheme (PTAS) for minimum dominating set in terrain-like graphs, which we will describe later. They then show that several families of polygons have a visibility graph that is terrain-like. One such family is WV-polygons. However, their analysis does not imply that the visibility polygon of vertex guarding a WV-polygon *with half-guards* is terrain-like. We provide additional observations in this paper that show the visibility polygon is terrain-like. There are WV-polygons P that can be completely guarded with one full-guard but require $\Omega(n)$ half-guards considered in this paper, see Figure 2.

The remainder of the paper is organized as follows. Section 2 provides a PTAS for vertex guarding a WV-polygon using half-guards. Section 3 shows NP-hardness for vertex guarding a WV-polygon using half-guards. Finally, Section 4 gives a conclusion and future work.

2 PTAS for Vertex Guarding a WV-Polygon with Half-Guards

In this section, we show that the visibility graph of a WV-polygon with half-guards is terrain-like. The visibility graph is a graph $G = (V, E)$ such that V corresponds to vertices in the WV-polygon and the edges E correspond to vertices that can be seen by other vertices. Then we use Theorem 1 from [2], see Appendix, to show that a PTAS exists for vertex guarding a WV-polygon with half-guards.

Using the definition from [2], a graph $G = (V, E)$ is *terrain-like* if one can assign a unique integer from the range $[1, |V|]$, where $|V|$ is the number of vertices in the polygon, to each vertex in V, such that, if both (i, k) and (j, l) are in E, for any $i < j < k < l$, then so is (i, l).

Much of the proof from [2] assumes that the WV-polygon lies above the WV-edge by placing guards at l and r and cutting off the portion of the polygon beneath the WV-edge, see Figure 3(left). When every vertex under consideration lies above this WV-edge, the order claim holds and the visibility graph is terrain-like. However, consider doing the same thing for a WV-polygon using half-guards that see to the right. In this case, the remaining portion of the polygon cannot be assumed to be above the WV-edge. As shown in the shaded parts of Figure 3(right), the regions to the left of the placed guards are not seen. More so, a guard placed at l will not dominate a guard placed in the shaded region to the left of l. If the guards were full-guards, these regions would be seen and the l guard would dominate any optimal guard placed in this region. We show that even though these portions of the polygon are unseen and optimal guards can lie in these regions, the visibility graph connecting vertices to the guards that see them is still terrain-like.

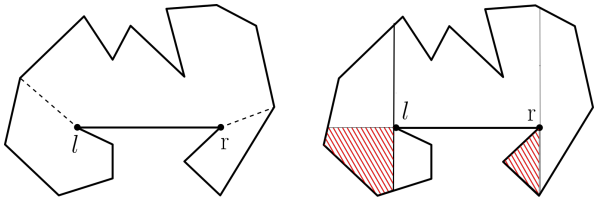


Figure 3: On the left, a full-guard placed at l and r cuts off the polygon below the WV-edge. On the right, the shaded regions are still unseen after half-guards are placed at l and r.

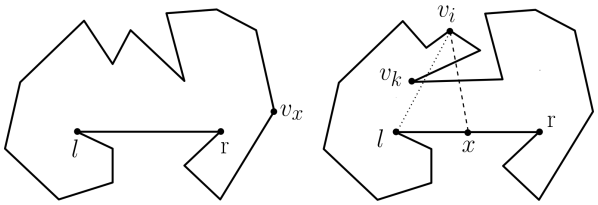


Figure 4: Example of v_x and Lemma 1.

2.1 Visibility Polygon is Terrain-Like

We will prove that the visibility polygon of the vertices is terrain-like by using the following modified order claim in WV-polygons that applies to full-guards as well as half-guards. Order the vertices walking in clockwise order starting from l: $(v_1 = l, v_2, v_2, \ldots, v_n = r)$.
Modified Order Claim: Assume that guards are placed at l and r and then consider the remaining unseen vertices. If 4 vertices are in order such that $a < b < c < d$, then if a sees c and b sees d, then a sees d.
As shown in [2], if all of a, b, c and d lie above the WV-edge, then the order claim holds and a sees d. If both a and d are below the WV-edge and both see l or both see r, then the same arguments from [2] hold for why a must see d.

The following lemmas are given for WV-polygons that apply to full-guards. For simplicity of the arguments, we assume that the WV-edge is parallel to the x-axis. Let v_x be the vertex such that every vertex $[v_{x+1}, v_{n-1}]$ is below the WV-edge. If no vertex meets this requirement, then $v_x = v_n = r$. In other words, v_x is the last vertex that is above the WV-edge when walking clockwise from l to r, see Figure 4(left).

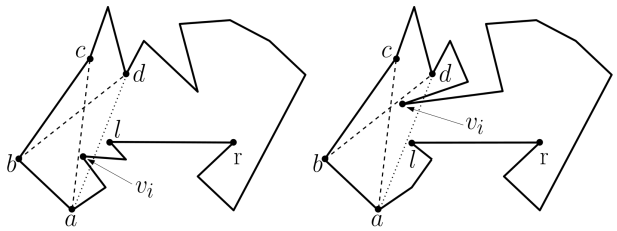


Figure 5: If a is blocked from d, then the blocker must be either l or r.

Lemma 1 *If $v_i \in [v_1, v_x]$, then no vertex $v_k \in [v_{i+1}, v_n]$ can block v_i from l.*

Proof. Assume that v_k blocks v_i from seeing l. If this happens, then v_k is to-the-left (when looking from l) of the $\overline{lv_i}$ line segment, see Figure 4(right). Let x be any point on the WV-edge that v_i sees. The $\overline{xv_i}$ line segment is to the right (when looking from l) of the $\overline{lv_i}$ line segment. This line segment is also blocked by v_k which means v_i does not see any point on the WV-edge. The polygon is not weakly-visible and we have a contradiction. □

Corollary 2 *Let v_w be the vertex such that every vertex $[v_2, v_{w-1}]$ is below the WV-edge. If no vertex meets this requirement, then $v_w = v_1 = l$. If $v_i \in [v_w, v_n]$, then no vertex $v_k \in [v_1, v_{i-1}]$ can block v_i from r.*

Lemma 3 *Consider 4 vertices in a WV-polygon such that $a < b < c < d$, a sees c, b sees d and a does not see d. If a is below the WV-edge and d is above the WV-edge, then a and d must both see l.*

Proof. Since a is below the WV-edge, a must see l. If a does not see l, then the polygon is not weakly-visible.

If d does not see l, then by Lemma 1, the v_i blocker for d must lie in the (l, d) range. If $v_i \in (b, d)$, then v_i would block b from d. If $v_i = (a, b]$, then a would not see c. It is not possible for $v_i = a$ since a is below the WV-edge. Lastly, if that vertex $v_i \in (l, a)$, then it would block a from seeing l, see Figure 5(left). Since there is no way to block d from l, d must see l. □

Corollary 4 *Consider 4 vertices in a WV-polygon such that $a < b < c < d$, a sees c, b sees d and a does not see d. If a is above the WV-edge and d is below the WV-edge, then a and d must both see r.*

This brings us to our final Lemma with full-guards:

Lemma 5 *If the order claim is broken, then a sees either l or r and also, d sees either l or r.*

Proof. If $a < b < c < d$, a sees c, b sees d, a does not see d and a and d are both below the WV-edge, then a sees l and d sees r. This, along with Lemma 3 and Corollary 4, cover the remaining cases. □

Returning to the discussion with respect to half-guards. Lemma 5 applies to full-guards. A modification of Lemma 5 is given to apply to half-guards:

Lemma 6 *If the order claim is broken, then at least one of a, b, c or d is seen by either l or r.*

The complete proof of Lemma 6 is given in the appendix. In short, if the order claim is broken, at least one of a, b, c or d must lie to the right of l or r and will necessarily be seen by one of l or r.

The analysis from [2] is now used to show a PTAS exists. First, one checks to see if the polygon can be guarded with a constant number of guards of some appropriate size. If an optimal guarding set of this size does not exist, then the first step of the algorithm is to place guards at l and r and remove the vertices that l or r see from consideration as they are already guarded. By Lemma 6, an order claim violation is not possible since at least one of the vertices involved in breaking the order claim has already been guarded by l or r and will not be in the modified problem instance. In other words, the visibility graph connecting vertices to the guards that see them are vertices that are not seen by l nor r. Since the order claim cannot be broken, when looking at the visibility graph of vertices connected to the leftmost and rightmost guards that see them, if $i < j < k < l$, $(i, k) \in E$ and $(j, l) \in E$, then $(i, l) \in E$. With this claim, the visibility graph for vertex guarding WV-polygons with half-guards that see to the right is terrain-like.

It should also be noted that the orientation of the polygon does not matter. For example, consider a polygon where the WV-edge is parallel to the y-axis and the "main" part of the polygon is to the right of the WV-edge, see Figure 6(left). In this instance, guards placed at l and r still cause the order claim to not be broken in the unguarded vertices that remain. Figure 6 shows polygons where the order claim will not be broken. No matter the orientation of the WV-edge, Lemma 6 holds for half-guards as well as full-guards.

Since the visibility graph is terrain-like, we use Theorem 1 from [2] to state the following:

Theorem 7 *There exists a PTAS for vertex guarding a weakly-visible polygon with half-guards where half-guards can only see to the right.*

3 NP-hardness for Vertex Guarding a WV-Polygon with Half-Guards

In this section, we provide a sketch for showing that vertex guarding a WV-polygon with half-guards is NP-hard. NP-hardness for terrain guarding with full-guards was shown in [7], however, the entire terrain is not seen if guards are only allowed to look down. In the appendix,

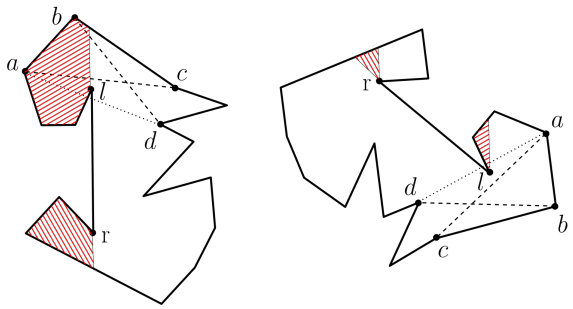


Figure 6: Shows the portion of the polygon "below" the WV-edge that is not seen by l or r.

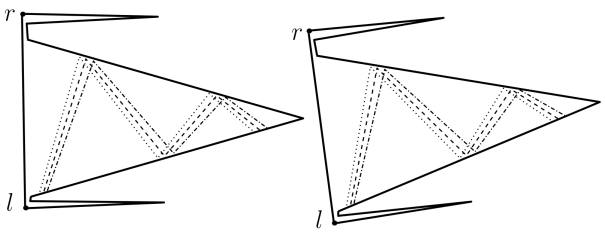


Figure 7: An overview of NP-hardness for WV-polygons.

we provide several new and updated gadgets to the reduction from [7] to show that vertex guarding a terrain with half-guards that only look down is NP-hard.

We will show how to use the terrain guarding hardness result for guards that only see down to show that vertex guarding a WV-polygon with half-guards that see to the right is NP-hard. One can take the modified terrain reduction, rotate it counterclockwise 90° and connect vertex l to vertex r to create a WV-polygon that is visible from the edge $e = (l, r)$, see Figure 7(left). The reduction holds the same way that it does for vertex guarding a terrain with half-guards that only see down.

One will notice that when the WV-edge is rotated slightly counterclockwise, the reduction still holds, see Figure 7(right). The key visibilities from the guards remain and the polygon is still weakly-visible. The reduction begins to fail whenever a guard visibility from the original reduction starts to have a negative slope. If this happens, the guard no longer sees the distinguished point(s) to its right. To account for this, the original terrain is "stretched" such that none of the guard visibilities have a negative slope. Details of this stretching are in the appendix.

4 Conclusion and Future Work

In this paper, we present a PTAS for vertex guarding WV-polygons with half-guards that see to the right. This algorithm works regardless of the orientation of the WV-edge. We also present an NP-hardness proof for vertex guarding a WV-polygon with half-guards that see to the right. Such a proof works for all instances except when the WV-edge is parallel to the y-axis and the "inside" of the polygon is to the left of the WV-edge. Whether or not this problem is NP-hard is left as an open problem. Future work might include finding a better approximation for the point guarding version of this problem. Insights provided in this paper may help with guarding polygons where the guard can choose to see either left or right, or in other natural directions. One may also be able to use these ideas when allowing guards to see 180° but guards can choose their own direction, i.e. 180°-floodlights.

References

[1] A. Aggarwal. *The art gallery theorem: its variations, applications and algorithmic aspects.* PhD thesis, The Johns Hopkins University, 1984.

[2] S. Ashur, O. Filtser, M. J. Katz, and R. Saban. Terrain-like graphs: Ptass for guarding weakly-visible polygons and terrains. *Computational Geometry*, 101:101832, 2022.

[3] P. Bhattacharya, S. Ghosh, and B. Roy. Approximability of guarding weak visibility polygons. *Discrete Applied Mathematics*, 228, 02 2017.

[4] M. C. Couto, P. J. de Rezende, and C. C. de Souza. An exact algorithm for minimizing vertex guards on art galleries. *Int. Trans. Oper. Res.*, 18(4):425–448, 2011.

[5] S. Eidenbenz. Inapproximability results for guarding polygons without holes. In *ISAAC*, pages 427–436, 1998.

[6] S. K. Ghosh. On recognizing and characterizing visibility graphs of simple polygons. *Discrete & Computational Geometry*, 17(2):143–162, 1997.

[7] J. King and E. Krohn. Terrain guarding is np-hard. *SIAM J. Comput.*, 40(5):1316–1339, 2011.

[8] E. Krohn and B. J. Nilsson. Approximate guarding of monotone and rectilinear polygons. *Algorithmica*, 66(3):564–594, 2013.

[9] A. Kröller, T. Baumgartner, S. P. Fekete, and C. Schmidt. Exact solutions and bounds for general art gallery problems. *ACM J. Exp. Algorithmics*, 17, may 2012.

[10] D. T. Lee and A. K. Lin. Computational complexity of art gallery problems. *IEEE Trans. Inform. Theory*, 32(2):276–282, March 1986.

[11] D. Lichtenstein. Planar formulae and their uses. *SIAM J. Comput.*, 11:329–343, 1982.

Appendix

Theorem 1 from [2]

Theorem *There exists a PTAS for (general) minimum dominating set in terrain-like graphs. That is, for any $\epsilon > 0$, there is a polynomial-time algorithm which, given a terrain-like graph $G = (V, E)$ and two sets $C, W \subseteq V$, returns $Q \subseteq C$ such that Q dominates W and $|Q| \leq (1+\epsilon)\cdot OPT$; here OPT is the size of a minimum subset of C that dominates W.*

Proof of Lemma 6

Proof. We break up this proof into several cases.

1. If a and d are both above the WV-edge, then the order claim cannot be broken as shown in [2].

2. If a is below the WV-edge and d is above the WV-edge, then by Lemma 3, a and d see l. It also must be the case that a and d are to the left of l (otherwise l would see one of them). Since a sees l, d sees l, a sees (or is seen by) c and b sees (or is seen by) d, the entire $\overline{ad}$ line segment is surrounded by visibility lines that cannot be pierced. Therefore, a must see (or be seen by) d, a contradiction that the order claim was broken. Since a and d cannot both be to the left of l, it must be that l sees one of them.

3. If a is above the WV-edge and d is below the WV-edge, then by Corollary 4, r sees (or is seen by) a and d. In order for r to not see a or d (and rather be seen by both a and d), both must be to the left of r. If b or c are below the WV-edge, then r will see them. Therefore, b and c must be above the WV-edge. Since d sees b, it must be the case that b is to the right of r. Since b is above and to the right of r, there must be a vertex that blocks r from seeing b. By Corollary 2, the blocker for r to b must be in the (b, r) range. If the blocker is in the (b, d) range, then d would not see b. If the blocker is in the (d, r) range, then d would not see r, a contradiction that the polygon is weakly-visible. It must be the case that r sees b.

4. If a and d are both below the WV-edge, a is to the left of l and d is to the left of r, then in order for d to see b, it must be the case that the $\overline{db}$ line segment goes below r forcing b to be to the right of r. Similar to case 3, r sees b. Although not necessary for the proof, using similar arguments, one can show that l sees c.

□

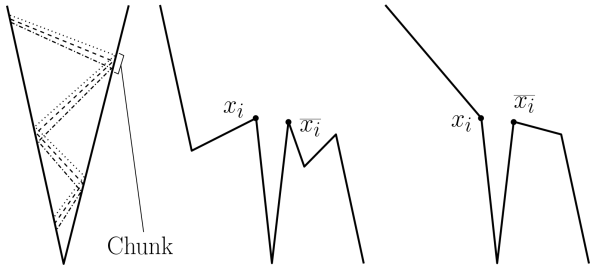


Figure 8: The left shows an overview of the NP-hardness reduction for terrain guarding. The middle is a variable gadget. The right is a starting gadget.

NP-hardness for Vertex Guarding a Terrain with Half-Guards

Abusing notation, only in this section, we will assume that half-guards can only see "down." If we restrict guards to be half-guards that see "down" in the terrain, then the terrain guarding problem is still NP-hard. In regular terrain guarding, a point p *sees* another point q if the line segment connecting p and q does not go below the terrain. In this half-guard variant, the point p sees q only if the y-coordinate of p is greater than or equal to the y-coordinate of q and the line segment connecting p and q does not go below the terrain.

Sketch of Reduction

The terrain guarding reduction is from PLANAR 3SAT [11] where an instance has n variables and m clauses. The reduction works by assigning vertices on the terrain to truth values of variables from the PLANAR 3SAT instance. For each variable in the PLANAR 3SAT instance, variable gadgets are created such that the gadget contains a vertex representing x_i and vertex representing $\overline{x_i}$, see Figure 8(middle). These variable gadgets are grouped together in chunks on the terrain. Figure 8(left) shows an example of one such chunk that contains one variable gadget for each variable in $\{x_1, x_2, \ldots, x_n\}$. These chunks are replicated on the terrain such that a guard placed at the x_i vertex in the variable gadget of x_i in chunk C_j would require a guard to be placed at the x_i vertex in another variable gadget in a different chunk C_k, see Figure 10 for an example of such mirroring of data. There are points on the terrain that correspond to clauses of the PLANAR 3SAT instance. For example, if clause $c_i = (x_{i+1} \vee \overline{x_{i+3}} \vee x_{i+4})$ were in the original PLANAR 3SAT instance, then a point on the terrain would exist that is seen by three vertices corresponding to x_i, $\overline{x_{i+3}}$ and x_{i+4}. If one of those vertices has a guard placed on it, then the clause would be satisfied. If none of those vertices has a guard placed on it, then an extra guard would be required to guard the terrain. If some minimum number of guards were placed and the entire

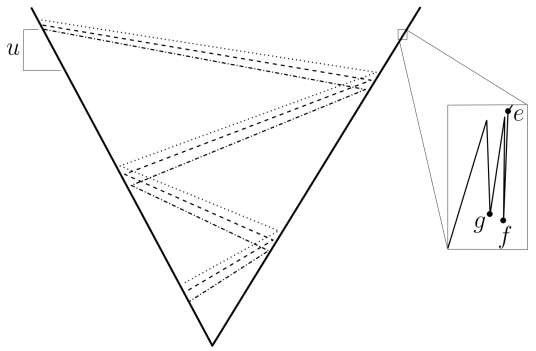


Figure 9: To ensure the entire terrain is seen, a guard is placed at e to see vertices f and g along with the u region.

terrain was seen, then the original instance was satisfied. If not, then the original instance was unsatisfiable.

The interested reader is encouraged to read [7] to see the full details of the original reduction. We modify the gadgets from [7] and show how these modified gadgets work for guards that can only see down.

Terrain Hardness Modifications

We describe the changes to the gadgets of [7] that must be made in order for the reduction to hold. Each part will explain why the original gadget doesn't work for this half-guard variant and what changes must be made in order to have the reduction hold.

Between Chunks: Let us order the chunks from top to bottom $(C_1, C_2, \ldots, C_m)$. We will assume that all variables gadgets in chunk C_i are below all variable gadgets in chunk C_{i-1}. In the original reduction, the terrain between C_i the C_{i+2} is seen by any guard placed in the C_{i+1} chunk. Since guards can only see down in this variant, a portion of the terrain, which we will call u, below the lowest variable gadget in C_i and above the highest variable gadget in C_{i+1} will be unseen, see Figure 9. To fix this, we place a new gadget on the other side of the terrain from C_i. This gadget is at the same y-coordinate of the lowest variable gadget in chunk C_i. This ensures that the new guard will not affect the mirroring. The vertices f and g can only be seen from vertex e. Placing a guard at e will see f and g and also see the unseen region u below chunk C_i.

Variable Gadget: The variable gadgets, shown in Figure 10, do not need to be tweaked much to work. Assume the $\overline{x_i}$ vertex in the variable gadget in chunk C_j has a guard placed on it and it sees b. In this case, a guard is placed at $\overline{x_i}$ in the variable gadget in chunk C_{j+1} to see a and c. The entire x_i variable gadget in chunk C_{j+1} is seen. Likewise, if a guard placed at x_i in chunk C_j sees a, then a guard is placed at x_i in chunk C_{j+1} sees b and c. A small portion of the terrain below $\overline{x_i}$ in chunk

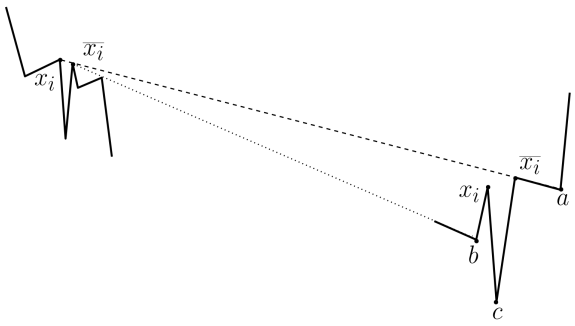


Figure 10: The variable gadget remains unchanged from [7].

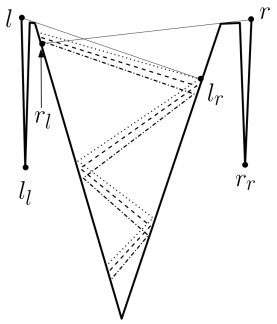


Figure 11: The overview of the terrain reduction where guards see down.

C_{j+1} may have been missed, see Figure 10. However, if the $\overline{x_i}$ vertex in chunk C_j is lowered just slightly, then the guard placed at x_i in C_j will see this region and an additional guard is not required. One must ensure the previously placed x_i does not see b but it can see anything in this gadget above b. Therefore, we ensure that $\overline{x_i}$ in the previous variable gadget is placed in such a way that it blocks x_i from seeing just above b in the subsequent variable gadget.

Removing a Variable: We will assume we are removing a variable from chunk C_i. When a variable gadget is removed going upwards, the gadget is modified slightly to remove the a and b vertices. Such a gadget is also called a starting gadget. When a guard is placed at the "lower" vertex in this gadget, a small portion of the terrain below the "higher" vertex remains unseen. In Figure 8(right), a starting gadget is shown. A guard placed at $\overline{x_i}$ would not see the small portion of the terrain below the x_i guard. To ensure this region is seen, we look at the variable patterns placed in the chunk above it, chunk C_{i-1}. The guard placed in the lowest variable pattern will see all of these potentially unseen portions. For example, in Figure 11, the guard placed in the variable gadget, x_k, directly above the r_l point will see all of these unseen regions in the variable patterns between l_r and the variable gadget for x_k in chunk C_2. Therefore, no modification is needed and no additional

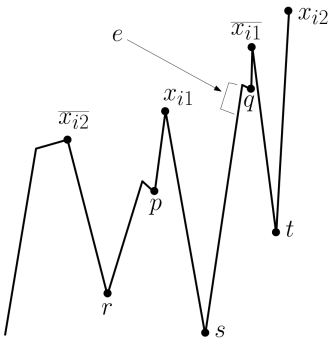


Figure 12: The inversion gadget for variable x_i in chunk C_{j+1}. This gadget is not tweaked but the variable gadget x_i in chunk C_j is modified slightly.

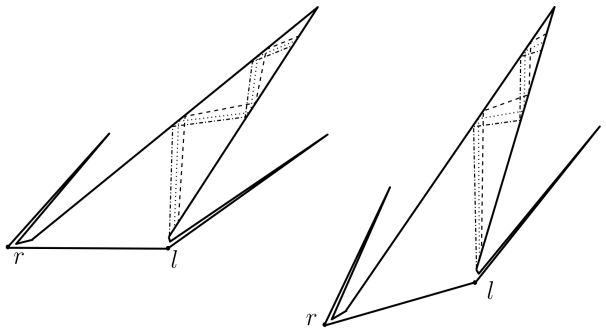


Figure 13: A stretched WV-polygon such that the NP-hardness result holds.

guard is needed.

When removing a variable going down, the variable gadget is simply removed. The guard placed in the previous chunk's lowest variable gadget will see this region. If the lowest variable was being removed, then the e guard in the gadget that sees between chunks will see this region (see Figure 9).

Above chunk C_1 and C_2: Since guards cannot see "up," a guard must be placed that guards the terrain above the first set of variable gadgets in chunk C_1 and above the variable gadgets in C_2. Assume WLOG that C_1 is on the left side of the terrain. Two guards are placed at points l and r as shown in Figure 11. These guards are required to see their own set of distinguished points, namely l_l, l_r, r_r and r_l. They see the "top" of the terrain above chunks C_1 and C_2. Since all gadgets in chunk C_1 are starting gadgets, r_l is placed below chunk C_1 to ensure the relevant part of the terrain in the starting gadgets and the terrain above the chunk are seen.

Clause Gadgets: No change is needed for the clause gadgets. All points in the clause gadgets are seen by the appropriate vertices above it.

Inversion Gadgets: A small update is needed for the inversion gadgets. See Figure 12 for a sample inversion gadget placed in chunk C_{j+1}. If a guard from the variable gadget x_i in chunk C_j sees point p, then when guards are placed at $\overline{x_{i1}}$ and $\overline{x_{i2}}$, the entire gadget is seen. If the guard from chunk C_j guard sees point q, then guards are placed at x_{i1} and x_{i2}. This leaves a small portion of the terrain unseen, the line segment e in Figure 12. To fix this, similar to the tweak of the variable gadget, the previously placed guard in chunk C_j is tweaked such that it sees just over the x_{i1} guard. In this example, the previously placed guard must see q and not see p. As long as it is blocked from p, this is all that matters. Therefore, it can be tweaked to see the e line segment.

Putting it all together: As seen above, certain tweaks and updates are made to ensure that the entire terrain is seen. Making these changes will cause the minimum number of guards that must be placed, k, to increase by $m+1$. An additional $m-1$ guards are needed to see the unseen regions between chunks. Two additional guards are added at l and r to see the "top" of the terrain. This gives us a total of $m-1+2 = m+1$ additional required guards. None of these additional required guards see any of the original distinguished points of the terrain. They see their own set of distinguished points and also see the portion of the terrain that would have been unseen. As shown in [7], if k guards can guard the entire terrain, the instance is satisfiable. If more than k are needed, then the instance is not satisfiable.

Theorem 8 *Finding the smallest vertex guard cover for guarding a terrain using half-guards that see down is NP-hard.*

Stretching WV-Polygon Hardness

An example of this stretching is seen in Figure 13(left). As seen in the example, the original bottom of the terrain is pulled up and to the right to ensure the polygon remains weakly-visible and all important lines of sight look to the right. The l and r vertices are tweaked slightly to ensure they also see their respective distinguished vertices to the right.

This stretching of the polygon works even if the WV-edge has a positive slope, see Figure 13(right). The polygon can continue to be stretched as far up and right as necessary. However, the tweak fails when the WV-edge is a vertical edge and the inside of the polygon is to the left of the edge. We leave this as an open problem.

A Randomized Algorithm for Non-crossing Matching of Online Points

Shahin Kamali* Pooya Nikbakht† Arezoo Sajadpour‡

Abstract

We study randomized algorithms for the online non-crossing matching problem. Given a sequence of n online points in general position, the goal is to create a matching of maximum size so that the line segments connecting pairs of matched points do not cross. In previous work, Bose et al. [CCCG 2020] showed that a simple greedy algorithm matches at least $\lceil 2n/3-1/3 \rceil \approx 0.\bar{6}n$ points, and it is the best that any deterministic algorithm can achieve. In this paper, we show that randomization helps to achieve a better competitive ratio, that is, we present a randomized algorithm that matches at least $235n/351 - 202/351 \approx 0.6695n$ points.

1 Introduction

In the geometric matching problems, the input is a set of geometric objects, and the goal is to create a pairwise matching of these objects under different restrictions and objectives. In the bottleneck matching problem, for example, the goal is to create a perfect matching of n points, assuming n is even, so as to minimize the maximum length of the line segments that connect matched pairs [8]. Using the same terminology as in graph theory, we refer to the line segments that connect pairs of matched vertices as the *edges* of the matching. Other variants of the geometric matching problems ask for perfect matchings that minimize the total length of edges [4] or maximize the length of the shortest edge [6]. Matching objects other than points are also studied (see, e.g., [1, 2])

In the non-crossing matching problem, the input is a set of points in general position, and the goal is to match points in a way that the edges between the matched pairs do not cross. In the offline setting, it is rather easy to solve the problem: one can sort all points by their x-coordinate and match pairs of consecutive points. All points, except possibly the last one, will be matched. The running time of this algorithm is $O(n \log n)$, which is asymptotically optimal [5]. Other variants of non-crossing matching have been studied in the offline setting (see [7]). For example, Aloupis et al. [1] considered the computational complexity of finding non-crossing matching of a set of points with a set of geometric objects that can be a line, a line segment, or a convex polygon.

Bose et al. [3] studied the online variant of the non-crossing matching. Under this setting, the input is a set of n points in general position that appear in an online, sequential manner. When a point x arrives, an online algorithm can match it with an existing unmatched point y, provided that the edge between them does not cross previous edges in the matching. Alternatively, the algorithm can leave the point unmatched to be matched later. In taking these decisions, the algorithm has no information about the forthcoming points or the length of the input. The algorithm's decisions are irrevocable in the sense that once a pair of points is matched, that pair cannot subsequently be removed from the matching. The objective is to find a maximum matching.

Under a worst-case analysis, where an adversary generates the online sequence, it is not possible to match all points. For example, consider an input that starts with two points x and y. If an online algorithm leaves the two points unmatched, then the adversary ends the sequence, and the matching is already sub-optimal. If the algorithm matches x and y, then the adversary generates the next two points on the opposite sides of the line between x and y, and the matching will be sub-optimal for this input of length $n = 4$. Bose et al. [3] extended this argument to show that in the worst case, no deterministic algorithm can match more than $\lceil 2n/3 - 1/3 \rceil$ points. Meanwhile, they showed that any greedy algorithm matches at least $\lceil 2n/3 - 1/3 \rceil$ points, and hence is optimal. An algorithm has the greedy property if it never leaves a point x unmatched if there is a suitable unmatched point y that x can be matched to (that is, the edge between x and y does not cross existing edges in the matching).

1.1 Contribution

We study randomized algorithms for the non-crossing matching problem. As in [3], we study worst-case scenarios, where the input is generated adversarially. We assume the adversary is *oblivious* to the random choices made by the algorithm, but it is aware of how the algorithms works (that is, the code of the algorithm).

We present a randomized algorithm that matches at

*Department of Computer Science, University of Manitoba, Winnipeg, Canada, shahin.kamali@umanitoba.ca

†Department of Computer Science, University of Manitoba, Winnipeg, Canada, nikbakhp@myumanitoba.ca

‡Department of Computer Science, University of Manitoba, Winnipeg, Canada, sajadpoa@myumanitoba.ca

least $\lfloor 235n/351 - 202/351 \rfloor \approx 0.6695n$ points on expectation for any input of size n. This shows the advantage of randomized algorithms over deterministic ones, which match roughly $0.\bar{6}n$ points in the worst case.

There are two main components in our randomized algorithm. First, the algorithm maintains a convex partitioning of the plane and matches two points only if they appear in the same partition. This is followed by updating the partitioning by extending the edge between the matched pair. This partitioning enables us to use a simple inductive argument to analyze the algorithm. Second, the algorithm deviates from the greedy strategy. In particular, the algorithm gives a chance for an incoming point x to stay unmatched even if there are one or two points in the same convex region that it can match. As we will see, this will be essential for any improvement over deterministic algorithms.

2 A Randomized Online Algorithm

We present and analyze a randomized online algorithm for the non-crossing matching problem. In what follows, for any $a \neq b$ we use L_{ab} to denote the line passing through a and b, and S_{ab} to denote the line segment between a and b.

2.1 Algorithm's description

The algorithm maintains a partitioning of the plane into convex regions and matches points only if they belong to the same region. In the beginning, there is only one region that is formed by the entire plane. After four points appear inside a convex region, one or two pairs of points are matched, and the convex region is partitioned into two or three convex regions by extending the line segments passing through the matched pairs.

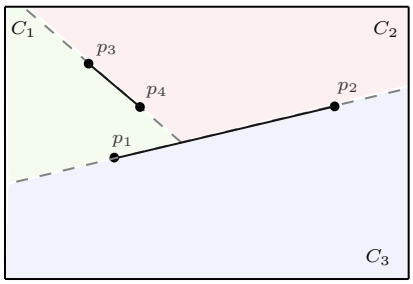
Let x, y, z, and w be the first four points inside a convex region C in the same order. In what follows, we describe how these four points are treated.

- Upon the arrival of x, there is no decision to make, given that there is no point inside C to be matched with x.
- Upon the arrival of y, it is matched with x with a probability of $1/2$ and stays unmatched with a probability of $1/2$.
- Upon the arrival of z, if the pair (x, y) is already matched, then there is no decision to make. Otherwise, z is matched with x with a probability of $1/3$, with y with a probability of $1/3$, and stays unmatched with a probability of $1/3$.
- Upon the arrival of w, there are two possibilities to consider:
  - First, suppose a pair of points $a, b \in \{x, y, z\}$ is already matched, while a third point $c \in \{x, y, z\}/\{a, b\}$ is unmatched. If it is possible to match w with c (that is, S_{wc} does not cross S_{ab}), then w is matched with c; otherwise, when S_{wc} and S_{ab} cross, there is no decision to make.
  - Second, suppose no pair of the first three points are matched. Then w is matched with a point $a \in \{x, y, z\}$ so that the two points $b, c \in \{x, y, z\}/\{a\}$ appear on different sides of the line L_{aw} (if there is more than one such point, w is matched with z).

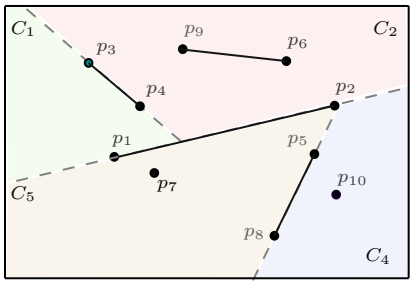
After the arrival of four points inside C, either all points are matched into two pairs, in which case we say a "double-pair is realized", or only two points are matched while the other two appear on different sides of the matched pair, in which case we say a "single-pair is realized". If a single-pair is realized, in this case, C is partitioned into two convex regions. If a double-pair is realized, then algorithm extends the line segments between the matched pairs until they hit the boundary of C or the (non-extended) segment between the other matched pair. This is followed by extending the line segment between the second pair until it hits the boundary of C or extended line that passes through the first matched pair. When a double-pair is realized, C is partitioned into three convex regions.

Assume $n \geq 8$. A single-pair is "good" if, after the appearance of all n points, both of the two regions resulted from extending the line segment of the matching contain at least 2 points, and it is "bad" otherwise. A double-pair is said to be "good" if, after the appearance of all n points, one of the three regions formed by extending the line segments of the two matchings is empty; otherwise, it is "bad". The presence of 2 or more than 2 points, or no points in a region provides a possibility of matching all pairs; hence we assert that a single/double pair is "good" or "bad" as specified above.

The following example illustrates the algorithm's steps. Consider an input formed by 10 points labeled from p_1 to p_{10} in the order of their appearance, as depicted in Figure 1. The convex regions maintained by the algorithm are highlighted in different colors. Initially, the entire plane is a convex region C_0, where point p_1 appears. Upon the arrival of p_2, the algorithm match it with p_1 with a probability of $1/2$. Suppose (p_1, p_2) are matched. Then, there is no decision to be made for p_3. Upon the arrival of p_4, the line segments $S_{p_1p_2}$ and $S_{p_3p_4}$ do not cross. Therefore, p_4 is matched with p_3. At this point, four points have appeared in C_0 and a double-pair (p_1, p_2) and (p_3, p_4) has been realized. Therefore, C_0 is partitioned into three smaller convex regions C_1, C_2, and C_3 by extending S_{p_1,p_2} and then S_{p_3,p_4} (Figure 1a). Points p_5 and p_6 appear respectively in C_3 and



(a) The state of the algorithm after processing $p_1, \ldots, p_4$.



(b) The state of the algorithm after processing $p_1, \ldots, p_{10}$.

Figure 1: One possible output of the algorithm when the input is a sequence of 10 points labeled as $p_1, \ldots, p_{10}$ in the order of their appearance.

C_2. Since these are the first points in their respective regions, there is no decision to be made, and they stay unmatched. Subsequently, p_7 appears in C_3 and the algorithm matches with p_5 with a probability of 1/2. Suppose these two points are not matched. Upon the arrival of p_8 in C_3, it is matched with p_5 or p_7, each with a probability of 1/3, and is left unmatched with a probability of 1/3. Suppose (p_5, p_8) are matched. Next, point p_9 appears in C_2 and is matched with p_6 with a probability of 1/2, and stays unmatched with a probability of 1/2. Suppose (p_6, p_9) are matched. Finally, point p_{10} appears on C_3. Given that the $S_{p_7 p_{10}}$ crosses $S_{p_5 p_8}$, there is no decision to be made, and p_{10} stays unmatched. At this point, four points have appeared in C_3, and a single-pair (p_5, p_8) has been realized. Therefore, C_3 is partitioned into two smaller convex regions C_4 and C_5 by extending S_{p_5, p_8} (Figure 1b).

2.2 Algorithm's analysis

Let $f(n)$ denote the expected number of unmatched points left by the algorithm when input is formed by n items. We use an inductive argument to find an upper bound for $f(n)$. First, we prove the following lemma. The proof is based on case analysis, and can be found in the appendix:

Lemma 1 *(appendix) We have $f(0) = 0$, $f(1) = 1$, $f(2) = 1$, $f(3) = 4/3$, $f(4) \leq 4/3, f(5) \leq 5/3, f(6) \leq 20/9$, and $f(7) \leq 52/18$.*

We use an inductive argument to prove $f(n) \leq cn + d$ where $c = 116/351 \approx 0.3304$ and $d = 32c - 10 = 202/351 \approx 0.5754$. First, we apply Lemma 1 to establish the base of induction in the following theorem.

Lemma 2 *(appendix) For $n \in [2, 7]$, it holds that $f(n) \leq cn + d$ where $c = 116/351$ and $d = 202/351$.*

Lemma 3 *For $n \geq 8$, after serving the first four points inside a convex region, at least one of the followings hold:*

- *A good single-pair is realized with a probability of at least 1/6*
- *A good double-pair is realized with a probability of at least 1/6.*

Proof. (sketch) We provide a sketch of the proof here. The detailed proof can be found in the appendix. Let x, y, z, and w denote the first four points in the same order that they appear.

First, suppose the convex hull formed by the four points is a triangle Δ. If w is inside Δ, then the pairs (x, y) and (w, z) form a double-pair that is realized with a probability of 1/2. If this is bad, then there should be at least one future point on each side of the line passing through (w, z), which means (w, z) is a good single-pair. We note that the single-pair formed by (w, z) is realized with a probability of 1/6. Next, suppose w is a vertex of Δ and another point $c \in \{x, y, z\}$ is inside Δ. Let a, b be the other two points in $\{x, y, z\}$. Then, the pairs (a, b) and (c, w) form a double-pair which is realized with a probability of at least 1/6. If this double-pair is not good, then (w, z) is a good single-pair which is realized with a probability of 1/6.

Next, suppose the convex hull formed by the four points is a quadrilateral and includes all of them as its vertices. In this case, each of the two single-pairs formed by the diagonals of the convex hull is realized by a probability of at least 1/6. If both of these single-pairs are bad, then all the remaining points in the input sequence must appear in one of the quarter-planes formed by extending these diagonals. Then, the double-pair formed by the pair of points on the boundary of the quarter-plane and the pair of points outside of the quarter-plain form a good double-pair. The probability of such a double-pair to be realized is at least 1/6. □

We are now ready to prove the main result.

Theorem 4 *There is a randomized algorithm that, for any input formed by $n \geq 2$ points, leaves at most $cn + d$ points unmatched, where $c = 116/351$ and $d = 202/351$.*

Proof. We use an inductive argument to show that our algorithm satisfies the conditions specified in the theorem. For $n \leq 7$, the claim holds by Lemma 1. Suppose

$n \geq 8$, and assume that for any $m < n$, it holds that $f(m) \leq cm + d$.

First, we claim that the number of unmatched points is at most $cn + d + (2 - 6c)$ when a bad single-pair is realized, or a bad double-pair is realized after the first four points appear. If a bad single-pair is realized, then either (I) there is one point on one side of the matched pair and $n - 3 > 2$ points on the other side, or (II) there is no point on one side of the matched pair and $n - 2 > 2$ points on the other side. For (I), by the induction hypothesis, the number of unmatched points on the side with $n-3$ points will be at most $f(n-3) \leq cn-3c+d$. Therefore, the number of unmatched points is at most $f(n-3)+1 \leq cn-3c+d+1 < cn+d+(2-6c)$. The last inequality holds because $c < 1/3$. For (II), the number of unmatched points will be at most $f(n-2) \leq cn+d-2c < cn+d+(2-6c)$. If a double-pair is realized which is not good, then one of the followings holds for the three regions formed by extending the line segments between the matched pairs:

i) One region contains $n-6$ points, and the other two regions each contains one point. Note that $n-6 \geq 2$ since $n \geq 8$. By the induction hypothesis, the number of unmatched points is at most $2 + f(n - 6) = cn + d + (2 - 6c)$.

ii) One region contains $m \geq 2$ points, another region contains one point, and the third region contains $n - m - 5 \geq 2$ points. The number of unmatched points is at most $f(m) + f(n - m - 5) + 1 \leq cn - 5c + 2d + 1 < cn + d + (2 - 6c)$. The last inequality holds because $c + d < 1$.

iii) One region contains $m_1 \geq 2$ points, one region contains $m_2 \geq 2$ points, and the third region contains $m_3 = n - m_1 - m_2 - 4 \geq 2$ points. The number of unmatched points is at most $f(m_1) + f(m_2) + f(m_3) \leq cn - 4c + 3d < cn + d + (2 - 6c)$. The last inequality holds because $c + d < 1$.

In summary, if a bad single-pair or a bad double-pair is realized, the number of unmatched points is at most $cn + d + (2 - 6c)$, and the claim holds.

By Lemma 3, after the appearance of the first four points, either a) a good pair or b) a good double-pair can be realized with a probability of at least 1/6.

Suppose case a) holds, that is, a good single-pair is realized with a probability of at least 1/6, which implies a bad single-pair or double-pair is realized with a probability of at most 5/6. In case the good single-pair is realized, there will be $m \geq 2$ points on one side of the line segment connecting matched pair, and $n - m - 2 \geq 2$ points on the other side. Therefore, the number of unmatched points will be at most $f(m)+f(n-m-2) \leq cn+2d-2c = (cn+d)+(d-2c)$. On expectation, the number of unmatched points will be at most $1/6((cn+d)+(d-2c))+5/6(cn+d+(2-6c)) = cn + d + 1/6(d - 32c + 10) = cn + d$. The last equality holds because $d = 32c - 10$.

Next, suppose case b) holds, that is, a good double-pair is realized with a probability of at least 1/6, which implies a bad single-pair or double-pair is realized with a probability of at most 5/6. In case the good double-pair is realized, by definition, at least one of the three convex regions formed by extending the double-pair will be empty. For the other two regions, we have the following cases:

i) One region is empty, and the other contains $n-4 \geq 2$ points, in which case the number of unmatched points becomes $f(n - 4) \leq cn + d - 4c < cn + d + (1 - 5c)$. The last inequality holds because $c < 1$.

ii) One region contains a single point, and the other one contains $n - 5 \geq 2$ points. The number of unmatched points will be at most $f(n - 5) + 1 \leq cn + d + (1 - 5c)$.

iii) Both regions include $m \geq 2$ and $n - m - 4 \geq 2$ points. In this case, the number of unmatched points will be at most $f(m) + f(n - m - 4) \leq cn + d + (d - 4c) < cn + d + (1 - 5c)$. The last inequality holds because $c + d < 1$.

Therefore, as long as the good double-pair is realized, the number of unmatched points will be at most $cn + d + (1 - 5c)$. On expectation, we can write $f(n) \leq 1/6((cn + d) + (1 - 5c)) + 5/6((cn + d) + (2 - 6c)) = cn + d + 1/6(11 - 35c) < cn + d$. The last inequality holds since $c > 11/35$. □

References

[1] G. Aloupis, J. Cardinal, S. Collette, E. D. Demaine, M. L. Demaine, M. Dulieu, R. F. Monroy, V. Hart, F. Hurtado, S. Langerman, M. Saumell, C. Seara, and P. Taslakian. Non-crossing matchings of points with geometric objects. *Comput. Geom.*, 46(1):78–92, 2013.

[2] A. Banik, M. J. Katz, and M. Simakov. Bottleneck segment matching. In *Proceedings of the 27th Canadian Conference on Computational Geometry CCCG*, 2015.

[3] P. Bose, P. Carmi, S. Durocher, S. Kamali, and A. Sajadpour. Non-crossing matching of online points. In *Proc. 32nd Canadian Conference on Computational Geometry (CCCG)*, 2020.

[4] A. Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.

[5] J. Erickson. https://mathoverflow.net/questions/86906. https://mathoverflow.net/questions/86906. Accessed: 2020-05-03.

[6] A. Kaneko and M. Kano. Discrete geometry on red and blue points in the plane—a survey. *Discrete & Computational Geometry*, 25:551–570, 2003.

[7] L. Lovász and M. Plummer. *Matching Theory*. AMS Chelsea Publishing Series. North-Holland, 2009.

[8] P. M. Vaidya. Geometry helps in matching. *SIAM J. Comput.*, 18(6):1201–1225, 1989.

Appendix

In order to prove Lemma 1, we first prove the following lemma:

Lemma 5 *After four points arrived in the convex region C, with a probability of at least 1/3, a double-pair is realized, and with a probability of at most 2/3, a single-pair is realized.*

Proof. Let x, y, z, and w denote the four points in the same order they appear. There are two cases to consider:

- Suppose S_{xy} crosses S_{wz}. With a probability of $1/2$, x and y are not matched. After that, with a probability of $2/3$, z is matched to x or y. Without loss of generality, assume z is matched with x. Given that S_{xy} crosses S_{wz}, line segments S_{xz} and S_{yw} will not cross, implying that w is matched to y, and a double-pair is realized. So, with a probability of at least $1/2 \cdot 2/3 = 1/3$, all points are matched, and a double-pair is realized.
- Suppose S_{xy} does not cross S_{xz}. Then, (x, y) are matched with a probability of $1/2$, and after that, (w, z) are matched, and a double-pair is realized.

□

Using Lemma 5, we can prove Lemma 1:

Lemma 1 *We have $f(0) = 0$, $f(1) = 1$, $f(2) = 1$, $f(3) = 4/3$, $f(4) \leq 4/3, f(5) \leq 5/3, f(6) \leq 20/9$, and $f(7) \leq 52/18$.*

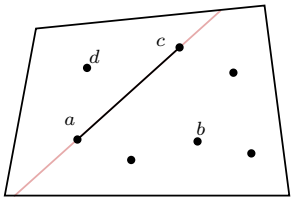
Proof. Suppose n items appear in a convex region C. The proof is trivial for $n \leq 2$. In what follows, we prove the lemma for other values of n.

- For $n = 3$, it is possible that all points stay unmatched, which happens when the second point is not matched with the first one (with a probability of $1/2$), and then the third point is not matched with any of the first two points (with a probability of $1/3$). Therefore, with a probability of $1/6$, all three points stay unmatched, and one point stays unmatched with a probability of $5/6$. We can write $f(3) = 1/6 \cdot 3 + 5/6 \cdot 1 = 4/3$.
- For $n = 4$, using Lemma 5, we can write $f(4) \leq 1/3 \cdot 0 + 2/3 \cdot 2 = 4/3$.
- For $n = 5$, after the first four points appeared, either a single-pair or a double-pair is realized:
  - Suppose a single-pair is realized. Then, C is partitioned into two regions, one containing one point and the other one containing two points. Therefore, it is expected that $f(1) + f(2) = 2$ points stay unmatched.
  - Suppose a double-pair is realized. Then, the first four points are matched, and only the fifth point stays unmatched.

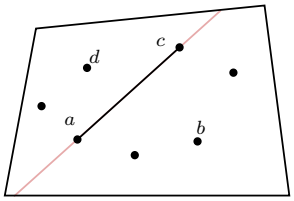
  By Lemma 5, with a probability of at least $1/3$, a double-pair is realized, and with a probability of at most $2/3$, a single-pair is realized. Therefore, we can write $f(5) \leq 1/3 \cdot 1 + 2/3 \cdot 2 = 5/3$.
- For $n = 6$, after the first four points appeared, either a single-pair or a double-pair is realized:
  - Suppose a single-pair is realized. Then, C is partitioned into two regions. Either (i) the fifth or the sixth points appear on the same region, in which case one region will have one point, and the other one will have three points, or (ii) the fifth and the sixth points appear in different regions, in which case each region contains two points. Therefore, it is expected that at most $\max\{f(1) + f(3), f(2) + f(2)\} = 7/3$ points stay unmatched.
  - Suppose a double-pair is realized. Then, at most 2 points (the last two points) stay unmatched.

  By Lemma 5, with a probability of at least $1/3$, a double-pair is realized, and with a probability of at most $2/3$, a single-pair is realized. Therefore, we can write $f(6) \leq 1/3 \cdot 2 + 2/3 \cdot 7/3 = 20/9$.
- For $n = 7$, after the first four points appeared, either a single-pair or a double-pair is realized:
  - Suppose a single-pair is realized. Then, C is partitioned into two regions. Either (i) the fifth, the sixth, and the seventh points all appear in the same region, in which case one region has one point, and the other one has four points (Figure 2a), or (ii) one of these points appear in one region, and the other two appear in the other region, in which case one region contains two points, and the other region contains three points (Figure 2b). Therefore, it is expected that at most $\max\{f(1)+f(4), f(2)+f(3)\} \leq \max\{1+4/3, 1+4/3\} = 7/3$ points stay unmatched.
  - Suppose a double-pair is realized. Then, at most three points stay unmatched, which happens when any of the three regions formed by partitioning of the first four points includes a single point (see Figure 2c).

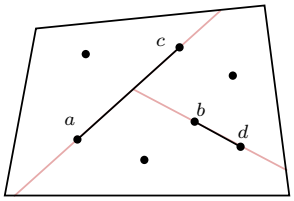
  Unlike other cases, here, the expected number of unmatched points is larger when a double-pair is realized, and hence we cannot use Lemma 5. Instead, we note that the probability of a single-pair being realized is at least $1/6$ This is because a single-pair is realized if either (i) the first two points are matched with a probability of $1/2$, and the other two points appear on opposite sides of the line passing through the matched points, happening with a total probability of $1/2$, (ii) the first two points are not matched with a probability of $1/2$, and the third point is matched to either of the first points with a probability of $1/3$, and the fourth point appears on the side of the matched line that the other unmatched point is not on, happening with a total probability of $1/6$, or (iii) the first three points stay unmatched with a probability of



(a) The case where a single-pair is realized, and the last three points appear in different regions.



(b) The case where a single-pair is realized, and the last three points appear in different regions.



(c) The case where a double-pair is realized, and the last three points appear in different regions.

Figure 2: The cases used in the calculation of $f(7)$; $a, b, c, d \in \{x, y, z, w\}$ where x, y, z, and w are the first four points in the same order of their appearance.

$1/2 \cdot 1/3 = 1/6$, and then the fourth point gets match to the point that bisects the unmatched points, happening with a total probability of 1/6. Therefore, we can write $f(7) \leq 5/6 \cdot 3 + 1/6 \cdot 7/3 = 52/18$ (see Figure 2).

□

Lemma 2 *For $n \in [2, 7]$, it holds that $f(n) \leq cn + d$ where $c = 116/351$ and $d = 202/351$.*

Proof. The proof follows from Lemma 1. For $n = 2$, we have $f(2) = 1 < 2c + d$ (since $2c + d > 1.2362$). For $n = 3$, we have $f(3) = 4/3 = 3c + d$ (since $3c + d > 1.5669$). For $n = 4$, we have $f(4) \leq 4/3 < 4c + d$ (since $4c + d > 1.8974$). For $n = 5$, we have $f(5) \leq 5/3 < 5c + d$ (since $5c + d > 2.2279$). For $n = 6$, we have $f(6) \leq 20/9 < 6c + d$ (since $6c + d > 2.5584$). For $n = 7$, we have $f(7) \leq 52/18 = 7c + d$ (note that $7c + d = 52/18$). □

Next, we provide the detailed proof of Lemma 3:

Lemma 3 *For $n \geq 8$, after serving the first four points inside a convex region, at least one of the followings hold:*

- *A good single-pair is realized with a probability of at least 1/6*
- *A good double-pair is realized with a probability of at least 1/6.*

Proof. Let x, y, z, and w denote the first four points in the same order that they appear.

First, suppose the convex hull formed by the four points is a triangle Δ which includes the fourth point inside it. We consider the following two cases:

- Assume w is the point that is inside Δ. Then the pairs (x, y) and (w, z) form a double-pair that is realized with a probability of 1/2. This is because the pair (x, y) is matched with a probability of 1/2, and then the pair (w, z) is matched with a probability of 1. Meanwhile, (w, z) is a single-pair which is realized with a probability of 1/6. This is because, with a probability of 1/6, the first three points stay unmatched, and then the algorithm matches w to z with a probability of 1. Now, if the double pair formed by the pairs (x, y) and (w, z) is bad, then there should be at least one future point on each side of the line passing through (w, z), which means (w, z) is a good single-pair (see Figure 3a).
- Assume w is a vertex of Δ and another point $c \in \{x, y, z\}$ is inside Δ. Let a, b be the other two points in $\{x, y, z\}$. Then, the pairs (a, b) and (c, w) form a double-pair which is realized with a probability of at least 1/6. This is because the pair (a, b) is matched with a probability of at least 1/6 (the pair (a, b) is matched with a probability of 1/2 if $z \notin \{a, b\}$, and with a probability of 1/6 if $z \in \{a, b\}$), and then w is matched with c with a probability of 1. Meanwhile, the pair (c, w) is a single-pair which is realized with a probability of 1/6. Similar to the previous case, if the double pair formed by the pairs (a, b) and (c, w) is bad, then there should be at least one future point on each side of (a, b), which means (a, b) is a good single-pair (see Figure 3b).

Next, suppose the convex hull formed by the four points is a quadrilateral and includes all of them. Consider the two single-pairs formed by the diagonals of the convex hull. Any of these pairs can be realized with a probability of at least 1/6. Specifically, the diagonal involving w is realized when no pair of points from $\{x, y, z\}$ are matched, which takes place with a probability of 1/6. The other diagonal is either between x and y, which is realized with a probability of 1/2, or between z and $a \in \{x, y\}$, which is realized with a probability of 1/6. Therefore, if any of the two diagonal forms a good single-pair, the statement of the lemma holds, and we are done (see Figure 3c). If none of the two diagonals is good, then all the remaining points in the input sequence should appear in one of the quarter-planes formed by extending these diagonals (see Figure 3d). Then, the double-pair formed by the pair of points on the boundary of the quarter-plane (points b and c in Figure 3d) and the pair of points outside of the quarter-plain (points w and a in Figure 3d) form a good double-pair. The probability of such a double-pair to be realized is at least 1/6. This is because one of the pairs in the double-pair involves two of the first three points. If these points are (x, y), the double-pair is

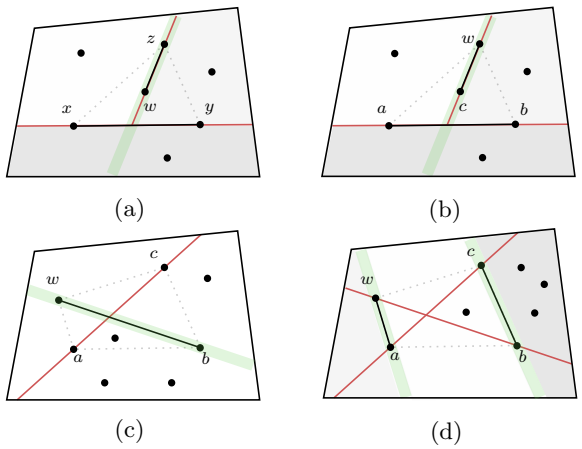


Figure 3: An illustration of the proof of Lemma 3. (a) when w is inside the triangle Δ, either the single-pair formed by (w, z) is a good single-pair, or the double-pair formed by $(x, y), (w, z)$ is a good double-pair. (b) when $c \in \{x, y, z\}$ is inside the triangle Δ, either the double pair formed by $(a, b), (w, c)$ is a good double-pair, or the single-pair formed by (w, c) is a good single-pair. (c) the case when at least one of the diagonals of the convex hull formed by the four points (here (w, b)) forms a good single-pair (d) when none of the single-pairs formed by the diagonals of the convex hull are good, all remaining points appear in one of the quarter-planes formed by extending these diagonals; therefore, the pair of points on the boundary of the quarter-plane (here (b, c)) and the pair of points outside the quarter-planes (here (w, a)) form a good double-pair.

realized with a probability of 1/2; otherwise, it is realized with a probability of 1/6. □

Online Square Packing With Rotation

Shahin Kamali* Pooya Nikbakht†

Abstract

In the square packing problem, the goal is to place a multi-set of square items of different side lengths in $(0, 1]$ into a minimum number of square bins of uniform side length 1. In the online setting, the multi-set of items forms a sequence that is revealed in an online and sequential manner. When an item is revealed, an online algorithm has to place it into a square bin without any prior knowledge of the forthcoming items. All existing results for the online square packing are restricted to the case when square items are placed orthogonally to the square bins. In this paper, we provide an algorithm with an asymptotic competitive ratio of 2.306 when squares are allowed to be rotated.

1 Introduction

An instance of the *square packing problem* is defined with a multiset of *squares-items* of different sizes in the range $(0, 1]$. The goal is to place these squares into a minimum number of unit *square bins* in a way that two squares-items placed in the same square bin do not intersect. The problem is a generalization of the classical bin packing problem into two dimensions. As such, we sometimes refer to the squares-items simply as *items* and square bins as *bins*. A square item can be recognized by the length of its side, which we refer to as the *size* of the square.

In the offline setting, all square items are given in advance, and the algorithm can process them as a whole before placing any item into a bin. In particular, the algorithm can sort squares in the decreasing order of their sizes, and this comes in handy in many settings. In the online setting, the multi-set of items forms a *sequence* that is revealed in an online and sequential manner. Items are revealed one by one; when an item is revealed, an online algorithm has to place it into a square bin without any prior knowledge of forthcoming items. The decisions of algorithms are irrevocable.

Square packing has many applications in practice. One prominent application is cutting stuck where bins represent stocks (e.g., wood boards) and items are requests to squares of specific sizes. When requests arrive, an algorithm has to 'cut' the stock to provide the pieces that match the requests. This cutting process is equivalent to 'placing' items into bins. The goal of cutting stock is to minimize the number of stocks which also matches a square packing goal. We note that in many practical applications, requests arrive in an online manner and the stock should be cut without priory knowledge about the future requests. It is needless to say that the cutting process is irrevocable which gives an inherently online nature to these applications of square packing.

There has been a rich body of research around square packing. To our knowledge, all existing results except for our previous work on offline square packing [17], assume that squares are not allowed to rotate, that is, sides of square items should be parallel to the square bins. While this assumption makes combinatorial analysis of the problem easier, might cause a higher cost. As an example, consider an instance of the problem formed by n items of size 0.36. If we do not allow rotation, any bin can include at most 4 items, which results a total cost of $n/4$ for any algorithm. Allowing rotation, however, 5 items fit in each bin and we can reduce the cost to $n/5$ (see Figure 1). As a result, the number of required bins is decreased by $n/20$ which is a notable saving in practice (e.g., for cutting stock applications).

In this paper, we consider the online square packing problem with rotation which is defined as follows.

Definition 1 *In the online square packing with rotation, the input is a multi-set of squares (items) with sizes $S = \{x_1, ..., x_n\}$ where $0 < x_i \leq 1$. The goal is to pack these squares into the minimum number of squares of unit size (bins). In the offline setting, S is available since the beginning. In the online setting, S is revealed as a sequence $\sigma = \langle x_1, x_2, \ldots, x_n \rangle$ which is revealed in an online manner. At time-step t, the value of x_t is revealed and an online algorithm has to place a square of*

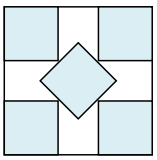


Figure 1: If all input items have length $\frac{\sqrt{2}}{2\sqrt{2}+1} \approx 0.35$, rotation allows packing 5 items per bin instead of 4.

*Department of Computer Science, University of Manitoba, Winnipeg, Canada, shahin.kamali@umanitoba.ca

†Department of Computer Science, University of Manitoba, Winnipeg, Canada, nikbakhp@myumanitoba.ca

size x ($1 \le t \le n$). The decisions of the algorithm are irrevocable and are made without knowing the values of $x_{t'}$ for $t' > t$.

The asymptotic competitive/approximation ratio is the standard method for analyzing packing problems. We say an algorithm ALG has a competitive ratio of c if there exists a constant c_0 such that, for all n and for all input sequences σ of length n, we have $\text{ALG}(\sigma) \le c \cdot \text{OPT}(\sigma) + c_0$ where $A(\sigma)$ and $Opt(\sigma)$ denote the costs of A and Opt for processing σ, respectively, and are both arbitrarily large.

1.1 Related work

The 1-dimensional bin packing has been studied extensively in both offline and online settings (see, e.g., [14, 13, 6, 7]). In the 1-dimensional setting, each item has simply a size $\in (0, 1]$, and each bin has a capacity of 1. In the offline setting, the problem is NP-hard, and the best existing result is an algorithm that opens at most $\text{OPT}(\sigma) + O(\log \text{OPT})$ bins for placing a sequence σ [16]. In the online setting, the best existing algorithm has a competitive ratio of 1.578 [2]. Meanwhile, it is known that no online algorithm has a competitive ratio better than 1.54278 [3].

There are many ways to extend bin packing to higher dimensions (see [5] for a survey). Orthogonal packing square items into square bins is perhaps the most straightforward extension. In the offline setting, the problem is known to be NP-hard [18]. Bansal et al. [4] provided an APTAS for this problem (indeed, for the more general d-dimensional cubes). In the online setting, the best existing upper and lower bounds have improved a few times [19, 10]. The best existing algorithm has a competitive ratio of 2.0885 [8] while the best existing lower bound is 1.6707 [15].

In a previous work [17], we studied offline square packing when the rotation of items is allowed. We showed that, while the problem remains NP-hard, it admits an APTAS under a resource-augmentation setting, where bins of the algorithm have size $1 + \alpha$, for an arbitrary small $\alpha > 0$, while bins of OPT have size 1. If resource augmentation is not allowed, the problem is likely $\exists\mathbb{R}$-hard [17, 1].

1.2 Contribution

We consider the online setting and provide an online algorithm that achieves a competitive ratio of 2.306 for the square packing problem. Our algorithm is based on classifying squares based on their sizes and placing squares of similar sizes tightly, possibly using rotations, in the same bins. This approach is previously used to introduce different families of *Harmonic algorithms* for bin packing in both one dimension and higher dimensions. The presence of rotations, however, make our classification and analysis different from the previous work.

2 Square-Rotate algorithm

In this section, we introduce our square packing algorithm called SQUARE-ROTATE.

2.1 Item classification

Similar to the Harmonic family of algorithms, we classify squares by the size of their side lengths (which we simply refer to as the *size* of the items).

SQUARE-ROTATE packs squares of each class separately from other classes. In total, there are 13 classes of squares (having more classes is possible but leads to none to small improvement of the final result). Square items with sizes in the range $(0, 0.1752]$ are in class 13. We refer to class 13 as the *tiny class*, and items that belong to this class are referred to as tiny items. We refer to items that belong to class $i \in [1, 12]$ as *regular items*. For each class $i \in [1, 12]$, the range of items in the class is specified as $(x_i, x_{i-1}]$ (for convenience, we define $x_0 = 1$). The values of x_i's are defined in a way that a certain number of items, denoted by S_i, of class i can fit in the same bin. The specific range of item sizes for each class $i \in [1, 12]$ and values of S_i is derived from the best-known or optimal results [12] on the congruent square packing problem [11], which asks for the minimum size $c(j)$ of a square that can contain j unit-sized squares. A scaling argument, where the container size is fixed to be 1, gives values of $u(j)$ when the goal is to pack j identical squares of maximum size $u(j)$ into a unit square.

In Figure 2, it is specified how S_i items of the largest size in class i can fit into a square bin. Therefore, the scaled best-known values of $u(j)$ for $1 \le j \le 36$ can be derived from the figure. These scaled numbers give the specific ranges that we used for classifying items as follows: Items of class 1 have sizes in the range $(1/2, 1]$, and we have $x_1 = 1/2$. Note that exactly $S_1 = 1$ item of class 1 can fit in the same bin. For $i \in [2, 12]$, S_i is the number of items of size x_{i-1} that fit in the same bin. For example, for $i = 2$, we have $S_2 = 4$ because $x_1 = 1/2$, and 4 items of size $1/2$ fit in the same bin. Moreover, x_i is defined as the largest value so that $S_i + 1$ items of size x_i cannot fit in the same bin. For example, we have $x_2 = 0.3694$ because according to Figure 2, $S_2 + 1 = 5$ squares of size 0.3694 cannot fit in the same bin.

The respective range of items for each class, as well as the values of S_i, are presented in Table 1. For example, a square belongs to class 1, 2, or 12 if its side size is in the interval $(0.5, 1]$, $(0.3694, .5]$, or $(0.1752, 0.1779]$, respectively.

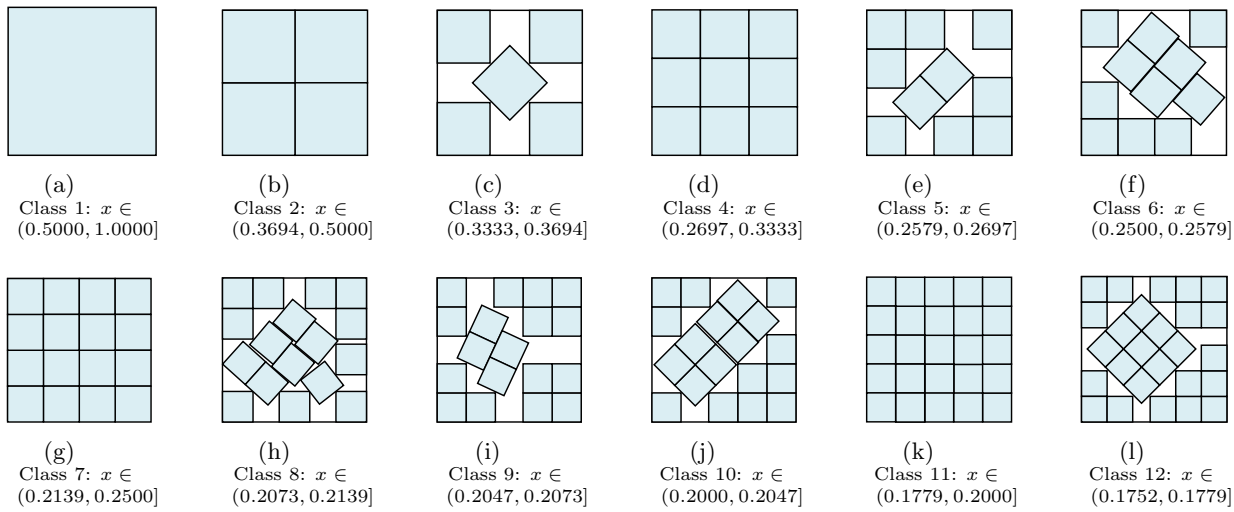


Figure 2: Placement of regular square items of class $i \in [1, 12]$ in their respective bin. It is possible to pack i square items of class i into a single square bin [12].

Class	Side length x	S_i	Occupied Area	Weight	Density
1	$(0.5000, 1.0000]$	1	$> 1(0.250)=0.250$	1	< 4.000
2	$(0.3694, 0.5000]$	4	$> 4(0.136)=0.544$	1/4	< 1.838
3	$(0.3333, 0.3694]$	5	$> 5(0.111)=0.555$	1/5	< 1.801
4	$(0.2697, 0.3333]$	9	$> 9(0.072)=0.648$	1/9	< 1.543
5	$(0.2579, 0.2697]$	10	$> 10(0.066)=0.660$	1/10	< 1.515
6	$(0.2500, 0.2579]$	11	$> 11(0.062)=0.682$	1/11	< 1.466
7	$(0.2139, 0.2500]$	16	$> 16(0.045)=0.720$	1/16	< 1.388
8	$(0.2073, 0.2139]$	17	$> 17(0.042)=0.714$	1/17	< 1.400
9	$(0.2047, 0.2073]$	18	$> 18(0.041)=0.738$	1/18	< 1.355
10	$(0.2000, 0.2047]$	19	$> 19(0.040)=0.760$	1/19	< 1.315
11	$(0.1779, 0.2000]$	25	$> 25(0.031)=0.775$	1/20	< 1.290
12	$(0.1752, 0.1779]$	26	$> 26(0.030)=0.780$	1/26	< 1.282
13	$(0, 0.1752]$		> 0.702	$1.425x^2$	≈ 1.425

Table 1: A summary of item classification and details on item weights and densities, as used in the definition and analysis of Square-Rotate.

2.2 Packing regular items

For each class i ($1 \leq i \leq 12$), the algorithm has at most one active bin of type i. When a bin of type i is opened, it is declared as the active bin of the class, and S_i square *spots*, each having a size equal to the largest square of class i, are reserved in the bin. Upon the arrival of an item of class i, it is placed in one of the S_i spots of the active bin. If all these spots are occupied by previous items, a new bin of type i is opened. This ensures that all bins of type i, except potentially the current active bin, include S_i items.

2.3 Packing tiny items

For the last class, i.e., tiny items, the algorithm uses a different approach, proposed by Epstein and van Stee [9]. Briefly, it maintains at most one active bin for placing tiny items. When a bin is opened for these items, the algorithm reserves four square spots of size 1/2, i.e., the four squares of class 2 in Figure 2b. These square spots are used as bins for placing tiny items. Then, the algorithm chooses one of the innermost sub-bin squares that has enough space for the arrived item and repeats the procedure for the selected sub-bin until it cannot split any of the innermost sub-squares into four new ones with enough space for the item. At this step, the item is placed in one of those smallest sub-bins. When the next item arrives, if there is a sub-bin of the smallest possible size in which the item can fit, the algorithm places the item in that spot. Otherwise, the algorithm finds the smallest sub-bin that can fit the item and repeats the previous procedure to split it into the smaller sub-bins to reach an appropriate spot for the item. If no sub-bin with enough empty space is available in the bin, the algorithm closes the current bin and opens a new empty active bin for the item and applies the whole process from the beginning (see [9], for details). Note that the algorithm does not rotate any of the tiny items to pack them. Epstein and van Stee proved the following result, which we will use in our analysis later.

Lemma 1 *[9] Consider the square packing problem (without rotation) in which all items are of size at most $1/M$ for some integer $M \geq 2$. There is an online algorithm (as described above) that creates a packing in which all bins, except possibly one, have an occupied area of size at least $(M^2 - 1)/(M + 1)^2$.*

2.4 Algorithm's analysis

In this section, we prove a competitive ratio of at most 2.306 for our algorithm. We use a *weighting function argument*. For each item of size x, we define a weight $w(x)$ for the item and prove that: (1) the total weight

of square items in each bin of the algorithm, except potentially a constant number of them, is at least 1, and (2) the total weight of items in each bin of an optimal packing is at most 2.306. If $w(\sigma)$ denote the total weight of items in an input sequence σ, then (1) implies that the number of bins opened by the algorithm is at most $w(\sigma) + c$, for some constant value of c, and (2) implies that the number of bins in an optimal packing is at least $w(\sigma)/2.306$. Therefore, the (asymptotic) competitive ratio of the algorithm would be at most 2.306.

Recall that all bins opened for squares of class i ($1 \leq i \leq 12$), except possibly the last active bin, include S_i squares. We define the weight of items of class i to be $1/S_i$. This way, the total weight of items in bins opened for all squares of classes 1 to 12, except possibly 12 of them (the last bin from each class), is exactly 1. Therefore, (1) holds for bins opened for regular items.

We define the weight of a tiny square of size x as $x^2/0.701 (= 1.425x^2)$. All tiny items are of size at most 0.1752. Therefore, by Lemma 1, the occupied area of all bins opened for tiny items (except possibly one of them) will be at least 0.701. This implies their total weight is at least $0.701/0.701 = 1$.

Table 1 gives a summary of the weights of items in different classes. From the above argument, we conclude the following lemma.

Lemma 2 *The total weight of squares in each bin opened by* Square-Rotate*, except possibly a constant number of them, is at least 1.*

The following lemma provides an upper bound for the total weight of items in a bin of the optimal offline algorithm (Opt). The proof works by case analysis and can be found in the appendix.

Lemma 3 *The total weight of items in every bin of* Opt *is less than* 2.306*.*

Provided with the above two lemmas, we can derive the main result of this section.

Theorem 4 *There is an online algorithm for the square packing problem with rotation problem which achieves a competitive ratio of at most* 2.306*.*

Proof. For an input σ, let $SR(\sigma)$ and $OPT(\sigma)$ denote the cost of Square-Rotate and Opt, respectively. Let $w(\sigma)$ denote the total weight of items of σ. Lemmas 2 implies that $SR(\sigma) \leq w(\sigma) + c$, where c is a constant independent of the length of σ. Meanwhile, Lemma 3 implies that $Opt(\sigma) \geq w(\sigma)/2.306$. From these inequalities, we conclude $SR(\sigma) \leq 2.306\ OPT(\sigma) + c$, which proves an upper bound 2.306 for the competitive ratio of Square-Rotate. □

References

[1] M. Abrahamsen, T. Miltzow, and N. Seiferth. Framework for ∃r-completeness of two-dimensional packing problems. In *2020 IEEE 61st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1014–1021. IEEE, 2020.

[2] J. Balogh, J. Békési, G. Dósa, L. Epstein, and A. Levin. A new and improved algorithm for online bin packing. In *26th Annual European Symposium on Algorithms (ESA)*, volume 112 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:14. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018.

[3] J. Balogh, J. Békési, G. Dósa, L. Epstein, and A. Levin. A new lower bound for classic online bin packing. *CoRR*, abs/1807.05554, 2018.

[4] N. Bansal, J. R. Correa, C. Kenyon, and M. Sviridenko. Bin packing in multiple dimensions: Inapproximability results and approximation schemes. *Mathematics of Operations Research*, 31(1):31–49, 2006.

[5] H. I. Christensen, A. Khan, S. Pokutta, and P. Tetali. Approximation and online algorithms for multidimensional bin packing: A survey. *Computer Science Review*, 24:63–79, 2017.

[6] E. G. Coffman, M. R. Garey, and D. S. Johnson. Approximation algorithms for bin packing: A survey. In *Approximation algorithms for NP-hard Problems*. PWS Publishing Co., 1997.

[7] E. G. Coffman Jr., J. Csirik, G. Galambos, S. Martello, and D. Vigo. Bin packing approximation algorithms: survey and classification. In P. M. Pardalos, D.-Z. Du, and R. L. Graham, editors, *Handbook of Combinatorial Optimization*, pages 455–531. Springer, 2013.

[8] L. Epstein and L. Mualem. Online bin packing of squares and cubes. *arXiv preprint arXiv:2105.08763*, 2021.

[9] L. Epstein and R. van Stee. Optimal online bounded space multidimensional packing. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 214–223, 2004.

[10] L. Epstein and R. van Stee. Online square and cube packing. *Acta Informatica*, 41(9):595–606, 2005.

[11] P. Erdős and R. L. Graham. On packing squares with equal squares. *Journal of Combinatorial Theory, Series A*, 19:119–123, 1975.

[12] E. Friedman. Packing unit squares in squares: A survey and new results. *The Electronic Journal of Combinatorics*, pages 1–24, 2000.

[13] G. Galambos and G. J. Woeginger. Online bin packing - a restricted survey. *ZOR*, 42:25–45, 1995.

[14] M. R. Garey and D. S. Johnson. Approximation algorithms for bin packing problems - a survey. In G. Ausiello and M. Lucertini, editors, *Analysis and Design of Algorithms in Combinatorial Optimization*, pages 147–172. Springer, New York, 1981.

[15] S. Heydrich and R. van Stee. Beating the harmonic lower bound for online bin packing. In *43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

[16] R. Hoberg and T. Rothvoss. A logarithmic additive integrality gap for bin packing. In *proc. the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2616–2625. SIAM, 2017.

[17] S. Kamali and P. Nikbakht. Cutting stock with rotation: Packing square items into square bins. In *International Conference on Combinatorial Optimization and Applications*, pages 530–544. Springer, 2020.

[18] J. Y. T. Leung, T. W. Tam, C. S. Wong, G. H. Young, and F. Y. L. Chin. Packing squares into a square. *Journal of Parallel and Distributed Computing*, 10(3):271–275, 1990.

[19] S. S. Seiden and R. van Stee. New bounds for multidimensional packing. *Algorithmica*, 36(3):261–293, 2003.

Appendix

Lemma 3 *The total weight of items in every bin of* Opt *is less than* 2.306.

Proof. We first define the *density* of an item of size x as the ratio between its weight and area, i.e., $w(x)/x^2$. Given the lower bound for the size of each square belonging to class i ($1 \leq i \leq 12$), we can calculate a lower bound for the density of each item in the class. For tiny items, the density is simply $1.425x^2/x^2 = 1.425$. Densities for all classes have been reported in Table 1.

Defining densities comes handy in the following case analysis to prove that the total weight of items in any bin B of an optimal packing is at most 2.306.

Case 1: First, assume there is no item of class 1 in B. Since the density of items of other classes are less than 1.838, even if B is fully packed with items of the largest density, the total weight of items cannot be more than 1.838 which is less than 2.306.

Case 2: In the second case, we assume there is one item x of class 1 (note that no two items of class 1 fit in the bin). Without loss of generality, we assume the size of x is $1/2 + \epsilon$, where ϵ is a small real value greater than zero. Clearly, a larger size for x does not increase the total weight of other items in B because it would leave less space to occupy more items in the bin (while the weight of x stays 1). Next, we consider all possible cases in which we have some items of class 2 and 3 together with x in B. As presented in Table 2, there will be 14 sub-cases to analyze. To see how we reach these 14 sub-cases, first note that it is not possible to accommodate 4 or more items of class 2 in addition to x in B (i.e., a total number of 5 or more items from these classes 1 and 2). This is because no five items with size larger than 0.3694 can fit in B [12]. A similar argument shows that we cannot have 6 or more items from classes 1, 2, and 3 together in a bin, otherwise we could accommodate 6 identical squares of size strictly larger than 0.3333 which is a contradiction to the fact that no six items of size larger than 0.3333 can fit in the same bin [12]. We can conclude that the 14 sub-cases summarized in Table 2 cover all possibilities for items of the first three classes in Case 2.

C1	C2	C3	total weight of items of class $c \leq 3$ (W)	area occupied by items of class $c \leq 3$ (A)	remaining remaining Area ($A_r = 1 - A$)	total weight in remaining area ($W_r = A_r \times 1.543$)	total weight of items in the bin ($W + W_r$)
1	0	0	1.00	> 0.250	< 0.750	< 1.157	< 2.157
1	0	1	1.20	> 0.361	< 0.639	< 0.986	< 2.186
1	0	2	1.40	> 0.472	< 0.528	< 0.815	< 2.215
1	0	3	1.60	> 0.583	< 0.417	< 0.644	< 2.244
1	0	4	1.80	> 0.694	< 0.306	< 0.472	< 2.272
1	1	0	1.25	> 0.386	< 0.614	< 0.948	< 2.198
1	1	1	1.45	> 0.497	< 0.503	< 0.776	< 2.226
1	1	2	1.65	> 0.608	< 0.392	< 0.605	< 2.255
1	1	3	1.85	> 0.719	< 0.281	< 0.434	< 2.284
1	2	0	1.50	> 0.522	< 0.478	< 0.738	< 2.238
1	2	1	1.70	> 0.633	< 0.367	< 0.566	< 2.266
1	2	2	1.90	> 0.744	< 0.256	< 0.395	< 2.295
1	3	0	1.75	> 0.658	< 0.342	< 0.528	< 2.278
1	3	1	1.95	> 0.769	< 0.231	< 0.356	< 2.306

Table 2: Fourteen possible cases in which we have a combination of items of class 2 (C2) and 3 (C3) together with one item x of class 1 (C1) in a single bin B. Here, "sum of weights (W)" and "sum of areas (A)" indicate, respectively, the total weight and area of items of the first three classes in B. "Remaining area" is the area left in the bin that is used for packing items of class 4 or higher. "Weight of items in the remaining area" is an upper bound for the total weight of items of class 4 or higher in B (these items have density no more than 1.543). Finally, "the total weight of items in the bin" indicate the sum of weights of all items (from all classes) in B.

According to Table 1, the density of items belonging to class i ($4 \leq i \leq 12$) as well as tiny items is at most 1.543 (which is the density of class-4 items). Using a similar argument made for Case 1, we suppose that, after placing a certain number of items of class 2 and 3 beside x in B, in each sub-case, we are able to completely fill the remaining empty space of B with the items of the maximum density 1.543. This makes us able to calculate an upper bound for the maximum total weight of items in B for each of the sub-cases. The resulting bounds for each sub-case can be found in the last column of Table 2, where the maximum upper bound among all sub-cases is 2.306, which happens when we have one item of class 1 in B together with 3 items of class 2 and one item of class 3.

As a result, in both Case 1 and Case 2, the total weight of items in B cannot be more than 2.306. □

Preprocessing Imprecise Points for Furthest Distance Queries

Vahideh Keikha* Sepehr Moradi† Ali Mohades‡

Abstract

Given is a set of regions in $\mathbb{R}^d$, in the region-based uncertainty model. We show here how to preprocess these regions so that if one point per region is specified with precise coordinates, in the query phase, the diameter of the query points can be computed faster than the scratch. We discuss a $(1+\epsilon)$-approximation algorithm with running time $O(\frac{n}{\epsilon^d})$ for answering such queries, for a set of pairwise disjoint unit balls, after spending $O(n \log n + \frac{n}{\epsilon^d})$ time for preprocessing.

1 Introduction

It is a common assumption in different areas of computational geometry that the input is a set of points. However, we usually face the problems at which the input data are not precise due to several resources, namely including bounded precision of measuring devices, rounding errors in calculations, etc. In some cases, we already know in which *region* each particular point would lie, however, the exact locations of the points are still unknown. One then may assume such a region as an *imprecise point*, that could be a disk, rectangle, line segment, etc.

In recent years, frequent analysis of uncertain data has been actively researched in computation modeling of real-world problems. There are numerous exact and approximation algorithms for processing uncertain data. Designing an exact algorithm that works for all possible instances may produce a big data structure and may need time-consuming calculations. As a result, these algorithms demand much time and space as their inputs are indeed superset compared to the standard algorithm, where the input is a set of points. There have been efforts to resolve this problem by careful analysis of the worst-case or the best-case behavior of the input instances, however, all cases are likely to happen. In some other scenarios, producing the lower and upper bound for feasible solutions may suffice. Another standpoint is preprocessing uncertain data for speeding-up the further computations on precise instances received later.

*Institute of Computer Science, Czech Academy of Sciences, Czech Republic keikha@cs.cas.cz

†Department of Computer Science, Amirkabir University of Technology semolation@gmail.com

‡Department of Computer Science, Amirkabir University of Technology mohades@aut.ac.ir

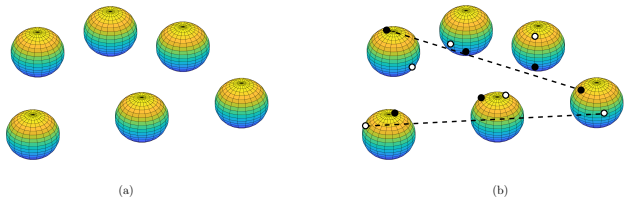


Figure 1: Problem definition: (a) A set D of 6 imprecise points modeled as unit balls, (b) and the diameter of two different realizations of D.

In this paper, we address the diameter problem in this setting.

Problem 1 (Diameter Query) *Let $D = \{d_1, \ldots, d_n\}$ be a set of balls in $\mathbb{R}^d$. For a given query point set $\mathcal{Q} = \{p_1, \ldots, p_n\}$, where $p_i \in d_i$, our objective is to find the diameter of $\mathcal{Q}$ in $o(dn^2)$ time, after preprocessing. We call the set $\mathcal{Q}$ a* realization *of D; see Figure 1.*

Related work. The region-based model of imprecision was introduced and extensively studied by Löffler and van Kreveld. Several models are already established for processing a set of imprecise points for (possibly) speeding up the sorting problem [17], computing an arbitrary triangulation [9, 18], the Delaunay triangulation [2, 11], and the convex hull of a query set in $\mathbb{R}^2$ [4]. In particular, it is previously shown that for a set of imprecise points modeled as convex polygons, with totally $O(n)$ vertices, an arbitrary triangulation of a query set with one point in each region can be computed in $O(n)$ time after spending $O(n \log n)$ for the preprocessing [18]. The same problem was also studied in [11] at which the same results also hold for computing a Delaunay triangulation. See also [2, 9]. For a set of imprecise points in the plane modeled as lines, it is shown that the preprocessing does not speed up the closest pair computation, the Delaunay triangulation, and the sorting problem on the realizations received later [4], where they lie on given lines known in advance. However, in the same paper, it is shown that preprocessing a set of lines, can speed up computing the convex hull of the points (on those lines) received later.

The diameter of a set of points is the maximum pairwise distance between the points in the set. Computing the diameter of a set of points has a long history. It is shown that computing the diameter in $\mathbb{R}^d$ needs

$\Omega(n \log n)$ time in any algebraic decision tree, by a reduction from the set disjointedness problem. But the best-known algorithm for computing the diameter in $\mathbb{R}^d$ takes $O(\min\{nd \log n, n^2 \log^{s-2} n, n^2 d^{s-2}\})$ time, where $s \approx 2.376$ [5, 13]. However, this running time can be improved for specific values of $d > 2$ [5]. For $d = 2$, near linear approximation algorithm exists for the diameter problem [8]. We refer the reader to [5, 12] for a complete list of algorithms for the diameter problem in different dimensions. We note that the diameter problem has extensively used as a black box in database queries. See, e.g., [6].

Contribution. We show there exists a $(1 + \epsilon)$-approximation for approximating the diameter queries on pairwise disjoint unit disks, that takes $O(\frac{n}{\epsilon^d})$ time, after spending $O(n \log n + \frac{n}{\epsilon^d})$ time for preprocessing (Sec 3.2).

2 Preliminaries

For a set $\mathcal{Q}$ of points in $\mathbb{R}^d$, let $diam(S)$ denote the diameter of $\mathcal{Q}$. In the following, we recall the definitions we use from the literature.

Let $G = (S, E)$ be a geometric graph on $\mathcal{Q}$. Let $d_G(p, q)$ denote the geodesic distance between any pair $p, q \in \mathcal{Q}$, that is defined as the length of the shortest path between these two points in G. The graph G is called a *t-spanner* for some $t \geq 1$, if for any two points $p, q \in \mathcal{Q}$ we have $d_G(p, q) \leq t|pq|$, where $|pq|$ is the Euclidean distance between p and q. The parameter t is refereed to as the stretch factor.

2.1 Well Separated Pair Decomposition (WSPD)

Let $\mathcal{Q}$ be a set of points in $\mathbb{R}^d$. Two sets $P_i, Q_i \subseteq \mathcal{Q}$ of points are s-well separated if they can be enclosed within balls of radius r such that the closest distance between these balls is at least sr. An s-well separated pair decomposition (s-WSPD) of size m for a point set $\mathcal{Q}$ is a set of *s-well-separated* pairs of subsets $\{(P_1, Q_1), \ldots, (P_m, Q_m)\}$, where each $(P_i, Q_i) \subset 2^{\mathcal{Q}} \times 2^{\mathcal{Q}}$, and for any pair of points $p, q \in \mathcal{Q}\, (p \neq q)$ there is a unique index i for which $p \in P_i, q \in Q_i$. See Figure 2. Moreover, for any s-well separated pair (P_i, Q_i), for a sufficiently large separation parameter s, we have approximately equal distances between any two points, where one lies in P_i and the other lies in Q_i. Furthermore, each pair P_i, Q_i has two *representatives* $p_i \in P_i$ and $q_i \in Q_i$, where p_i, q_i gives an approximation for distances between any two points from P_i to Q_i. It has been shown that an s-WSPD of $O(s^d n)$ pairs can be computed in $O(n \log n + s^d n)$ [3].

We start stating our results with a related question: Given is a set D of imprecise points modeled by disjoint unit balls. The question is determining whether there exists an spanner G for an arbitrary realization $\mathcal{Q}$ of D such that for any other realization $\mathcal{Q}'$ of D where $\mathcal{Q}' \neq \mathcal{Q}$, the graph G remains an spanner for $\mathcal{Q}'$ with the same stretch factor. Abam et al. in [1] answered this question positively by introducing a method for computing the WSPD with respect to a separation ratio s' on the center of the balls. They proved that the computed WSPD remains valid for any realization $\mathcal{Q}$ of D, where the separation ratio s of the WSPD on instances is calculated according to $s = \frac{s'-2}{2}$. Then they create a spanner that is valid for any realization.

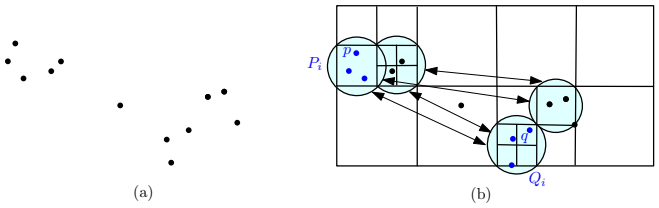


Figure 2: (a) Illustration of a point set and (b) a well-separated pair decomposition of it with 4 pairs (computed from the quadtree [14]).

Let D be a set of n unit balls, and let s' be the separation ratio, for which one make a WSPD on the centers. The following result exist:

Lemma 1 (Lemma 1 [1]) *Let D be a set of disjoint unit balls, and let $\{(P_i, Q_i) | 1 \leq i \leq m\}$ be a WSPD for the set $\{c_1, \ldots, c_n\}$ as the centers of the balls in D, with respect to $s' = 2s + 2$. Let $\mathcal{Q} = \{p_1, \ldots, p_n\}$ be a set of points, where $p_j \in D_j$, for $1 \leq j \leq n$. For $1 \leq i \leq m$, let $P'_i = \{p_j | c_j \in P_i\}$ and $Q'_i = \{p_j | c_j \in Q_i\}$. Then $\{(P'_i, Q'_i) | 1 \leq i \leq m\}$ is a WSPD for $\mathcal{Q}$ with respect to s.*

2.2 Point Set Diameter Approximation

A $(1 + \epsilon)$-approximation algorithm already exists for approximating the diameter of a point set in $\mathbb{R}^d$ using WSPD [7] (Chapter 3, Lemma 3.14). Let $\mathcal{Q}$ be a set of n points in $\mathbb{R}^d$. For a given $0 \leq \epsilon \leq 1$, the objective is computing a pair $p_u, p_v \in \mathcal{Q}$ such that $\frac{diam(\mathcal{Q})}{1+\epsilon} \leq \|p_u p_v\| \leq diam(\mathcal{Q})$. In the following, we recall the algorithm.

Algorithm: Approximating the Diameter [7]. We first compute an s-WSPD for a point set $\mathcal{Q}$, where $s = 4/\epsilon$. For each WSPD pair (P_i, Q_i), associate a pair of points as representative points $p_u \in P_i, p_v \in Q_i$ and compute the distance between them. See Figure 3. We then remember the maximum distance among all representative points and return it in the end. This would give a $(1 + \epsilon)$-approximation for the diameter of $\mathcal{Q}$ [7]. It is shown that the running time of this algorithm is $O(n \log n + s^d n)$ as the WSPD needs to be computed. Although, the number of candidate pairs realizing the diameter is only $O(s^d n)$.

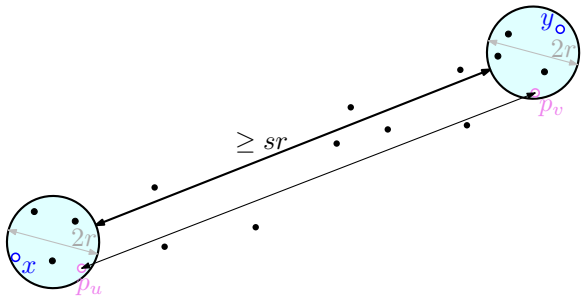


Figure 3: Diameter approximation using WSPD. The points x, y determine the diameter, and p_u, p_v approximate the diameter within a factor $1+\epsilon$.

Our method is in fact a combination of Abam et al. [1] technique in the preprocessing phase for computing a *persistent* WSPD which is computed on the disk centers, and the diameter approximation algorithm [7] in the query phase, using the computed WSPD in the preprocessing step.

3 Computing the Diameter after Preprocessing

Observe that for any set D of n imprecise points in $\mathbb{R}^2$, there is no preprocessing with running time $o(n \log n)$ on D to speed-up answering the diameter queries on D to $o(n \log n)$ time. That is because all such problems simulate the point set case, and it is known that there is a lower bound $\Omega(n \log n)$ for the diameter problem in any algebraic decision tree [12]. In other words, if the preprocessing takes $o(n \log n)$ time, this would result in an $o(n \log n)$ time algorithm for the diameter of a set of points in the plane. As another variant consider the input regions as a set of parallel lines in the plane. If the $2D$ points are sorted in just a single direction, one cannot compute their diameter in less than $\Omega(n \log n)$ time [15]. Because, if D is a set of parallel lines, e.g., along the x-axis, we can only anticipate the x-order of the points (received later), from which the lower bound follows.

For a set of unit disks in $\mathbb{R}^2$, the diameter query problem can be solved in $O(n)$ time after spending $O(n \log n)$ time for preprocessing. Let D be a set of unit disks in the plane. It is known that the Delaunay triangulation of a realization of D, as the query set, can be computed in $O(n)$ time after spending $O(n \log n)$ time for preprocessing. Hence, the convex hull can be extracted in $O(n)$ time. Having the convex hull, the diameter also can be computed in $O(n)$ time, as all the antipodal pairs of a convex polygon can be computed in $O(n)$ time and the diameter is among them [16].

In $\mathbb{R}^d$, we focus on approximation algorithms. An obvious constant factor approximation algorithm for this problem is the smallest enclosing ball (SEB) of a set of points that approximates the diameter of the points within a factor $\frac{\sqrt{3}}{3}$. Consider the configuration at which four points on the boundary of the SEB forms an equilateral triangular-based pyramid, and the side length of each triangular face determines the diameter. If one translates any pair of these points on the boundary of the SEB, to get closer, the diameter enlarges between at least one pair. Hence, a $(\frac{\sqrt{3}}{3}+\epsilon)$-approximation of the diameter of any set of points in $\mathbb{R}^d$ can be computed in $O(dnz/\epsilon^{O(1)})$ time by using the randomized $O(1+\epsilon)$-approximation algorithm in [10] for computing the SEB of a set of points in $\mathbb{R}^d$, at which z is a parameter depending on the input [1]. Next, we discuss a $(1+\epsilon)$-approximation algorithm.

3.1 Preprocessing

In this section, our objective is to preprocess the regions, such that when the exact position of points are given, one can compute and return an approximation of the diameter in $o(n \log n)$ time. To solve the problem, in our algorithm we use the aforementioned technique that returns a $(1+\epsilon)$-approximation of diameter using WSPD on the point set in $O(n/\epsilon^2)$. However, we need to compute the WSPD on the point set according to a specific separation factor $s = \frac{4}{\epsilon}$, but it takes $O(n \log n + s^2 n)$ time and makes the algorithm useless. Therefore, in the case where the input is a set of disks, we use Abam et al. [1] technique for computing a WSPD on the center points of the disks with the separation parameter $s' = 2s + 2$, which has been proved that would be valid for any realization according to separation factor s. Hence, we do not need to compute the WSPD on each instance, and the WSPD is computed only once in the preprocessing phase.

Lemma 2 *Let $\{(A_i, B_i) | 1 \leq i \leq m\}$ be a WSPD on the set $\{c_1, \ldots, c_n\}$ of given disjoint unit disks with respect to $s' = \frac{8}{\epsilon} + 2$. Let $\mathcal{Q} = \{p_1, \ldots, p_n\}$ be a set of points, where $p_j \in D_j$, for $1 \leq j \leq n$. For $1 \leq i \leq m$, let $A'_i = \{p_j | c_j \in A_i\}$ and $B'_i = \{p_j | c_j \in B_i\}$. Then $\{(A'_i, B'_i) | 1 \leq i \leq m\}$ is a WSPD for $\mathcal{Q}$ with respect to $s = \frac{4}{\epsilon}$.*

Proof. According to Lemma 1 the $\{(A'_i, B'_i) | 1 \leq i \leq m\}$ would be a valid WSPD for any instance with respect to separate factor $s = \frac{s'-2}{2}$. We assumed the separate factor of the WSPD on the center points is $s' = \frac{8}{\epsilon} + 2$, so we have: $s = \frac{s'-2}{2} = \frac{(\frac{8}{\epsilon}+2)-2}{2} = \frac{\frac{8}{\epsilon}}{2} = \frac{4}{\epsilon}$. □

3.2 Query Phase

Now, when we are given a realization of the balls, we wish to compute a $(1+\epsilon)$-approximation of the diameter in $O(n/\epsilon^d)$ time. It follows from Lemma 2 that we can

[1] We note this is the best-known algorithm for computing the SEB, that has a linear dependency on d.

do this by having a WSPD on the center points with respect to separation factor $s = \frac{4}{\epsilon}$. In addition, our WSPD is valid for any other realization.

Theorem 3 *For any given set $D = \{D_1, \ldots, D_n\}$ of n imprecise points modeled as the same size balls which are pairwise disjoint, a $(1+\epsilon)$-approximation of the diameter of a realization $\mathcal{Q}$ of D can be computed in $O(\frac{n}{\epsilon^d})$ time, after $O(n \log n + \frac{n}{\epsilon^d})$ preprocessing time.*

Proof. Let $s = \frac{4}{\epsilon}$ and $s' = 2s + 2 = \frac{8}{\epsilon} + 2$ and $\mathcal{Q} = \{p_1, \ldots, p_n\}$ be the set of precise points. Let $\{(A_i, B_i)\}_{i=1,\ldots,m}$ be an s'-WSPD for the center points, of size $m = O(s'^2 n)$, and let $A'_i = \{p_j | c_j \in A_i\}$, $B'_i = \{p_j | c_j \in B_i\}$. It follows from Lemma 1 that $\{(A'_i, B'_i) | 1 \leq i \leq m\}$ is a WSPD for $\mathcal{Q}$ with respect to separation parameter $s = \frac{4}{\epsilon}$.

Then, we associate one point to each set as the representative point, let $p_a \in A'_i$ and $p_b \in B'_i$ be the representative points of the sets A'_i and B'_i respectively. From the presented approximation algorithm for the diameter in [7], by calculating the distance between representative points of each pair and computing the maximum distance among all, we have a $(1+\epsilon)$-approximation of the diameter of any realization in $O(s'^d n)$ time. □

4 Discussion

The main open question is finding an algorithm for the general version, as our approach cannot be extended to overlapping disks or disks of arbitrary size. We note that since the WSPD gives the nearest neighbour pairs in a set of points, our results give also an $O(1)$-approximation algorithm for the nearest neighbour query for a realization of D in high dimensions.

Acknowledgement The authors would like to thank an anonymous reviewer for pointing out the efficient exact algorithm for the 2-d case. V.K is Supported by the Czech Science Foundation, grant number GJ19-06792Y, and by institutional support RVO: 67985807.

References

[1] M. A. Abam, P. Carmi, M. Farshi, and M. Smid. On the power of the semi-separated pair decomposition. *Comput. Geom.*, 46(6):631–639, 2013.

[2] K. Buchin, M. Löffler, P. Morin, and W. Mulzer. Delaunay triangulation of imprecise points simplified and extended. In *WADS*, pages 131–143. Springer, 2009.

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

[4] E. Ezra and W. Mulzer. Convex hull of points lying on lines in $o(n \log n)$ time after preprocessing. *Comput. Geom.*, 46(4):417–434, 2013.

[5] D. V. Finocchiaro and M. Pellegrini. On computing the diameter of a point set in high dimensional euclidean space. *Theoretical Computer Science*, 287(2):501–514, 2002.

[6] X. Guo, X. Yang, D. Chen, and C. Chen. Diameter-aware extreme group queries. *IEEE Access*, 6:58687–58701, 2018.

[7] S. Har-Peled. *Geometric approximation algorithms*. Number 173. American Mathematical Soc., 2011.

[8] J. Hong, Z. Wang, and W. Niu. A simple approximation algorithm for the diameter of a set of points in an euclidean plane. *Plos one*, 14(2):e0211201, 2019.

[9] V. Keikha, A. Mohades, and M. D. Monfared. On the triangulation of non-fat imprecise points. In *CCCG*, pages 114–121, 2016.

[10] A. Krivošija. Probabilistic smallest enclosing ball in high dimensions. *Technical report for Collaborative Research Center SFB 876 Providing Information by Resource-Constrained Data Analysis*, page 13, 2019.

[11] M. Löffler and J. Snoeyink. Delaunay triangulation of imprecise points in linear time after preprocessing. *Comput. Geom.*, 43(3):234–242, 2010.

[12] G. Malandain and J.-D. Boissonnat. Computing the diameter of a point set. *Internat. J. Comput. Geom. Appl.*, 12(06):489–509, 2002.

[13] F. P. Preparata and M. I. Shamos. Computational Geometry: an Introduction. pages 95–149. Springer, 1985.

[14] H. Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys (CSUR)*, 16(2):187–260, 1984.

[15] R. Seidel. A method for proving lower bounds for certain geometric problems. In *Machine Intelligence and Pattern Recognition*, volume 2, pages 319–334. Elsevier, 1985.

[16] M. I. Shamos. Computational geometry. Ph.D. Thesis. 1978.

[17] I. van der Hoog, I. Kostitsyna, M. Löffler, and B. Speckmann. Preprocessing ambiguous imprecise points. *arXiv preprint arXiv:1903.08280*, 2019.

[18] M. Van Kreveld, M. Löffler, and J. S. Mitchell. Preprocessing imprecise points and splitting triangulations. *SIAM J. Comput.*, 39(7):2990–3000, 2010.

The final problem I worked on with Saeed Mehrabi*

Prosenjit Bose†

The last problem that I worked on with Saeed was a problem called the "Boundary Labeling problem". The essence of the problem is the following: Given a set of objects inside a rectangle, we wish to label these objects where the labels are located outside the rectangle. The goal is to draw a curve from each object inside of the rectangle to the label outside the rectangle. There are many variants of the problem, such as restricting which sides of the rectangle the curve is allowed to intersect, forbidding curves from intersecting or restricting the type of curve that is allowed. There are also some optimization questions such as whether one can minimize the total length of the curves. I will review the results that Saeed and I and our co-authors obtained for this family of problems.

*In memory of Saeed Mehrabi, former PC of ICCG, who passed away unexcpectedly.

†School of Computer Science, Carleton University, Ottawa, Canada, jit@scs.carleton.ca

Index of Authors