38th European Workshop on Computational Geometry

March 14-16, 2022, Perugia, Italy

https://eurocg2022.unipg.it/

Booklet of abstracts

Preface

The 38th European Workshop on Computational Geometry (EuroCG 2022) was held on March 14-16 in Perugia, Italy. EuroCG is an annual, informal workshop whose goal is to provide a forum for scientists to meet, present their work, interact, and establish collaborations, in order to promote research in the field of Computational Geometry, within Europe and beyond.

After two on-line only editions due to the COVID pandemic, this year it was again possible to have an in-person meeting, although we kept the possibility, for people who could not travel, to attend the workshop and to give talks on-line. We had 91 registered participants for the in-person workshop and 104 for the on-line event.

Concerning the scientific program, we received 76 submissions, which underwent a limited refereeing process by the program committee in order to ensure some minimal standards and to check for plausibility. We selected 68 submissions for presentation at the workshop. EuroCG does not have formally published proceedings; therefore, we expect most of the results outlined here to be also submitted to peer-reviewed conferences and/or journals. This book of abstracts, available through the EuroCG 2022 web site, should be regarded as a collection of preprints. In addition to the 68 contributed talks, this book contains abstracts of the invited lectures. The invited speakers were originally Leila De Floriani, Michael Hoffmann, and Maurizio Patrignani. Unfortunately, shortly before the workshop Leila had to cancel her participation. Thus we asked Stefan Felsner and he kindly accepted to replace Leila. We thank Stefan for having accepted to give an invited talk even with a very short notice.

Many thanks to all authors and to the members of the program committee and all external reviewers for their insightful comments. We also thank the organizing committee members: Carla Binucci, Luca Grilli, Giacomo Ortali, Alessandra Tappini and Tommaso Piselli. Finally, we are very grateful for the generous support of our sponsors: ITS Umbria Smart Academy, Slope, Ordine degli Ingegneri Provincia di Perugia, Bazuel. We would like to thank also the Algorithms journal, which sponsored the *Best Student Presentation Award*. The prize was voted by the EuroCG 2022 attendees to recognize the effort of young researchers to present their work clearly and elegantly. The ex aequo winners were Paul Jungeblut, for the presentation of paper "The Complexity of the Hausdorff Distance" and Soeren Nickel for the presentation of paper "Removing Popular Faces in Curve Arrangements by Inserting one more Curve". Congratulations to Paul and Soeren!

During the business meeting Rodrigo Silveira presented the 2023 edition of EuroCG, which will take place in Barcelona, Spain. A single bid was presented for 2024 by Michael Bekos and Charis Papadopoulos and, as a consequence, EuroCG 2024 will take place in Ioannina, Greece.

Looking forward to seeing you all in Barcelona!

March 2022,

Emilio Di Giacomo and Fabrizio Montecchiani

Program Committee

Peyman Afshani, Aarhus University, DK
Dominique Attali, Université Grenoble Alpes, FR
Martin Balko, Charles University, CZ
Michael Bekos, Universität Tübingen, DE
Sergio Cabello, University of Ljubljana, SI
Éric Colin de Verdière, CNRS, LIGM, Marne-la-Vallée, FR
Sabine Cornelsen, Universität Konstanz, DE
Giordano Da Lozzo, Università degli Studi di Roma Tre, IT
Emilio Di Giacomo, CO-CHAIR, University of Perugia, IT
Stephane Durocher, University of Manitoba, CA
William Evans, The University of British Columbia, CA
Sándor Fekete, TU Braunschweig, DE
Silvia Fernández-Merchant, California State University Northridge, US
Joachim Gudmundsson, The University of Sydney, AU
Philipp Kindermann, Universität Trier, DE
Irina Kostitsyna, TU Eindhoven, NL
Stefan Langerman, Université Libre de Bruxelles, BE
Maarten Löffler, Utrecht University, NL
Fabrizio Montecchiani, CO-CHAIR, University of Perugia, IT
Dömötör Pálvölgyi, Eötvös Loránd University, HU
Chrysanthi Raftopoulou, National Technical University of Athens, GR
Christiane Schmidt, Linköping University, SE
Monique Teillaud, INRIA, Loria, FR
Torsten Ueckerdt, Karlsruhe Institute of Technology, DE
Ryuhei Uehara, Japan Advanced Institute of Science and Technology, JP
Meirav Zehavi, Ben-Gurion University, IL

Organizing Committee

Carla Binucci, University of Perugia, IT
Emilio Di Giacomo, CO-CHAIR, University of Perugia, IT
Luca Grilli, University of Perugia, IT
Fabrizio Montecchiani, CO-CHAIR, University of Perugia, IT
Giacomo Ortali, University of Perugia, IT
Alessandra Tappini, University of Perugia, IT

Table of Contents

Invited talks

Contributed papers

Arc diagrams, flip distances, and Hamiltonian triangulations.

Michael Hoffmann

ETH Zurich

Abstract

How many edge flips are needed to transform one combinatorial triangulation into another? How many spine crossings are needed in a topological book embedding of a planar graph? Under what conditions can a given planar graph be transformed into a Hamiltonian planar graph? In this talk I will discuss connections between these three questions along with some partial answers and open problems concerning specific variations of them.

Contact Representations of Planar Graphs - Combinatorial Structure and Algorithm X

Stefan Felsner

Technische Universität Berlin

Abstract

The main player in this talk is algorithm X. In its various disguises this algorithm can be used to compute contact representations of planar graphs with squares, homothetic triangles, pentagons and other shapes. To this end the algorithm exploits combinatorial structures such as transversal structures, Schnyder woods, and five-color forests. We survey what is known about algorithm X and what remains mysterious.

Visual Analysis of Large Networks - Strategies and Challenges

Maurizio Patrignani

Roma Tre University

Abstract

The visual analysis of large networks plays a critical role in today's applications and its relevance is doomed to grow in the next future. The incredibly-vast amount of networked data produced by real-world applications poses unprecedented challenges that standard graph-visualization paradigms seem unprepared to address. Indeed, although several approaches have been proposed, an effective solution appears still elusive. In this talk we will discuss the requirements of such an analysis and we will review the most promising techniques and tools that have been proposed so far to cope with such new challenges. It will be apparent that, in addition to efficiency, visual analytics tools must also be based on a combination of abstraction and modeling. Further, the goal of producing readable representations of the inner structure of large networks has lead to formalizing several combinatorial problems that need to be addressed. Therefore, this domain has both a deep impact on applications and an intriguing theoretical appeal. We will review recent results and highlight the main open questions in this domain.

The Complexity of the Hausdorff Distance

Paul Jungeblut[1], Linda Kleist[2], and Tillmann Miltzow[3]

1 Karlsruhe Institute of Technology, Germany
paul.jungeblut@kit.edu
2 Technische Universität Braunschweig, Germany
kleist@ibr.cs.tu-bs.de
3 Utrecht University, The Netherlands
t.miltzow@uu.nl

Abstract

We determine the computational complexity of computing the Hausdorff distance. Specifically, we show that the decision problem of whether the Hausdorff distance of two semi-algebraic sets is bounded by a given threshold is complete for the complexity class $\forall\exists_{<}\mathbb{R}$. This implies that the problem is NP-, co-NP-, $\exists\mathbb{R}$- and $\forall\mathbb{R}$-hard.

Related Version Full version available on arXiv: 2112.04343.

1 Introduction

The question of 'how similar are two given objects' occurs in numerous settings. A typical tool to quantify their similarity is the Hausdorff distance. Two sets have a small Hausdorff distance if every point of one set is close to some point of the other set and vice versa. The Hausdorff distance appears in many branches of science. To illustrate the range of use cases, we consider two examples. For illustrations consider Figure 1. In mathematics, the Hausdorff distance provides a metric on sets and henceforth also a topology. This topology can be used to discuss continuous transformations of one set to another [7]. In computer vision and geographical information science, the Hausdorff distance is used to measure the similarity between spacial objects [17, 18], for example the quality of quadrangulations of complex 3D models [20]. In this paper, we study the computational complexity of the Hausdorff distance from a theoretical perspective.

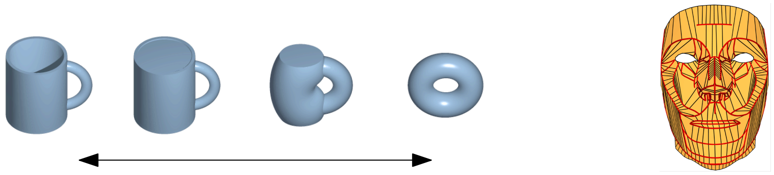


Figure 1 Left: Continuous deformation of a cup into a doughnut [10]. Right: Quadrangulation of a smooth surface used for rendering [20].

Definition. The *directed Hausdorff distance* between a non-empty set $A \subseteq \mathbb{R}^n$ and a non-empty set $B \subseteq \mathbb{R}^n$ is defined as

$$\vec{d}_{\mathrm{H}}(A,B) := \sup_{a\in A}\inf_{b\in B} \|a-b\|.$$

The directed Hausdorff distance between A and B can be interpreted as the smallest value $\varepsilon \geq 0$ such that the (closed) ε-neighborhood of B contains A. Hence, it nicely captures the intuition of how much B has to be blown up to contain A. Note that $\vec{d}_{\mathrm{H}}(A,B)$ and $\vec{d}_{\mathrm{H}}(B,A)$

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

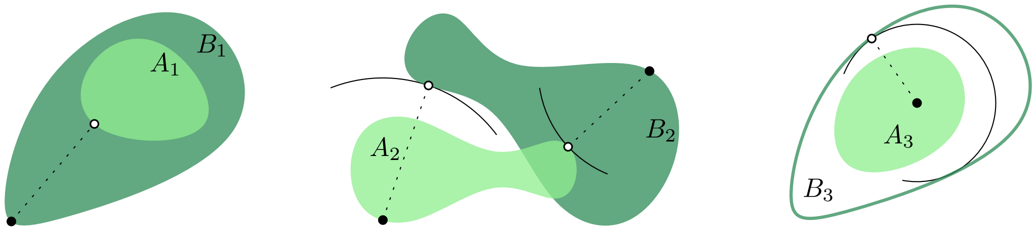


Figure 2 How similar are these sets?

must not be equal. For an example, consider Fig. 2; while $A_1 \subset B_1$ and thus $\vec{d}_{\mathrm{H}}(A_1, B_1) = 0$, it holds that $\vec{d}_{\mathrm{H}}(B_1, A_1) > 0$. The (undirected) *Hausdorff distance* is symmetric and defined as $d_{\mathrm{H}}(A, B) := \max\{\vec{d}_{\mathrm{H}}(A, B), \vec{d}_{\mathrm{H}}(B, A)\}$. In this paper, we investigate the *computational complexity* of deciding whether the Hausdorff distance of two sets is at most a given threshold.

Semi-Algebraic Sets. The algorithmic complexity of the Hausdorff distance clearly depends on the type of the considered sets. If we are given the sets in a way that we cannot even decide if they are empty, it seems near impossible to compute their Hausdorff distance. However, if the sets consists of finitely many points, their Hausdorff distance can be easily computed by checking all pairs of points. In practice, we are often somewhere between those two extreme situations. For instance, the sets could be a collection of disks in the plane or cubic splines, describing a surface in three dimensions, see also Fig. 3.

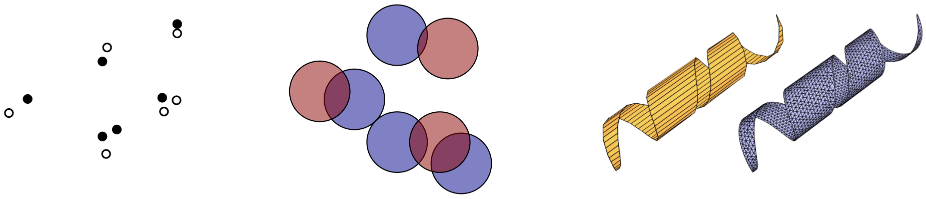


Figure 3 The Hausdorff distance can appear in simpler or more complicated settings. Left: Two finite point sets (black and white) in the plane. Middle: Two sets of blue and red disks in the plane. Right: Two surfaces in 3-space with different meshes, image taken from [20].

In this paper, we focus on semi-algebraic sets, i.e., sets that can be described by polynomial inequalities. Formally, a semi-algebraic set is the finite union of basic semi-algebraic sets. A basic semi-algebraic set S is specified by two families of polynomials $\mathcal{P}$ and $\mathcal{Q}$ such that

$$S = \Big\{x \in \mathbb{R}^n \;\Big|\; \bigwedge_{P \in \mathcal{P}} P(x) \leq 0 \wedge \bigwedge_{Q \in \mathcal{Q}} Q(x) < 0\Big\}.$$

Semi-algebraic sets cover clearly the vast majority of practical cases and finding efficient algorithms for this problem would be a tremendous contribution. Simultaneously, when considering smooth sets, one is quickly in the situation that one needs to deal with polynomials anyway. So the step to general semi-algebraic sets is not a very big one.

General Decision Algorithm. We consider a situation where we are given two semi-algebraic sets A and B as well as a threshold t; for simplicity, we assume here (only in this paragraph) that A and B are closed. The statement $\vec{d}_{\mathrm{H}}(A, B) \leq t$ can be encoded into a logical sentence

$$\forall a \in A \,.\, \exists b \in B : \|a - b\|^2 \leq t^2,$$

where $\|x\|$ denotes the Euclidean norm of the vector x. We can decide the truth of this sentence by employing sophisticated algorithms from real algebraic geometry that can deal with two blocks of quantifiers [5, Chapter 14]. These algorithms are impractical for all non-trivial instances. Our main result roughly states that in general there is little hope for an improvement. To state this formally, we continue by defining suitable complexity classes.

Algorithmic Complexity. Let φ be a *quantifier-free formula in the first-order theory of the reals*, i.e., a formula formed over the alphabet $\Sigma = \{0, 1, +, \cdot, =, \leq, <, \vee, \wedge, \neg\}$ together with symbols for the variables. The Universal Existential Theory of the Reals (UETR) asks to decide the truth value of a sentence

$$\Phi := \forall X \in \mathbb{R}^n \,.\, \exists Y \in \mathbb{R}^m : \varphi(X, Y).$$

An instance of UETR belongs to Strict-UETR if the corresponding formula φ is over the alphabet $\Sigma = \{0, 1, +, \cdot, <, \vee, \wedge\}$, i.e., if every atom is a strict inequality and there are no negations. The complexity classes $\forall\exists\mathbb{R}$ and $\forall\exists_<\mathbb{R}$ contain all decision problems for which there exists a polynomial-time many-one reduction to UETR and Strict-UETR, respectively. We propose to pronounce the complexity classe $\forall\exists\mathbb{R}$ as 'UER' or 'forall exists R' and $\forall\exists_<\mathbb{R}$ as 'Strict-UER' or 'strict forall exists R'. To the best of our knowledge, $\forall\exists\mathbb{R}$ was first introduced by Bürgisser and Cucker [9, Section 9] under the name $\mathsf{BP}^0(\forall\exists)$ (in the constant-free Blum-Shub-Smale-model [6]). The notation $\forall\exists\mathbb{R}$ arised later in [13] extending the notation from Schaefer and Števankovič [19]. The sister class co-$\forall\exists_<\mathbb{R} = \exists\forall_\leq\mathbb{R}$ was first studied by D'Costa, Lefaucheux, Neumann, Ouaknine and Worrel [12].

Problem and Results. We now have all ingredients to state our problem and main results. Let $\Phi_A(X)$ and $\Phi_B(X)$ be two quantifier-free formulas defining the semi-algebraic sets $A = \{x \in \mathbb{R}^n \mid \Phi_A(x)\}$ and $B = \{x \in \mathbb{R}^n \mid \Phi_B(x)\}$, and let $t \in \mathbb{Q}$ be a rational number. The Hausdorff problem asks whether $d_\mathrm{H}(A, B) \leq t$. Here the dimension n of the ambient space of A and B is part of the input (there is a polynomial-time algorithm for every fixed n, see the related work in Section 2). The computational complexity of this problem was posed as an open question by Dobbins, Kleist, Miltzow and Rzążewski [13].

▶ **Theorem 1.1.** *The* Hausdorff *problem is $\forall\exists_<\mathbb{R}$-complete.*

Note that prior to our result, it was not even known if computing the Hausdorff distance was NP-hard. As $\forall\exists_<\mathbb{R}$ contains, NP, co-NP, $\exists\mathbb{R}$ and $\forall\mathbb{R}$, we also get hardness for all of these complexity classes. In the proof of $\forall\exists_<\mathbb{R}$-hardness for Theorem 1.1, we create instances with some additional properties. In particular, we can guarantee a gap, i.e., the Hausdorff distance is either below the threshold t or at least $t \cdot 2^{2^{\Omega(d)}}$, where d denotes the number of variables of Φ_A and Φ_B. Thus our result also rules out approximation algorithms.

▶ **Corollary 1.2.** *Let A and B be two semi-algebraic sets in $\mathbb{R}^d$ and $f(d) = 2^{2^{o(d)}}$. Then there is no polynomial time $f(d)$-approximation algorithm to compute $d_\mathrm{H}(A, B)$, unless* $\mathsf{P} = \forall\exists_<\mathbb{R}$.

2 Related Work

This section reviews previous work concerning two directions. First, we discuss the complexity of computing the Hausdorff distance for special sets. Afterwards, we investigate previous work on the complexity class $\forall\exists\mathbb{R}$.

Computing the Hausdorff Distance. The notion of the Hausdorff distance was introduced by Felix Hausdorff in 1914 [16]. Most of the early works focused on the Hausdorff distance for finite point sets. For a set of n points and a set of m points in any fixed dimension, the Hausdorff distance can be easily computed by checking all pairs, i.e., in time $O(mn)$. In the plane, this can be improved to $O((n+m)\log(m+n))$ by using Voronoi diagrams [1]. In fact, this method can be extended to sets consisting of pairwise non-crossing line segments in the plane, e.g., simple polygons and polygonal chains fulfill this property. If the polygons are additionally convex, their Hausdorff distance can even be computed in linear time [4].

More generally, the Hausdorff distance can be computed in polynomial time whenever the two sets can be described by a simplicial complex of fixed dimension. Based on the PhD thesis of Godau [15], Alt et al. [2, Theorem 3.3] show how to compute the directed Hausdorff distance between two sets in $\mathbb{R}^d$ consisting of n and m k-dimensional simplices in time $O(nm^{k+2})$ (assuming d is constant). Using a Las Vegas algorithm for computing the vertices of the lower envelope, similar ideas yield an approach with randomized expected time in $O(nm^{k+\varepsilon})$ for $k > 1$ and every $\varepsilon > 0$ [2, Theorem 3.4]. They additionally present algorithms with better randomized expected running times for sets of triangles in $\mathbb{R}^3$ and point sets in $\mathbb{R}^d$.

Given two semi-algebraic sets $A, B \subseteq \mathbb{R}^n$, the HAUSDORFF problem can be encoded as a sentence of the form $\forall X \in \mathbb{R}^n \,.\, \exists Y \in \mathbb{R}^n : \varphi(X, Y)$ with $\Theta(n)$ variables, where φ is quantifier-free. Such a sentence can be decided in time roughly equal to $(sd)^{O(n^2)}$ [5, Theorem 14.14] where d denotes the maximum degree of any polynomial in φ and s denotes the number of atoms.

In other contexts the two sets are allowed to undergo certain transformations (e.g. translations) such that the Hausdorff distance is minimized [8]. See Alt [3] for a survey.

Universal Existential Theory of the Reals. As mentioned above, the complexity class $\forall\exists\mathbb{R}$ was first studied by Bürgisser and Cucker who prove complexity results for many decision problems involving circuits [9]. Dobbins, Kleist, Miltzow, and Rzążewski [14, 13] consider $\forall\exists\mathbb{R}$ in the context of area-universality of graphs. A plane graph is *area-universal* if for every assignment of reals to the inner faces of a plane graph, there exists a straight-line drawing such that the area of each inner face equals the assigned number. Dobbins et al. conjecture that the decision problem whether a given plane graph is area-universal is complete for $\forall\exists\mathbb{R}$. They support this conjecture by proving hardness for several related notions [13]. Additionally, for future research directions, they present a number of candidates for potentially $\forall\exists\mathbb{R}$-hard problems. Among them, they stated a question motivating this paper as an open problem, namely whether the HAUSDORFF problem is $\forall\exists\mathbb{R}$-complete. The other candidates exhibit intrinsic connections to imprecision, robustness and extendability.

The sister class $\exists\forall\mathbb{R}$ was recently investigated by D'Costa et al. [12]. They show that it is $\exists\forall_{<}\mathbb{R}$-complete to decide for a given rational matrix A and a compact semi-algebraic set $K \subseteq R^n$, whether there exists a starting point $x \in K$ such that $x_n := A^n x$ is contained in K for all $n \in N$.

3 Techniques and Proof Overview

In this section, we present the general idea behind the hardness reduction for the HAUSDORFF problem. The goal is to convey the intuition and to motivate the technical intermediate steps needed. The sketched reduction is oversimplified and thus neither in polynomial time nor fully correct. We point out both of these issues and give first ideas on how to solve them.

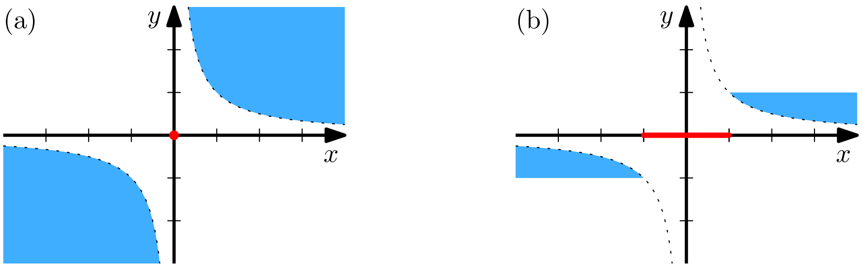


Figure 4 Consider the formula $\forall X \in \mathbb{R} \,.\, \exists Y \in \mathbb{R} : XY > 1$. (a) Each point $(x, y) \in \mathbb{R}^2$ in the blue open region satisfies $xy > 1$. Only for $x = 0$ (in red) no suitable $y \in \mathbb{R}$ exists. (b) Restricting the range of Y to $[-1, 1]$, then for all $x \in [-1, 1]$ (in red) no y with $xy > 1$ exists.

Let $\Phi := \forall X \in \mathbb{R}^n \,.\, \exists Y \in \mathbb{R}^m : \varphi(X, Y)$ be a Strict-UETR instance. We define two sets

$$\begin{aligned} A &:= \{x \in \mathbb{R}^n \mid \exists Y \in \mathbb{R}^m : \varphi(x, Y)\} \quad \text{and} \\ B &:= \mathbb{R}^n \end{aligned}$$

and ask whether $d_{\mathrm{H}}(A, B) = 0$. If Φ is true, then $A = \mathbb{R}^n$ and we have $d_{\mathrm{H}}(A, B) = 0$ because both sets are equal. Otherwise, if Φ is false, then there exists some $x \in \mathbb{R}^n$ for which there is no $y \in \mathbb{R}^m$ satisfying $\varphi(x, y)$ and we conclude that $A \neq \mathbb{R}^n$. In general we call the set of all $x \in \mathbb{R}^n$ for which there is no $y \in \mathbb{R}^m$ satisfying $\varphi(x, y)$ the *counterexamples* $\perp(\Phi)$ of Φ. One might hope that $\perp(\Phi) \neq \emptyset$ is enough to obtain $d_{\mathrm{H}}(A, B) > 0$, but this is not the case. To this end, consider the formula $\Psi := \forall X \in \mathbb{R} \,.\, \exists Y \in \mathbb{R} : XY > 0$, which is false. The set $\perp(\Psi) = \{0\}$ contains only a single element, so we have $A = \mathbb{R} \setminus \{0\}$ and $B = \mathbb{R}$. However, their Hausdorff distance also evaluates to $d_{\mathrm{H}}(A, B) = 0$. We conclude that above reduction does not (yet completely) work, because it maps a yes- and a no-instances of Strict-UETR to a yes-instance of Hausdorff.

We solve this issue by blowing up the set of counterexamples. Specifically, Theorem 12 (in the full version) establishes a polynomial time algorithm to transform a Strict-UETR instance Φ into an equivalent formula Φ' such that the set of counterexamples is either empty (if Φ' is true) or contains an open ball of positive radius (if Φ' is false). The radius of the ball serves as a lower bound on the Hausdorff distance $d_{\mathrm{H}}(A, B)$. Thus a reduction starting with Φ' is correct. As a key tool for this step, we restrict the variable ranges from $\mathbb{R}^n$ and $\mathbb{R}^m$ to small and compact intervals. Fig. 4 presents an an example on how such a range restriction may enlarge the set of counterexamples from a single point to an interval. We think that the special property of blown up counterexamples can prove useful in future reductions to show $\forall\exists_<\mathbb{R}$-hardness of other problems because it makes handling the no-instances easier.

A further challenge is given by the definition of the sets A and B. While the description complexity of B depends only on n, the definition of A contains an existential quantifier. This is troublesome because our definition of the Hausdorff problem requires quantifier-free formulas as its input, and in general there is no equivalent quantifier-free formula of polynomial length which describes the set A [11]. We overcome this issue by taking the existentially quantified variables as additional dimensions into account. We cannot know their precise values for each possible choice of the universally quantified variables. But by scaling them to a tiny range, their influence on the Hausdorff distance becomes negligible. Therefore instead of the above we work with sets similar to

$$\begin{aligned} A &:= \{(x, y) \in [-1, 1]^n \times [-\varepsilon, \varepsilon]^m \mid \varphi(x, y)\} \quad \text{and} \\ B &:= [-1, 1]^n \times \{0\}^m \end{aligned}$$

for some tiny value ε depending on the radius r (of the ball contained in the counterexamples). This definition of A and B introduces the new issue that even if Φ is true, the Hausdorff distance $d_{\mathrm{H}}(A,B)$ might be strictly positive. However, we manage to identify a threshold t, such that $d_{\mathrm{H}}(A,B) \leq t$ if and only if Φ is true. This completes the proof of $\forall\exists_<\mathbb{R}$-hardness.

$\forall\exists_<\mathbb{R}$-membership is shown by formulating the Hausdorff problems as an equivalent Strict-UETR instance (see Section 6 of the full version).

References

1 Helmut Alt, Bernd Behrends, and Johannes Blömer. Approximate Matching of Polygonal Shapes. *Annals of Mathematics and Artificial Intelligence*, 13(3):251–265, 1995. doi:10.1007/BF01530830.

2 Helmut Alt, Peter Braß, Michael Godau, Christian Knauer, and Carola Wenk. Computing the Hausdorff Distance of Geometric Patterns and Shapes. In Boris Aronov, Saugata Basu, János Pach, and Micha Sharir, editors, *Discrete and Computational Geometry: The Goodman-Pollack Festschrift*, volume 25 of *Algorithms and Combinatorics*, pages 65–76. Springer, 2003. doi:10.1007/978-3-642-55566-4_4.

3 Helmut Alt and Leonidas J. Guibas. Discrete Geometric Shapes: Matching, Interpolation, and Approximation. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 121–153. Elsevier, 2000. doi:B978-044482537-7/50004-8.

4 Mikhail J. Atallah. A Linear Time Algorithm for the Hausdorff Distance Between Convex Polygons. *Information Processing Letters*, 17(4):207–209, 1983. doi:10.1016/0020-0190(83)90042-X.

5 Sauguta Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer, 2006. doi:10.1007/3-540-33099-2.

6 Lenore Blum, Mike Shub, and Steve Smale. On a Theory of Computation and Complexity over the Real Numbers: NP-Completeness, Recursive Functions and Universal Machines. *Bulletin of the American Mathematical Society*, 21:1–46, 1989. doi:10.1090/S0273-0979-1989-15750-9.

7 Glen E. Bredon. *Topology and Geometry*, volume 139 of *Graduate Texts in Mathematics*. Springer Science & Business Media, 1st edition, 2013. doi:10.1007/978-1-4757-6848-0.

8 Karl Bringmann and André Nusser. Translating Hausdorff Is Hard: Fine-Grained Lower Bounds for Hausdorff Distance Under Translation. arXiv preprint, 2021. arXiv:2101.07696.

9 Peter Bürgisser and Felipe Cucker. Exotic Quantifiers, Complexity Classes, and Complete Problems. *Foundations of Computational Mathematics*, 9:135–170, April 2009. doi:10.1007/s10208-007-9006-9.

10 Wiki Community. Homotopy. accessed 2021 November. URL: https://en.wikipedia.org/wiki/Homotopy.

11 James H. Davenport and Joos Heintz. Real Quantifier Elimination is Doubly Exponential. *Journal of Symbolic Computation*, 5(1–2):29–35, 1988. doi:10.1016/S0747-7171(88)80004-X.

12 Julian D'Costa, Engel Lefaucheux, Eike Neumann, Joël Ouaknine, and James Worrel. On the Complexity of the Escape Problem for Linear Dynamical Systems over Compact Semialgebraic Sets. In Filippo Bonchi and Simon J. Puglisi, editors, *International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 202 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 33:1–33:21, 2021. doi:10.4230/LIPIcs.MFCS.2021.33.

13 Michael G. Dobbins, Linda Kleist, Tillmann Miltzow, and Paweł Rzążewski. $\forall\exists\mathbb{R}$-Completeness and Area-Universality. In Andreas Brandstädt, Ekkehard Köhler, and Klaus Meer, editors, *Graph-Theoretic Concepts in Computer Science (WG)*, volume 11159 of *Lecture Notes in Computer Science*, pages 164–175. Springer, 2018. doi:10.1007/978-3-030-00256-5_14.

14 Michael G. Dobbins, Linda Kleist, Tillmann Miltzow, and Paweł Rzążewski. Completeness for the Complexity Class $\forall\exists\mathbb{R}$ and Area-Universality. ArXiv preprint, 2021. arXiv:1712.05142v3.

15 Michael Godau. *On the complexity of measuring the similarity between geometric objects in higher dimensions*. PhD thesis, Freie Universtät Berlin, 1999. doi:10.17169/refubium-7780.

16 Felix Hausdorff. *Grundzüge der Mengenlehre*. Von Veit & Company, 1914.

17 Deng Min, Li Zhilin, and Chen Xiaoyong. Extended Hausdorff distance for spatial objects in GIS. *International Journal of Geographical Information Science*, 21(4):459–475, 2007. doi:10.1080/13658810601073315.

18 William Rucklidge. *Efficient Visual Recognition Using the Hausdorff Distance*, volume 1173 of *Lecture Notes in Computer Science*. Springer, 1996. doi:10.1007/BFb0015091.

19 Marcus Schaefer and Daniel Štefankovič. Fixed Points, Nash Equilibria, and the Existential Theory of the Reals. *Theory of Computing Systems*, 60:172–193, 2017. doi:10.1007/s00224-015-9662-0.

20 Floor Verhoeven, Amir Vaxman, Tim Hoffmann, and Olga Sorkine-Hornung. Dev2PQ: Planar Quadrilateral Strip Remeshing of Developable Surfaces. arXiv preprint, 2021. arXiv:2103.00239.

APX-Hardness of the Minimum Vision Points Problem

Mayank Chaturvedi[1] and Bengt J. Nilsson[2]

1 **Birla Institute of Technology and Science Pilani, Goa Campus, India**
f20170548@goa.bits-pilani.ac.in

2 **Department of Computer Science and Media Technology, Malmö University, Sweden**
bengt.nilsson.TS@mau.se

Abstract

Placing a minimum number of guards on a given watchman route in a polygonal domain is called the *minimum vision points problem.* We prove that finding the minimum number of vision points on a shortest watchman route in a simple polygon is APX-Hard. We then extend the proof to the class of rectilinear polygons having at most three dent orientations.

1 Introduction

The problem of guarding polygonal domains is known as the *Art Gallery Problem.* A *guard* is a point in the domain and the visibility of the guard is defined to be those points that can be reached from the guard by line segments that do not intersect the exterior of the domain. When the domain is a simple polygon, Aggarwal [2] and Lee and Lin [13] independently prove that finding the minimum number of guards is NP-hard; see also O'Rourke [16], this is later strengthened to APX-hardness and $\exists\mathbb{R}$-hardness [1, 5, 10].

Computing the watchman route is another way to solve the guarding problem. A *watchman route* is a closed tour traced by a moving guard who sees the complete polygon while tracing the tour. There exist polynomial time algorithms that compute the shortest watchman route for simple polygons [9, 17, 18].

Surveillance devices that trace the watchman route to guard a polygonal domain may be unable to accurately engage their vision systems continuously and could potentially only do so at discrete points along the tour. Such points are called *vision points.* The optimization problem of finding a minimum number of vision points on a shortest watchman route is denoted the *minimum vision points problem* (MVPP) and is the focus of our current results.

Carlsson *et al.* [7] prove the NP-hardness of finding a minimum number of vision points on a shortest watchman route in a simple polygon. Algorithmic results for special classes of polygons for which the watchman route is a shortest path between two points have also been developed [6, 7]; see also Ghosh and Burdick [11] for results for polygons with holes.

We show that MVPP is APX-hard for simple polygons and extend the result to also hold for rectilinear polygons having three dent orientations [8, 14, 15].

2 A Reduction for Simple Polygons

We make a gap preserving reduction [3] from MAX2SAT(3) to MVPP in simple polygons, where MAX2SAT(3) is the following problem.

MAX2SAT(3)
INSTANCE: a set of n boolean variables $u_1, \ldots, u_n$ and a set of m clauses $c_1, \ldots, c_m$,

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

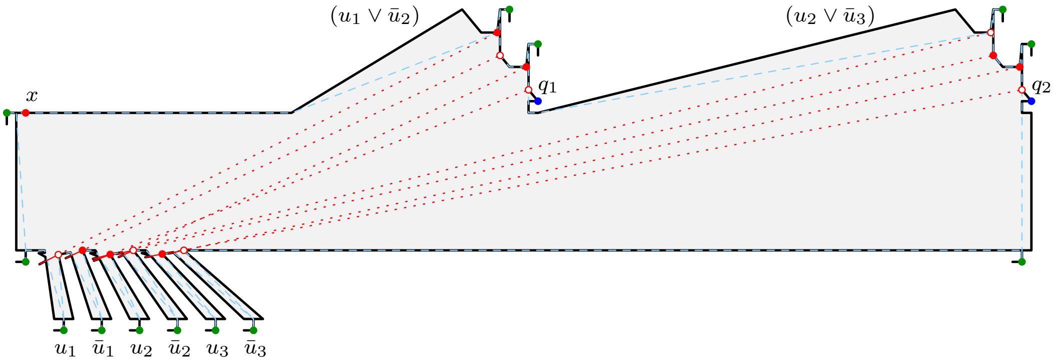


Figure 1 The corresponding MVPP instance for the MAX2SAT(3) instance $(u_1 \vee \bar{u}_2), (u_2 \vee \bar{u}_3)$.

each consisting of a disjunction of exactly two distinct literals formed from the n variables such that each variable occurs at most three times in the clauses.
SOLUTION: an assignment to the variables that satisfies the largest number of clauses.

We can assume that no variable occurs only non-negated or negated in the clauses, otherwise we simply assign it the appropriate truth value to satisfy those clauses it is contained in. Since a variable then occurs two or three times in the clauses, there is one version, non-negated or negated, that occurs exactly once. We call this the *lone literal* of the variable. We also assume that at least one variable occurs three times in the clauses and without loss of generality that u_1 is such a variable with $\bar{u}_1$ as the lone literal in clause c_1. This will be used later to argue the structure of a canonical set of vision points.

▶ **Lemma 2.1.** *For an instance of* MAX2SAT(3) *having n variables and m clauses, such that M of them are satisfiable, it holds that: 1. $n \leq m$, 2. $M \geq 3m/4$, and 3. there is a solution of size at least M such that any unsatisfied clause consists only of lone literals.*

Proof. Claim 1. holds since each variable appears as at least two literals in the clauses.

For Claim 2., a random assignment will satisfy each clause with probability $3/4$ so in total $3m/4$ clauses are satisfied in expectation. Therefore, least one assignment must exist having at least $3m/4$ clauses satisfied.

For Claim 3., any unsatisfied clause that contains a non-lone literal can be satisfied by reversing the assignment of that variable. Only the clause containing the lone literal can become unsatisfied by this so the number of satisfied clauses does not decrease. ◀

We call a solution to a MAX2SAT(3) instance that obeys Claims 1., 2., and 3. in Lemma 2.1, *locally maximal*.

We modify the NP-hardness proof by Aggarwal [2] and Lee and Lin [13] in the same way as Carlsson *et al.* [7]; see Figure 1. The original NP-hardness proof constructs a *reduction polygon* for a MAX2SAT(3) instance consisting of a base rectangle with clause gadgets along the upper segment and variable gadgets along the lower segment. Each clause gadget c_k is a structure with two *chimneys* corresponding to the literals in the clause having a designated point q_k that is seen using two guards if and only if the clause is satisfied. Each variable gadget consists of two *wells*, visible from the point x, one corresponding to the literal u_i and the other corresponding to the literal $\bar{u}_i$. Each variable gadget has a spike t, the only vertices seeing t being its adjacent vertices along the polygon boundary and the two vertices v_i and v_i' marked red in Figure 2. Each literal chimney in a clause c_k has two red vertices

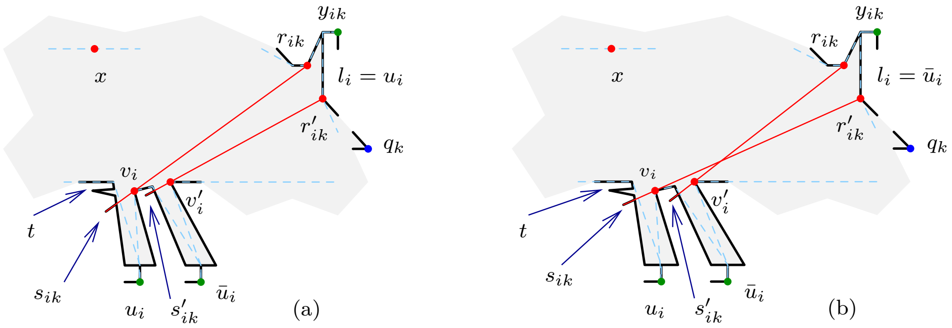


Figure 2 Connecting variables and clauses in the construction.

r_{ik} and r'_{ik} that see it and they are connected to the corresponding variable gadget of u_i. For the clause $c_k = (l_i \vee l_j)$ the literal chimney of l_i is connected to variable gadget u_i by adding spikes s_{ik} and s'_{ik} depending on whether l_i is u_i or $\bar{u}_i$ as illustrated in Figures 2(a) and (b). At least one guard in the clause gadget c_k must be placed at the lower vertex r'_{ik} or r'_{jk} to see both the chimney and the point q_k, the rightmost point of the clause gadget. The chimney is made thin enough so that no point sees the top y_{ik} of more than one chimney; see Figures 1 and 2.

To adapt the construction for MVPP, we extend it by adding two *caves* (thin corridors each with a 90° bend) to each clause structure, one at the top of each chimney, one cave is added at the bottom of each well structure, and three extra caves, on the top right sides of the well structures and one at x, are also added. This gives us $2n + 2m + 3$ caves. To guard the polygon using vision points, the shortest watchman route must enter each of the caves and thus has a vision point in each cave, these are marked green in Figures 1 and 2. The caves tie down the shortest watchman route to ensure that the route passes the *critical guard points* that are used in the constructions of Aggarwal [2] and Lee and Lin [13]. These critical guard points are marked red in Figures 1 and 2. The polygon and shortest watchman route have polynomial sized descriptions in the size of the MAX2SAT(3) instance.

Disregarding the vision points in the caves, each clause gadget requires at least two more vision points (and at most three) and each variable gadget requires at least one vision point given a vision point at x. Thus, if an assignment to the MAX2SAT(3) instance satisfies M clauses, we can guard the polygon using $V = 2n + 2m + 3 + n + 2M + 3(m - M) + 1 = 3n + 5m - M + 4$ vision points by placing one in each cave, one at x, one on the critical guard point corresponding to the assigned truth value in the variable gadget, one on the matching critical guard point of each literal chimney and, if the clause c_k is not satisfied, one extra vision point at the rightmost of the critical guard points r'_{ik} in the clause structure to see q_k. We call such a placement a *canonical vision point set* and prove that we can assume that any vision point set is canonical. (A canonical vision point set is given in Figure 1 consisting of the cave guards in green, the point x and the subset of red points that have white centers.)

Given a set of vision points, we modify it to be canonical without increasing its size as follows. Clearly each green point must be a vision point otherwise not all caves are seen. We can also assume that point x is a vision point, otherwise each variable gadget must have two further vision points and, since each clause gadget must also have two further vision points, we obtain at least $4n + 4m + 3$ vision points. Without loss of generality, these are the critical guard points v_i and v'_i in the variable gadgets and r'_{ik} and r'_{jk} in each clause gadget $c_k = (l_i \vee l_j)$, giving exactly $4n + 4m + 3$ vision points guarding the polygon. Since u_1 occurs in three clauses and has $\bar{u}_1$ as lone literal in clause c_1, we remove the vision point at v'_1, place it at x, and move the vision point at $r'_{1,1}$ to $r_{1,1}$ if necessary, thus neither decreasing coverage nor increasing the size of the vision point set.

The top point y_{ik} of a clause gadget chimney of l_i in clause c_k sees two connected components of the watchman route that we denote w and w', w containing y_{ik}. The component w contains the chimney's two critical guard points r_{ik} and r'_{ik}, r'_{ik} seeing q_k. If w' contains vision points, we move them to r'_{ik}, if the path from y_{ik} to r_{ik} of w contains vision points, we move them to r_{ik}, and if the path from y_{ik} to r'_{ik} of w contains vision points, we move them to r'_{ik}. If r'_{ik} has a vision point after these moves, we remove all other vision points that see y_{ik} (except the green cave one), otherwise we keep one at r_{ik}. Together with x, this vision point will guard at least as much as the original vision points on w and w' (except for a disregardable portion of the other literal chimney in the clause gadget).

The apex of the spike t in a variable gadget u_i sees three connected components of the watchman route. We denote these by w_1, w_2, and w_3 in increasing order of distance to t and note that w_2 contains critical guard point v_i and w_3 contains v'_i. If w_1 or w_2 have vision points, we move them to v_i and if w_3 has vision points, we move them to v'_i and remove any duplicates from v_i and v'_i. Together with x, these points will guard at least as much as the original vision points on w_1, w_2, and w_3. If both v_i and v'_i have vision points, we remove the one that corresponds to the lone literal in the clause gadget of some clause c_k and place one at r'_{ik} unless point q_k is already seen by the other vision points in the clause gadget. The process described above never adds vision points so the size of a canonical vision point set is no larger than the original set. We state this as a lemma.

▶ **Lemma 2.2.** *Any vision point set on a shortest watchman route in a reduction polygon can be transformed to a canonical vision point set of no larger size than the original set.*

Berman and Karpinski [4] show that it is NP-hard to approximate MAX2SAT(3) by a factor $2012/2011 - \epsilon$, for any $\epsilon > 0$. Assume from the discussion above that we have a polynomial time approximation algorithm for MVPP that produces $V = 3n + 5m - M + 4$ canonical vision points for some value M. We can assume that M corresponds to some locally maximal solution of the MAX2SAT(3) instance for which the optimum is M_{opt}. Given an optimal solution to the MAX2SAT(3) instance, we construct a canonical vision point set in the reduction polygon by assigning vision points according to the truth values in the MAX2SAT(3) solution. Let $V' = 3n + 5m - M_{\text{opt}} + 4$ be the number of vision points placed in this way in the reduction polygon and let V_{opt} be the minimum number of vision points in the reduction polygon. Since $V' \geq V_{\text{opt}}$, $M_{\text{opt}}/M \geq 2012/2011 - \epsilon$, and $m \geq 4$, we have by Lemmata 2.1 and 2.2 the ratio

$$\frac{V}{V_{\text{opt}}} \geq \frac{V}{V'} = \frac{3n + 5m - M + 4}{3n + 5m - M_{\text{opt}} + 4} \geq \frac{9m - M}{9m - M(2012/2011 - \epsilon)} \geq \frac{22121}{22120} - \delta, \qquad (1)$$

for any $\delta > 0$ dependent on ϵ, which proves the APX-hardness of MVPP in simple polygons.

3 The MVPP in Rectilinear Polygons with Three Dents

The concept of *dents* in rectilinear polygons was introduced by Culberson and Reckhow [8] and Motwani *et al.* [14, 15] and they develop algorithms for orthogonal covering problems in rectilinear polygons with restricted number of dent orientations. A *dent* in a rectilinear polygon is simply a boundary edge where both endpoints are reflex. Thus, we identify dents with four different orientations, *north*, *south*, *east*, and *west*; see Figure 3(a).

Monotone rectilinear polygons have dents of one or two (opposite) orientations and for these, optimal linear time algorithms for MVPP exist [6, 7]. We settle the complexity

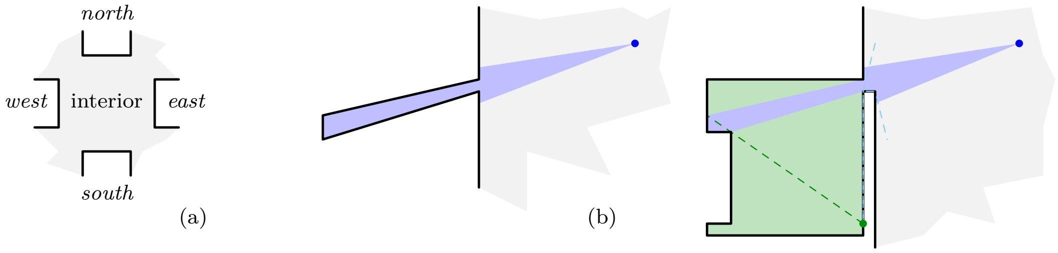


Figure 3 Illustrating the concepts of dents and the rectilinear spike emulator.

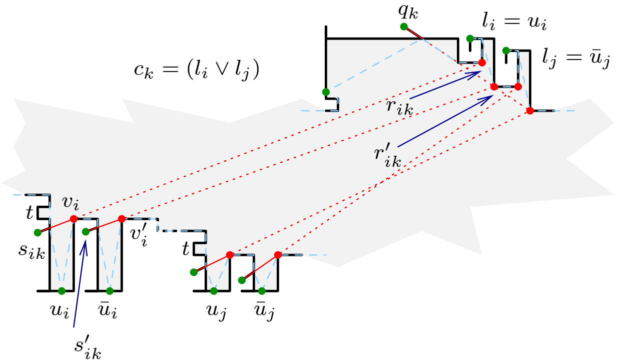


Figure 4 Illustrating the variable and clause gadgets in the rectilinear construction. Spikes at s_{ik}, s'_{ik}, and q_k are replaced by small rectilinear spike emulators.

status for polygons with three dent orientations here but for rectilinear polygons having two (non-opposite) dent orientations the complexity status remains unknown. This should be contrasted with the classical art gallery problem, where linear time algorithms for computing the minimum number of point guards exist only for rectilinear polygons having one dent orientation (histograms) [7], for rectilinear polygons having two non-opposite dent orientations, the art gallery problem can be shown to be APX-hard by modifying the proof by Brodén *et al.* [5]. For the classical art gallery problem, the complexity is unknown for rectilinear monotone polygons having two opposite dent orientations.

We modify the reduction introduced in the previous section to be rectilinear and furthermore to only contain dents of three different orientations. To this end, we introduce the *rectilinear spike emulator*, also used by Katz and Roisman [12]. A spike as used in Section 2 is a thin corridor that can only be seen along a thin visibility cone. We can emulate the effect with a rectilinear gadget as shown in Figure 3(b) using one extra guard (green in the figure) and, as long as the original spike has reflex vertices with larger x-coordinates than its convex vertices, the rectilinear spike gadget never introduces an east dent.

As in Section 2, each variable gadget consists of two rectilinear wells, corresponding to the literals u_i and $\bar{u}_i$. The point x is not necessary, since each well is covered by a green guard at the bottom. Each variable gadget has a rectilinear spike t seen by the two critical guard points v_i and v'_i marked red in Figure 4. As before, each clause gadget has two rectilinear chimneys corresponding to the literals in the clause and each chimney in a clause c_k has two critical guard points r_{ik} and r'_{ik} that see it and they are connected to the corresponding variable gadget of u_i by adding rectilinear spike emulators s_{ik} and s'_{ik} as illustrated in Figure 4. Again, we note that at least one guard in a clause gadget $c_k = (l_i \vee l_j)$ must be

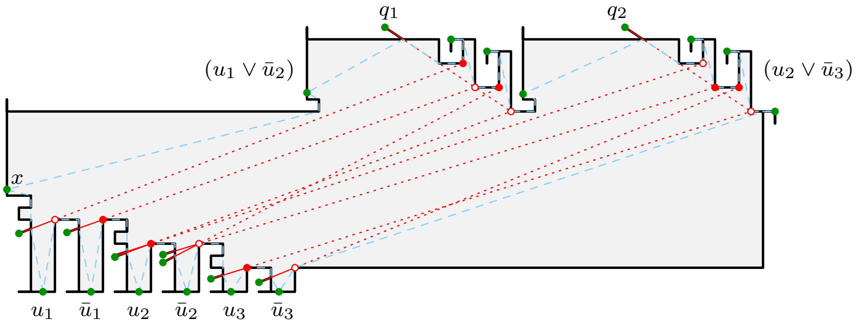


Figure 5 A rectilinear MVPP instance for the MAX2SAT(3) instance $(u_1 \vee \bar{u}_2), (u_2 \vee \bar{u}_3)$.

placed at the lower vertex r'_{ik} or r'_{jk} to see both the chimney and the point q_k, placed in rectilinear spike emulator at the top edge of the clause gadget; see Figure 4.

We add caves at the top of the chimneys, at the bottom of the variable gadgets, on the right side of the base rectangle and two caves, ensuring that these do not introduce east dents. These tie down the shortest watchman route to make it pass all the critical guard points. The convex vertices of the shortest watchman route each require a vision point, giving us $2n + 8m + 2$ such green vision points. In the same way as in Section 2, we can argue that any algorithm produces a canonical vision point set consisting of $V = 2n+8m+2+n+2M+3(m-M) = 3n+11m-M+2$ vision points, choosing the remaining ones from the set of critical guard points; see Figure 5 for a full example of a canonical vision point set consisting of the green points and the subset of the red points that have white centers in a rectilinear polygon with three dent orientations.

Using the result by Berman and Karpinski [4], that it is NP-hard to approximate MAX2SAT(3) by a factor $2012/2011 - \epsilon$, for any $\epsilon > 0$, we obtain as before the ratio

$$\frac{V}{V_{\text{opt}}} \geq \frac{3n + 11m - M + 2}{3n + 11m - M_{\text{opt}} + 2} \geq \frac{15m - M}{15m - M(2012/2011 - \epsilon)} \geq \frac{38209}{38208} - \delta, \tag{2}$$

for any $\delta > 0$ dependent on ϵ, proving the APX-hardness of MVPP in rectilinear polygons having three dent orientations.

References

1 M. Abrahamsen, A. Adamaszek, and T. Miltzow. The art gallery problem is ∃ℝ-complete. *Journal of the ACM*, 69(1):1–70, 2021.

2 A. Aggarwal. *The Art Gallery Theorem: It's Variations, Applications and Algorithmic Aspects*. PhD thesis, Department of Electrical Engineering and Computer Science, Johns Hopkins University, 1984.

3 G. Ausiello, A. Marchetti-Spaccamela, P. Crescenzi, G. Gambosi, M. Protasi, and V. Kann. *Complexity and Approximation — Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999.

4 P. Berman and M. Karpinski. On some tighter inapproximability results. In *Proc.* 26th *International Colloquium on Automata, Languages and Programming, ICALP'99*, pages 200–209, 1999.

5 B. Brodén, M. Hammar, and B.J. Nilsson. Guarding lines and 2-link polygons is APX-hard. In *Proc.* 13th *Canadian Conference on Computational Geometry, CCCG'01*, pages 45–48, 2001.

6 S. Carlsson and B.J. Nilsson. Computing vision points in polygons. *Algorithmica*, 24(1):50–75, 1999.

7 S. Carlsson, B.J. Nilsson, and S. Ntafos. Optimum guard covers and m-watchmen routes for restricted polygons. *International Journal of Computational Geometry and Applications*, 3(1):85–105, 1993.

8 J.C. Culberson and R.A. Reckhow. Orthogonally convex coverings of orthogonal polygons without holes. *Journal of Computer and System Sciences*, 39:166–204, 1989.

9 M. Dror, A. Efrat, A. Lubiw, and J.S.B. Mitchell. Touring a sequence of polygons. In *Proc. 35th ACM Symposium on Theory of Computing, STOC'03*, pages 473–482, 2003.

10 S. Eidenbenz. Inapproximability results for guarding polygons without holes. In *Proc.* 9th *International Symposium on Algorithms and Computation, ISAAC'98*, volume LNCS 1533, pages 427–437. Springer, 1998.

11 S.K. Ghosh and J.W. Burdick. An on-line algorithm for exploring an unknown polygonal environment by a point robot. In *Proc.* 9th *Canadian Conference on Computational Geometry, CCCG'97*, pages 100–106, 1997.

12 M.J. Katz and G.S. Roisman. On guarding the vertices of rectilinear domains. *Computational Geometry*, 39(3):219–228, 2008.

13 D.T. Lee and A.K. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, IT-32:276–282, 1986.

14 R. Motwani, A. Raghunathan, and H. Saran. Covering orthogonal polygons with star polygons: The perfect graph approach. *Journal of Computer and System Sciences*, 40:19–48, 1989.

15 R. Motwani, A. Raghunathan, and H. Saran. Perfect graphs and orthogonally convex covers. *SIAM Journal on Algebraic Discrete Methods*, 2:371–392, 1989.

16 J. O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, 1987.

17 X.-H. Tan. Fast computation of shortest watchman routes in simple polygons. *Information Processing Letters*, 77(1):27–33, 2001.

18 X.-H. Tan, T. Hirata, and Y. Inagaki. Corrigendum to "an incremental algorithm for constructing shortest watchman routes". *International Journal of Computational Geometry and Applications*, 9(3):319–324, 1999.

Reflection Helps Guarding an Art Gallery

Arash Vaezi[1], **Bodhayan Roy**[2], **and Mohammad Ghodsi**[3]

1 **Sharif University of Technology**
avaezi@ce.sharif.edu
2 **Indian Institute of Technology Kharagpur**
broy@maths.iitkgp.ac.in
3 **Sharif University of Technology,**
ghodsi@sharif.edu

Abstract

This paper studies a variant of the Art Gallery problem in which the "walls" can be replaced by *reflecting edges*, which allows the guard to see further and thereby see a larger portion of the gallery.

Chao Xu proved that although reflection helps the visibility of guards to be expanded, similar to the normal guarding problem, even considering r specular reflections we may need $\lfloor \frac{\mathbf{n}}{\mathbf{3}} \rfloor$ guards to cover the polygon, where r is the number of times the visibility ray from a guard may reflect on edges. In this article, we prove that considering r *diffuse* reflections the minimum number of *vertex or boundary* guards required to cover a given simple polygon $\mathcal{P}$ decreases to $\lceil \frac{\alpha}{1+\lfloor \frac{\mathbf{r}}{\mathbf{4}} \rfloor} \rceil$, where α indicates the minimum number of guards required to cover the polygon without reflection. Furthermore, we generalize the $\mathcal{O}(\log n)$-approximation ratio algorithm of the vertex guarding problem to work in the presence of reflection. For a bounded r, the generalization gives a polynomial-time algorithm with $\mathcal{O}(\log n)$-approximation ratio for several special cases of the generalized problem.

1 Introduction

Consider a simple polygon $\mathcal{P}$ with n vertices. Suppose $int(\mathcal{P})$ denotes $\mathcal{P}$'s interior. Two points x and y are visible to each other, if and only if the relatively open line segment $\overline{xy}$ lies completely in $int(\mathcal{P})$. The visibility polygon of a point q in $\mathcal{P}$, denoted as $VP(q)$, consists of all points of $\mathcal{P}$ visible to q. Many problems concerning visibility polygons have been studied so far. There are linear time algorithms to compute $VP(q)$ ([9], [15]). Edges of $VP(q)$ that are not edges of $\mathcal{P}$ are called *window*.

If some of the edges of $\mathcal{P}$ are made into mirrors, then $VP(q)$ may enlarge. Klee first introduced visibility in the presence of mirrors in 1969 [13]. He asked whether every polygon whose edges are all mirrors is illuminable from every interior point. In 1995 Tokarsky constructed an all-mirror polygon inside which there exists a dark point [18]. Visibility with reflecting edges subject to different types of reflections has been studied earlier [5]:

1. *Specular-reflection* in which the direction light is reflected is defined by the law-of-reflection. Since we are working in the plane, this law states that the angle of incidence and the angle of reflection of the visibility rays with the normal through the polygonal edge are the same.
2. *Diffuse-reflection* that is to reflect light with all possible angles from a given surface. The diffuse type is where the angle between the incident and reflected ray may assume all possible values between 0 and π. So, a segment's endpoints that reflect light with diffuse-type defined here will not reflect light behind that segment.

We may count on a single reflection or multiple reflections per visibility ray in each type of reflection. Some papers have specified the maximum number of allowed reflections via reflectors in between [4]. In multiple reflections, we restrict the path of a ray coming from

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.

the viewer to turn at the polygon boundary more than one time. We denote the allowed number of reflections by r. Each time this ray will reflect based on the type of reflection specified in a problem (specular or diffuse).

Every edge of $\mathcal{P}$ can potentially become a reflector. We can assume that all edges are reflecting edges. However, the viewer can only see some edges of $\mathcal{P}$. When we talk about an edge, and we want to consider it as a reflector, we call it a *reflecting edge* (or a *mirror-edge* considering specular reflections). We use the words "reflecting edge" and "reflected" in general, but the word "mirror" is used only when we deal with specular reflections.

Two points x and y inside $\mathcal{P}$ can see each other through a reflecting edge e, if and only if they are reflected-visible with a specified type of reflection. We call these points *reflected-visible* (or mirror-visible).

The Art Gallery problem is to determine the minimum number of guards that are sufficient to see every point in the interior of an art gallery room. The art gallery can be viewed as a polygon $\mathcal{P}$ of n vertices, and the guards are stationary points in $\mathcal{P}$. If guards are placed at vertices of $\mathcal{P}$, they are called *vertex guards*. If guards are placed at any point of $\mathcal{P}$, they are called *point guards*. If guards are allowed to be placed along the boundary of $\mathcal{P}$, they are called *boundary guards* (on the perimeter). To know more details on the history of this problem see [19]. For guarding simple polygons, the problem was proved to be NP-complete for vertex guards by [16]. This proof was generalized to work for point guards by [1]. Ghosh [7] provided an $\mathcal{O}(\log n)$-approximation algorithm for guarding polygons with or without holes with *vertex* guards. King and Kirkpatrick obtained an approximation factor of $\mathcal{O}(\log(OPT))$ for vertex guarding simple polygons [12]. They presented an $\mathcal{O}(\log\log(OPT))$-approximation algorithm for guarding simple polygons, using either vertex guards or perimeter guards.

1.1 Our Settings

Every guard can see a point if it is directly visible to it or if it is reflected-visible. This is a natural and non-trivial extension of the classical art gallery setting. The problem of visibility via reflection has many applications in wireless networks. Also, reflection is a natural issue in computer graphics, where a common rendering technique is to trace the path of light arriving on each pixel of the screen, backwards through multiple reflections [6]. There is a large literature on geometric optics (such as [3], [17]), and on the chaotic behavior of a reflecting ray of light or a bouncing billiard ball (see, e.g., [2], [10], [11], [14]). Particularly, regarding the art gallery problem, reflection helps in decreasing the number of guards (see Figure 1). A special case of the this problem is described by Chao Xu in 2011 [23]. Since we want to generalize the notion of guarding a simple polygon, if the edges become mirrors instead of walls, the light loses intensity every time it gets reflected on the mirror. Therefore after r reflections, it becomes undetectable to a guard. Chao Xu proved that regarding r specular reflections, for any n there exist polygons with n vertices that need $\lfloor \frac{n}{3} \rfloor$ guards. For more information on combining reflection with the art gallery problem see [22], [20], [21], [4], and [5].

2 Regular Visibility vs Reflection

Under some settings, visibility with reflections can be seen as a general case of regular visibility. For example, consider guarding a polygon $\mathcal{P}$ with vertex guards, where all the edges of the polygons are diffuse reflecting edges, and r reflections are allowed for each ray. Let S be the set of guards in an optimal solution, if we do not consider reflection. Consider any guard $v \in S$. The visibility polygon $VP(v)$ of v must have at least one window.

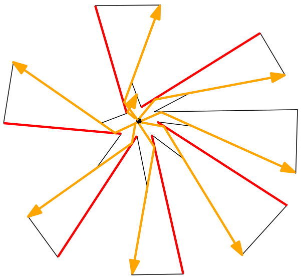


Figure 1 This figure illustrates a situation where a single guard is required if we use reflection-edges; $\Theta(n)$ guards are required if we do not consider reflection. Red segments illustrate the reflected-edges.

Otherwise, v is the only guard of $\mathcal{P}$. Consider such a window, say, w. Then there must be another guard $u \in S$ such that w lies in both $VP(u)$ and $VP(v)$. Then an edge e of $VP(v)$ adjacent to w must see u. This edge (e) must be a polygonal edge since w is a window. So, v can see u by diffuse reflection through this edge. If u is a reflex-vertex, then through another diffuse reflection by any of the two edges of the polygon incident on u, v can see the whole of $VP(u)$.

▶ **Theorem 2.1.** *If $\mathcal{P}$ (a given polygon possibly with holes) can be guarded by α vertex-guards without reflections, then $\mathcal{P}$ can be guarded by at most $\lceil \frac{\alpha}{1+\lfloor \frac{r}{4} \rfloor} \rceil$ guards when r diffuse reflections are permitted.*

To prove this theorem see the following lemmas first:

▶ **Lemma 2.2.** *If there is no single guard that sees the whole (boundary and interior) of a polygon $\mathcal{P}$, then for every optimum vertex guard set $\mathcal{S}$ guarding $\mathcal{P}$, if $u \in \mathcal{S}$, then there is a $y(\neq u) \in \mathcal{S}$ such that u and y can see each other through one diffuse reflection. Furthermore, u and y can fully see each other's visibility polygons with four diffuse reflections.*

Proof. Consider any vertex u of $\mathcal{P}$. Since, by our assumption, u cannot see the whole of $\mathcal{P}$, $VP(u)$ must have a window. This window must intersect the boundary of $\mathcal{P}$ to describe a vertex (say, x) of $VP(u)$ (see Figure 2). Since, by definition, visibility polygons are closed regions, the visibility polygon of some other vertex guard (say, y) of $\mathcal{P}$ must contain x and a non-zero region around x. So, making a polygonal edge of $\mathcal{P}$ containing x a reflecting edge result in a diffuse reflection through which u can see y.

Two edges contain the vertex guard y. Since y sees a small region around x, x must see a non-zero portion of the interior of at least one of these two edges. If x sees a non-zero portion of both edges' interiors, then through diffuse reflections on both edges, u sees $VP(y)$. Now suppose that x sees a non-zero portion of the interiors of only one edge (say, $\overline{yc}$) containing y. Call the other edge $\overline{yd}$, considering a vertex d of $\mathcal{P}$ as its other endpoint. Extend the ray $\overrightarrow{dy}$. Suppose that $\overrightarrow{dy}$ hits the boundary of $\mathcal{P}$ at a point q (see Figure 2). Then an interior point of the edge containing q, very near to q, is visible from the interior of $\overline{yc}$. So, by a diffuse reflection through a polygonal edge containing c, $\overline{yd}$ is visible from u via three diffuse reflections. Hence, by turning $\overline{yd}$ into a reflecting edge, $VP(y)$ becomes visible to u via four diffuse reflections.

Consider a special case where x, y, and a reflex-vertex of the polygon are collinear. In such a case, y does not see an open neighborhood of x. Consider a point p on $\overline{yx}$, where p is closer to x. As there is a reflex-vertex on $\overline{yx}$ which is closer to y, y cannot see an open neighborhood of p, so this neighborhood must be covered by another guard that we can call it u. Although an open neighborhood of x is not visible to y, one side of x is visible to y. If we consider p close enough to x, the ray from u to p will hit the polygon boundary on that same side that y sees. So, we can use this edge on the boundary, and u can see y with only one diffuse reflection. So, again only four diffuse reflections are enough. ◀

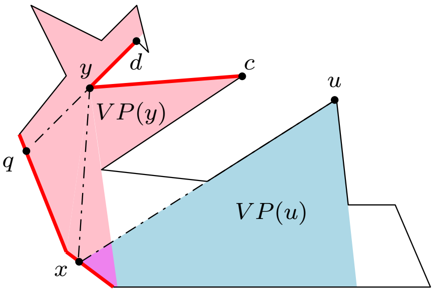


Figure 2 $VP(y)$ is visible from u via four diffuse reflections. The reflecting edges are drawn in red.

Now we build a graph G as follows. We consider the vertex guards in $\mathcal{S}$ as the vertices of G, and add an edge between two vertices of G if and only if the two corresponding vertex guards in $\mathcal{S}$ can see each other directly or through at most one reflection. We have the following Lemma.

▶ **Lemma 2.3.** *The graph G is connected.*

Proof. Consider any two guards g_i and g_j of $\mathcal{S}$. Draw a polygonal path π from g_i to g_j. The path begins in $VP(g_i)$ and ends in $VP(g_j)$. Since $\mathcal{S}$ sees the whole of $\mathcal{P}$, π always travels through the visibility polygon of some or the other guard in $\mathcal{S}$. Then π can travel through two neighboring visibility polygons if and only if it travels through their intersection. This means, by Lemma 2.2 and the definition of G, that if π travels consecutively through the visibility polygons of two guards in $\mathcal{S}$, then the corresponding vertices are adjacent in V. Thus, there is a path between every pair of vertices in G. So, G is connected. ◀

Consider any optimum solution $\mathcal{S}$ of the Art Gallery problem on the polygon $\mathcal{P}$, where $\mid \mathcal{S} \mid = \alpha$. Build a graph G with the vertex guards in $\mathcal{S}$ as its vertices, and add an edge between two vertices of G if and only if the two corresponding vertex guards in $\mathcal{S}$ can see each other directly or through at most one reflection. Due to Lemma 2.3, G is connected. Find a spanning tree T of G and root it at any vertex. Denote the i^{th} level of vertices of T by L_i. Given a value of r, divide the levels of T into $1 + \lfloor \frac{r}{4} \rfloor$ classes, such that the class $\mathcal{C}_i$ contains all the vertices of all levels of T of the form $L_{i+x(1+\lfloor \frac{r}{4} \rfloor)}$, where $x \in \mathbb{Z}_0^+$. By the pigeonhole principle, one of these classes will have at most $\lceil \frac{\alpha}{1+\lfloor \frac{r}{4} \rfloor} \rceil$ vertices. Again, by Lemma 2.2, given any vertex class $\mathcal{C}$ of T, all of $\mathcal{P}$ can be seen by the vertices of $\mathcal{C}$ when x diffuse reflections are allowed. The theorem follows.

▶ Corollary 2.4. The above bound (mentioned in Theorem 2.1) holds even if the guards are allowed to be placed anywhere on the boundary of the polygon.

Proof. The proof follows directly from the proof of Theorem 2.1 since Lemmas 2.2 and 2.3 are valid for boundary guards as well. ◀

▶ **Observation 2.5.** *The above bound (mentioned in Theorem 2.1) does not hold in the case of point guards.*

Proof. See Figure 3. The two guards, colored red, see the whole of the polygons. Clearly, if we seek to replace them with one guard and allow one reflection, then the new guard must lie somewhere near the polygon's lowest vertex. From there, it can see both the red guards. On the left and right sides of the red guards are two funnels whose apices are visible only from certain edges of their respective opposite funnels. These edges are not visible to any point near the lowest vertex of the polygon. So, it is impossible for only one guard to see the whole polygon through one reflection. For any given k, the funnels can be made narrower so that even k reflections are not enough to see the whole polygon.

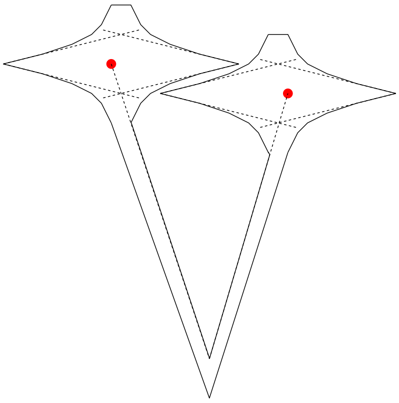


Figure 3 Two point guards cannot be replaced by one despite allowing reflections.

◀

To find an approximate solution to the vertex guard problem with r diffuse reflections, we have a straight-forward generalization of Ghosh's discretization algorithm in [8].

▶ **Theorem 2.6.** *For vertex guards, the art gallery problem considering r reflections, for both the diffuse and specular reflection are solvable in $\mathcal{O}(n^{4^{r+1}+2})$ time giving an approximation ratio of $\mathcal{O}(\log n)$.*

Proof. We begin by drawing all possible windows and decomposing the polygon into convex polygons, as it is mentioned in [8]. Denote the set of these convex polygons by R_0. Denote the set of vertices of these convex polygons of R_0, that lie on the boundary of $\mathcal{P}$ by Q_0. Note that in the initial step, zero diffuse reflections are considered. In the next step, allow a single diffuse reflection for each ray. Each vertex's visibility polygons are extended and have new points on the boundary of $\mathcal{P}$ as their vertices. Join all such pairs of vertices, whenever possible, by drawing windows, to get a new larger set of convex polygons. Denote the set of convex polygons so obtained by R_1, and the set of all their vertices lying on the boundary of $\mathcal{P}$ by Q_1. Analogously, we associate with i diffuse reflections the sets R_i and Q_i.

For r diffuse reflections, as before, draw all possible windows from Q_{r-1} and compute the minimal convex polygons formed as a result. Start by Updating R_{r-1} to R_r and then update Q_{r-1} to Q_r by updating their constituent convex polygons and their vertices lying on the boundary of $\mathcal{P}$, respectively. The lines in R_r are formed by joining together the end-points of the lines of R_{r-1}. So, the cardinality of R_r is at most $4|R_{r-1}|^2$. The cardinality of Q_r is at most $|R_r|^2$, due to being formed by the intersections of the lines of R_r.

Hence, by the argument in Theorem 2.1 of [8], the algorithm takes a total time of $\mathcal{O}(n^{4^{r+1}+2})$. The extra factor of 2 comes due to traversal following the method of [8] again. Our algorithm also gives an approximation ratio of $\mathcal{O}(\log n)$ due to being reduced from the greedy algorithm of Set Cover.

◀

3 Conclusion

This paper addresses a variant of the art gallery guarding problem in which walls are assumed to support diffuse reflections. It was previously established that mirrored walls (supporting specular reflections) do not reduce the number of guards needed in the worst case, even though in some cases, the number of guards required can be decreased substantially. The number of boundary guards (or vertex guards) needed is reduced even in the worst case by a factor proportional to the length r of diffuse reflection sequences permitted. For point guards, it is shown that in some cases no reduction in the guarding number is possible using diffuse reflections.

Various articles worked on approximating the art gallery problem. Accordingly, the generalized version of the art gallery regarding either specular or diffuse reflection has to be approximated by a suitable factor. In this article, we presented a $\mathcal{O}(\log n)$-approximation factor considering both types of reflection that runs in $\mathcal{O}(n^{4^{r+1}+2})$ time if at most r reflection is allowed.

References

1 A. Aggarwal. The art gallery theorem: its variations, applications and algorithmic aspects. *Ph.D. thesis, The Johns Hopkins University, Baltimore, MD*, 1984.

2 C. Boldrighini, M. Keane, , and F. Marchetti. Billiards in polygons. *Ann.Probab.*, (6):532—-540, 1978.

3 M. Born and E. Wolf. Principles of optics. *6th edn. Pergamon Press, Oxford*, 1980.

4 B. Aronovand A. R. Davis, T. K. Dey, S. P. Pal, and D. Prasad. Visibility with multiple specular reflections. *Discrete & computational Geometry.*, (20):62–78, 1998.

5 B. Aronovand A. R. Davis, T. K. Dey, S. P. Pal, and D. Prasad. Visibility with one reflection. *Discrete & computational Geometry.*, (19):553–574, 1998.

6 J. Foley, A. van Dam, S. Feiner, J. Hughes, and R. Phillips. Introduction to computer graphics. *Addison-Wesley, Reading, MA*, 1994.

7 S. K. Ghosh. Approximation algorithms for art gallery problems. *Proc. Canadian Information Processing Society Congress*, pages 429–436, 1987.

8 S. K. Ghosh. Approximation algorithms for art gallery problems in polygons. *Discrete Applied Mathematics*, (158(6)):718–722, 2010.

9 L. J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, (2):209–233, 1987.

10 E. Gutkin. Billiards in polygons. *Phys. D*, 19:311—-333, 1986.

11 S. Kerckhoff, H. Masur, and J. Smillie. Ergodicity of billiard flows and quadratic differentials. . *Ann. of Math.*, 124:293–311, 1986.

12 J. King and D. Kirkpatrick. Improved approximation for guarding simple galleries from the perimeter. *Discrete Computer Geometry*, (46):252–269, 2011. doi:10.1007/s00454-011-9352-x.

13 V. Klee. Is every polygonal region illuminable from some point? *Computational Geometry: Amer.Math. Monthly*, 76:180, 1969.

14 V. V. Kozlov and D. V. Treshchev. Billiards: A genetic introduction to the dynamics of systems with impacts. *Translations of Mathematical Monographs, American Mathematical Society, Providence, RI*, 89:62–78, 1991.

15 D. T. Lee. Visibility of a simple polygon. *Computer Vision, Graphics, and Image Processing*, (22):207—-221, 1983.

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

17 I. Newton. Opticks, or a treatise of the reflections, refractions, inflections and colours of light,. *4th edn. London*, 1730.

18 G. T. Tokarsky. Polygonal rooms not illuminable from every point. *American Mathematical Monthly.*, 102:867–879, 1995.

19 J. Urrutia. *Handbook of Computational Geometry.* 2000. doi:10.1016/B978-044482537-7/50023-1.

20 A. Vaezi and M. Ghodsi. How to extend visibility polygons by mirrors to cover invisible segments.

21 A. Vaezi and M. Ghodsi. Extending visibility polygons by mirrors to cover specific targets. *EuroCG*, pages 13–16, 2013.

22 A. Vaezi and M. Ghodsi. Visibility extension via reflection-edges to cover invisible segments. *Theoretical Computer Science*, 2019. doi:10.1016/j.tcs.2019.02.011.

23 Chao Xu. A generalization of the art gallery theorem with reflection and a cool problem. *https://chaoxuprime.com/posts/2011-06-06-a-generalization-of-the-art-gallery-theorem-with-reflection-and-a-cool-problem.html*, 2011.

Rectilinear Planarity Testing of Independent-Parallel SP-Graphs

Walter Didimo[1], Michael Kaufmann[2], Giuseppe Liotta[1], and Giacomo Ortali[1]

1 University of Perugia, Italy
{walter.didimo,giuseppe.liotta}@unipg.it, giacomo.ortali@gmail.com

2 University of Tübingen, Germany
mk@informatik.uni-tuebingen.de

Abstract

We shed new light on the long-standing open problem of efficiently testing rectilinear planarity of series-parallel graphs (SP-graphs). Namely, we establish new results for a subfamily of such graphs, called *independent-parallel*, where no two parallel components share a pole. When the maximum degree of a vertex is three, a key ingredient behind the design of the known linear-time testing algorithm for general SP-graphs is that one can restrict the attention to a constant number of "rectilinear shapes" for each series or parallel component. To formally describe these shapes the notion of spirality can be used. This key ingredient no longer holds for SP-graphs with vertices of degree four, as we prove a logarithmic lower bound on the spirality of their components. Although this bound holds even for independent-parallel SP-graphs, for this graph family we are able to design a linear-time rectilinear planarity testing algorithm by carefully analyzing their spirality properties.

1 Introduction

Rectilinear planarity testing asks whether a planar 4-graph (i.e., with vertex-degree at most four) admits a planar orthogonal drawing without edge bends. It is a classical subject of study, at the heart of algorithms that compute bend-minimum orthogonal drawings, which find applications in several domains (see, e.g. [4, 7, 11, 16, 17, 18]).

Rectilinear planarity testing in the variable embedding setting is NP-complete [14], it belongs to the XP-class when parameterized by treewidth [6], and it is FPT tractable when parameterized by the number of degree-4 vertices [9]. In the fixed-embedding setting (i.e., when the algorithm must preserve a given planar embedding), the problem can be solved in subquadratic time for general graphs [2, 13], and in linear time for planar 3-graphs [20] and for biconnected SP-graphs [8]. In the variable-embedding setting, linear-time solutions exist only for planar 3-graphs [10, 15, 19, 21] and for outerplanar graphs [12]. A polynomial-time solution for SP-graphs has been known for a long time [5], but establishing whether there is a linear-time algorithm for this graph family remains a long-standing open problem [1]; to date, the most efficient algorithm for n-vertex SP-graphs has complexity $O(n^3 \log n)$ [6].

This paper sheds new light on this long-standing open problem, studying it along the lines that led to linear-time testing algorithms for degree-3 SP-graphs [21] and for general planar 3-graphs [10]. We highlight some of the difficulties that stem from the degree-4 vertices and show how to overcome them to design a linear-time algorithm for a class of degree-4 SP-graphs that properly includes all biconnected degree-3 SP-graphs. To better describe our contribution, we briefly recall some fundamental aspects of these previous approaches.

In a nutshell, the linear-time algorithms of [10, 21] are based on recursive approaches that, at each step, determine if a (series or parallel) component of the graph is rectilinear planar by suitably combining rectilinear representations of its sub-components. For both

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

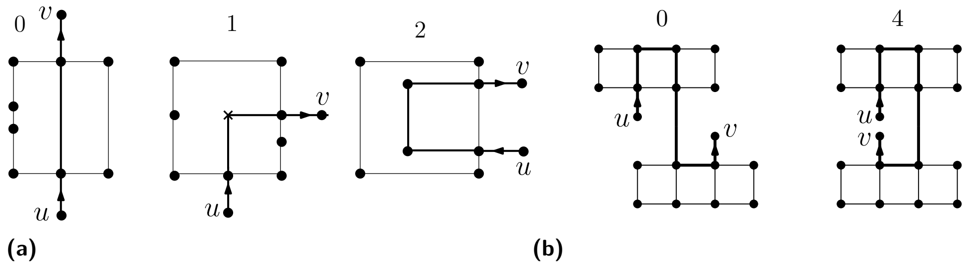


Figure 1 (a) Three orthogonal representations of the same component having spiralities 0, 1, 2, respectively. In bold, an arbitrary path from the pole u to the pole v. A bend, depicted as a cross, is needed for spirality 1. (b) A component that is rectilinear planar only for spiralities 0 and 4.

algorithms, a key ingredient to achieve linear-time complexity is that it is enough to consider a constant number of rectilinear planar representations at each composition step. Another key ingredient is that the "shapes" of these representations can be succinctly described in $O(1)$ space. In [10] the shape is described through the concept of *spirality*, a number that specifies how much a rectilinear planar representation is "rolled up". Roughly, the spirality of a representation is the number of right minus left turns in any oriented path between the poles of its corresponding component (see Fig. 1 for an example). Also the shapes considered in [21] can be described in terms of spirality. Hence, the possible representations for each component are succinctly described by a set of spirality values of constant size.

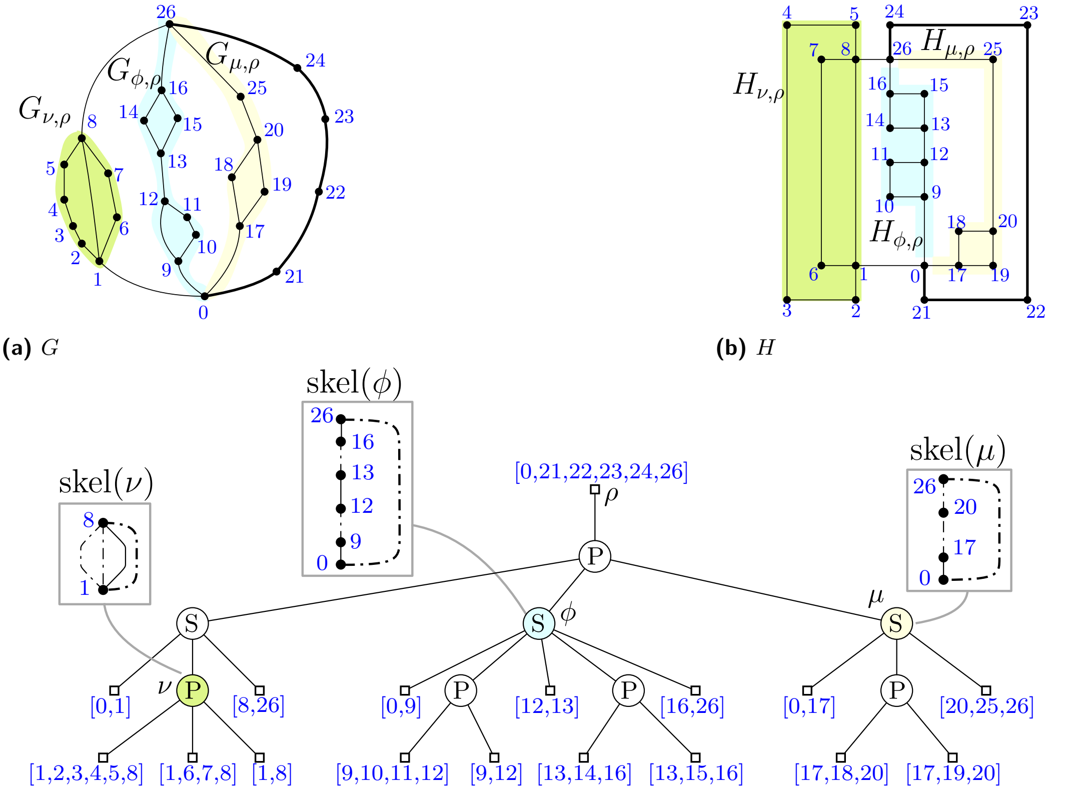
A first difficulty in extending the above approaches to degree-4 SP-graphs is that we lose one of the key ingredients: As stated in Theorem 3.1, there exist n-vertex SP-graphs whose rectilinear planar representations require components with $\Omega(\log n)$ spirality. For these instances a testing algorithm may need to consider $\Omega(\log n)$ rectilinear planar representations per component. To complicate matters even further, it is not obvious how to use the spirality to construct a succinct description of these $\Omega(\log n)$ representations. For example, the component of Fig. 1a is rectilinear planar for spirality 0 and for spirality 2, but not for spirality 1. As another example, the component of Fig. 1b is rectilinear planar for spirality 0 and 4, but not for any intermediate value of spirality. The absence of regularity is an obstacle to the design of a succinct description based on whether a component is rectilinear planar for consecutive spirality values.

We study SP-graphs of vertex-degree four where no two parallel components share a pole, which we call *independent-parallel SP-graphs*; see Fig. 2a. The component in Fig. 1a and the graphs used to prove the $\Omega(\log n)$ spirality lower bound are independent-parallel. By carefully analyzing the spirality properties of independent-parallel SP-graphs, we can overcome the previously described difficulties and design a linear-time rectilinear planarity testing for this graph family. The algorithm uses a set of composition techniques to compute in constant time a succinct description of the rectilinear representations of each component.

2 Preliminaries

We assume familiarity with orthogonal drawings and representations, and with the concepts of SP-graphs and SPQ-trees (see [4]). Testing whether a simple cycle is rectilinear planar is trivial. Hence, we shall assume that G is a biconnected SP-graph different from a simple cycle and we use a variant of the SPQ-tree called *SPQ*-tree* (refer to Fig. 2).

In an SPQ*-tree, each degree-1 node of T is a *Q*-node*, and represents a maximal chain of edges of G (possibly a single edge) starting and ending at vertices of degree larger than



(a) G

(b) H

(c) T_ρ

Figure 2 (a) An (independent-parallel) SP-graph G. (b) A rectilinear representation H of G. (c) The SPQ*-tree T_ρ of G, where ρ represents the thick chain; Q*-nodes are small squares; the left-to-right order of the children of each P-node reflects the embedding of H. The components and skeletons of nodes ν, μ, ϕ are shown: virtual edges are dashed and the reference is thicker.

two and passing through a sequence of degree-2 vertices only (possibly none). If ν is an S- or a P-node, an edge of skel(ν) corresponding to a Q*-node μ is virtual if μ is a chain of at least two edges, else it is a real edge.

For any given Q*-node ρ of T, denote by T_ρ the tree T rooted at ρ. The chain of edges represented by ρ is the *reference chain* of G with respect to T_ρ. If ν is an S- or a P-node distinct from the root child of T_ρ, then skel(ν) contains a virtual edge that has a counterpart in the skeleton of its parent; this edge is the *reference edge* of skel(ν). If ν is the root child, the *reference edge* of skel(ν) is the edge corresponding to ρ. For any S- or P-node ν of T_ρ, the end-vertices of the reference edge of skel(ν) are the *poles* of ν and of skel(ν). We remark that skel(ν) does not change if we change ρ. However, if ν is an S-node, its poles depend on ρ; namely, if ρ' is a Q*-node in the subtree of T_ρ rooted at ν, the poles of ν in $T_{\rho'}$ are different from those in T_ρ. Conversely, the poles of a P-node stay the same independent of the root of T. For a Q*-node ν of T_ρ (including ρ), the *poles* of ν are the end-vertices of the corresponding chain, and do not change when the root of T changes. For any S- or P-node ν of T_ρ, the *pertinent graph* $G_{\nu,\rho}$ of ν is the subgraph of G formed by the union of the chains represented by the leaves in the subtree of T_ρ rooted at ν. The *poles* of $G_{\nu,\rho}$ are the poles of ν. The *pertinent graph* of a Q*-node ν (including the root) is the chain represented by ν, and its *poles* are the poles of ν. Any graph $G_{\nu,\rho}$ is also called a *component* of G (with respect to ρ). If μ is a child of ν, we call $G_{\mu,\rho}$ a *child component* of ν. If H is a rectilinear representation of G, for any node ν of T_ρ, the restriction $H_{\nu,\rho}$ of H to $G_{\nu,\rho}$ is a *component* of H (with respect to ρ). Tree T_ρ is used to describe all planar embeddings of G having the reference chain on the external face. These embeddings are obtained by permuting in all possible ways the non-reference edges of the skeletons of the P-nodes. For each P-node ν, each permutation of the edges in skel(ν) corresponds to a different left-to-right order of the children of ν in T_ρ and of their associated components.

Independent-parallel SP-graphs. Let G be an SP-graph and let T be its SPQ*-tree. We say that G is *independent-parallel* if no two P-nodes of T have a pole in common (see, e.g., Fig. 2a). Let ρ be a Q*-node of T. For a pole w of a node ν of T_ρ, let $\text{indeg}_\nu(w)$ and $\text{outdeg}_\nu(w)$ be the degree of w inside and outside $G_{\nu,\rho}$, respectively. If G is independent-parallel, each pole w of a P-node ν of T_ρ is such that $\text{outdeg}_\nu(w) = 1$; if ν is an S-node, either $\text{indeg}_\nu(w) = 1$ or $\text{outdeg}_\nu(w) = 1$. In all cases, $\text{outdeg}_\nu(w) = 1$ when $\text{indeg}_\nu(w) > 1$.

3 Results

Spirality of Independent-Parallel SP-graphs. Let G be a degree-4 SP-graph and let H be a rectilinear planar representation of G. Let T_ρ be a rooted SPQ*-tree of G, let $H_{\nu,\rho}$ be a component of H, and let $\{u, v\}$ be the poles of ν, conventionally ordered according to an st-numbering of G, where s and t are the poles of ρ. Since we deal with independent-parallel SP-graphs, $\text{outdeg}_\nu(w) = 1$ when $\text{indeg}_\nu(w) > 1$. Define the *alias vertex* w' of w as follows: If $\text{indeg}_\nu(w) = 1$, then $w' = w$; else w' is a dummy vertex that subdivides the edge incident to w outside $H_{\nu,\rho}$. Let P^{uv} be any simple path from u to v inside $H_{\nu,\rho}$ and let u' (resp. v') be the alias vertex of u (resp. of v). The path $S^{u'v'}$ obtained concatenating (u', u), P^{uv}, and (v, v') is a *spine* of $H_{\nu,\rho}$. The *spirality* $\sigma(H_{\nu,\rho})$ of $H_{\nu,\rho}$ in H is the number of right turns minus the number of left turns along $S^{u'v'}$ while moving from u' to v'.

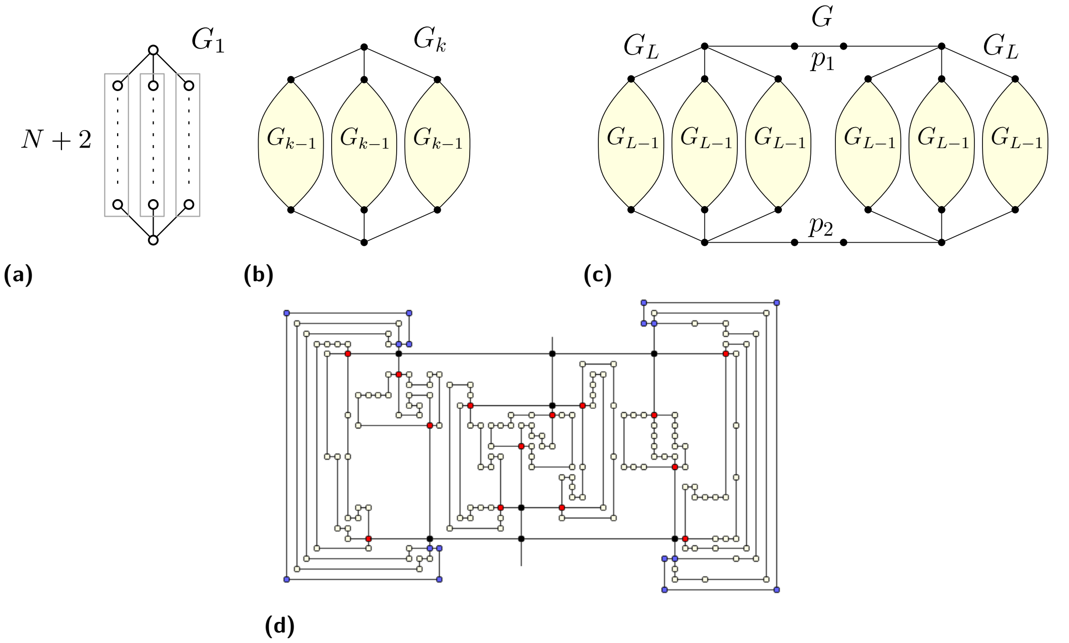


Figure 3 (a)–(c) The graph family of Theorem 3.1, where $L = \frac{N}{2} + 1$. (d) A rectilinear planar representation of G_L (computed by the GDToolkit library [3]), for $N = 4$; the two G_0 components with blue vertices have spirality $N + 2 = 6$ (left) and $-(N + 2) = -6$ (right), respectively.

See, e.g., Fig. 1. Di Battista et al. [5] show that the spirality of $H_{\nu,\rho}$ does not depend on the choice of P^{uv}; also any component of H can be replaced by another component with the same spirality. In Fig. 2b, the spiralities of $H_{\nu,\rho}$, $H_{\mu,\rho}$, and $H_{\phi,\rho}$ are 2, -2, and 0, respectively. For brevity, we shall denote by σ_ν the spirality of a rectilinear representation of $G_{\nu,\rho}$. We say that $G_{\nu,\rho}$ *admits spirality* σ_ν or, equivalently, that ν *admits spirality* σ_ν, if there exists a rectilinear planar representation $H_{\nu,\rho}$ with spirality σ_ν in some rectilinear planar representation H of G.

The proof of the lower bound of Theorem 3.1 uses an infinite family of graphs that have components whose spirality is not bounded by a constant in any rectilinear planar representation. The graph family is schematically illustrated in Fig. 3: For any even integer $N \geq 2$, we construct an independent-parallel SP-graph G with $n = O(3^N)$ vertices whose rectilinear planar representations require a component with spirality larger than N. Namely, let $L = \frac{N}{2} + 1$. For $k \in \{0, \dots, L\}$, let G_k be the SP-graph inductively defined as follows: (i) G_0 is a chain of $N + 4$ vertices; (ii) G_1 is a parallel of three copies of G_0, with coincident poles (Fig. 3a); (iii) for $k \geq 2$, G_k is a parallel composition of three series, each starting and ending with an edge, and having G_{k-1} in the middle (Fig. 3b). Graph G is obtained by composing in a cycle two chains p_1 and p_2 of length three, with two copies of G_L (Fig. 3c). In any representation of G, at least one of the G_0 components requires spirality larger than N.

▶ **Theorem 3.1.** *For infinitely many integer values of n, there exists an n-vertex independent-parallel SP-graph for which every rectilinear planar representation has a component with spirality* $\Omega(\log n)$.

Rectilinear Planarity Testing. Let G be a rectilinear planar SP-graph, T_ρ be a rooted SPQ*-tree of G, and $\nu \neq \rho$ be a node of T_ρ. The *rectilinear spirality set* $\Sigma_{\nu,\rho}$ of ν in T_ρ (and of $G_{\nu,\rho}$) is the set of spirality values for which $G_{\nu,\rho}$ admits a rectilinear planar representation. We are able to prove that there is some regularity in the rectilinear spirality sets of independent-parallel SP-graphs. Denote by $\Sigma^+_{\nu,\rho}$ (resp. $\Sigma^-_{\nu,\rho}$) the subset of non-negative (resp. non-positive) values of $\Sigma_{\nu,\rho}$. Clearly, $\Sigma_{\nu,\rho} = \Sigma^+_{\nu,\rho} \cup \Sigma^-_{\nu,\rho}$. Note that, for any value $\sigma_\nu \in \Sigma_{\nu,\rho}$, we also have that $-\sigma_\nu \in \Sigma_{\nu,\rho}$. Indeed, if $G_{\nu,\rho}$ admits a rectilinear representation with spirality σ_ν for some embedding, by flipping this embedding around the poles of $G_{\nu,\rho}$, we can obtain a rectilinear representation of $G_{\nu,\rho}$ with spirality $-\sigma_\nu$. Hence, $\sigma_\nu \in \Sigma^+_{\nu,\rho}$ if and only if $-\sigma_\nu \in \Sigma^-_{\nu,\rho}$, and we can restrict the study of the properties of $\Sigma_{\nu,\rho}$ to $\Sigma^+_{\nu,\rho}$, which we call the *non-negative rectilinear spirality set* of ν in T_ρ (or of $G_{\nu,\rho}$).

We prove that if G is an independent-parallel SP-graph, there is a limited number of possible structures for the sets $\Sigma^+_{\nu,\rho}$. Let $m < M$ be two non-negative integers: (i) $[M]$ is a *trivial interval* and denotes the singleton $\{M\}$; (ii) $[m, M]^1$ is a *jump-1 interval* and denotes the set of all integers in the interval $[m, M]$; (iii) If m and M have the same parity, $[m, M]^2$ is a *jump-2 interval* and denotes the set of values $\{m, m + 2, \dots, M - 2, M\}$.

▶ **Theorem 3.2.** *Let G be a rectilinear planar independent-parallel SP-graph and let $G_{\nu,\rho}$ be a component of G. The non-negative rectilinear spirality set $\Sigma^+_{\nu,\rho}$ of $G_{\nu,\rho}$ has one the following six structures:* $[0]$, $[1]$, $[1, 2]^1$, $[0, M]^1$, $[0, M]^2$, $[1, M]^2$.

Let G be a biconnected independent-parallel SP-graph that is not a simple cycle, T be its SPQ*-tree, and $\{\rho_1, \dots, \rho_h\}$ be the Q*-nodes of T. Based on Theorem 3.2, for each possible choice of the root $\rho \in \{\rho_1, \dots, \rho_h\}$, the algorithm visits T_ρ bottom-up in post-order and computes, for each visited node ν, the non-negative spirality set $\Sigma^+_{\nu,\rho}$, based on the sets of the children of ν. This computation is done in $O(1)$ time. $\Sigma^+_{\nu,\rho}$ is representative of all

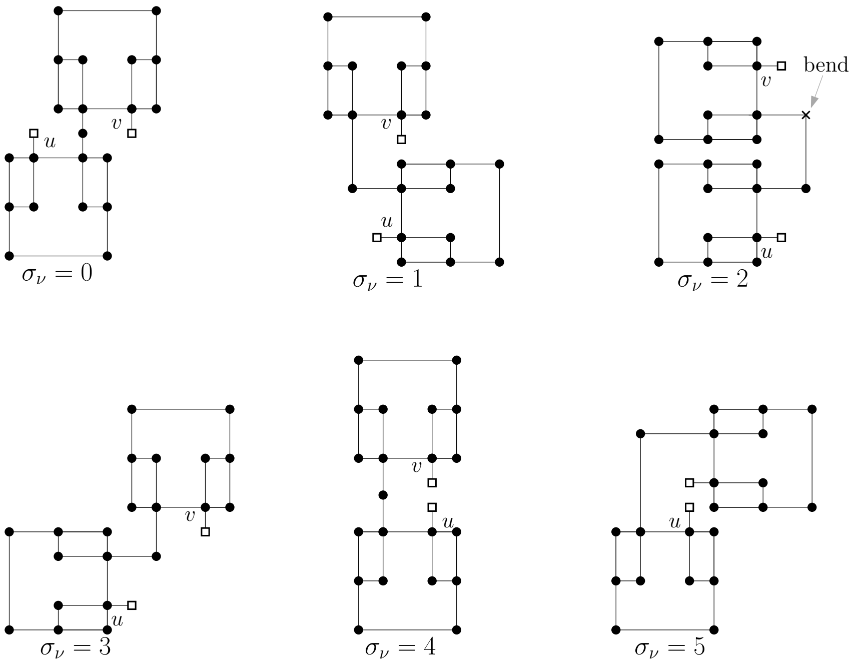


Figure 4 Component that admits spiralities 0,1,3,4,5. Spirality 2 needs a bend (×).

"shapes" that $G_{\nu,\rho}$ can take in a rectilinear planar representation of G with the reference chain on the external face. At the level of the root the test consists of verifying whether ν and ρ admit spirality values $\sigma_\nu \in \Sigma^+_{\nu,\rho}$ and $\sigma_\rho \in \Sigma^+_{\rho,\rho}$, respectively, such that $\sigma_\nu + \sigma_\rho = 4$. An $O(n)$-time testing algorithm over all choices of the root ρ is achieved by exploiting a re-usability principle similar to the one in [10].

▶ **Theorem 3.3.** *Let G be an independent-parallel SP-graph with n vertices. There exists an $O(n)$-time algorithm that tests whether G is rectilinear planar.*

It is not difficult to see that if the testing is positive, a rectilinear planar representation of G can also be constructed in linear time by a variant of the technique described in [8]: Visit T_ρ top-down and for each node ν compute a target value of spirality in $\Sigma^+_{\nu,\rho}$, based on whether ν is an S-node, a P-node, or a Q*-node.

4 Final Remarks

The problem about whether the result of Theorem 3.3 can be extended to every SP-graph remains open. We just observe here that the spirality set of a component of an SP-graph that is not independent-parallel may not exhibit a behavior like the one described in Theorem 3.2. For example, the component shown in Fig. 4 is rectilinear planar for all spirality values from 0 to 5 except 2.

References

1 Franz-Josef Brandenburg, David Eppstein, Michael T. Goodrich, Stephen G. Kobourov, Giuseppe Liotta, and Petra Mutzel. Selected open problems in graph drawing. In Giuseppe Liotta, editor, *Graph Drawing, 11th International Symposium, GD 2003, Perugia, Italy, September 21-24, 2003, Revised Papers*, volume 2912 of *Lecture Notes in Computer Science*, pages 515–539. Springer, 2003. doi:10.1007/978-3-540-24595-7_55.

2 Sabine Cornelsen and Andreas Karrenbauer. Accelerated bend minimization. *J. Graph Algorithms Appl.*, 16(3):635–650, 2012. doi:10.7155/jgaa.00265.

3 Giuseppe Di Battista and Walter Didimo. Gdtoolkit. In *Handbook of Graph Drawing and Visualization*, pages 571–597. Chapman and Hall/CRC, 2013.

4 Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999.

5 Giuseppe Di Battista, Giuseppe Liotta, and Francesco Vargiu. Spirality and optimal orthogonal drawings. *SIAM J. Comput.*, 27(6):1764–1811, 1998. doi:10.1137/S0097539794262847.

6 Emilio Di Giacomo, Giuseppe Liotta, and Fabrizio Montecchiani. Sketched representations and orthogonal planarity of bounded treewidth graphs. In *Graph Drawing*, volume 11904 of *Lecture Notes in Computer Science*, pages 379–392. Springer, 2019.

7 W. Didimo and G. Liotta. Mining graph data. In Diane J. Cook and Lawrence B. Holder, editors, *Graph Visualization and Data Mining*, pages 35–64. Wiley, 2007.

8 Walter Didimo, Michael Kaufmann, Giuseppe Liotta, and Giacomo Ortali. Rectilinear planarity testing of plane series-parallel graphs in linear time. In *Graph Drawing*, volume 12590 of *Lecture Notes in Computer Science*, pages 436–449. Springer, 2020.

9 Walter Didimo and Giuseppe Liotta. Computing orthogonal drawings in a variable embedding setting. In *ISAAC*, volume 1533 of *Lecture Notes in Computer Science*, pages 79–88. Springer, 1998.

10 Walter Didimo, Giuseppe Liotta, Giacomo Ortali, and Maurizio Patrignani. Optimal orthogonal drawings of planar 3-graphs in linear time. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 806–825. SIAM, 2020. doi:10.1137/1.9781611975994.49.

11 Christian A. Duncan and Michael T. Goodrich. Planar orthogonal and polyline drawing algorithms. In Roberto Tamassia, editor, *Handbook on Graph Drawing and Visualization.*, pages 223–246. Chapman and Hall/CRC, 2013. URL: https://www.crcpress.com/Handbook-of-Graph-Drawing-and-Visualization/Tamassia/9781584884125.

12 Fabrizio Frati. Planar rectilinear drawings of outerplanar graphs in linear time. In *Graph Drawing*, volume 12590 of *Lecture Notes in Computer Science*, pages 423–435. Springer, 2020.

13 Ashim Garg and Roberto Tamassia. A new minimum cost flow algorithm with applications to graph drawing. In Stephen C. North, editor, *Graph Drawing, Symposium on Graph Drawing, GD '96, Berkeley, California, USA, September 18-20, Proceedings*, volume 1190 of *Lecture Notes in Computer Science*, pages 201–216. Springer, 1996. doi:10.1007/3-540-62495-3_49.

14 Ashim Garg and Roberto Tamassia. On the computational complexity of upward and rectilinear planarity testing. *SIAM J. Comput.*, 31(2):601–625, 2001. doi:10.1137/S0097539794277123.

15 Md. Manzurul Hasan and Md. Saidur Rahman. No-bend orthogonal drawings and no-bend orthogonally convex drawings of planar graphs (extended abstract). In Ding-Zhu Du, Zhenhua Duan, and Cong Tian, editors, *Computing and Combinatorics - 25th International Conference, COCOON 2019, Xi'an, China, July 29-31, 2019, Proceedings*,

volume 11653 of *Lecture Notes in Computer Science*, pages 254–265. Springer, 2019. doi:10.1007/978-3-030-26176-4_21.

16 Michael Jünger and Petra Mutzel, editors. *Graph Drawing Software*. Springer, 2004. doi:10.1007/978-3-642-18638-7.

17 Michael Kaufmann and Dorothea Wagner, editors. *Drawing Graphs, Methods and Models (the book grow out of a Dagstuhl Seminar, April 1999)*, volume 2025 of *Lecture Notes in Computer Science*. Springer, 2001. doi:10.1007/3-540-44969-8.

18 Takao Nishizeki and Md. Saidur Rahman. *Planar Graph Drawing*, volume 12 of *Lecture Notes Series on Computing*. World Scientific, 2004.

19 Md. Saidur Rahman, Noritsugu Egi, and Takao Nishizeki. No-bend orthogonal drawings of subdivisions of planar triconnected cubic graphs. *IEICE Trans. Inf. Syst.*, 88-D(1):23–30, 2005. URL: http://search.ieice.org/bin/summary.php?id=e88-d_1_23&category=D&year=2005&lang=E&abst=.

20 Md. Saidur Rahman, Takao Nishizeki, and Mahmuda Naznin. Orthogonal drawings of plane graphs without bends. *J. Graph Algorithms Appl.*, 7(4):335–362, 2003. URL: http://jgaa.info/accepted/2003/Rahman+2003.7.4.pdf.

21 Xiao Zhou and Takao Nishizeki. Orthogonal drawings of series-parallel graphs with minimum bends. *SIAM J. Discret. Math.*, 22(4):1570–1604, 2008. doi:10.1137/060667621.

Euclidean Bipartite Edge Cover in Subcubic Time*

Rodrigo Castro[1], José M. Díaz-Báñez[2], Marco A. Heredia[1], Jorge Urrutia[3], Inmaculada Ventura[2], and Francisco J. Zaragoza[1]

1 **Departamento de Sistemas, Universidad Autónoma Metropolitana Azcapotzalco**
`racc,hvma,franz@azc.uam.mx`

2 **Departamento de Matemática Aplicada II, Universidad de Sevilla**
`dbanez,iventura@us.es`

3 **Instituto de Matemáticas, Universidad Nacional Autónoma de México**
`urrutia@matem.unam.mx`

Abstract

Given a graph $G = (V, E)$ with costs on its edges, the minimum-cost edge cover problem consists of finding a subset of E covering all vertices in V at minimum cost. If G is bipartite, this problem can be solved in time $O(|V|^3)$ via a well-known reduction to a maximum-cost matching problem on G. If in addition V is a set of points on the Euclidean line, Collanino *et al.* showed that the problem can be solved in time $O(|V| \log |V|)$ and asked whether it can be solved in time $o(|V|^3)$ if V is a set of points on the Euclidean plane. We answer this in the affirmative, giving an $O(|V|^{2.5} \log |V|)$ algorithm based on the Hungarian method using weighted Voronoi diagrams.

1 Introduction

Let $G = (V, E)$ be a simple graph with no isolated vertices and with cost $d_{uv} \geq 0$ for each $uv \in E$. The *minimum-cost edge cover problem* consists of finding a subset $C \subseteq E$ covering all vertices in V at minimum cost $d(C) = \sum_{\{u,v\} \in C} d_{uv}$. We direct the interested reader to a recent survey on exact and approximation algorithms for this and related problems [10].

Let $C \subseteq E$ be a minimum-cost edge cover of G. To each $v \in V$, assign an edge $e \in C$ which covers v. Note that some edges will be assigned to two vertices, while the rest will be assigned to exactly one vertex. Note further that the former constitute a matching M of G, while each of the latter must be a minimum-cost edge incident to its assigned vertex. For each $v \in V$, define d_v to be the minimum cost among the edges incident to v and, for each $\{u, v\} \in E$, define the *reduced* cost $c_{uv} = d_u + d_v - d_{uv}$. It follows that the cost $d(C)$ of C equals $\sum_{v \in V} d_v - \sum_{\{u,v\} \in M} c_{uv}$. Since the first term is constant, the cost $d(C)$ is minimized when the reduced cost $c(M)$ of M is maximized. This $O(|V| + |E|)$ time transformation due to Geelen (see [3, page 165]) implies that one can solve the minimum-cost edge cover problem on general graphs in time $O(|V|^2|E|)$ using the blossom algorithm [5], and on bipartite graphs in time $O(|V||E|)$ using a well-known improvement of the Hungarian method [6].

When G is bipartite, the minimum-cost edge cover problem has sometimes been studied under the name *many-to-many* matching. In particular, Collanino *et al.* studied this problem

* R. Castro, M.A. Heredia, and F.J. Zaragoza are partially supported by Sistema Nacional de Investigadores, Consejo Nacional de Ciencia y Tecnología, México. J.M. Díaz-Báñez and I. Ventura are partially supported by Ministerio de Ciencia e Innovación CIN/AEI/10.13039/501100011033/ (PID2020-114154RB-I00). J. Urrutia is partially supported by PAPIIT IN105221 Programa de Apoyo a la Investigación e Innovación Tecnológica, UNAM, Mexico.

J.M. Díaz-Báñez, J. Urrutia and I. Ventura were also supported by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 734922.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

when V is a set of points on the Euclidean line, that is, when the cost of edges between points in different partitions are *implicitly* given by the Euclidean lengths of the respective segments [2]. They showed that the problem can be solved in time $O(|V|\log|V|)$ and asked whether it can be solved in time $o(|V|^3)$ if V is a set of points on the Euclidean plane.

Our main result is an affirmative answer to this question. The rest of this work is structured as follows. In Section 2 we setup the Hungarian method and show that the use of reduced costs implies that the dual variables are nicely bounded above. In Section 3 we show how to modify the Hungarian method with reduced costs to avoid many dual updates. In Section 4 we present our $O(|V|^{2.5}\log|V|)$ algorithm using weighted Voronoi diagrams.

2 The Hungarian Method with Reduced Costs

Let G be a complete bipartite graph with partition $R = \{r_1, \ldots, r_n\}$ and $B = \{b_1, \ldots, b_m\}$ $(n \geq m)$. For each $1 \leq i \leq n$ and $1 \leq j \leq m$, let $c(r_i, b_j)$ be the cost of edge $\{r_i, b_j\}$. The maximum-cost bipartite matching problem with cost c has a well-known formulation as a pair of primal and dual linear programs with primal variables $x_{i,j}$ for each $1 \leq i \leq n$ and $1 \leq j \leq m$ and dual variables α_i for each $1 \leq i \leq n$ and β_j for each $1 \leq j \leq m$. Any integral solution of the primal corresponds to a matching $M = \{\{r_i, b_j\} : x_{i,j} = 1\}$. Given a matching M, we say that it *covers* the ends of its edges. We say that the vertices not covered by M are *exposed*. An edge $\{r_i, b_j\}$ such that $\alpha_i + \beta_j = c(r_i, b_j)$ is called an *equality* edge. In these terms, we can write the complementary slackness conditions at primal-dual optimality as:

First condition Each edge $\{r_i, b_j\} \in M$ is an equality edge.

Second condition For each $1 \leq i \leq n$, $\alpha_i > 0$ implies r_i is covered.

Third condition For each $1 \leq j \leq m$, $\beta_j > 0$ implies b_j is covered.

We say that vertex b_j is *bad* if it fails the last condition, that is, if $\beta_j > 0$ but it is exposed.

We follow the well-known Hungarian method for maximum-cost bipartite matching [8, 9], that is, we construct a series of integral primal solutions and dual solutions satisfying the first two complementary slackness conditions and, at the end, also the third complementary slackness condition (no bad vertices). Our starting primal solution is $x_{i,j} \leftarrow 0$ for all $1 \leq i \leq n, 1 \leq j \leq m$ (the empty matching M), and our starting dual solution is $\alpha_i \leftarrow 0$ for all $1 \leq i \leq n$ and $\beta_j \leftarrow \max\{c(r_i, b_j) : 1 \leq i \leq n\}$ for all $1 \leq j \leq m$ (if $\beta_j < 0$, then $\beta_j \leftarrow 0$).

A path is *alternating* if it consists of edges that are alternately in M and not in M. An alternating path P is *augmenting* if it starts and ends in two distinct exposed vertices. Observe that augmenting paths must start and end on distinct partitions. They are called augmenting since $M' = P \triangle M$ is a matching larger than M. An *alternating tree* is a tree rooted at an exposed vertex in B, such that all its paths from the root to its leaves are alternating and consist of equality edges. If any such path P is augmenting, then $M' = P \triangle M$ also satisfies the first two complementary slackness conditions.

Each iteration of the Hungarian method consists of growing a forest of alternating trees, rooted at bad vertices and formed of equality edges, until the amount of bad vertices decreases. The method stops when no bad vertices remain. There are at most m iterations. At the start of each iteration, let $S \neq \emptyset$ be the set of bad vertices of B, let $F \leftarrow R$, let $T \leftarrow \emptyset$, let $b_s \in S$ be such that $\beta_s \leq \beta_j$ for all $b_j \in S$, and let $\epsilon \leftarrow \beta_s > 0$. As long as F is non-empty, let $\{r_i, b_j\}$ be an edge that achieves the minimum dual slack as

$$\delta \leftarrow \min\{\alpha_i + \beta_j - c(r_i, b_j) : r_i \in F, b_j \in S\} \tag{1}$$

Four mutually exclusive cases may occur:

Case 1 ($\delta = 0$ and r_i is exposed): Edge $\{r_i, b_j\}$ is an equality edge which is added to the alternating forest. Furthermore, the path P from r_i to the root b_t of its alternating tree is an augmenting path. Let $M \leftarrow M \triangle P$. Now b_t is not bad. Stop the iteration.

Case 2 ($\delta = 0$ and r_i is covered): In this case r_i is matched in M to $b_k \notin S$. Edges $\{b_k, r_i\} \in M$ and $\{r_i, b_j\}$ are equality edges which are added to the alternating forest. Let $F \leftarrow F \setminus \{r_i\}$, $T \leftarrow T \cup \{r_i\}$, and $S \leftarrow S \cup \{b_k\}$. If $\beta_k < \epsilon$, then let $s \leftarrow k$ and $\epsilon \leftarrow \beta_k$. Continue with the iteration.

Case 3 ($\epsilon > \delta$): Let $\alpha_i \leftarrow \alpha_i + \delta$ for each $r_i \in T$, $\beta_j \leftarrow \beta_j - \delta$ for each $b_j \in S$, and $\epsilon \leftarrow \epsilon - \delta$. This keeps all equality edges in the alternating forest and creates at least one equality edge. Continue with the iteration.

Case 4 ($\delta \geq \epsilon$): Let $\alpha_i \leftarrow \alpha_i + \epsilon$ for each $r_i \in T$, $\beta_j \leftarrow \beta_j - \epsilon$ for each $b_j \in S$, and $\epsilon \leftarrow 0$. This keeps all equality edges in the alternating forest. Let P be the path from b_s to the root b_t of its alternating tree. Let $M \leftarrow M \triangle P$. Now b_t is not bad. Stop the iteration.

In our case, we start with an instance of the minimum-cost edge cover problem where, for each $1 \leq i \leq n$ and $1 \leq j \leq m$, the *original* cost of edge $\{r_i, b_j\}$ is given by $d(r_i, b_j) \geq 0$. Following the reduction, we compute the minimum cost of the edges incident to each vertex of G. That is, for each $1 \leq i \leq n$, let $d(r_i) = \min\{d(r_i, b_j) : 1 \leq j \leq m\}$ and, for each $1 \leq j \leq m$, let $d(b_j) = \min\{d(r_i, b_j) : 1 \leq i \leq n\}$. Finally, for each $1 \leq i \leq n$ and $1 \leq j \leq m$, the *reduced* cost of edge $\{r_i, b_j\}$ is given by $c(r_i, b_j) = d(r_i) + d(b_j) - d(r_i, b_j)$.

It turns out that, under these special conditions, the dual variables remain within certain nice bounds during the execution of the Hungarian method. In particular, $\beta_j < 0$ cannot occur during its initialization, even though some reduced costs might be negative.

▶ **Lemma 2.1.** *During an execution of the Hungarian method with reduced costs, the dual variables satisfy* $0 \leq \alpha_i \leq d(r_i)$ *for all* $1 \leq i \leq n$ *and* $0 \leq \beta_j \leq d(b_j)$ *for all* $1 \leq j \leq m$.

Proof. Recall that $0 \leq d(r_i) \leq d(r_i, b_j)$ and $0 \leq d(b_j) \leq d(r_i, b_j)$ for all $1 \leq i \leq n$ and all $1 \leq j \leq m$. At the start of the method $\beta_j = \max\{c(r_i, b_j) : 1 \leq i \leq n\}$. Since $c(r_i, b_j) = d(b_j) - [d(r_i, b_j) - d(r_i)] \leq d(b_j)$ for all $1 \leq i \leq n$, it follows that $\beta_j \leq d(b_j)$. Moreover, if $r_k \in R$ is closest to b_j, then $d(b_j) = d(r_k, b_j)$ and $\beta_j \geq c(r_k, b_j) = d(r_k) - [d(r_k, b_j) - d(b_j)] = d(r_k) \geq 0$. Since β_j never grows and never becomes negative, $0 \leq \beta_j \leq d(b_j)$ remains true during the execution of the algorithm. At the start of the method $\alpha_i = 0$. Note that α_i can increase only when we consider an equality edge $\{r_i, b_j\}$. In this case $\alpha_i = c(r_i, b_j) - \beta_j = d(r_i) - [d(r_i, b_j) - d(b_j)] - \beta_j \leq d(r_i)$. Since α_i never decreases, $0 \leq \alpha_i \leq d(r_i)$ remains true during the execution of the algorithm. ◀

3 Avoiding Dual Updates

In order to accelerate Case 3, we want to avoid updating α_i and β_j more than once during an iteration of the Hungarian method. To this end, we introduce weights $w(r_i) \leftarrow d(r_i) - \alpha_i$ for each $r_i \in R$ and $w(b_j) \leftarrow d(b_j) - \beta_j$ for each $b_j \in B$. We also introduce the total dual change $\Delta \leftarrow 0$, used at the end of an iteration to update α_i and β_j. Consider the following adaptation to the Hungarian method: As long as F is non-empty, let $\{r_i, b_j\}$ be an edge that achieves the minimum dual slack as

$$\delta \leftarrow \min\{d(r_i, b_j) - w(r_i) - w(b_j) : r_i \in F, b_j \in S\} - \Delta \tag{2}$$

Four mutually exclusive cases may occur:

Case 1 ($\delta = 0$ and r_i is exposed): Edge $\{r_i, b_j\}$ is an equality edge which is added to the alternating forest. Let $F \leftarrow F \setminus \{r_i\}$, $T \leftarrow T \cup \{r_i\}$, and $w(r_i) \leftarrow w(r_i) + \Delta$. Furthermore, the path P from r_i to the root b_t of its alternating tree is an augmenting path. Let $M \leftarrow M \triangle P$. Now b_t is not bad. Let $\alpha_i \leftarrow d(r_i) - w(r_i) + \Delta$ for each $r_i \in T$ and $\beta_j \leftarrow d(b_j) - w(b_j) - \Delta$ for each $b_j \in S$. Stop the iteration.

Case 2 ($\delta = 0$ and r_i is covered): In this case r_i is matched in M to $b_k \notin S$. Edges $\{b_k, r_i\} \in M$ and $\{r_i, b_j\}$ are equality edges which are added to the alternating forest. Let $F \leftarrow F \setminus \{r_i\}$, $T \leftarrow T \cup \{r_i\}$, $S \leftarrow S \cup \{b_k\}$. Let $w(r_i) \leftarrow w(r_i) + \Delta$ and $w(b_k) \leftarrow w(b_k) - \Delta$. If $\beta_k < \epsilon$, then let $s \leftarrow k$ and $\epsilon \leftarrow \beta_k$. Continue with the iteration.

Case 3 ($\epsilon > \delta$): Let $\Delta \leftarrow \Delta + \delta$ and $\epsilon \leftarrow \epsilon - \delta$. This keeps all equality edges in the alternating forest and creates at least one equality edge. Continue with the iteration.

Case 4 ($\delta \geq \epsilon$): Let $\Delta \leftarrow \Delta + \epsilon$ and $\epsilon \leftarrow 0$. This keeps all equality edges in the alternating forest. Let P be the path from b_s to the root b_t of its alternating tree. Let $M \leftarrow M \triangle P$. Now b_t is not bad. Let $\alpha_i \leftarrow d(r_i) - w(r_i) + \Delta$ for each $r_i \in T$ and $\beta_j \leftarrow d(b_j) - w(b_j) - \Delta$ for each $b_j \in S$. Stop the iteration.

Note that, by Lemma 2.1, the weights $w(r_i)$ and $w(b_j)$ remain non-negative. Also note that each time we evaluate (2), its right-hand side is identical to the right-hand side of (1), that is, $\alpha_i + \beta_j - c(r_i, b_j) = d(r_i, b_j) - w(r_i) - w(b_j) - \Delta$ for all $r_i \in F$ and $b_j \in S$.

4 Weighted Voronoi Diagrams and a Subcubic Algorithm

In this section, we assume that R and B are sets of points in the plane and that d is given by the Euclidean distances between pairs of points. We shall closely adapt Vaidya's $O(|V|^{5/2} \log |V|)$ algorithm for the minimum-cost bipartite *perfect* matching problem with Euclidean costs [11]. This algorithm has also been adapted to solve the transportation problem with Euclidean costs [1]. Recall that our problem of interest is a maximum-cost bipartite, non-necessarily perfect matching problem with non-Euclidean costs (in fact, some costs are positive and some others are negative). However, the original Euclidean costs together with Lemma 2.1 will allow us to compute δ using the same data structures as in [11].

It turns out that the computation of the minimum slack δ is equivalent to the computation of the minimum distance between pairs of circles with disjoint interiors, one centered at $r_i \in F$ with radius $w(r_i)$, the other centered at $b_j \in S$ with radius $w(b_j)$.

Let P be a set of points in the plane with weights $w(p) \geq 0$ for each $p \in P$. A *weighted Voronoi diagram* divides the plane into $|P|$ possibly empty regions V_p for each $p \in P$, given by $V_p = \{q : d(q, p) - w(p) \leq d(q, p') - w(p') \text{ for all } p' \in P\}$. A weighted Voronoi diagram can be constructed and preprocessed in time $O(|P| \log |P|)$ so that, given any point q, it is possible to find $p \in P$ such that $q \in V_p$ in time $O(\log |P|)$ [4, 7]. In this case, we denote the point p by $\text{nearest}(q, P)$ and the edge $\{q, p\}$ by $\text{shortest}(q, P)$. Similarly, for $P_1, P_2 \subseteq P$, we denote by $\text{shortest}(P_1, P_2)$ the edge $\arg\min\{d(p_1, p_2) - w(p_1) - w(p_2) : p_1 \in P_1, p_2 \in P_2\}$.

Let $h = \lceil \sqrt{n} \rceil$ and consider an iteration of the Hungarian method with reduced costs. After computing the set S of bad vertices, we partition it into S_1 and S_2, ensuring that $|S_2| \leq h$. We also partition F into $F_1, \ldots, F_h$, ensuring that each part has cardinality $\leq h$. We compute the following data structures:

1. A weighted Voronoi diagram for S_1 with weights w in time $O(m \log m)$.
2. A minimum heap H_1 containing the edge $\{r, b\} = \text{shortest}(r, S_1)$ for each $r \in F$ with priority $d(r, b) - w(r) - w(b)$. Each of the $n = |F|$ edges can be computed in time $O(\log m)$ using the weighted Voronoi diagram for S_1. The total time is $O(n \log m)$

3. A weighted Voronoi diagram for each $F_1, \dots, F_h$ with weights w. Since $|F_i| \leq h$, this takes time $O(h^2 \log h) = O(n \log n)$
4. A minimum heap H_2 containing the edge $\{r, b\} = \text{shortest}(b, F_i)$ for each $b \in S_2$ and each F_i with priority $d(r, b) - w(r) - w(b)$. Each of the $h|S_2| \leq h^2$ edges can be computed in time $O(\log h)$ using the weighted Voronoi diagram for F_i. The corresponding heap can be constructed in time $O(h^2) = O(n)$. The total time is $O(h^2 \log h) = O(n \log n)$.

We can compute δ and a corresponding edge $\{r, b\}$ by examining H_1 and H_2 in time $O(\log n)$. The previously computed data structures need to be updated as follows:

Delete r from F: Assume $r \in F_i$. The weighted Voronoi diagram for F_i needs to be recomputed. This takes time $O(\sqrt{n} \log n)$. We also need to recompute $\text{shortest}(b, F_i)$ for each $b \in S_2$ and update the minimum heap H_2. This also takes time $O(\sqrt{n} \log n)$.

Insert b into S_2: We need to compute $\text{shortest}(b, F_i)$ for each $1 \leq i \leq h$ and insert it into the minimum heap H_2. This can be done in time $O(\sqrt{n} \log n)$.

Flush S_2: If $|S_2| = h$, then move all vertices from S_2 into S_1. Recompute $\text{shortest}(r, S_1)$ for each $r \in F$. This is done in time $O(n \log m)$ using the weighted Voronoi diagram for S_1.

Since an iteration of the Hungarian method with reduced costs has $O(n)$ deletions and insertions, the total time spent in the first two of these updates is $O(n\sqrt{n} \log n)$. Since S_2 can reach size h at most h times during an iteration, the total time spent in the last update is also $O(n\sqrt{n} \log m)$. Since there are at most m iterations, we obtain our main result.

▶ **Theorem 4.1.** *The minimum-cost edge cover problem with Euclidean costs on a complete bipartite graph $G = (V, E)$ can be solved in time $O(|V|^{2.5} \log |V|)$.*

5 Conclusions and Further Work

Our subcubic algorithm for the minimum-cost edge cover problem with Euclidean costs is based on the linear programming result in Lemma 2.1, which allowed us to use the techniques of [11]. We implemented four algorithms (Hungarian vs Voronoi and one bad vertex vs all). The versions starting with one bad vertex were faster than those starting with all bad vertices. Moreover, the Hungarian method seems to run in *quadratic* time on random instances.

References

1 D. S. Atkinson and P. M. Vaidya. Using geometry to solve the transportation problem in the plane. *Algorithmica*, 13(5):442–461, May 1995. doi:10.1007/BF01190848.

2 Justin Colannino, Mirela Damian, Ferran Hurtado, Stefan Langerman, Henk Meijer, Suneeta Ramaswami, Diane Souvaine, and Godfried Toussaint. Efficient many-to-many point matching in one dimension. *Graphs and Combinatorics*, 23(1):169–178, 2007. doi:10.1007/s00373-007-0714-3.

3 William J. Cook, William H. Cunningham, William R. Pulleyblank, and Alexander Schrijver. *Combinatorial Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley, 1997.

4 Herbert Edelsbrunner, Leonidas J. Guibas, and Jorge Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15(2):317–340, 1986. doi:10.1137/0215023.

5 Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965. doi:10.4153/CJM-1965-045-4.

6 Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, Apr 1972. doi:10.1145/321694.321699.

7 Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2(1):153, Nov 1987. doi:10.1007/BF01840357.

8 H. W. Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97, 1955. doi:10.1002/nav.3800020109.

9 James Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957. doi:10.1137/0105003.

10 Alex Pothen, S. M. Ferdous, and Fredrik Manne. Approximation algorithms in combinatorial scientific computing. *Acta Numerica*, 28:541–633, 2019. doi:10.1017/S0962492919000035.

11 Pravin M. Vaidya. Geometry helps in matching. *SIAM Journal on Computing*, 18(6):1201–1225, 1989. doi:10.1137/0218080.

Ruler Wrapping

Travis Gagie[1], Mozhgan Saeidi[1], and Allan Sapucaia[2]

1 **Faculty of Computer Science, Dalhousie University, Canada**
`firstname.lastname@dal.ca`

2 **Institute of Computing, University of Campinas, Brazil**
`allansapucaia@gmail.com`

Abstract

In 1985 Hopcroft, Joseph and Whitesides showed it is NP-complete to decide whether a carpenter's ruler with segments of given positive lengths can be folded into an interval of at most a given length, such that the folded hinges alternate between 180 degrees clockwise and 180 degrees counter-clockwise. At the open-problem session of 33rd Canadian Conference on Computational Geometry (CCCG '21), O'Rourke proposed a natural variation of this problem called *ruler wrapping*, in which all folded hinges must be folded the same way. In this paper we show O'Rourke's variation has a linear-time solution.

1 Introduction

Problems about carpenters' rulers are a staple of computational geometry. For example, in 1985 Hopcroft, Joseph and Whitesides [3] posed the following question: can a carpenter's ruler whose segments have given positive lengths be *folded* into an interval of at most a given length, with folded hinges alternating between 180 degrees clockwise and 180 degrees counter-clockwise (segments of the ruler having width 0 and folds being points)? They showed this problem is NP-complete in the weak sense via a reduction from Partition, illustrated in Figure 1; gave a pseudo-polynomial algorithm for it; and gave a linear-time 2-approximation algorithm. Călinescu and Dumitrescu [1] later gave an FPTAS for it.

, as illustrated in Figure 1

At the open-problem session of the 33rd Canadian Conference on Computational Geometry (CCCG '21), Professor Joseph O'Rourke proposed a natural variation of this problem, in which all folded hinges must be folded the same way (either all 180 degrees clockwise or all 180 degrees counter-clockwise); he named this variation *ruler wrapping* and, as far as we know, it had not been considered before.

2 A quadratic algorithm

Suppose the ruler has n segments and we lay it out flat, considering the hinges from left to right and pretending (for convenience) that there is a hinge 0 at the left end and a hinge n at the right end. For each hinge, we define the *wrapping length* at that hinge to be the smallest length into which the segments to its left can be wrapped such that we can still fold the hinge.

If we wrap the segments to the left of hinge i into its wrapping length and then fold it, the wrapped segments trace an arc with apex (x_i, y_i), where x_i is the position of the hinge (the sum of the lengths of the segments to its left) and y_i is its wrapping length. Figure 2 shows the arcs for the first five hinges (including hinge 0) of the ruler with segments of lengths 5, 6, 3, 4, 8, 6, 2, 1, 8 and 5, with the red dots marking the apexes. Notice that if we fold hinge 2 in Figure 2 then we cannot fold hinge 3, because $x_2 + y_2 = 17 > 14 = x_3$, and if we fold

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

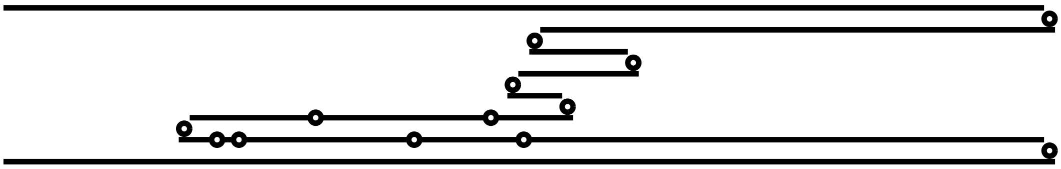


Figure 1 Hopcroft et al. showed that a multiset $M = \{m_1, \ldots, m_n\}$ of positive positive numbers can be partitioned into two subsets with equal sums, if and only a carpenter's ruler with segments of lengths $(m_1 + \cdots + m_n), (m_1 + \cdots + m_n)/2, m_1, \ldots, m_n, (m_1 + \cdots + m_n)/2, (m_1 + \cdots + m_n)$ can be folded — with folded hinges alternating between 180 degrees clockwise and 180 degrees counter-clockwise — into an interval of length $(m_1 + \cdots + m_n)$. For example, $\{5, 6, 3, 4, 8, 6, 2, 1, 8, 5\}$ can be partitioned into two subsets each summing to 24 if and only if a carpenter's ruler with segments of lengths $48, 24, 5, 6, 3, 4, 8, 6, 2, 1, 8, 5, 24, 48$ can be folded into an interval of length 48.

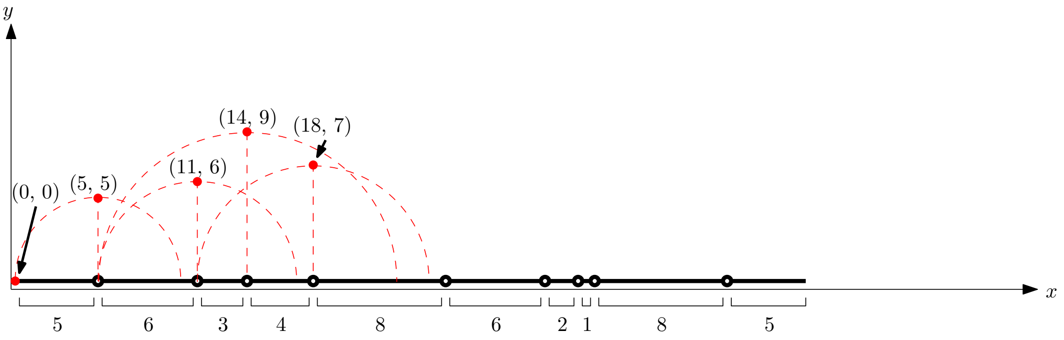


Figure 2 Arcs for the first five hinges (including hinge 0) of the ruler with segments of lengths 5, 6, 3, 4, 8, 6, 2, 1, 8 and 5.

hinge 3 then we cannot fold hinge 4, because $x_3 + y_3 = 23 > 18 = x_4$. When $x_h + y_h \leq x_i$, on the other hand, we can fold the segments to the left of hinge h into its wrapping length y_h, fold hinge h, and then fold hinge i, so $y_i \leq x_i - x_h$.

▶ **Observation 2.1.** *For $i > 0$, the wrapping length y_i of hinge i is $x_i - x_h$, where hinge h is the last previous hinge such that $x_h + y_h \leq x_i$.*

Figure 2 suggests a simple quadratic-time dynamic program for computing the wrapping lengths: set $x_0 = 0$ and $y_0 = 0$ (because hinge 0 at the left end of the ruler has no segments to its left); for i from 1 to n, set x_i to the position of hinge i and set y_i to

$$x_i - \max\{x_h \ : \ h \leq i,\ x_h + y_h \leq x_i\}$$

It may seem at first that we can simply choose the last arc that ends before or at each hinge, but Figure 3 shows we are sometimes better off choosing an arc that ends earlier: the blue arc ends closer to the end of the ruler but the green arc has a later center and yields a smaller wrapping length.

It may also seem at first that the wrapping length y_n of hinge n should always be the smallest length into which we can wrap the whole ruler, but Figure 4 shows this is guaranteed only when we require the distance from the last folded hinge to the end of the ruler to be at least the distance between the last two folded hinges. (We display the wrappings as triangular spirals here to make it easier to show the segments' lengths.) When we do not require this, we can scan the hinges in linear time to find the one that minimizes the maximum of its

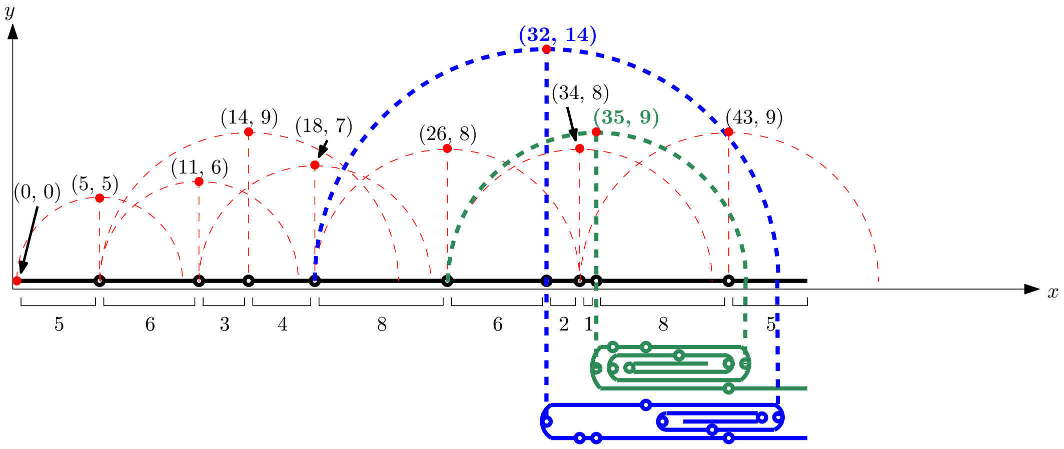


Figure 3 A case in which choosing the last arc **(blue)** that ends before or at hinge n yields a larger wrapping length than choosing an arc **(green)** that ends earlier.

wrapping length and the distance to the end of the ruler; that maximum is the smallest length into which we can wrap the ruler when that hinge is the last one folded. Summing up, so far we have the following result:

▶ **Theorem 2.2.** *Given the positive lengths of the n segments of a carpenter's ruler, in $O(n^2)$ time we can compute the shortest length into which it can be wrapped.*

3 An $O(n \log n)$-time algorithm

We can use a range-minimum data structure to reduce the running time in Theorem 2.2 to $O(n \log n)$, but this complicates the implementation somewhat. Instead, recall the $O(n \log n)$-time array-based algorithm for finding a longest increasing subsequence of a list $L[1..n]$ of numbers, which Fredman [2] analyzed and attributed to Knuth [4]. That algorithm starts with an array $T[1..n]$ with $T[1]$ set to $L[1]$ and the other entries empty; for $i > 1$, it compares $L[i]$ against the rightmost non-empty value $T[k]$ in T and, if $L[i] > T[k]$, sets $T[k+1] = L[i]$; otherwise, it performs a binary search in $T[1..k]$ — which is always sorted — to find the leftmost value $T[h] > L[i]$ and sets $T[h] = L[i]$. By induction, this maintains the invariant that each $T[j]$ is always the smallest value that ends an increasing subsequence of $L[1..i]$ of length j.

A key idea behind Knuth's algorithm is that if we find $T[j] > L[i]$ then we need not keep the current value of $T[j]$ because any extension of a subsequence ending with $T[j]$ is also an extension of a subsequence ending with $L[i]$. We can apply a similar idea to obtain an $O(n \log n)$-time array-based algorithm for ruler wrapping: we start with an array $P[0..n]$ of pairs with $P[0] = (x_0, y_0) = (0, 0)$ and the other entries empty; for $i \geq 1$, we

1. add the length of the ith segment of the ruler to x_{i-1} to obtain x_i,
2. perform a binary search in the non-empty prefix $P[0..k]$ of P — which is always sorted both by x-coordinate and by sum $x + y$ — to find the rightmost pair (x_h, y_h) with $x_h + y_h \leq x_i$ (there always is such a pair, since $x_0 + y_0 = 0$),
3. set $y_i = x_i - x_h$,
4. scan leftward from $P[k]$ discarding pairs (x_j, y_j) with $x_j + y_j \geq x_i + y_i$,
5. insert (x_i, y_i) immediately to the right of the rightmost undiscarded pair.

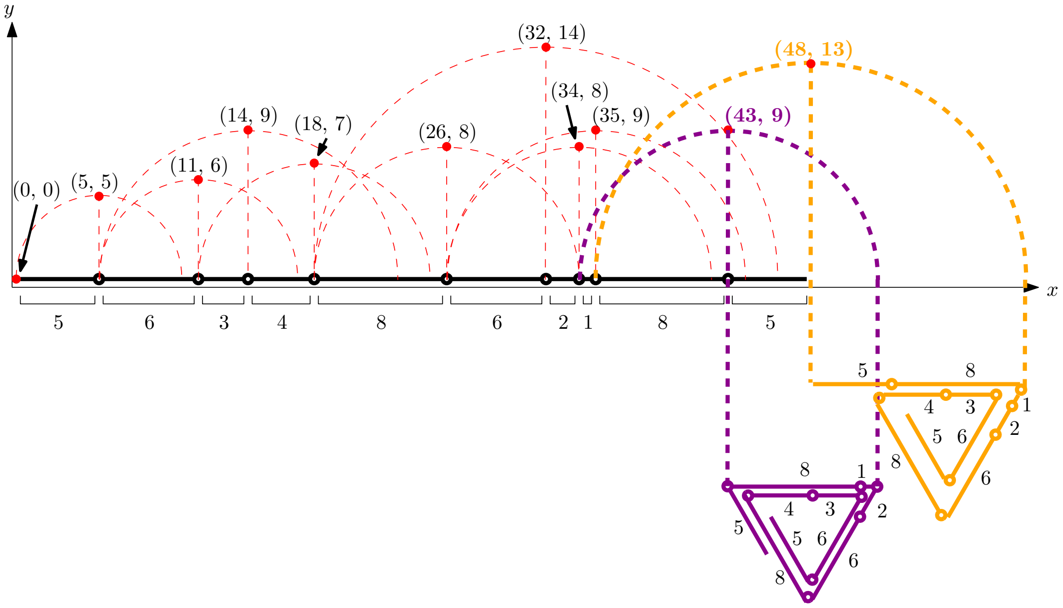


Figure 4 A case in which the wrapping length of hinge n **(orange)** is more than the shortest length into which we can wrap the whole ruler **(magenta)** when we do not require the distance from the last folded hinge to the end of the ruler to be at least the distance between the last two folded hinges.

To see why $P[0..k]$ is always sorted both by x-coordinate and by sum $x+y$, suppose it is sorted before we process the length of the ith segment of the ruler, and consider that x_i is larger than any previous x-coordinate and we discard all the pairs (x_j, y_j) with $x_j + y_j \geq x_i + y_i$. To see why we can discard any such pair (x_j, y_j), consider that we will never choose the arc centered at $x_j < x_i$ that ends at $x_j + y_j \geq x_i + y_i$, when we can choose the arc centered at x_i. Finally, to see why processing the length of the ith segment of the ruler takes us $O(\log n)$ amortized time, consider that the binary search takes $O(\log n)$ time, we can stop discarding pairs as soon as we encounter one that sums to less than $x_i + y_i$, and we can charge each pair we discard to the segment of the ruler for which we inserted it.

We wrote earlier that when we do not require the distance from the last folded hinge to the end of the ruler to be at least the distance between the last two folded hinges, we can scan the hinges in linear time to find the one that minimizes the maximum of its wrapping length and the distance to the end of the ruler. We can no longer scan all of the hinges easily if we discard some pairs, but we claim that we cannot discard the pair for what should be the last folded hinge. To see why, consider that our algorithm works online (at each hinge we compute the wrapping length of the prefix of the ruler ending at that hinge) and let hinges g and h be the last folded hinges in a shortest wrapping; if a segment of length $(x_h - x_h) - (x_n - x_h)$ were appended to the ruler (which is allowed in the online setting) then we would need to have (x_h, y_h) in the array in order to compute $y_{n+1} = x_{n+1} - x_h$. If a segment of length 3 were appended to the ruler in Figure 4, for example, then because $x_9 + y_9 = 43 + 9 \leq x_{11} = 51$, we would have $y_{11} = x_{11} - x_9 = 8$ — so our algorithm cannot discard (x_9, y_9).

▶ **Theorem 3.1.** *Given the positive lengths of the n segments of a carpenter's ruler, in $O(n \log n)$ time we can compute the shortest length into which it can be wrapped.*

Table 1 The contents of our array P while processing our running example.

step	$P[0]$	$P[1]$	$P[2]$	$P[3]$	$P[4]$	$P[5]$	$P[6]$	$P[7]$	$P[8]$	$P[9]$	$P[10]$
0	$(0,0)$										
1	$(0,0)$	$(5,5)$									
2		$(5,5)$	$(11,6)$								
3		$(5,5)$	$(11,6)$	$(14,9)$							
4			$(11,6)$	$(14,9)$	$(18,7)$						
5					$(18,7)$	$(26,8)$					
6					$(18,7)$	$(26,8)$	$(32,14)$				
7						$(26,8)$	$(34,8)$				
8						$(26,8)$	$(34,8)$	$(35,9)$			
9							$(34,8)$	$(35,9)$	$(43,9)$		
10								$(35,9)$	$(43,9)$	$(48,13)$	

4 A linear algorithm

Fredman showed the number of comparisons Knuth's algorithm performs is the best possible for finding a longest increasing subsequence, to within a linear term. Rather surprisingly, however, we can further reduce the running time in Theorem 3.1 to $O(n)$. To see why, consider that for ruler wrapping, because the segments' lengths are positive, the x_is are themselves an increasing sequence. Therefore, if we are currently processing x_i and we have (x_g, y_g) and (x_h, y_h) in our array with $x_g < x_h < x_i$ and $x_h + y_h \leq x_i$, then not only will we not choose the arc centered on x_g to compute y_i, we will never choose it to compute any other y-coordinate in the future, either. It follows that instead of using binary search to find the rightmost pair (x_h, y_h) in our array with $x_h + y_h \leq x_i$, we can scan rightward from the first non-empty cell in our array, discarding pairs until we find that rightmost pair (x_h, y_h) with $x_h + y_h \leq x_i$. Again, we charge each pair we discard to the segment of the ruler for which we inserted it. Table 1 shows the contents of our array while we process our running example. We note that the rightmost cell $P[10]$ is always empty in this example, but its presence guarantees we have space for all the pairs even if we never discard pairs on the right.

Even when we do not require the distance from the last folded hinge to the end of the ruler to be at least the distance between the last two folded hinges, during our rightward scan we cannot accidentally discard a pair we might need later, this time because we discard only pairs (x_g, y_g) with $x_g + y_g \leq x_i$ — so the distance from hinge g to the end of the ruler is at least hinge g's wrapping length, which is the distance from the previous folded hinge if g is the last folded one.

▶ **Theorem 4.1.** *Given the positive lengths of the n segments of a carpenter's ruler, in $O(n)$ time we can compute the shortest length into which it can be wrapped.*

5 Acknowledgments

Many thanks to Joseph O'Rourke for proposing ruler wrapping; to Joseph Mitchell, David Wagner and the other participants of the CCCG '21 open-problem session for helpful discussions; and to the reviewers for their constructive criticism. The first author thanks NSERC for his Discovery Grant, RGPIN-2020-07185, which funded this research; his mother, Meg Gagie, for proofreading; and his CSCI 3110 class, for trying so desperately to avoid building range-minimum data structures that they implemented Knuth's algorithm instead — and thus inadvertently taught it to him.

References

1 Gruia Călinescu and Adrian Dumitrescu. The carpenter's ruler folding problem. *Combinatorial and Computational Geometry*, 52:155, 2005.

2 Michael L. Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11(1):29–35, 1975.

3 John Hopcroft, Deborah Joseph, and Sue Whitesides. On the movement of robot arms in 2-dimensional bounded regions. *SIAM Journal on Computing*, 14(2):315–333, 1985.

4 Donald E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, 1973.

Approximating the discrete center line segment in linear time

Joachim Gudmundsson[1] **and Yuan Sha**[2]

1 **The University of Sydney**
joachim.gudmundsson@sydney.edu.au

2 **The University of Sydney**
ysha3185@sydney.edu.au

Abstract

Let P be a set of n points in the plane. The discrete center line segment of P is the line segment bounded by two points in P such that the maximum distance from any point in P to it is minimized. Previously, an $O(n^2)$ time $O(n^2)$ space algorithm is given [6]. In this paper, we give a $(1+\varepsilon)$-approximation algorithm for the discrete center line segment problem which runs in $O(n + \frac{1}{\varepsilon^4}\log\frac{1}{\varepsilon})$ time and uses linear space.

1 Introduction

The general discrete p-center problem is a fundamental problem in clustering and facility location. The problem is shown to be NP-complete by Fowler et al. in [8]. Approximation hardness results are also known. Megiddo and Supowit [11] showed that the problem has no polynomial time 1.154-approximation algorithm unless P=NP. By using a reduction from the planar vertex cover problem, Feder and Greene [7] showed that there is no polynomial time 1.822-approximation algorithm for the problem (which they referred to as the central clustering problem) unless P=NP.

Due to the hardness of the general discrete p-center problem, the problem for constant p where $p = 1$ or 2 is also considered in the literature. The discrete 1-center problem can be solved easily in $O(n \log n)$ time by using farthest-neighbour Voronoi diagram. The discrete 2-center problem is first solved in near-quadratic time by Hershberger and Suri in [10]. Later Aggarwal et al. [1] gave a much improved $O(n^{4/3}\log^5 n)$ time algorithm for the discrete 2-center problem. They also noticed that the discrete 2-center problem is harder to solve efficiently than the standard (continuous) 2-center problem. The discrete center line segment problem is connected to the discrete 2-center problem in that the former searches for a center segment which has a minimum maximum distance from a point to it while the latter searches for two centers that minimize the maximum distance from a point to them.

In [9], Gudmundsson et al. gave a strong linear-time approximation scheme for the geometric minimum diameter spanning tree problem and the discrete 2-center problem. A strong linear-time approximation scheme is a $(1+\varepsilon)$ approximation scheme with a running time of the form $O^*(n + \frac{1}{\varepsilon^c})$ [4].

In this paper, we consider the following *discrete center line segment problem*: given a set P of n points in $\mathbb{R}^2$, find a segment bounded by two points in P such that the maximum distance from a point in P to the segment is minimized. This problem is proposed by Daescu and Teo [6], who give an $O(n^2)$ time, $O(n^2)$ space exact algorithm. In this paper, we give a $(1+\varepsilon)$-approximation algorithm for the discrete center line segment problem that runs in $O(n + \frac{1}{\varepsilon^4}\log\frac{1}{\varepsilon})$ time and uses linear space.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

1.1 Our Approach

The main idea of our approach is to compute an approximate point set of the input point set P whose approximate center segment is easier to compute. This is done in two steps. We first compute a $(1+\varepsilon)$-approximate diametral point pair of P. Next an approximate convex hull of P is computed. The approximate convex hull algorithm in [3] uses strips. We apply the algorithm in [3] by requiring the orientation of the strips to be aligned with the line through the approximate diametral point pair. The approximate point set is obtained from P by shifting every point outside the approximate convex hull by a small enough distance until it lies on the approximate convex hull. By the orientation requirement, we are assured that we can get a $(1+\varepsilon)$-approximate center segment of P from a $(1+\varepsilon)$-approximate center segment of the approximate point set.

The convex hull of the approximate point set has only $O(\frac{1}{\varepsilon})$ vertices. Combined with other observations, one only need to consider a small number of candidate segments which only depends on ε, in order to get a $(1+\varepsilon)$-approximate center segment of the approximate point set.

We further use techniques to reduce the number of candidate segments and search the farthest point to a candidate segment efficiently.

2 Preliminaries

Let P be a set of n points in $\mathbb{R}^2$. A segment of P is a segment bounded by two points of P. A *center segment* of P is a segment of P such that the maximum distance from a point in P to this segment is minimized. For convenience, we call a segment bounded by two points in P as a segment of P. For any two points a, b in P, let ab denote the line segment joining a and b and let $\bar{ab}$ denote the line going through a and b. For any point p in $\mathbb{R}^2$, let $d(p, ab)$ denote the distance from p to segment ab and let $d_\perp(p, ab)$ denote the distance from p to line $\bar{ab}$. Let $|ab|$ denote the length of segment ab.

The *diameter* of P is the maximum distance between two points of P. A diametral point pair of P is a pair of points in P that realize the diameter. A pair of P is said to be α-approximate diametral point pair of P if the distance between the pair is at least $1/\alpha$ of the diameter of P. The *width* of P along an orientation (direction) is the distance between parallel lines that are support of P and perpendicular to the orientation. Let d^* denote the maximum distance from a point in P to a center segment of P. A segment of P is called an α-approximate center segment if the maximum distance from a point in P to the segment is at most αd^*.

The convex hull of a point set is the minimum convex set containing all the points. We use $CH(P)$ to denote the convex hull of P.

All the precision parameters ε', ε_1, $\ldots$ are constants between 0 and 1.

3 The algorithm

The algorithm consists of several parts. The first part concerns with computing an approximate point set of the input point set and is presented in Section 3.1. The second part concerns with computing a $(1+\varepsilon)$-approximate center segment of the approximate point set and is presented in Section 3.2. Details of the third part are omitted in this abstract.

3.1 Compute an approximate point set of P

We start by finding an orientation for computing an approximate convex hull of P. The width of P along a desirable orientation is some constant times d^*. We first give two facts.

▶ **Fact 3.1.** Let p, q be any two points in $\mathbb{R}^2$. ab is a segment in the plane. Then $d(p, ab) \leqslant |pq| + d(q, ab)$ and $d_\perp(p, ab) \leqslant |pq| + d_\perp(q, ab)$.

The following lemma is the main lemma of this section. The proof can be found in Appendix **??**.

▶ **Lemma 3.2.** *The width of P along the orientation perpendicular to the line through a $(1+\varepsilon)$-approximate diametral point pair of P is at most 10 times d^*, where d^* is the maximum distance from a point in P to a center segment of P.*

We can use the approximate diameter algorithm of [3] to get an approximate diametral point pair of P. Let (a, b) be the computed pair. Rotate P around the origin until segment ab is parallel to the y-axis. Let P' be P after rotation. We compute an approximate convex hull of P' in $O(n + \frac{1}{\varepsilon'})$ time by using the algorithm in [3], where ε' is the precision parameter. The strips which the algorithm uses are parallel to the y-axis and have width ε' times the width of P' along the x-axis.

The computed approximate convex hull $\tilde{CH}(P')$ has $O(\frac{1}{\varepsilon'})$ vertices [3]. We set the precision parameter ε' to $\varepsilon_1/10$. By Lemma 3.2, the width of the strips is at most $\varepsilon_1 \cdot d^*$. For any point in P' that lies outside $\tilde{CH}(P')$, shift it along the positive or negative x-axis direction by at most $\varepsilon_1 d^*$ until it lies on the boundary of $\tilde{CH}(P')$. See Figure 1 for an illustration. Let $\tilde{P}$ denote P' after the shifting. For any point in $\tilde{P}$ that is a shifted point, we keep track of its original point in P'. Thus each segment of $\tilde{P}$ corresponds to a segment of P'. Let $\tilde{c}\tilde{e}$ be a center segment of $\tilde{P}$. Assume that the original points of $\tilde{c}$ and $\tilde{e}$ are c' and e', respectively. $c'e'$ is a $(1 + 4\varepsilon_1)$-approximate center segment of P', as stated in the following lemma.

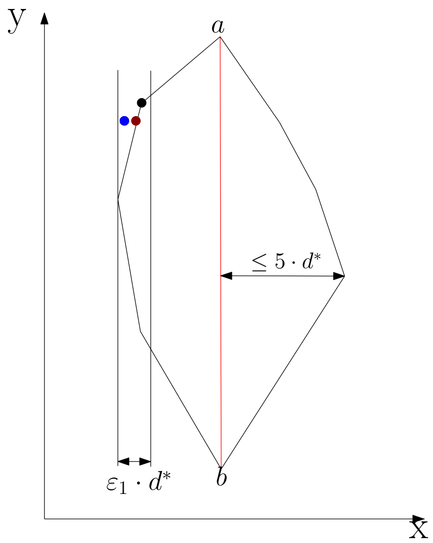


Figure 1 The blue point is in P' and lies outside $\tilde{CH}(P')$. We shift it along the positive x-axis direction by at most $\varepsilon_1 d^*$ until it lies on the boundary of $\tilde{CH}(P')$. The red point is the shifted blue point.

▶ **Lemma 3.3.** *The maximum distance from a point in P' to $c'e'$ is at most $(1+4\varepsilon_1)\cdot d^*$.*

Because of Lemma 3.3, we can compute a $(1+\varepsilon_2)$-approximate center segment of $\tilde{P}$ to get a $(1+\varepsilon)$-approximate center segment of P. Note that we can get $\tilde{P}$ from P' in $O(n+\frac{1}{\varepsilon_1})$ time by sweeping P' and $\tilde{CH}(P')$ along the x-axis.

▶ **Lemma 3.4.** *In $O(n+\frac{1}{\varepsilon_1})$ time, we can compute an approximate point set $\tilde{P}$ of P'. Let $\varepsilon_1=\varepsilon_2=\frac{\varepsilon}{6}$. The segment of P' that corresponds to a $(1+\varepsilon_2)$-approximate center segment of $\tilde{P}$ is a $(1+\varepsilon)$-approximate center segment of P'.*

3.2 Compute a $(1+\varepsilon)$-approximate center segment of $\tilde{P}$

Let $\tilde{d}$ be the maximum distance from a point in $\tilde{P}$ to a center segment of $\tilde{P}$. To compute a $(1+\varepsilon)$-approximate center segment of $\tilde{P}$, we need a good estimation of $\tilde{d}$. We can get one by examining the diagonals of $CH(\tilde{P})$. We call a diagonal of $CH(\tilde{P})$ with the minimum maximum distance to a point in $\tilde{P}$ among all the diagonals a *center diagonal* of $CH(\tilde{P})$. The following lemma shows that the maximum distance from a point in $\tilde{P}$ to a center diagonal is at most $2\tilde{d}$. The proof can be found in Appendix **??**.

▶ **Lemma 3.5.** *Let $\mathcal{G}$ be a center diagonal of $CH(\tilde{P})$. The maximum distance from a point in $\tilde{P}$ to $\mathcal{G}$ is at most $2\tilde{d}$.*

There are $O(\frac{1}{\varepsilon_1^2})$ diagonals, but we only need to consider $O(\frac{1}{\varepsilon_1})$ diagonals to find the center diagonal. We first show some monotone properties of the diagonals. Let $m=|CH(\tilde{P})|$ and let $\tilde{v}_1,\tilde{v}_2,\ldots,\tilde{v}_m$ be a counterclockwise ordering of the hull vertices along $CH(\tilde{P})$. Consider all diagonals with vertex $\tilde{v}_i$ as one end. Let $\tilde{v}_j$, $i<j<i+m$, be the other end of the diagonal. Let $ccw(i,j)$ denote the counterclockwise chain from $\tilde{v}_i$ to $\tilde{v}_j$ (with wrap around) on the boundary of $CH(\tilde{P})$ and $cw(i,j)$ denote the clockwise chain from $\tilde{v}_i$ to $\tilde{v}_j$(with wrap around) on the boundary. We can prove the following monotone properties.

▶ **Lemma 3.6.** *Let $f(i,j)$ be the maximum distance from a vertex on $ccw(i,j)$ to diagonal $\tilde{v}_i\tilde{v}_j$ and $g(i,j)$ be the maximum distance from a vertex on $cw(i,j)$ to $\tilde{v}_i\tilde{v}_j$. Then*[1]

(a) $f(i,j)\leqslant f(i,j+1)$.
(b) $g(i,j)\geqslant g(i,j+1)$.
(c) $f(i+1,j)\leqslant f(i,j)$.
(d) $g(i+1,j)\geqslant g(i,j)$

Let $f(i)$ be the minimum j such that $f(i,j)$ is greater than $g(i,j)$. Then

(e) $f(i+1)\geqslant f(i)$.

$\max\{f(i,j),g(i,j)\}$ is the maximum distance from a hull vertex to diagonal $\tilde{v}_i\tilde{v}_j$. Fix i, $\max\{f(i,j),g(i,j)\}$ is a unimodal function of j and its minimum takes place at either $j=f(i)$ or $j=f(i)-1$. After we get $f(i)$, we can search for $f(i+1)$ starting from $j=f(i)$ rather than from $j=i+2$, by Lemma 3.6(e). In this way, we only consider $O(\frac{1}{\varepsilon_1})$ candidate diagonals in search of a center diagonal.

We now discuss how to compute the farthest point and the maximum distance in $\tilde{P}$ to a segment $\tilde{p}\tilde{q}$ of $\tilde{P}$. We have the following lemma.

[1] $i+1$, $j+1$ should take modulo m, we omit for brevity.

▶ **Lemma 3.7.** *The farthest point in $\tilde{P}$ to a segment $\tilde{p}\tilde{q}$ of $\tilde{P}$ is a vertex of $CH(\tilde{P})$.*

Thus we only consider vertices of $CH(\tilde{P})$ in search of the farthest point in $\tilde{P}$. By using the data structure in [5], we can compute the farthest point to $\tilde{p}\tilde{q}$ in $O(\log^2 \frac{1}{\varepsilon_1})$ time.

Let $\bar{d}$ denote the maximum distance from a point in $\tilde{P}$ to the center diagonal of $CH(\tilde{P})$. For each of the $O(\frac{1}{\varepsilon_1})$ diagonals we consider, $O(\frac{1}{\varepsilon_1^2})$ time is spent on computing the maximum distance from a $CH(\tilde{P})$ vertex to it. We can compute the center diagonal and $\bar{d}$ in $O(\frac{1}{\varepsilon_1}\log^2\frac{1}{\varepsilon_1})$ time.

By Lemma 3.5, $\bar{d}$ is a 2-approximation of $\tilde{d}$. If we lay squares with side length $2\bar{d}$ such that the square centers are at vertices of $CH(\tilde{P})$, the center segment of $\tilde{P}$ must have both ends inside the squares, as suggested by the following lemma.

▶ **Lemma 3.8.** *The center segment of $\tilde{P}$ must have both ends inside squares that have centers at vertices of $CH(\tilde{P})$ and have radius $\bar{d}$.*

Proof. Let $\tilde{o}\tilde{p}$ be a center segment of $\tilde{P}$. Assume the extension of $\tilde{o}\tilde{p}$ out of $\tilde{o}$ intersects edge $\tilde{f}\tilde{g}$ of $CH(\tilde{P})$ and the extension of $\tilde{o}\tilde{p}$ out of $\tilde{p}$ intersects edge $\tilde{h}\tilde{i}$ of $CH(\tilde{P})$. See Figure 2.

Either $\angle\tilde{f}o'\tilde{o}$ or $\angle\tilde{g}o'\tilde{o}$ is at least $\pi/2$. WLOG, assume $\angle\tilde{f}o'\tilde{o} \geqslant \pi/2$. The distance from $\tilde{f}$ to $\tilde{o}\tilde{p}$ equals $|\tilde{f}\tilde{o}|$. Thus $|\tilde{f}\tilde{o}| \leqslant \tilde{d} \leqslant \bar{d}$. $\tilde{o}$ lies within the square that has center at $\tilde{f}$ and has radius $\bar{d}$. In the same way, we can show that $\tilde{p}$ lies within either the square with center at $\tilde{i}$ or the square with center at $\tilde{h}$. ◀

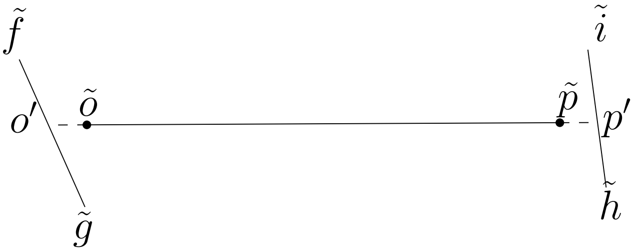


Figure 2 $\tilde{o}$ is within the square centered at $\tilde{f}$ and $\tilde{p}$ is within the square centered at $\tilde{h}$

Now we can find a $(1+\varepsilon_2)$-approximate center segment of $\tilde{P}$.

▶ **Lemma 3.9.** *A $(1+\varepsilon)$-approximate center segment for P can be computed in $O(n+\frac{1}{\varepsilon^7})$ time.*

Proof. Lay a grid of cell side length $\frac{\varepsilon_2\bar{d}}{4\sqrt{2}}$ over the plane. Index all points in $\tilde{P}$ into cells of the grid. A square with center at a vertex of $CH(\tilde{P})$ and with radius $\bar{d}$ intersects $O(\frac{1}{\varepsilon_2^2})$ cells of the grid. Call the square with center at a vertex *the vertex's $\bar{d}$-square*. Call the intersection of the global grid and a vertex's $\bar{d}$-square the *vertex's grid*. For each cell of a vertex's grid, maintain one point indexed into the cell (if any) and call the point the *representative* of the cell. A vertex grid has $O(\frac{1}{\varepsilon_2^2})$ representatives. By Lemma 3.8, the center segment of $\tilde{P}$ must have each end inside a cell of a vertex grid. Let these two cells be A and B. The segment joining the representatives of A and B is a $(1+\varepsilon_2)$-approximate center segment of $\tilde{P}$. Since $CH(\tilde{P})$ has $O(\frac{1}{\varepsilon_1})$ vertices and each vertex has $O(\frac{1}{\varepsilon_2^2})$ cells in its grid, we can consider $O(\frac{1}{\varepsilon_1^2})$ pairs of vertex grids and for each pair consider $\frac{1}{\varepsilon_2^4}$ candidate segments, to get a $(1+\varepsilon_2)$-approximate center segment of $\tilde{P}$. By Lemma 3.7, we can compute the maximum distance from a point in $\tilde{P}$ to a candidate segment in $O(\frac{1}{\varepsilon_1})$ time. Thus given $CH(\tilde{P})$ and

$\bar{d}$, we can compute a $(1+\varepsilon_2)$-approximate center segment of $\tilde{P}$ in $O(\frac{1}{\varepsilon_1^3} \cdot \frac{1}{\varepsilon_2^4})$ time. With Lemma 3.4, we have proved the lemma. ◀

We can further reduce the running time. First, we can consider $O(\frac{1}{\varepsilon_2^3})$ candidate segments for a pair of vertex grids. Second, we only need to consider $O(\frac{1}{\varepsilon_1})$ pairs of vertex grids by using monotone properties similar to Lemma 3.6. Third, we can compute the farthest point to a segment in $O(\log \frac{1}{\varepsilon_1})$ time by using the data structure in [2]. We obtain the main result of the paper.

▶ **Theorem 3.10.** *A $(1+\varepsilon)$-approximate discrete center line segment for n points in the plane can be computed in $O(n + \frac{1}{\varepsilon^4} \log \frac{1}{\varepsilon})$ time and linear space.*

References

1 Pankaj K. Agarwal, Micha Sharir, and Emo Welzl. The discrete 2-center problem. *Discret. Comput. Geom.*, 20(3):287–305, 1998. doi:10.1007/PL00009387.

2 Boris Aronov, Prosenjit Bose, Erik D. Demaine, Joachim Gudmundsson, John Iacono, Stefan Langerman, and Michiel H. M. Smid. Data structures for halfplane proximity queries and incremental voronoi diagrams. *Algorithmica*, 80(11):3316–3334, 2018. doi:10.1007/s00453-017-0389-y.

3 Jon Louis Bentley, Mark G. Faust, and Franco P. Preparata. Approximation algorithms for convex hulls. *Commun. ACM*, 25(1):64–68, 1982. doi:10.1145/358315.358392.

4 Timothy M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. *Int. J. Comput. Geom. Appl.*, 12(1-2):67–85, 2002. doi:10.1142/S0218195902000748.

5 Ovidiu Daescu, Ningfang Mi, Chan-Su Shin, and Alexander Wolff. Farthest-point queries with geometric and combinatorial constraints. *Comput. Geom.*, 33(3):174–185, 2006. doi:10.1016/j.comgeo.2005.07.002.

6 Ovidiu Daescu and Ka Yaw Teo. The discrete median and center line segment problems in the plane. In Meng He and Don Sheehy, editors, *Proceedings of the 33rd Canadian Conference on Computational Geometry, CCCG 2021, August 10-12, 2021, Dalhousie University, Halifax, Nova Scotia, Canada*, pages 312–319, 2021.

7 Tomás Feder and Daniel H. Greene. Optimal algorithms for approximate clustering. In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 434–444. ACM, 1988. doi:10.1145/62212.62255.

8 Robert J. Fowler, Mike Paterson, and Steven L. Tanimoto. Optimal packing and covering in the plane are np-complete. *Inf. Process. Lett.*, 12(3):133–137, 1981. doi:10.1016/0020-0190(81)90111-3.

9 Joachim Gudmundsson, Herman J. Haverkort, Sang-Min Park, Chan-Su Shin, and Alexander Wolff. Facility location and the geometric minimum-diameter spanning tree. *Comput. Geom.*, 27(1):87–106, 2004. doi:10.1016/j.comgeo.2003.07.007.

10 John Hershberger and Subhash Suri. Finding tailored partitions. *J. Algorithms*, 12(3):431–463, 1991. doi:10.1016/0196-6774(91)90013-O.

11 Nimrod Megiddo and Kenneth J. Supowit. On the complexity of some common geometric location problems. *SIAM J. Comput.*, 13(1):182–196, 1984. doi:10.1137/0213014.

Efficiently Enumerating Scaled Copies of Point Set Patterns

Aya Bernstine[1] and Yehonatan Mizrahi[2]

1 School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel
aya.bernstine@mail.huji.ac.il

2 School of Computer Science and Engineering, The Hebrew University, Jerusalem, Israel
yehonatan.mizrahi@mail.huji.ac.il

Abstract

Problems on repeated geometric patterns in finite point sets in Euclidean space are extensively studied in the literature of combinatorial and computational geometry. Such problems trace their inspiration back to Erdős' original work on this topic. In this paper, we investigate the problem of finding scaled copies of any pattern within a set of n points, that is, the algorithmic task of efficiently enumerating all such copies. We initially focus on one particularly simple pattern of axis-parallel squares, and present an algorithm with an $O(n\sqrt{n})$ running time and $O(n)$ space for this task, involving various bucket-based and sweep-line techniques. Our algorithm's running time is worst-case optimal, as it matches the known lower bound of $\Omega(n\sqrt{n})$ on the maximum number of axis-parallel squares determined by n points in the plane, thereby solving an open question for more than three decades of realizing that bound for this pattern. We extend our result to an algorithm that enumerates all copies, up to scaling, of any full-dimensional fixed set of points in d-dimensional Euclidean space, that runs in time $O(n^{1+1/d})$ and space $O(n)$, matching the more general lower bound due to Elekes and Erdős.

Related Version arXiv:2112.14980

1 Introduction

The problems of geometric point pattern matching and the identification of repeated geometric patterns are fundamental computational problems with a myriad of applications [11, 9, 1, 13, 10, 7]. Such problems were motivated in part by questions regarding the maximal number of occurrences of a given pattern determined by a set of points, a field historically inspired by Erdős' well-known Unit Distance Problem (1946) regarding the maximal number of unit distance pairs induced by such sets [6]. Our paper approaches the computational problems of identifying patterns using computational geometric tools.

In this paper, we analyze the problem of identifying translated and scaled copies of *any* point set pattern in Euclidean space, where the scaling is applied identically in all axes. We begin with focusing on the problem of repeated patterns of squares having axis-parallel edges in the plane. As articulated in 1990 by van Kreveld and de Berg [12], the maximum possible number of axis-parallel squares determined by n points in the plane is $\Theta(n\sqrt{n})$ (attained, for example, in a regular $\sqrt{n} \times \sqrt{n}$ grid), and those can be enumerated in time $O(n\sqrt{n}\log n)$[1] and space $O(n)$.

[1] The analysis given throughout this paper of time and space complexities is based on the conventional word-RAM model of computation [8].

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

They show an extension to full-dimensional axis-parallel d-dimensional hypercubes in time $O(n^{1+1/d}\log n)$, with a gap separating this computational result from the lower bound of a maximum of $\Theta(n^{1+1/d})$ possible hypercubes. The latter combinatorial result was further extended by Elekes and Erdős [5], establishing a bound of $\Theta(n^{1+1/d})$ on the maximum number of copies of *any* full-dimensional pattern (i.e., a set of points that generates the vector space) in $\mathbb{Q}^d$. The computational aspect of it occurs in [3], providing an algorithm that works in time $O(n^{1+1/d}\log n)$ for the task of enumerating all such copies, exhibiting the same logarithmic-factor gap between the two results.

1.1 Our Results

Our main result of this paper is an efficient deterministic algorithm that enumerates all scaled copies of any fixed d-dimensional pattern. The treatment of such general patterns appeared, e.g., in [3], but [12] were the first to raise the question of whether it is computationally feasible to realize the combinatorial bound of $\Theta(n\sqrt{n})$ possible axis-parallel *squares*, thereby improving their algorithmic result. Our algorithm fully answers this question which was open for more than three decades. To this end, we use in our algorithm a reduction from arbitrary input points to points having ”compressed” coordinates, allowing the use of linear sorting methods. Second, we deploy a sweep-line sub-procedure that marks points forming a square, instead of searching those in a set, avoiding the logarithmic cost. Third, we relabel the sum and the difference of the input coordinates, in addition to the relabeling of the coordinates themselves. We show why the last step is crucial for the algorithm to succeed in Section 2.

Theorem: *Given a planar set P of points of size n, all axis-parallel squares defined by points from P can be enumerated in time $O(n\sqrt{n})$ and $O(n)$ space.*

Our main result for general patterns relies on the ideas from the previous theorem. Specifically, we relabel some affine transformations of the input coordinates, a relabeling that creates a representation of the points for the purpose of sweep-line scanning them.

Theorem: *Given a fixed set Q of points of full dimension in the d-dimensional Euclidean space, and a set P of points of size n, all scaled copies of Q determined by subsets of P can be enumerated in time $O(n^{1+1/d})$ and $O(n)$ space.*

The running time in this theorem matches the corresponding lower bound of the same magnitude, and improves the best known running time of $O(n^{1+1/d}\log n)$ for the specific case of d-dimensional hypercubes [12], extended later for general arbitrary patterns [3]. Note that although the improvement suggested is by a logarithmic factor, the upshot is an asymptotically *worst-case optimal* algorithm[2] in terms of running time analysis, even for the most general case of arbitrary patterns. This can be compared with [4], where the authors studied the problem of enumerating all *rotated* copies of a given pattern, improving the running time of the trivial algorithm for this task by a logarithmic factor as well. An excellent survey that covers this variant of our problem can be found in [2].

[2] For the task of outputting an explicit representation of all copies of the pattern, rather than some other representation of this set of copies, that later needs to be further parsed.

2 Axis-Parallel Squares

In this section, we present an efficient algorithm that reports all axis-parallel squares defined by a planar set of n points. A relatively efficient algorithm, devised by van Kreveld and de Berg [12], works as follows (Note that we refer, for any x_0, to the set of all points whose x coordinate is x_0, as the "column" corresponding to x_0. Moreover, we refer to columns with at most $\sqrt{n}$ points as "short columns").

```
Squares-Listing($p_1, \ldots, p_n$):
1. Build a balanced search tree $T$ and an array $A$ on the input, sorted by the $x$ coordinate.
2. For every pair of points $p$ and $q$ in $A$ residing in a short column, search in $T$ whether they
   can be complemented to a square from the right or from the left. Report each square
   found unless the other two vertices defining it are on a short column to the left of $p$ and $q$.
3. Delete all short columns from $T$ and $A$, and convert each remaining point $(x, y)$ to $(y, x)$.
4. Apply step 2 on the remaining converted points.
```

It operates correctly with a running time of $O(n\sqrt{n}\log n)$ and $O(n)$ space, in essence, since the total number of searched points defined in each of the two iterations of step 2 is

$$O\left(\sum_i s_i^2\right) \le O\left(\sum_i s_i\sqrt{n}\right) = O\left(\sqrt{n}\cdot\sum_i s_i\right) \le O\left(n\sqrt{n}\right)$$

where s_i denotes the length of the i'th column scanned. Every pair is scanned during its course, since there are at most $\frac{n}{\sqrt{n}}$ original long columns (otherwise there are more than n points), so the length of each column in step 4 is at most $\frac{n}{\sqrt{n}} = \sqrt{n}$. We strive for an algorithm with a running time of $O(n\sqrt{n})$ and space $O(n)$. As shown in [12]:

▶ **Theorem 2.1.** *(van Kreveld, de Berg) For a set P of n points in d-dimensional space, the maximal number of 2^d points that are subsets of P and that form the vertices of an axis-parallel hypercube is $\Theta(n^{1+1/d})$.*

This theorem induces a lower bound on the running time of the optimal relevant algorithm. Our result bridges the gap between this bound, and the previously best known upper bound.

2.1 Main Ideas Towards an Improvement

Assume that all input points have coordinates in $\{1, \ldots, n\}$. Instead of searching in a set the two *query* pairs that complement the pair $(x, y), (x, y + \delta)$ to a square, i.e., the pair $(x+\delta, y), (x+\delta, y+\delta)$ and the pair $(x-\delta, y), (x-\delta, y+\delta)$, we put all query points along with the original points in an array, apply radix sort on it, treating each point as a two-digit number, and then scan and mark all positive queries that define the appropriate squares. However, we cannot generally assume that all coordinates are taken from $\{1, \ldots, n\}$. We address this issue by "shrinking" the coordinates of all input points by relabeling their coordinates to values in $\{1, \ldots, n\}$. The main caveat, though, is that arithmetical considerations regarding labels are invalid. So, we avoid using arithmetic considerations when defining a query pair of points q_1, q_2 that complement the pair $p_1 = (x, y), p_2 = (x, y + \delta)$ to a square (from the right, assuming $\delta > 0$). Instead of using the invalid label $x + \delta$ as a coordinate, we define the pair q_1, q_2 (with q_2 above q_1) using *identical* labels as those of p_1, p_2. So, the query point q_2 is defined having the same y label as p_2 and the same $x + y$ label as p_1, with a similar treatment (using subtraction) for q_1. Another observation is that the linear transformation that rotates a vector (x, y) in the plane by 45^o and stretches it by $\sqrt{2}$ yields the vector $(x + y, y - x)$, as illustrated in Figure 1. So, this process is in fact a labeling of the post-rotated points.

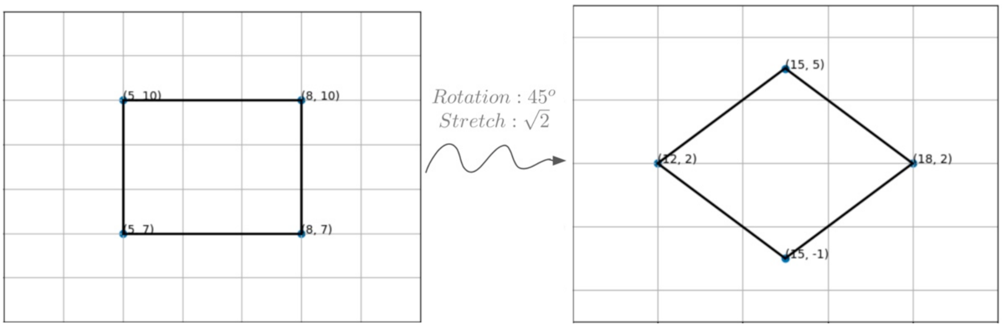


Figure 1 Illustrating the rotation by 45^o and the stretch by a factor of $\sqrt{2}$ applied on four points in the plane. Each point (x, y) was converted to the point $(x + y, y - x)$ as a result.

2.2 The Efficient Solution

Having in mind the main ideas of the following algorithm's correctness and complexity[3], we are ready to present it fully.

▶ **Theorem 2.2.** *Given a planar set P of points of size n, all axis-parallel squares defined by points from P can be enumerated in time $O(n\sqrt{n})$ and $O(n)$ space.*

Proof. The following algorithm is considered:

Amplified-Squares-Listing$(p_1, \dots, p_n)$:

1. Change the representation of each point $p = (x, y)$ to the representation $(x, y, x+y, y-x)$. Map each x coordinate in the input to a value in $\{1, \dots, n\}$. Perform a similar procedure for the y coordinates, the $x + y$ coordinates and the $y - x$ coordinates.
2. Build an array A on the input points, sorted by the x coordinate.
3. For each pair of post-labeled points $p_1 = (x, y_1, x+y_1, y_1-x)$ and $p_2 = (x, y_2, x+y_2, y_2-x)$ with $y_2 > y_1$, out of the first n pairs of points in A that reside in a short column – define the query pair $q_1 = (*, y_1, x + y_2, *), q_2 = (*, y_2, *, y_1 - x)$ that complements p_1, p_2 to a square from the right, and the query pair $q'_1 = (*, y_1, *, y_2 - x), q'_2 = (*, y_2, x + y_1, *)$ that complements to a square from the left. The wildcards replace the unknown coordinates.
4. Place each query point defined by its y and $x + y$ coordinates in an array B_1 along with all input points, and apply radix sort on B_1 based on those two coordinates. Perform a similar procedure for points of the form of q_2 and q'_1 in another array B_2.
5. Scan B_1 and mark each query point adjacent to an input point sharing the same coordinates, or to an already marked identical query point. Act similarly on B_2. Report each square found during this scanning, unless the other two vertices defining it are on a short column and complement to a square from the left.
6. Perform steps 3-5 iteratively on each subsequent n pairs of points in A in a short column.
7. Delete all points that are on short columns from A. Convert each remaining point (x, y) to (y, x). Apply steps 1-6 on the remaining converted points. ◀

[3] Some further elaboration and an extended description of those appears in the full version of this paper.

3 The General Case

In this section, we describe an algorithm that enumerates all scaled copies of any fixed arbitrary full-dimensional pattern in d-dimensions[4]. For general patterns, our algorithm works in time $O(n^{1+1/d})$ and $O(n)$ space. This answers the open question of realizing the lower bound of [5].

▶ **Theorem 3.1.** *Given a fixed set Q of points of full dimension in the d-dimensional Euclidean space, and a set P of points of size n, all scaled copies of Q determined by subsets of P can be enumerated in time $O(n^{1+1/d})$ and $O(n)$ space.*

Proof. Assume there are more than two pattern points, as the other case is easily handled. Moreover, assume that no three points in Q are on the same line. Otherwise, perturb Q and P appropriately so this condition is not met, and apply the inverse perturbation on the result, to obtain the correct output. The following algorithm is used to prove the theorem:

Amplified-Patterns-Listing$(p_1, \ldots, p_n)$:

1. Rotate the input points and the pattern points simultaneously, such that two of the pattern points, p and q, share afterwards all coordinates except the last one.
2. For each point $r \neq p, q$ in Q, define $d-1$ hyperplanes of dimension $d-1$ that include p and r, and an additional hyperplane including q and r, altogether defining r uniquely. Apply each of the $d \cdot |Q|$ transformations corresponding to those hyperplanes on each input point, and label the resulting values – the augmented coordinates – using $\{1, \ldots, n\}$.
3. Build an array A on the input points, sorted by all original coordinates by their order.
4. Scan A. For each pair r and t on an axis-parallel line that corresponds to step 1 that also has at most $n^{1/d}$ points (”short” line), define the other $|Q|-2$ points that complement to a pattern using the labels obtained from step 2, until defining n such sets of queries.
5. Place all query points that are defined by the same augmented coordinates in an array with all input points, forming several such arrays. Apply radix sort on each such array.
6. Scan each array from step 5, and mark each query point adjacent to an identical input point or an already marked query point. Report all copies that were found, only after applying on those the rotation which is inverse to that of step 1.
7. Perform steps 4-6 on each subsequent n pairs of points in A of the form of step 4.
8. Apply steps 2-7 for each pair among the pattern points that determines a line parallel to that through p and q, excluding enumeration of duplicate copies (similarly to the identification of duplicate squares). Delete all points on those ”short” lines from step 4. Apply steps 1-7 on the remainder, for a different pattern pair. Perform $d-1$ times.

Analysis: The main ideas already appeared in Section 2. Aside from those ideas, note that step 2, in fact, defines each point as the intersection of a line and a $(d-1)$-dimensional hyperplane, and since no three points are on the same line, the points are uniquely defined in that manner. As for the running time, note that steps involving only Q cost $O(1)$. Moreover, note that there is no remaining long line analyzed at the ultimate iteration. Otherwise, all points on it are on another long line defined by a linearly independent vector. This induces more points on a different long line, and so forth, yielding that there are more than $\left(n^{1/d}\right)^d = n$ input points, a contradiction. Other than that, we did not need the lines to be axis-parallel, but rather merely that the corresponding vectors form an independent set. ◀

[4] We individually treat hypercubes in the full version of this paper, as their analysis motivates and simplifies some of the ideas that lead to the general solution.

4 Conclusion and Further Work

In this paper, we analyzed the problem of enumerating all scaled copies of a pattern in a set of n points in time $O(n^{1+1/d})$, answering open questions from [12] and [3] by realizing the lower bound due to Elekes and Erdős [5]. We relied on some existing ideas, amplified using bucket-based methods, sweep-line scanning and more. As far as we are aware of, the combinations of these techniques this way was not noted in the literature so far for similar tasks. One open question is whether these techniques can be adjusted for different pattern matching problems. Other questions include whether the task of finding one copy of a pattern is an easier task than enumerating all copies of it, and similarly for the task of counting the number of copies instead of outputting them. In addition, the existence of an output-sensitive algorithm for our problem, and the existence of an efficient enumeration algorithm for patterns not of a constant size, form another two open questions for further research.

References

1 M. Alzina, W. Szpankowski, and A. Grama. 2d-pattern matching image and video compression: theory, algorithms, and experiments. *IEEE Transactions on Image Processing*, 11(3):318–331, 2002.

2 D. Avis, A. Hertz, and O. Marcotte. *Graph theory and combinatorial optimization.* Springer Science & Business Media, 2005.

3 P. Braß. Combinatorial geometry problems in pattern recognition. *Discrete & Computational Geometry*, 28:495–510, 2002.

4 P. de Rezende and D. Lee. Point set pattern matching in d-dimensions. *Algorithmica*, 13:387–404, 1995.

5 G. Elekes and P. Erdős. Similar configurations and pseudo grids. *Intuitive geometry*, 63:85–104, 1994.

6 P. Erdős. On sets of distances of n points. *American Mathematical Monthly*, 53:248–250, 1946.

7 P. Finn, L. Kavraki, J. Latombe, Rajeev Motwani, C. Shelton, S. Venkatasubramanian, and A. Yao. Rapid: Randomized pharmacophore identification for drug design. *Computational Geometry: Theory and Applications*, 10:324–333, 1997.

8 M. Fredman and D. Willard. Blasting through the information theoretic barrier with fusion trees. In *Proceedings of the twenty-second annual ACM symposium on Theory of Computing*, pages 1–7, 1990.

9 D. Mount, N. Netanyahu, and J. Le Moigne. Efficient algorithms for robust feature matching. *Pattern Recognition*, 32(1):17–38, 1999.

10 R. Norel, D. Fischer, H. Wolfson, and R. Nussinov. Molecular surface-recognition by a computer vision-based technique. *Protein engineering*, 7:39–46, 1994.

11 G. Schindler, P. Krishnamurthy, R. Lublinerman, Y. Liu, and F. Dellaert. Detecting and matching repeated patterns for automatic geo-tagging in urban environments. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–7, 2008.

12 M. van Kreveld and M. de Berg. Finding squares and rectangles in sets of points. In *Graph-Theoretic Concepts in Computer Science*, 1990.

13 V. Zografos and B. Buxton. Affine invariant, model-based object recognition using robust metrics and bayesian statistics. In *Kamel M., Campilho A. (eds) Image Analysis and Recognition. ICIAR 2005. Lecture Notes in Computer Science*, volume 3656, 2005.

Blocking Delaunay Triangulations from the Exterior*

Oswin Aichholzer[1], Thomas Hackl[1], Maarten Löffler[2], Alexander Pilz[1], Irene Parada[3], Manfred Scheucher[4], and Birgit Vogtenhuber[1]

1 **Institute for Software Technology,**
Graz University of Technology, Austria,
{oaich,thackl,apilz,bvogt}@ist.tugraz.at

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

3 **Department of Applied Mathematics and Computer Science,**
Technical University of Denmark
irmde@dtu.dk

4 **Institut für Mathematik,**
Technische Universität Berlin, Germany,
scheucher@math.tu-berlin.de

Abstract

Given two distinct point sets P and Q in the plane, we say that Q *blocks* P if no two points of P are adjacent in any Delaunay triangulation of $P \cup Q$. Aichholzer et al. (2013) showed that (the Delaunay triangulation of) any set P of n points in general position (that is, no three collinear and no four cocircular) can be blocked by $\frac{3}{2}n$ points and that every set P of n points in convex position can be blocked by $\frac{5}{4}n$ points. Moreover, they conjectured that, if P is in convex position, n blocking points are sufficient and necessary. The necessity was recently shown by Biniaz (2021) who proved that every point set in general position requires n blocking points.

Here we investigate the variant, where blocking points can only lie outside of the convex hull of the given point set. We show that $\frac{5}{4}n - O(1)$ such *exterior-blocking* points are sometimes necessary, even if the given point set is in convex position. As a consequence we obtain that, if the conjecture of Aichholzer et al. for the original setting was true, then minimal blocking sets of some point configurations P would have to contain points inside of the convex hull of P.

1 Introduction

Delaunay triangulations, Delaunay graphs, Voronoi diagrams (their dual structures), and various generalizations have been intensively studied in the last century; see for example the standard textbook in Computation Geometry [6]. A *Delaunay triangulation* $DT(P)$ of a given point set P in the plane is a triangulation of P in which for every edge between two distinct points $p_1, p_2 \in P$ there exists a circle through p_1, p_2 that contains no point of $P \setminus \{p_1, p_2\}$ in its interior. An edge spanned by P with this property is called *Delaunay edge*. For a point set *in general position*, that is, no three points of P lie on a common line and no four points of P lie on a common circle, the Delaunay triangulation is unique. Figure 1(a)

* We thank anonymous reviewers for valuable comments. Parada was supported by the Austrian Science Fund (FWF): W1230 and by Independent Research Fund Denmark grant 2020-2023 (9131-00044B) "Dynamic Network Analysis". Scheucher, Parada, and Vogtenhuber were partially supported within the collaborative D-A-CH project *Arrangements and Drawings*, by grants DFG: FE 340/12-1 and FWF: I 3340-N35, respectively. Scheucher was supported by the DFG Grant SCHE 2214/1-1.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.

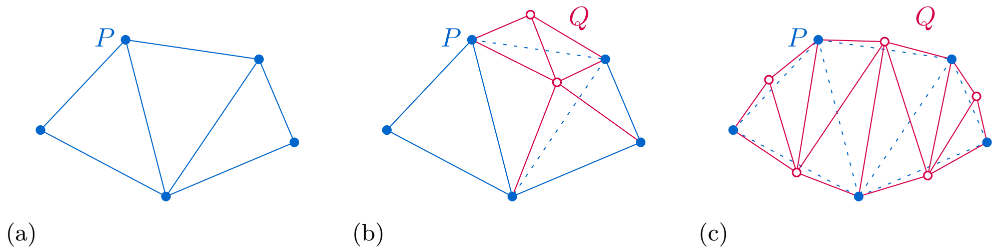


Figure 1 (a) A set P (blue) of five points in convex position, and its unique Delaunay triangulation $DT(P)$. (b) A set Q (red) of two points that blocks two of the edges of $DT(P)$. (c) A set Q of five points from the exterior of $\mathrm{conv}(P)$ that blocks P.

shows the unique Delaunay triangulation of a point set *in convex position*, that is, the points are the vertices of a convex polygon.

In this article we continue the investigation of blocking points for Delaunay edges. For two point sets P, Q, we say that Q *blocks an edge* p_1p_2 spanned by P if every circle through p_1, p_2 contains at least one point of $P \cup Q$ in its interior. Equivalently, p_1p_2 is not an edge of any Delaunay triangulation of $P \cup Q$. We say that Q *blocks* P if Q blocks all edges spanned by P. Equivalently, no two points of P are adjacent in any Delaunay triangulation of $P \cup Q$. If moreover no point of Q lies in the interior of the convex hull of P, we say that Q *blocks* P *from the exterior.* Figures 1(b) and 1(c) shows examples where Q blocks P.

Aronov et al. [3] showed that every set P of n points in general position can be blocked by a set of $2n - 2$ points, and that, if P is in convex position, $\frac{4}{3}n$ blocking points are sufficient. Both of their bounds were improved by Aichholzer et al. [1], who showed that, for general position, $\frac{3}{2}n$ blocking points are sufficient, and that, for convex position, $\frac{5}{4}n$ blocking points are sufficient. They also showed that $n - 1$ blocking points are always needed and posed the following conjecture.

▶ **Conjecture 1.1** ([1]). *If P is a set of n points in convex position in the plane, then n blocking points are necessary and sufficient, that is, every blocking set of P contains at least n points and this bound is tight.*

Biniaz [5] recently strengthened the lower bound by showing that, for every set of n points in general position, n blocking points are necessary and that there are sets of n points in convex position which can be blocked by n points. While this confirms the necessity part from Conjecture 1.1, the question about sufficiency remains open.

For many sets P of n points in convex position, a simple construction suffices to indeed block all Delaunay edges with exactly n points: place a single point of Q close to the mid point of each edge of the convex hull of P, on the outer side; see Figure 1(c). Placing the points arbitrary close to the convex hull edges ensures that all those edges are blocked, and indeed every convex hull edge requires at least one point somewhere outside the convex hull to be blocked. Moreover, this simple construction often enough also blocks all interior edges of $DT(P)$. This may suggest that a similar approach could actually always work.

Inspired by these observations, we investigate the variant where blocking points have to lie outside of the convex hull of the given n-point set P. We show that $\frac{5}{4}n - O(1)$ such *exterior-blocking* points are sometimes necessary, even if P is in convex position.

▶ **Theorem 1.2.** *For $k \in \mathbb{N}$, there is a set P of $4k$ points in general position that requires at least $5k - 5$ exterior-blocking points.*

As a direct consequence of Theorem 1.2 we obtain for the original setting that, if Conjecture 1.1 was true, then minimal blocking sets of certain point sets P would have to contain points inside of the convex hull of P.

Note that the construction of size $\lfloor \frac{5}{4}n \rfloor$ for convex position in [1] might contain interior points. The reason is that in the induction blocking points placed for a subproblem in the exterior of an edge (Case (a) in the proof of Theorem 3 in [1]) might end up to be interior for the overall triangulation. Modifying their approach, a blocking set of size $\approx \frac{4}{3}n$ can be obtained by iteratively cutting ears ($(n, 3, 4)$-cuts in the terminology of [1]).

2 Proof of Theorem 1.2

To prove Theorem 1.2, we first give a configuration with collinear points in Section 2.1, which we then perturb in Section 2.2 to obtain a configuration which is in general position.

2.1 Construction with Collinear Points

Our construction consists of k gadgets, each containing 4 points (a *top point* t_i, a *left point* ℓ_i, *middle point* m_i, and a *right point* r_i, where the latter three are called *bottom points*), which gives us a set P_0 of $n = 4k$ points in total. We place all $3k$ bottom points on the x-axis and all k top points on a line segment (above the x-axis) with negative slope.

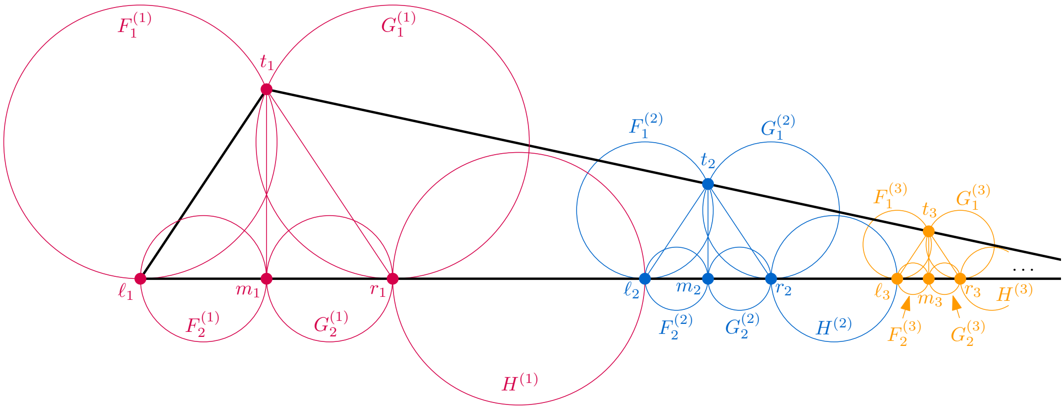


Figure 2 An illustration of the point set P_0 of size $4k$ and the set of circles $\mathcal{C}_0$ where at least $5k - 3$ exterior-blocking points are required. The red, blue, and yellow points and circles illustrate the first, second, and third gadget of the construction, respectively.

Explicit coordinates for the points $\{\ell_i, m_i, r_i, t_i\}$ in the i-th gadget are $\{(-2, 0), (0, 0), (2, 0), (0, 3)\}$, scaled by 2^{-i}, and with x-offset of $3 + 14 \sum_{j=1}^{i} 2^{-j} = 3 + 14(1 - 2^{-i})$. By construction, all points have positive x-coordinate, all top points lie on the x-axis, and all bottom points lie on the line $\{(x, y) : 3x + 14y = 51\}$.

Further, each gadget i with $1 \leq i < k$ contains 5 circles and the k-th gadget contains 4 circles, which gives us a set $\mathcal{C}_0$ of $5k - 1$ circles in total. They are defined as follows:

- a circle $F_1^{(i)}$ through t_i and ℓ_i, which is tangent to the x-axis in ℓ_i;
- a circle $G_1^{(i)}$ through t_i and r_i, which is tangent to the x-axis in r_i,
- a circle $F_2^{(i)}$ with the segment $\ell_i m_i$ as diameter,
- a circle $G_2^{(i)}$ with the segment $m_i r_i$ as diameter; and
- a circle $H^{(i)}$ with the segment $r_i \ell_{i+1}$ as diameter.

See Figure 2 for an illustration of the construction. On each circle, there are exactly two points of P_0 and no circle contains points of P_0 in its interior. Further, any two "neighboring" bottom circles are tangent in their common point of P_0, that is, $F_2^{(i)} \cap G_2^{(i)} = \{m_i\}$, $G_2^{(i)} \cap H_2^{(i)} = \{r_i\}$, and $H_2^{(i)} \cap F_2^{(i+1)} = \{\ell_{i+1}\}$.

It is necessary that each of the circles contains a blocking point of Q in its interior as otherwise there is an edge in the Delaunay graph of P_0 (and hence in any Delaunay triangulation). For each circle C, we denote the region in the interior of C and in the exterior of the convex hull of P_0 as its *blocking area*. Note that the circles $F_1^{(i)}$ and $G_1^{(i)}$ are both tangent the x-axis and thus only contain points above the x-axis in their interior, and that the circles H^i can only be blocked from points below x-axis. Therefore no two circles (except in the first and last gadget) have a common exterior-blocking area. Therefore, five exterior-blocking points are required to block all circles of a gadget for $1 < i < k$. For the first and last gadget, 4 and 3 exterior-blocking points are required, respectively. As none of these points can be used for two gadgets simultaneously, a total of $5k - 3$ points is required to block P_0 from the exterior.

2.2 Transformation to General Position

We will slightly perturb the point set P_0 such that all points are in convex position. We also add two more circles for each gadget i with $1 < i < k$ to the set $\mathcal{C}_0$ and remove the circles $F_1^{(i)}$ and $G_1^{(i)}$ for $i = 1, k$. We denote the resulting set or circles by $\mathcal{C}_0'$. The new circles are defined as follows; see Figures 3(b) and 3(c) for an illustration.

- a circle $F_3^{(i)}$ through t_i and m_i, which is tangent to the segment $t_i t_{i+1}$; and
- a circle $G_3^{(i)}$ through t_i and m_i, which is tangent to the segment $\ell_i m_i$.

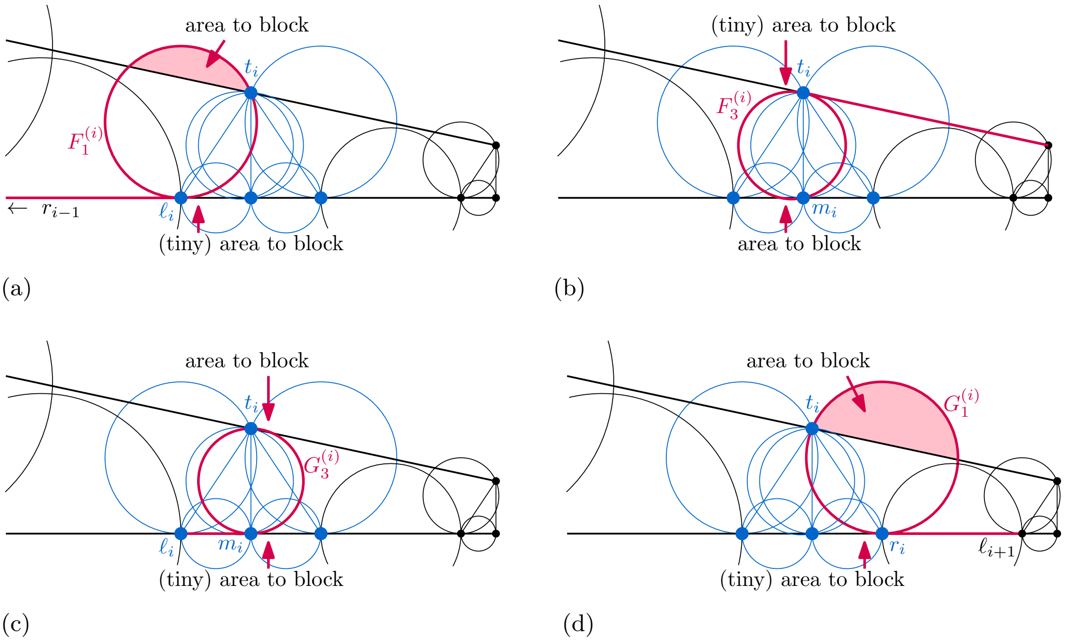


Figure 3 The gadget for the general case construction. (a) – (d) show how to align circles (the red circle is always tangent to the red line) and highlight the exterior blocking area using red arrows.

Note that a circle C through a point p cannot simultaneously be tangent to two line segments at p with different slopes. Thus, the arguments from Section 2.1 will not apply anymore after we perturb P_0, because circles $F_1^{(i)}$ and $G_1^{(i)}$ will intersect other circles outside the convex hull of P_0. In the following we will deal with this issue.

Transformation. We define $P(\tau)$ as the continuous transformation of $P_0 = P(0)$ where

- all bottom points are transformed as $(x, y) \mapsto (x, y + \tau x^3)$ and
- all top points are transformed as $(x, y) \mapsto (x, y - \tau x^3)$.

The transformation is illustrated in Figure 4.

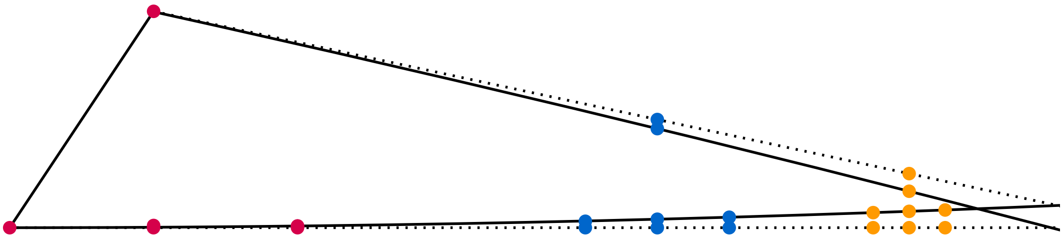


Figure 4 An illustration of the point set $P_0 = P(0)$ and the perturbed set $P(\tau)$ for sufficiently small τ. Note that, if τ is not small enough, the resulting set might not be in convex position and hence might not have the desired properties .

Analogously, we define $\mathcal{C}(\tau)$ as the transformation of $\mathcal{C}'_0$, which preserves the defined properties of the circles, where for $1 < i < k$, we keep the tangency of $F_1^{(i)}$ with $r_{i-1}\ell_i$ and the one of $G_1^{(i)}$ with $r_i\ell_{i+1}$. See Figures 3(a) and 3(d). Since all circles in $\mathcal{C}'_0$ have finite radii, we can choose $\tau_{max} > 0$ such that all points of $P(\tau)$ are in general position and lie on the boundary of the convex hull and all circles of $\mathcal{C}(\tau)$ have finite radii for $0 \leq \tau \leq \tau_{max}$. Details are deferred to the full version; see [2] for a version with appendix.

In the following, we denote by $c(C)$ the center of a circle C and by $r(C)$ the radius of C, and we define $d_{C,p} := \|p - c(C)\| - r(C)$ to indicate whether the point p lies

- inside the circle C ($d_{C,p} < 0$),
- on the circle C ($d_{C,p} = 0$), or
- outside the circle C ($d_{C,p} > 0$).

Since every circle C in $\mathcal{C}'_0$ contains exactly 2 points a, b of P_0 (and no points of P_0 in its interior), we have $d_{C,a} = d_{C,b} = 0$ and $d_{C,p} > 0$ for every other point p of P_0. Analogously, we define $d_{C,p}(\tau)$ at time τ. As $d_{C,p}(\tau)$ and $P(\tau)$ are both continuous functions, there exists $0 < \varepsilon_{C,p} \leq \tau_{max}$ such that $d_{C,p}(\tau)$ has the same sign for any $0 \leq \tau \leq \varepsilon_{C,p}$. We remark that $\varepsilon_{C,p}$ does not need to be maximal – we just need some $\varepsilon_{C,p} > 0$ for our purposes.

Note that in the i-th gadget ($1 < i < k$) the lower intersection point of the circles $F_1^{(i)}$ and $F_3^{(i)}$ (as depicted in Figure 5) lies inside the convex hull of $P(\tau)$ at time $\tau = 0$. Moreover, as this intersection point moves continuously on time, we can choose $\varepsilon_i > 0$ such that at any time $0 \leq \tau \leq \varepsilon_i$ this intersection point lies inside the convex hull. In an analogous manner, we can choose $\varepsilon'_i > 0$ for $1 < i < k$ such that at any time $0 \leq \tau \leq \varepsilon'_i$ the lower intersection point of the circles $G_1^{(i)}$ and $G_3^{(i)}$ (as depicted in Figure 5) lies inside the convex hull.

Since we have a finite number of points and a finite number of circles, we can choose a common $\varepsilon > 0$ small enough such that at any time $0 \leq \tau \leq \varepsilon$

- every circle in $\mathcal{C}(\tau)$ contains exactly 2 points of $P(\tau)$ (and no point in its interior), and
- no two exterior blocking areas overlap for $1 < i < k$, except for the blocking areas of $F_1^{(i)}$ and $F_3^{(i)}$ on top, and the blocking areas of $G_1^{(i)}$ and $G_3^{(i)}$ on top.

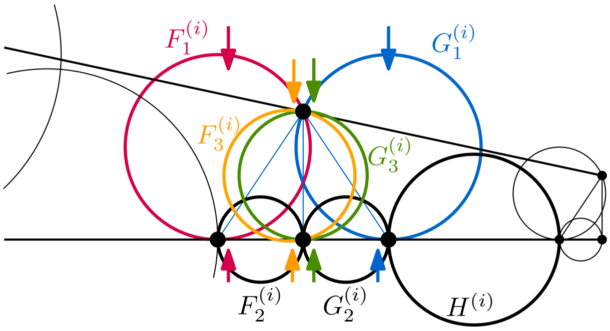


Figure 5 Analysis of a gadget and its corresponding circles. The colored arrows indicate the regions of the disks, which can be blocked by exterior points after the perturbation.

Analysis. We first show that two points are required to block the circles $F_1^{(i)}$, $F_2^{(i)}$, and $F_3^{(i)}$. If $F_1^{(i)}$ is blocked from above then we need at least a second point to block $F_2^{(i)}$. Thus assume there is no point blocking $F_1^{(i)}$ from above. Since the above blocking area of $F_3^{(i)}$ is fully contained in $F_1^{(i)}$, the circle $F_3^{(i)}$ is also not blocked from above. Since the bottom blocking areas of $F_1^{(i)}$ and $F_3^{(i)}$ are disjoint, at least two blocking points have to be placed in $F_2^{(i)}$. As a consequence, two points are required to block $F_1^{(i)}$, $F_3^{(i)}$, and $F_2^{(i)}$.

In an analogous manner one can show that two points are required to block the circles $G_1^{(i)}$, $G_3^{(i)}$, and $G_2^{(i)}$. It is easy to see, that

- the union of blocking areas of $F_1^{(i)}$, $F_3^{(i)}$, and $F_2^{(i)}$,
- the union of blocking areas of $G_1^{(i)}$, $G_3^{(i)}$, and $G_2^{(i)}$, and
- the blocking area of $H^{(i)}$

are mutually disjoint. Consequently, at least five exterior blocking points are required for the i-th gadget $(1 < i < k)$. Further, the blocking areas of the bottom circles of the first and last gadget ($F_2^{(1)}$, $G_2^{(1)}$, $H^{(1)}$, $F_2^{(k)}$, and $G_2^{(k)}$) are all disjoint from all other blocking areas. Hence, at least $5k - 5$ points are required in total, which completes the proof of Theorem 1.2.

3 Discussion and Further Related Work.

The idea of blocking points can also be extended to other graph classes. For example, Biedl et al. [4] investigated blocking sets of so-called Θ_6-graphs, a structure related to Delaunay graphs: In a Θ_6-graph of a point set, every pair of points shares an edge if there is an empty equilateral triangle (instead of an empty disks).

From an algorithmic point of view, we can ask how fast a minimal blocking set can be computed. For the general problem, where blocking points can also be placed in the interior of the convex hull of the Delaunay triangulation, this would help to identify cases where many blocking points are needed. In fact, we tried several approaches to find a set of n points which requires more than n points to be blocked, but without success. We therefore would not be surprised if Conjecture 1.1 always holds. But even if Conjecture 1.1 is true, then there is still the algorithmic question how fast a blocking set of n points can be found.

The anonymous reviewers pointed out that the degenerate construction from Section 2.1 can be improved as follows. By removing the "middle" point m_i from gadget i and replacing the circles $F_2^{(i)}$ and $G_2^{(i)}$ by a circle $I_2^{(i)}$ with the segment $\ell_i r_i$ as diameter, the constructed set of $3k$ points (depicted in Figure 7) requires $4k - 2$ exterior-blocking points. However, when making this construction non-degenerate via a perturbation as in Section 2.2, the number of required exterior-blocking points also drops significantly.

▶ **Theorem 3.1.** *For $k \in \mathbb{N}$, there is a set P of $3k$ points that requires at least $4k-2$ exterior-blocking points.*

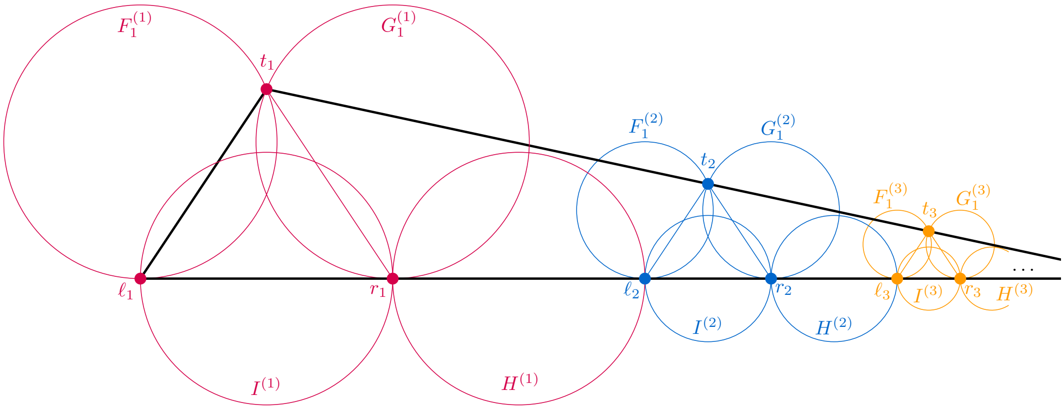


Figure 6 A degenerate construction with $3k$ points where at least $4k-2$ exterior-blocking points are required. The red, blue, and yellow points and circles illustrate the first, second, and third gadget of the construction, respectively.

A reviewer also pointed out that the gadgets in the degenerate construction need not to be scaled. Figure 7 gives an illustration of the alternative construction. However, when making this construction non-degenerate via a perturbation as in Section 2.2, the number of required exterior-blocking points significantly drops because, for $1 < i \leq k$, the circles $G_3^{(i)}$ can be blocked by points that are to the left of t_i and slightly above $t_{i-1}t_i$. Also note that, in contrast to our construction from Section 2.1, here the four points m_i, t_i, m_j, t_j lie on a common circle for every $1 \leq i < j \leq k$.

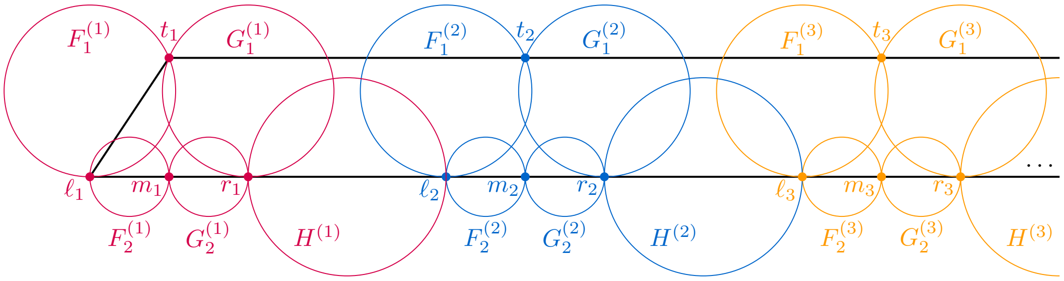


Figure 7 An alternative construction with isometric gadgets. The red, blue, and yellow points and circles illustrate the first, second, and third gadget of the construction, respectively.

References

1 Oswin Aichholzer, Ruy Fabila-Monroy, Thomas Hackl, Marc van Kreveld, Alexander Pilz, Pedro Ramos, and Birgit Vogtenhuber. Blocking Delaunay triangulations. *Computational Geometry: Theory and Applications*, 46(2):154–159, 2013. doi:10.1016/j.comgeo.2012.02.005.

2 Oswin Aichholzer, Thomas Hackl, Maarten Löffler, Alexander Pilz, Irene Parada, Manfred Scheucher, and Birgit Vogtenhuber. Blocking Delaunay Triangulations from the

Exterior (with Appendix), 2022. http://page.math.tu-berlin.de/~scheuch/publ/ahlppsv-bd-eurocg22.pdf.

3 Boris Aronov, Muriel Dulieu, and Ferran Hurtado. Witness (Delaunay) graphs. *Computational Geometry*, 44(6):329–344, 2011. doi:10.1016/j.comgeo.2011.01.001.

4 Therese Biedl, Ahmad Biniaz, Veronika Irvine, Kshitij Jain, Philipp Kindermann, and Anna Lubiw. Maximum matchings and minimum blocking sets in Θ_6-graphs. In *Graph-Theoretic Concepts in Computer Science*, volume 11789 of *LNCS*, pages 258–270. Springer, 2019. doi:10.1007/978-3-030-30786-8_20.

5 Ahmad Biniaz. A short proof of the toughness of Delaunay triangulations. *Journal of Computational Geometry*, 12:5, 2021. doi:10.20382/jocg.v12i1a2.

6 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications, Third Edition.* Springer, third edition, 2008. doi:10.1007/978-3-540-77974-2.

The Computational Complexity of the ChordLink Model

Philipp Kindermann[1], **Jan Sauer**[2], and **Alexander Wolff**[2]

1 **Universität Trier, Trier, Germany**
`kindermann@uni-trier.de`

2 **Universität Würzburg, Würzburg, Germany**
`firstname.lastname@uni-wuerzburg.de`

Abstract

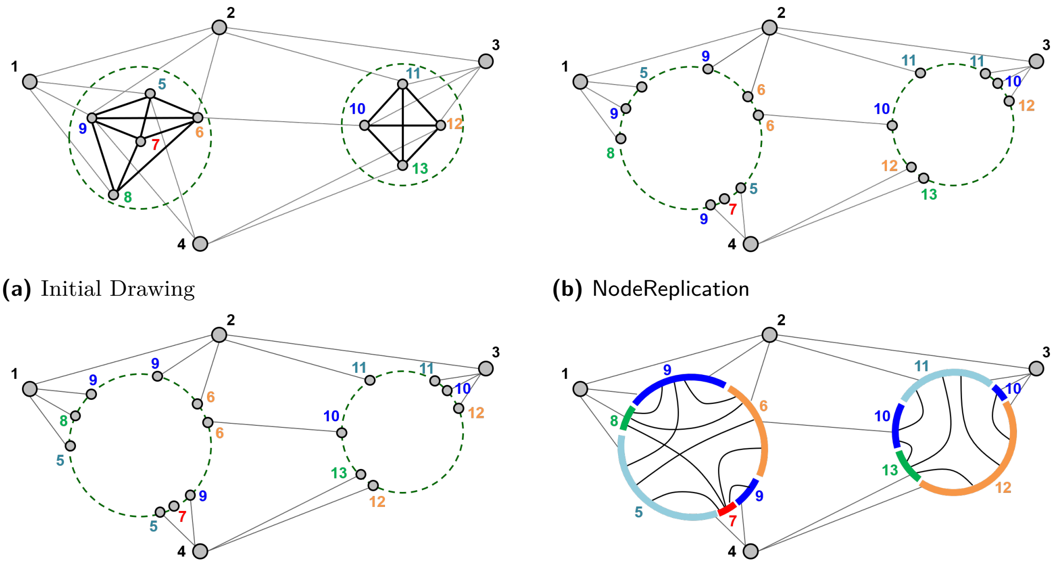
In order to visualize well-clustered graphs with many intra-cluster but few inter-cluster edges, hybrid approaches have been proposed. For example, ChordLink draws the clusters as chord diagrams and embeds these into a node-link diagram that represents the overall structure of the clustered graph. The ChordLink approach consists of four steps; node replication, node permutation, node merging, and chord insertion. In this paper, we focus on the optimization problems defined by two of these steps. We show that the decision version of the problem defined by node permutation is NP-complete and present an efficient algorithm for a special case. For chord insertion, we show that it is NP-complete to decide whether a crossing-free placement of the chords exists. Moreover, it is APX-hard to minimize the number of crossings among the chords. Our results answer an open question posed by Angori, Didimo, Montecchiani, Pagliuca, and Tappini, who introduced ChordLink [TVCG 2021].

1 Introduction

Node-link diagrams represent an intuitive tool for visualizing graphs. For dense graphs, however, node-link diagrams tend to degenerate into unintelligible hairballs. Less intuitive, but more robust visualization paradigms such as adjacency matrices can be a remedy. In practice, however, large graphs are often "globally sparse" and just "locally dense" [3]. This is the case, for example, in social networks such as collaboration and financial networks [4], but also in biological networks [9]. For visualizing such graphs, hybrid representations have been invented. In intersection-link representations, for example, each vertex is represented by a geometric object and each each is either represented by a curve connecting the two objects or, if it belongs to a dense subgraph, by a non-empty intersection of the two objects [2, 1]. Another example of a hybrid representation is NodeTrix [7], which uses matrices for dense subgraphs and links between the matrices for the global graph structure. ChordLink, recently introduced by Angori, Didimo, Montecchiani, Pagliuca, and Tappini [3], combines very effectively so-called chord diagrams [8] for dense subgraphs, again with links between them for the overall graph structure. In a chord diagram, each vertex is represented by one or several circular arcs, and each edge is a chord between any two arcs that represent the endpoints of the edge. Turning a given node-link diagram into a ChordLink visualization can, for example, be triggered by a user in an interactive system. We now formalize the ChordLink model and the four steps that are performed in order to compute a ChordLink visualization from a node-link diagram. To this end, for a graph G, let $V(G)$ be its vertex set and let $E(G)$ be its edge set. For a positive integer k, let $[k] = \{1, 2, \dots, k\}$.

The ChordLink Model. Given a node-link diagram Γ of a graph G, a cluster $C \subseteq V(G)$, and a circle R that contains only the vertices in C (at their positions in Γ), a ChordLink

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.



(a) Initial Drawing

(b) NodeReplication

(c) NodePermutation

(d) NodeMerging+ChordInsertion

Figure 1 The steps of the ChordLink approach (drawings from Angori et al. [3]).

visualization of G locally modifies Γ such that $G[C]$ is drawn as a chord diagram with the vertices of C on R. There are four steps; see Fig. 1, which treats two clusters simultaneously.

NodeReplication: For each node $w \in C$ connected to a node $u \notin C$, create a copy w_u of w on the intersection of the edge (w, u) with R, and add the edge (w_u, u); see Fig. 1b.

NodePermutation: Copies v_u and w_u of different nodes v and w that are connected to the same node $u \notin C$ can be exchanged in the order of the node copies on R. This step naturally defines the optimization problem NodePermutation, where the aim is to find a permutation of the node copies on R that exchanges only copies connected to the same node outside of R and maximizes the total number of pairs of consecutive copies of the same node; see Fig. 1c. (The number of such pairs increases by 3 when going from Fig. 1b to Fig. 1c.)

NodeMerging: Replace each maximal subsequence of consecutive copies of a node w along R by a circular arc c_w; see Fig. 1d.

ChordInsertion: For each edge $(v, w) \in G[C]$, select an arc c_v representing v and an arc c_w representing w, and insert a chord that connects c_w and c_z in the interior of R; see Fig. 1d. Angori et al. [3] suggest to minimize the total number of crossings among the chords. (They also suggest maximizing the smallest angle formed by any pair of crossing chords, but we do not consider this here.) This defines the optimization problem CrossingMinimal ChordInsertion.

For NodePermutation, Angori et al. [3] describe a dynamic program that yields optimal solutions if the node copies that are adjacent to the same external node form intervals along R. If this condition does not hold, they simply split R into maximal pieces where the condition does hold and treat each piece seperately, which yields a heuristic overall solution.

For CrossingMinimal ChordInsertion, Angori et al. suggest a greedy algorithm that first draws the chords whose endpoints are both represented by unique arcs. Then it adds the other chords one by one, making the currently best choice in terms of crossings (and, with lower weight, in terms of crossing angles). It draws chords as Bézier curves; see Fig. 1d.

Contribution. First, we prove that NodePermutation is NP-complete; see Section 2. Then, we give an efficient algorithm for NodePermutation for the special case that the neighborhood of C contains only two vertices; see Section 3. Finally, we show that (even a rather special case of) CrossingMinimal ChordInsertion is APX-hard; see Section 4.

2 NP-Completeness of NodePermutation

Above, we have stated NodePermutation as an optimization problem. We now formally define the corresponding decision problem. In the ChordLink model, for every vertex c in the cluster C and each neighbor $g \notin C$ of c, a copy of c is placed on the circle R. Since G is simple, each copy can be described as a unique pair (c, g). Abstracting from the original problem, we call c the *color* and g the *group* of the pair (c, g). This leads to the following formulation of the problem, where we associate every vertex of C with a distinct color.

Let $\mathcal{C}$ be a set of colors, let $\mathcal{G}$ be a set of groups, and let $L = (L_1, \dots, L_n, L_1)$ be a circular list of *distinct* pairs where, for $i \in [n]$, $L_i = (c_i, g_i) \in \mathcal{C} \times \mathcal{G}$. Define $\mathcal{G}(L) = (g_1, \dots, g_n, g_1)$, $\mathcal{C}(L) = (c_1, \dots, c_n, c_1)$, and let $N(L)$ be the number of pairs of consecutive equal entries of $\mathcal{C}(L)$. Given $\mathcal{C}$, $\mathcal{G}$, L, and an integer $K > 0$, find a permutation π of L such that (i) $\mathcal{G}(\pi(L)) = \mathcal{G}(L)$ and (ii) $N(\pi(L)) \geq K$. Note that requirement (i) ensures that we can permute only elements of L that belong to the same group.

▶ **Theorem 1.** *NodePermutation is NP-complete.*

Proof. The problem is in NP since we can verify a permutation easily.

To show hardness, we reduce from 3SetCover. This problem generalizes VertexCover in cubic graphs, which is NP-hard [5]. In the decision version of 3SetCover, given a finite universe U (the edge set of the cubic graph), a family $\mathcal{S}$ of size-3 subsets of U (for each vertex, its three incident edges), and an integer $k > 0$, the task is to find a subfamily $\mathcal{S}'$ (corresponding to a vertex cover) of $\mathcal{S}$ of size at most k that covers U. (In the special case of VertexCover, each element of the universe appears in exactly two elements of $\mathcal{S}$.)

Given an instance $(U, \mathcal{S}, k)$ of 3SetCover, we construct an instance $(\mathcal{G}, \mathcal{C}, L, K)$ of NodePermutation such that one is a yes-instance if and only if the other is a yes-instance. Let $U = \{u_1, \dots, u_n\}$, $\mathcal{S} = \{S_1, \dots, S_m\}$, and, for $i \in [m]$, let $S_i = \{u_{i_1}, u_{i_2}, u_{i_3}\}$ with $i_1, i_2, i_3 \in [n]$. To construct an instance of NodePermutation, we use only a single color $c^\star$ that appears in more than one entry; all other entries have a unique color. Furthermore, we have one group for every $u_i \in U$, and there is exactly one entry $(c^\star, u_i)$, and we have one additional group z such that there is no entry $(c^\star, z)$; the entries with group z basically serve as *blockers* between the gadgets. The full reduction is as follows:

$$
\begin{aligned}
K &= n - k,\\
\mathcal{G} &= U \cup \{z\}, \text{where } z \notin U, \text{ and}\\
\mathcal{C} &= \{c^\star\} \cup \{c_1, \dots, c_n\} \cup \textstyle\bigcup_{i=1}^{m} \{c_{i,1}, c_{i,2}, c_{i,3}, c_{i,4}, c_{i,5}\}, \text{and}\\
L^0 &= \big((c^\star, u_1), (c_1, z), (c^\star, u_2), (c_2, z), \dots, (c^\star, u_n), (c_n, z)\big)\\
L^i &= \big((c_{i,1}, u_{i_1}), (c_{i,2}, u_{i_2}), (c_{i,3}, u_{i_3}), (c_{i,4}, u_{i_1}), (c_{i,5}, z)\big) \text{ for each } i \in [m]\\
L &= L^0 \oplus L^1 \oplus \dots \oplus L^m, \text{where } \oplus \text{ concatenates lists.}
\end{aligned}
$$

Clearly, this reduction can be performed in polynomial time. Intuitively, every sublist L^i that contains a color-$c^\star$ entry corresponds to a set S_i in a solution $\mathcal{S}'$ of $\mathcal{S}$. If these sublists contain K consecutive color-$c^\star$ entries, then there are $2K$ elements that are covered by K sets in $\mathcal{S}'$, so $|\mathcal{S}'| \leq n - K = k$.

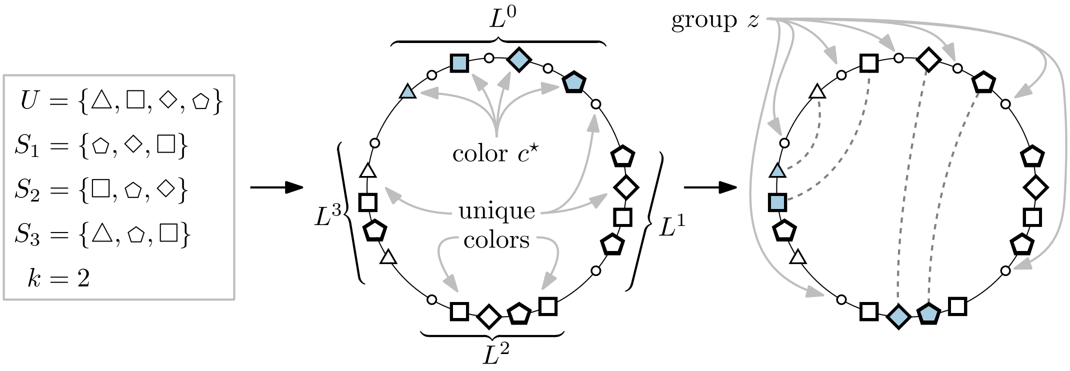

$U = \{\triangle, \square, \diamond, ⬠\}$
$S_1 = \{⬠, \diamond, \square\}$
$S_2 = \{\square, ⬠, \diamond\}$
$S_3 = \{\triangle, ⬠, \square\}$
$k = 2$
L^0
L^1
L^2
L^3
color $c^\star$
unique colors
group z

Figure 2 Here, the reduction from 3SetCover to NodePermutation yields the cover $\{S_2, S_3\}$.

First, we assume that $(U, \mathcal{S}, k)$ is a yes-instance of 3SetCover, that is, there is a size-k subfamily $\mathcal{S}'$ of $\mathcal{S}$ that covers U. We need to construct a permutation π of L such that (i) $\mathcal{G}(\pi(L)) = \mathcal{G}(L)$ and (ii) $N(\pi(L)) \geq K$.

For each $j \in [n]$, let i be the index of an arbitrary set in $\mathcal{S}'$ that contains u_j. Swap $(c^\star, u_j)$ in L^0 with an entry in L^i with group u_j. There is a choice only if $u_j = u_{i_1}$. If $(c_{i,2}, u_{i_2})$ has been or will be swapped with another entry, too, then swap $(c^\star, u_j)$ with $(c_{i,1}, u_{i_1})$; otherwise, swap $(c^\star, u_j)$ with $(c_{i,4}, u_{i_1})$. This makes sure that all swapped entries in L^i are consecutive. In total, we make n swaps. In the resulting permutation of L, the elements of color u form at most k groups of consecutive entries (as $\mathcal{S}'$ might contain "unnecessary" sets in the decision version). Hence, the number of pairs of consecutive entries with the same color $c^\star$ is at least $n - k = K$.

Now assume that $(\mathcal{G}, \mathcal{C}, L, K)$ is a yes-instance of NodePermutation, that is, there is a permutation π of L such that (i) $\mathcal{G}(\pi(L)) = \mathcal{G}(L)$ and (ii) $N(\pi(L)) \geq K$. We have to show that then $(U, \mathcal{S})$ admits a set cover of size $k = n - K$. Without loss of generality, we can assume that π swaps *each* color-$c^\star$ element of L^0 with a color-$c^\star$ element of $L^1 \oplus \cdots \oplus L^m$ (because such a swap does not decrease $N(\pi(L))$) and that π does not swap any other entries of L (because swapping group-z elements does not change $N(\pi(L))$). Consider the family $\mathcal{S}'$ of those sets $S_i \in \mathcal{S}$ such that π modifies L^i. We claim that (i) $\mathcal{S}'$ covers U and (ii) $|\mathcal{S}'| \leq k$.

Property (i) holds due to our assumption that π swaps all color-$c^\star$ entries of L^0 with a sublist L^i with $i \in [m]$. Thus, every element of U is contained in an element of $\mathcal{S}'$. For property (ii), note that the only pairs of consecutive entries with equal color in $\pi(L)$ are pairs of color-$c^\star$ entries in sublists L^i. Among the n color-$c^\star$ entries, at least K pairs are consecutive. Let K_1, K_2 and K_3 be the number of sublists L^i that contain one, two and three color-$c^\star$ entries, respectively. Then we have $K_2 + 2K_3 \geq K$ and $K_1 = n - 2K_2 - 3K_3$. Hence, the total number of sublists L^i that contain at least one color-$c^\star$ entry is $K_1 + K_2 + K_3 = n - K_2 - 2K_3 \leq n - K = k$, so $|\mathcal{S}'| \leq k$. ◀

3 An Algorithm for a Special Case of NodePermutation

In this section, we describe a linear-time algorithm to solve the optimization version of NodePermutation for the special case that there are only two groups (but an arbitrary number of colors). In ChordLink, this means that the cluster C has a neighborhood of size 2.

▶ **Theorem 2.** *NodePermutation can be solved in $O(n)$ time for two groups.*

Proof. Let $\mathcal{G} = \{x, y\}$. The goal is to find a permutation π of L that maximizes $N(\pi(L))$.

Recall that the elements of L are pairwise disjoint. Thus, $c_{\pi^{-1}(i)} = c_{\pi^{-1}(i+1)}$ implies that $g_i \neq g_{i+1}$. Since there are only two groups, we have either $g_{i+2} = g_i$ or $g_{i+2} = g_{i+1}$, so $c_{\pi^{-1}(i+2)} \neq c_{\pi^{-1}(i+1)}$. Hence, we cannot have $c_{\pi^{-1}(i)} = c_{\pi^{-1}(i+1)} = c_{\pi^{-1}(i+2)}$.

For $\circ \in \{x, y\}$, let $\mathcal{C}_\circ$ be the set of colors c_i such that there exists some list element $(c_i, \circ)$, and let $S = \mathcal{C}_x \cap \mathcal{C}_y$. We say that the color c_j is *assigned* to index i if $c_{\pi^{-1}(i)} = c_j$.

We can formulate this problem as a maximum independent set problem on a graph G'. The graph contains a vertex v'_i for every pair of consecutive indices $i, i+1$ with $g_i \neq g_{i+1}$, and a vertex v'_n if $g_n \neq g_1$. If $v'_i, v'_{i+1} \in V(G')$, then $E(G')$ contains the edge (v'_i, v'_{i+1}). Any assignment of colors to K pairs of consecutive indices $(i_1, i_1+1), \ldots, (i_K, i_K+1)$ with $g_{i_1} \neq g_{i_1+1}, \ldots, g_{i_K} \neq g_{i_K+1}$ induces an independent set $v'_{i_1}, \ldots, v'_{i_K}$ in G'.

Hence, if we find an independent set $(v'_{i_1}, \ldots, v'_{i_k})$ of size k in G', then we can find an assignment of $K = \min\{|S|, k\}$ colors in S to consecutive pairs of indices $(i_1, i_1 + 1), \ldots, (i_K, i_K + 1)$. By construction, G' is either a cycle or a linear forest, so we can find a maximum independent set of G' in linear time with a simple greedy algorithm. To obtain a permutation π of L with $N(\pi(L)) = K$, we arbitrarily assign the remaining colors of $\mathcal{C}_x$ and $\mathcal{C}_y$ to the remaining list elements of types (c_i, x) and (c_i, y), respectively. ◀

4 APX-Hardness of CrossingMinimal ChordInsertion

In this section, we focus on the optimization problem CrossingMinimal ChordInsertion: Given a graph G with at least one copy of each of its vertices placed on a circle R, insert every edge between a copy of each of its endvertices such that the total number of crossings between the edges is minimized. Since the number of crossings only depends on the order of the vertex copies along R, we can also assume them to be drawn as points (rather than circular arcs) on R. We first prove that finding a crossing-free solution is NP-complete.

▶ **Theorem 3.** *It is NP-complete to decide whether a given graph (with node copies on R) admits a crossing-free solution for ChordInsertion.*

Proof. Membership in NP is obvious since we can verify an assignment easily.

To show hardness, we reduce from SAT. Let $(\mathcal{X}, \mathcal{C})$ be a SAT instance with variables $\mathcal{X} = X_0, \ldots, X_n$ and clauses $\mathcal{C} = C_0, \ldots, C_m$. We create a graph G as follows. The vertex set $V(G)$ consists of (i) a vertex s; (ii) for every clause C_i a *clause vertex* c_i; and (iii) for every variable X_j two *variable vertices* x_j and y_j. The edge set $E(G)$ consists of (i) for every clause C_i a *clause edge* (s, c_i); and (ii) for every variable X_j a *variable edge* (x_j, y_j). For every variable X_j, let T_j be the set of clauses that contain the literal X_j, and let F_j be the set of clauses that contain the literal $\neg X_j$.

To create an instance I of ChordInsertion, we place copies of the vertices of $V(G)$ along R as follows. We start with the vertex s, and then append for every variable X_j (i) one copy y_j^{F} of y_j, (ii) for every clause $C_i \in T_j$ a copy c_i^j of c_i, (iii) the vertex x_j, (iv) for every clause $C_i \in F_j$ a copy c_i^j of c_i, (v) a second copy y_j^{T} of y_j; see Fig. 3. Observe that, by the placement of the vertices along R, the only crossings that can occur are between a clause edge (s, c_i) and a variable edge (x_j, y_j) such that clause C_i contains variable X_j.

Assume that I is a yes-instance, i.e., it admits a crossing-free drawing Γ. For every variable X_i, Γ must contain either the chord (x, y_i^{F}) or the chord (x, y_i^{T}); see Fig. 4.

Consider a clause C_i. In Γ, there has to be one chord between s and one of the copies of c_i. Consider any literal X_j contained in C_i. Then, the edge (s, c_i) can be drawn as the chord (s, c_i^j) only if Γ contains the chord (x_i, y_i^{T}); otherwise, there would be a crossing

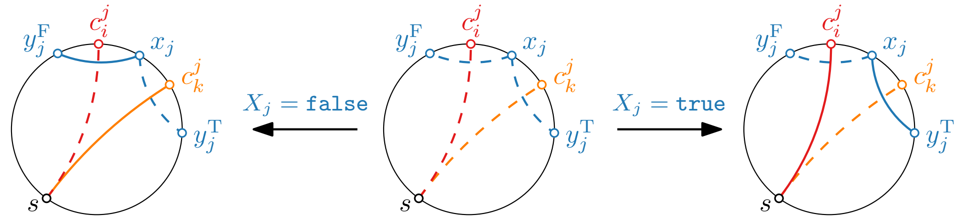


Figure 3 Illustration for the part corresponding to variable X_j in the reduction from SAT to ChordInsertion for the formula $c_i \wedge c_k \wedge \cdots = (X_j \vee \ldots) \wedge (\overline{X_j} \vee \ldots) \wedge \ldots$.

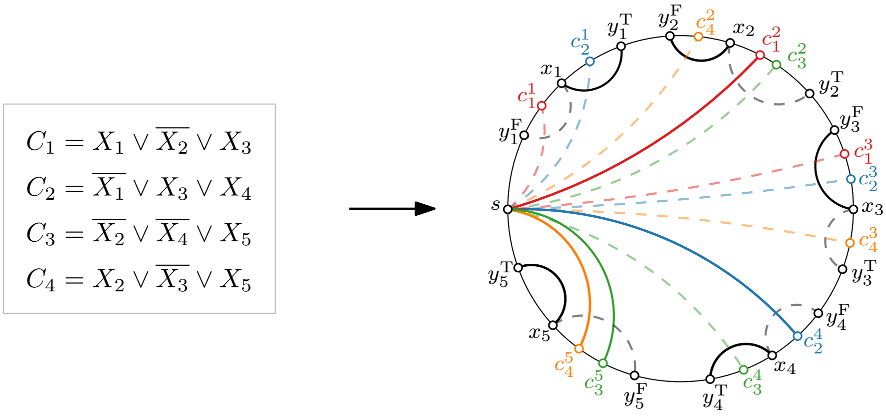


Figure 4 Example of the reduction from SAT to ChordInsertion

between (s, c_i^j) and (x_i, y_i^{F}). Conversely, for any literal $\neg X_j$ contained in C_i, the edge (s, c_i) can be drawn as the chord (s, c_i^j) only if Γ contains the chord (x_i, y_i^{F}). Hence, for the edge (s, c_i) to be drawn in Γ, for at least one of its literals X_j there must be the chord (x_j, y_j^{T}), or for at least one of its literals $\neg X_j$ there must be the chord (x_j, y_j^{F}).

Thus, we can obtain a feasible solution for the SAT instance $(\mathcal{X}, \mathcal{C})$ as follows: if Γ contains the chord (x_j, y_j^{T}), then set $X_j = \texttt{true}$, otherwise (Γ contains the chord (x_j, y_j^{F})), set $X_j = \texttt{false}$. Since every edge (s, c_i) is drawn as a chord in Γ, at least one literal in every clause C_i must be true, so the solution is feasible.

Conversely, if $(\mathcal{X}, \mathcal{C})$ is a yes-instance, then we can obtain a crossing-free drawing Γ for I by using the chord $(x_j, y_j)^{\mathrm{T}}$ for every `true` variable X_j, the chord $(x_j, y_i)^{\mathrm{F}}$ for every `false` variable X_j, and a chord (s, c_i^j) for every clause C_i with satisfied literal X_j. ◀

In the above proof we reduced from SAT. If we instead reduce from Max-2-SAT, which is APX-hard [6], we can show that our problem is even APX-hard.

▶ **Theorem 4.** *CrossingMinimal ChordInsertion is APX-hard even if there are at most two possible choices for every edge.*

References

1 Patrizio Angelini and Giordano Da Lozzo. Beyond clustered planar graphs. In Seok-Hee Hong and Takeshi Tokuyama, editors, *Beyond Planar Graphs: Communications of NII Shonan Meetings*, pages 211–235. Springer, 2020. doi:10.1007/978-981-15-6533-5_12.

2 Patrizio Angelini, Giordano Da Lozzo, Giuseppe Battista, Fabrizio Frati, Maurizio Patrignani, and Ignaz Rutter. Intersection-link representations of graphs. *J. Graph Algorithms Appl.*, 21:731–755, 2017. doi:10.7155/jgaa.00437.

3 Lorenzo Angori, Walter Didimo, Fabrizio Montecchiani, Daniele Pagliuca, and Alessandra Tappini. Hybrid graph visualizations with ChordLink: Algorithms, experiments, and applications. *IEEE Trans. Vis. Comput. Graphics*, 28(2):1288–1300, 2020. URL: https://arxiv.org/abs/1908.08412, doi:10.1109/TVCG.2020.3016055.

4 Punam Bedi and Chhavi Sharma. Community detection in social networks. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, 6(3):115–135, 2016. doi:10.1002/widm.1178.

5 Michael R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete problems. In Robert L. Constable, Robert W. Ritchie, Jack W. Carlyle, and Michael A. Harrison, editors, *Proc. 6th Ann. ACM Symp. Theory Comput. (STOC)*, pages 47–63, 1974. doi:10.1145/800119.803884.

6 Johan Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001. doi:10.1145/502090.502098.

7 Nathalie Henry Riche, Jean-Daniel Fekete, and Michael McGuffin. Nodetrix: A hybrid visualization of social networks. *IEEE Trans. Vis. Comput. Graphics*, 13(6):1302–1309, 2007. doi:10.1109/TVCG.2007.70582.

8 Martin Krzywinski, Jacqueline Schein, Inanç Birol, Joseph Connors, Randy Gascoyne, Doug Horsman, Steven J. Jones, and Marco A. Marra. Circos: An information aesthetic for comparative genomics. *Genome Res.*, 19(9):1639–1645, 2009. doi:10.1101/gr.092759.109.

9 Hassan Mahmoud, Francesco Masulli, Stefano Rovetta, and Giuseppe Russo. Community detection in protein-protein interaction networks using spectral and graph approaches. In Enrico Formenti, Roberto Tagliaferri, and Ernst Wit, editors, *Proc. 10th Int. Meeting Comput. Intell. Methods for Bioinf. Biostat. (CIBB)*, volume 8452 of *Lect. Notes Comput. Sci.*, pages 62–75. Springer, 2013. doi:10.1007/978-3-319-09042-9_5.

Unfolding the Simplex and Orthoplex

Satyan L. Devadoss[1] **and Matthew Harvey**[2]

1 **University of San Diego**
devadoss@sandiego.edu

2 **The University of Virginia's College at Wise**
msh3e@uvawise.edu

Abstract

Over a decade ago, it was shown that every edge unfolding of the Platonic solids was without self-overlap, yielding a valid net. We consider this property for regular polytopes in higher dimensions, notably the simplex, the cube, and the orthoplex. It was recently proven that all unfoldings of the n-cube yield nets. We show that this property holds for the n-simplex and the 4-orthoplex but fails for any orthoplex of higher dimension.

Related Version arXiv:2111.01359

1 Introduction

The study of unfolding polyhedra was popularized by Albrecht Dürer in the early 16th century who first recorded examples of polyhedral *nets*, connected edge unfoldings of polyhedra that lay flat on the plane without overlap. Motivated by this, Shephard [7] conjectures that every convex polyhedron can be cut along certain edges to admit a net. This claim remains tantalizingly open and has resulted in numerous areas of exploration.

We consider this question for higher-dimensional *polytopes*: The codimension-one faces of a polytope are *facets* and its codimension-two faces are *ridges*. The analog of an edge unfolding of polyhedron is the *ridge unfolding* of an n-dimensional polytope: the process of cutting the polytope along a collection of its ridges so that the resulting (connected) arrangement of its facets develops isometrically into an $\mathbb{R}^{n-1}$ hyperplane. In our work, instead of trying to find one valid net for each convex polyhedron (as posed by Shephard), we consider a more aggressive property:

▶ **Definition.** *A polytope P is* all-net *if every ridge unfolding of P yields a valid net.*[1]

A decade ago, Horiyama and Shoji [4] showed that the five Platonic solids are all-net. Figure 1 shows the 11 different unfoldings (up to symmetry) of the octahedron, all of which are nets. The higher-dimensional analogs of the Platonic solids are the regular polytopes. Three classes of regular polytopes exist for all dimensions: the n-simplex, n-cube, and n-orthoplex.[2] It was recently shown that the n-cube is all-net [2]. We prove that the n-simplex and 4-orthoplex are as well. Surprisingly, for all $n > 4$, the n-orthoplex fails to be all-net.

▶ Remark. Sam Zhang [9] has built a lovely interactive software that creates every net of the 4-cube, 4-simplex, and 4-orthoplex by drawing spanning trees on its dual 1-skeleton.

[1] This nomenclature comes from Joe O'Rourke: a basketball "all-net" shot scores by not touching the rim, as all unfoldings become successful nets by facets not overlapping and touching each other.

[2] The orthoplex is dual to the cube and is sometimes called the *cross-polytope* or the *cocube*.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

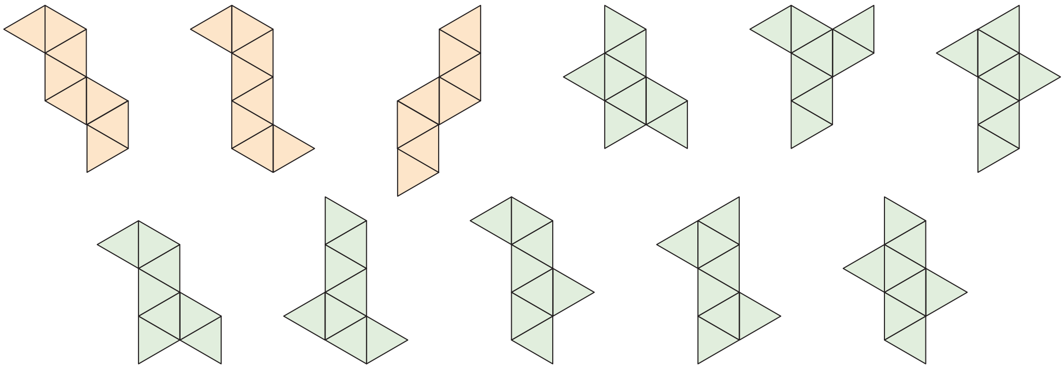


Figure 1 The 11 nets of the octahedron, also known as the 3-orthoplex.

2 Unfolding the Simplex

We explore ridge unfoldings of a convex polytope P by focusing on the combinatorics of the arrangement of its facets in the unfolding. In particular, a ridge unfolding induces a spanning tree in the 1-skeleton of the dual of P: a tree whose nodes are the facets of the polytope and whose edges are the uncut ridges between the facets [6]. Our focus throughout this paper will be on the n-simplex and the n-orthoplex, both of whose facets are $(n-1)$-simplices. First, we study paths in the 1-skeleton, corresponding to a chain of unfolded simplicial facets.

▶ **Definition.** *A* list $\mathcal{L} = \langle a_1, a_2, \ldots, a_k \rangle$ *is a sequence of numbers from* $\{1, \ldots, n\}$ *(possibly with repeats) where no number is listed twice in a row.*

Label the vertices of the $(n-1)$-simplex S with the numbers $1, \ldots, n$. Given a list $\mathcal{L}$ with k elements, we construct a chain $C(\mathcal{L})$ of $k+1$ simplices from the list as follows: Starting with $S = S_1$, attach a simplex S_2 to S_1 on the facet of S_1 that is opposite vertex a_1. Note that all but one of the vertices of S_2 will inherit a label from S_1 and we label the remaining one a_1. Attach a third simplex S_3 to S_2 on the facet opposite vertex a_2, and extend the labeling from S_2 to S_3 as before, and continue in this matter until the list is exhausted. Figure 2 shows this process in action for the list $\langle 3, 2, 3 \rangle$, creating a chain of four 2-simplices.

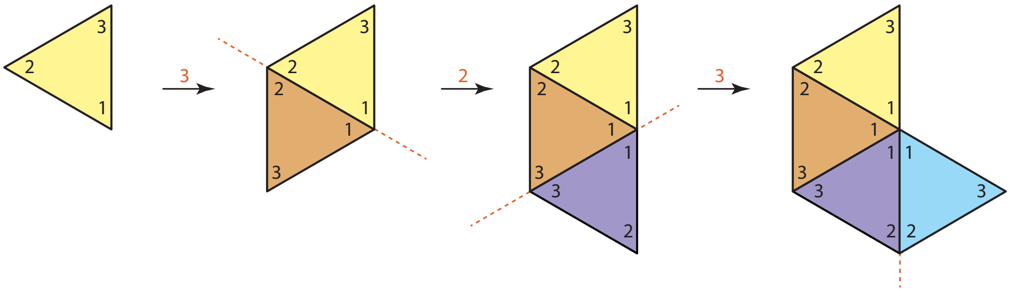


Figure 2 The chain of simplices assembled from the list $\langle 3, 2, 3 \rangle$.

We now introduce a coordinate system to capture the geometry. Begin by placing the n vertices of the $(n-1)$-simplex S at the standard basis vectors e_i of $\mathbb{R}^n$. Note that the coordinates of its vertices are recorded as the column vectors of the $n \times n$ identity matrix. The rest of the chain is then placed in the hyperplane $x_1 + \cdots + x_n = 1$ by a sequence of reflections. Let ρ denote the reflection of S across its facet opposite the vertex (say v) labeled with number a_1. Thus, ρ fixes all vertices except for v; see Figure 3.

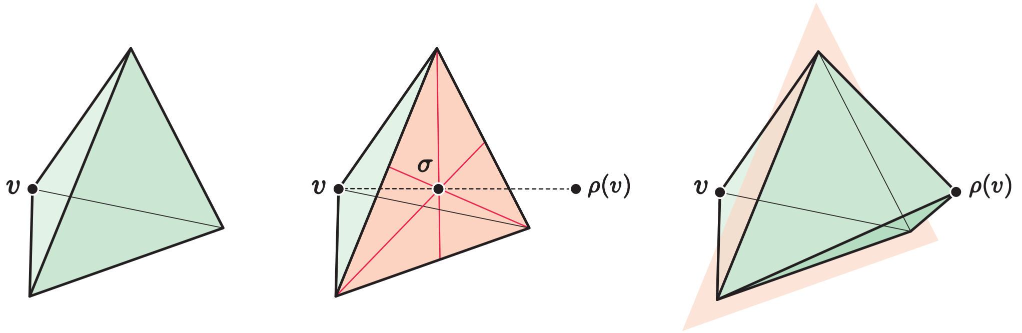


Figure 3 The reflection of the vertex across the opposite face.

To calculate the coordinate of $\rho(v)$ in $\mathbb{R}^n$, we first find the center σ of the facet opposite v, given by $\sigma = 1/(n-1) \cdot (1, \ldots,\ 0,\ \ldots, 1)$, where 0 occurs in the a_1-th coordinate. Since σ bisects the segment from v to $\rho(v)$,

$$\rho(v) = \ v + 2\ \overrightarrow{v\,\rho(v)} \ = \ (0, \ldots, 1, \ldots 0) \ + \ 2 \left(\frac{1}{n-1}, \ldots, -1, \ldots, \frac{1}{n-1} \right) ,$$

where the -1 occurs in the a_1-th coordinate. Hence the reflection ρ is given by a matrix M_{a_1}, which is the identity except for $\rho(v)$ in the a_1-th column. Thus the coordinates of the i-th vertex of S_2 are recorded in the i-th column v_i of $N_1 = M_{a_1}$. By change of coordinates, its image under the reflection from S_2 to S_3 is

$$N_1 M_{a_2} N_1^{-1} v_i = N_1 M_{a_2} e_i \, ,$$

and thus, the coordinates of the i-th vertex of S_3 are recorded in the i-th column of $N_2 = N_1 M_{a_2}$. Note that because M_{a_2} affects only the a_2 column, N_1 and N_2 differ only in the a_2 column. Continuing in this way, the vertices of S_{k+1} are recorded as the columns of $N_k = N_{k-1} M_{a_k}$.

An n-simplex has $n+1$ facets, and each is adjacent to every other. Thus, any listing of the facets (without repeat) describes a chain. However, because the full symmetric group acts transitively on the simplex, there is essentially only one chain, say $\langle 1, 2, \ldots, n \rangle$. Since the first facet is exactly the portion of $\sum x_i = 1$ that lies in the first orthant, a subsequent facet will only intersect the first if it contains a point that has all positive coordinates. This never happens, and a detailed proof is given in [3, Section 2.3]. Hence:

▶ **Theorem 1.** *Every unfolding of the n-simplex yields a net.*

3 Orthoplex Combinatorics and Geometry

In contrast to the simplex, both the unfoldings of the n-orthoplex and the chains within these unfoldings exhibit considerable variety. Unfoldings of the n-orthoplex are in bijection with spanning trees of the 1-skeleton of the n-cube. Consider the following approach to record paths on this skeletal structure: Position the n-cube with antipodal vertices at $(0, \ldots, 0)$ and $(1, \ldots, 1)$. A path along the edges of this cube is encoded as a list of binary numbers (sometimes called a *Gray code*) where exactly one digit changes from one entry to the next. For our work, our list $\mathcal{L}$ simply records the digit entry that changes in moving from one vertex to another. By duality, the ridges of the orthoplex inherit these labels and the process of unfolding the chain corresponds to the construction of $C(\mathcal{L})$.

▶ **Example.** *Consider the Gray code* $\langle 101, 100, 110, 111\rangle$ *associated to the list* $\langle 3, 2, 3\rangle$*. Figure 4(a) shows the path on four vertices of the cube, (b) corresponding to four adjacent facets of the octahedron, (c) resulting in a partial chain unfolding. Compare to Figure 2.*

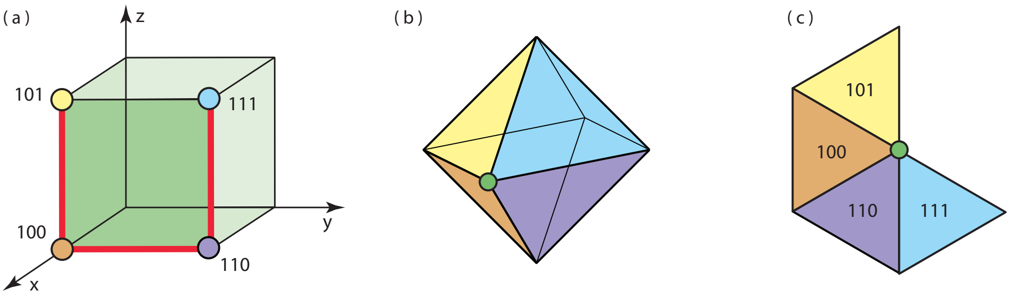


Figure 4 Path on the 3-cube and a partial unfolding of the octahedron.

▶ Remark. Up to symmetry, there are just three spanning paths on the 1-skeleton of the 3-cube: $\langle 1, 2, 1, 3, 1, 2, 1\rangle$, $\langle 1, 2, 1, 3, 2, 1, 2\rangle$, and $\langle 1, 2, 3, 2, 1, 2, 3\rangle$, corresponding to the first three highlighted nets shown in Figure 1. The situation escalates rapidly as n increases: there are 238 spanning paths on the 4-cube and 48,828,036 on the 5-cube [5].

▶ **Definition.** *A list of numbers from* $\{1, \ldots, n\}$ *is* valid *if it corresponds to a path on the* n*-cube.*

A list is valid as long as the route it describes on the cube does not cross itself, which can be characterized as follows:

▶ **Lemma 2.** *A list is valid if and only if it contains no sublist of consecutive entries in which each entry occurs an even number of times.*

▶ Remark. With this characterization, it is straightforward to create an algorithm to build valid lists: recursively append numbers $\{1, \ldots, n\}$ and check whether any of the new consecutive sublists have entries that occur an even number of times.

The question of whether two facets overlap depends on how close they are to each other, which can be estimated by calculating the distance between their centroids. If the vertices are $v_i = (a_{i1}, \ldots a_{in})$, the centroid is found by averaging their coordinates:

$$\left(\frac{1}{n}\sum a_{1j},\ \ldots,\ \frac{1}{n}\sum a_{nj}\right).$$

It is straightforward to calculate the necessary distances:

▶ **Lemma 3.** *Let* d *denote the distance between the centroids of two* $(n-1)$*-simplex facets of the* n*-orthoplex in an unfolding. If* $d < 2/\sqrt{n(n-1)}$*, the facets must intersect. If* $d > 2\sqrt{(n-1)/n}$*, the facets cannot intersect.*

4 Orthoplex Unfolding

This section proves that the 4-orthoplex is all-net. We do this by extending paths on the skeleton of the 4-cube. While any path along a 3-cube can always be extended to a spanning path, this is not true for $n \geq 4$. For example, Figure 5(a) shows the 1-skeleton of the 4-cube, and the blue path shown in (b) cannot be extended further.

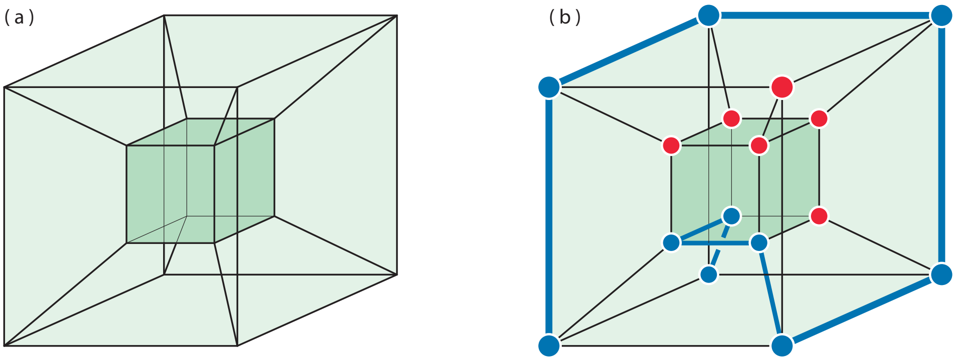


Figure 5 A path in the 4-cube that cannot be further extended.

Notice that a path can no longer be extended only when it has already crossed through all vertices adjacent to its two endpoints. Each vertex of the 4-cube is adjacent to four others, so roughly speaking, we might expect a path to pass through eight additional points before reaching its end. It is not quite this simple, because some of these points may overlap, but by considering the possible configurations, we arrive at a slightly weaker result.

▶ **Lemma 4.** *A path on the skeleton of the 4-cube can be extended to connect at least nine vertices.*

Rephrasing Lemma 4, any valid list can be extended to a valid list with at least eight entries. There are relatively few valid lists with eight entries, and by direct inspection it can be seen that they all yield nets, so:

▶ **Lemma 5.** *Every valid list containing exactly eight entries unfolds to form a partial net of the 4-orthoplex.*

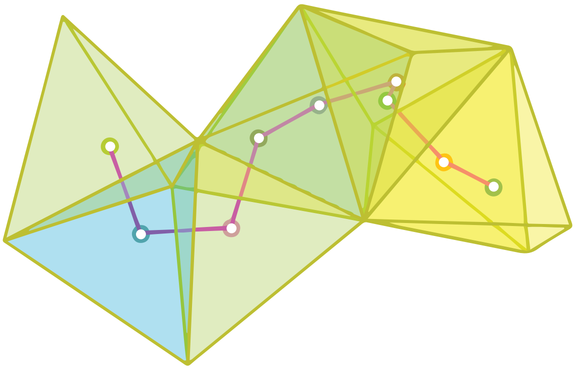


Figure 6 The partial unfolding corresponding to a valid list with length eight.

In unfoldings corresponding to longer lists, individual facets may be separated by more than eight facets. In these cases, we can calculate the distance between centroids. In every case, the distance is large enough to guarantee that the facets do not intersect, so:

▶ **Lemma 6.** *If two facets of the 4-orthoplex are separated by eight or more facets, they cannot overlap.*

Buekenhout and Parker [1] enumerate 110,912 ridge unfoldings of the 4-orthoplex. The following guarantees that they are all valid nets.

▶ **Theorem 7.** *The 4-orthoplex is all-net.*

Proof. If there were an unfolding that did not yield a net, then there would be a path between two of its overlapping facets. By Lemma 6, those facets must be separated by fewer than eight intervening facets along the path, corresponding to a valid list $\mathcal{L}$ whose length is at most eight. By Lemma 4, that list can be extended to one whose length is exactly eight. As described in Lemma 5, none of the unfolds generated by these lists exhibit overlap. ◀

Moving to higher dimensions, although the n-cube is all-net [2], its dual is not:

▶ **Theorem 8.** *For each $n > 4$, the n-orthoplex is not all-net.*

Proof. In dimensions 5 – 9, specific lists demonstrate overlap using centroid arguments:

$$\begin{array}{lcl} \text{dim. } 5 & : & \langle 1,2,3,4,2,1,5,4,2,4,5,4,2,1,5,4,3,1,5 \rangle \\ \text{dim. } 6 & : & \langle 1,2,3,1,4,5,4,3,5,4,1,3,2,1,4 \rangle \\ \text{dim. } 7,\ 8 & : & \langle 1,2,3,4,1,5,3,5,4,3,2,1 \rangle \\ \text{dim. } 9 & : & \langle 1,2,3,4,2,4,1,2,3 \rangle. \end{array}$$

It turns out that the dimension 9 example fails to unfold to a net for any $n > 9$. However, in higher dimensions, the centroid measurements become less robust. Instead, we return to the idea used in the simplex proof. It suffices to show that a point in the tenth facet has all positive coordinates. The point $v = 1/(n-1)\langle 1,0,1,1,\ldots,1\rangle$ is the midpoint of the ridge of the first facet. It can be shown that its image

$$M_1 \cdot M_2 \cdot M_3 \cdot M_4 \cdot M_2 \cdot M_4 \cdot M_1 \cdot M_2 \cdot M_3 \cdot v$$

in the tenth facet has all positive coordinates. Details are provided in [3, Section 4.3]. ◀

There are only three additional regular polytopes whose all-net property has not been studied, all of which are four-dimensional: the 24-cell, 120-cell, and 600-cell. The number of distinct unfoldings of these three polytopes are enumerated in [1]:

$$\begin{array}{rcl} \text{24-cell} & : & 6\ (2^{19} \cdot 5688888889\ +\ 347) \\ \text{120-cell} & : & 2^7 \cdot 5^2 \cdot 7^3\ (2^{114} \cdot 3^{78} \cdot 5^{20} \cdot 7^{33}\ +\ 2^{47} \cdot 3^{18} \cdot 5^2 \cdot 7^{12} \cdot 53^5 \cdot 2311^3\ +\ 239^2 \cdot 3931^2) \\ \text{600-cell} & : & 2^{188} \cdot 3^{102} \cdot 5^{20} \cdot 7^{36} \cdot 11^{48} \cdot 23^{48} \cdot 29^{30} \end{array}$$

The unfolding enumerations for these three exceptional polytopes encourage us to conjecture that all of them will fail to be all-net.

▶ Acknowledgments. We thank Nick Bail, Zihan Miao, Andy Nelson, and Joe O'Rourke for helpful conversations. The first author was partially supported by an endowment from the Fletcher Jones Foundation.

References

1 F. Buekenhout and M. Parker. The number of nets of the regular convex polytopes in dimension ≤ 4, *Discrete Mathematics* **186** (1998) 69–94, doi:10.1016/S0012-365X(97)00225-2.

2 K. DeSplinter, S. Devadoss, J. Readyhough, B. Wimberly. Unfolding cubes: nets, packings, partitions, chords, *Electronic Journal of Combinatorics* **27** (2020) 4–41, doi:10.37236/9796.

3 S. Devadoss and M. Harvey. Unfoldings and nets of regular polytopes, arXiv:2111.01359.

4 T. Horiyama and W. Shoji. Edge unfoldings of Platonic solids never overlap, *Canadian Conference on Computational Geometry* (2011).

5 Online Encyclopedia of Integer Sequences. https://oeis.org/A342631.

6 G. Shephard. Angle deficiencies of convex polytopes, *Journal of the London Mathematical Society* **43** (1969) 325–336, doi:10.1112/jlms/s1-43.1.325.

7 G. Shephard. Convex polytopes with convex nets, *Mathematical Proceedings of the Cambridge Philosophical Society* **78** (1975) 389–403, doi:10.1017/s0305004100051860.

8 Sympy: Open source Python library. https://www.sympy.org/.

9 S. Zhang. Unfolding software. https://sam.zhang.fyi/html/unfolding/index.html.

Explicit Dynamic Schnyder Woods Require Linear (Amortized) Update Time *

Aleksander B. G. Christiansen[1], **Jacob Holm**[2], **Eva Rotenberg**[1], **and Carsten Thomassen**[1]

1 **Technical University of Denmark, Lyngby, Denmark**
{abgch,erot,ctho}@dtu.dk
2 **University of Copenhagen (DIKU), Copenhagen, Denmark**
jaho@di.ku.dk

Abstract

In the dynamic edge orientation problem, the usual goal is to orient edges in a way that bounds the maximum out-degree as the graph is subject to insertions and deletions of edges. Brodal & Fagerberg showed that explicitly maintaining an α-bounded out-degree orientation for dynamic graphs of arboricity α requires amortized linear update time [8]. While all planar graphs have arboricity ≤ 3, the construction in [8] is nonplanar. We show that the same lower bound holds for dynamic planar graphs. This immediately implies that one cannot hope to explicitly maintain a dynamic Schnyder wood of a graph with sub-linear amortized update time.

Related Version Partially based on the master's thesis by Christiansen [9, Chapter 3].

1 Introduction

The study of Schnyder woods began with Schnyder's result on realizers [24]; He showed that every planar graph on at least 3 vertices admits a straight-line embedding in the $(n-2)$ by $(n-2)$ grid, and that this embedding can be computed in linear time. Soon a plethora of applications of Schnyder woods were established: From graph drawing and dimension theory [3, 13, 14] to combinatorics and algorithmics [4, 22] and the list continues. See, for instance, Bhore et al. [2] for a more in-depth overview. Schnyder showed that every plane triangulation admits a decomposition of its *inner edges*, i.e. those not incident to the outer face, into three forests. He called such a decomposition a *realizer*. The three forests have since then commonly been referred to as Schnyder woods.

A 3-*orientation* of a plane graph is an orientation of the inner edges such that all vertices have out-degree 3. A realizer (see Figure 1) of a plane graph is then a 3-orientation together with a 3-edge colouring (with colours, say, $c_i, i \in \mathbb{Z}/3$) of all of the inner edges such that:

- Every inner vertex has an out-edge coloured c_0, c_1 and c_2.
- The colours of the out-edges incident to an inner vertex v always appear in the same counter-clockwise ordering (say c_0, c_1, c_2).
- In-edges of colour c_i appear exactly between the out-edges of colour c_{i+1} and c_{i+2}

Brehm [7], Mendez [21] and Bonichon et al. [5] studied different notions of flips that move one Schnyder wood to another. One of the flips they studied is the diagonal flip, on which there is a rich body of research on in its own - both in the combinatorial setting and in the geometrical setting, where the graphs are embedded in the plane. A *diagonal flip*

* Partially supported by Independent Research Fund Denmark grant 2018-2021 (8021-00249B), "Algo-Graph", the VILLUM Foundation grant 16582, "BARC", and the VILLUM Foundation grant 37507, "Efficient Recomputations for Changeful Problems".

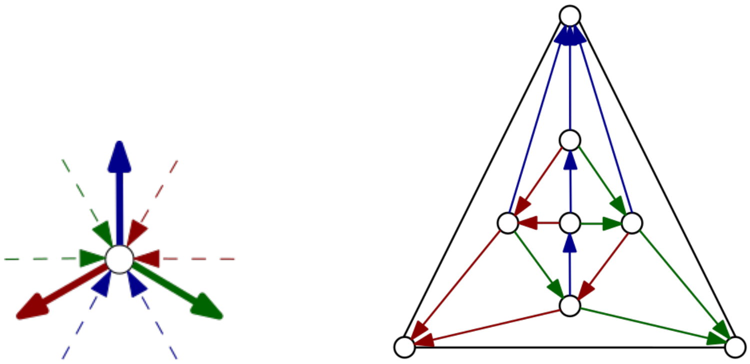


Figure 1 The local structure at a vertex (left), and an example of a realizer (right).

in a triangulation is the action of removing an edge xy incident on faces xyz and vxy and replacing it with the edge vz (see Figure 2, left). These flips also move one realizer to another. Wagner [26] initiated this study by showing that one can go from any triangulation to another using only diagonal flips. See, for example, [6] for a review of the literature on flips. Finally, Bhore et al. [2] studied certain diagonal flips called coloured flips (Figure 2, right). There are two types of *coloured flips* in a plane triangulation G: 1) in a quadrilateral $vyzx$ with diagonal vz oriented from v to z and coloured, say, c_i, yv oriented from y to v and coloured c_{i+1} and $yx \notin G$, we can remove vz and insert yx oriented from y towards x coloured c_{i+1} and reorient yv to be oriented from v towards y and colour it c_i, 2) a symmetrical version if xv is oriented from x towards v (see for example [2]). Bhore et al. [2] show that one can go between any two plane triangulations using only coloured flips. They also give a dynamic algorithm able to maintain a Schnyder wood over a sequence of coloured flips in $O(\log n)$ time per flip. Note that there exist diagonal flips which are not coloured flips, and while they can be simulated by a sequence of coloured flips [2], as we shall later see, this sequence may have length $\Omega(n)$.

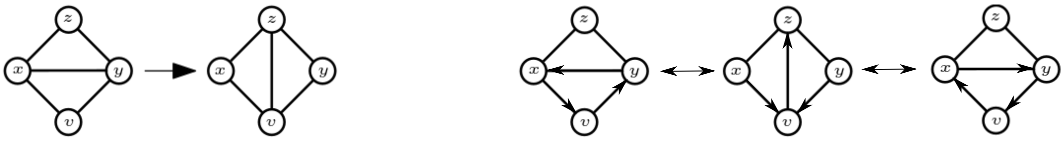


Figure 2 A diagonal flip (left). Coloured flips (right) w.r.t. the specified orientations [2].

Dynamic planar graphs have been studied both in the incremental (edge-insertion only) [10, 17, 23, 27] and fully-dynamic (insertion/deletion) setting [11, 12, 15, 16, 18]. The existence of efficient algorithms for testing planarity of a fully-dynamic graph [12, 16] motivate efforts to dynamically maintain well-known properties of planar graphs, such as e.g. bounded out-degree orientations, colourings, and Schnyder woods.

A k-bounded out-degree orientation of a graph is an orientation of the edges s.t. every vertex v has out-degree $d^+(v) \leq k$ (here $d^+(v)$ denotes the out-degree of v). It is *implicit* if querying an edge-orientation requires computation, and *explicit* otherwise. Maintaining bounded out-degree orientations of dynamic graphs is well-studied [1, 8, 19, 20, 25]. Brodal & Fagerberg showed [8] that it takes linear (even amortized) update time to maintain an explicit α-bounded out-degree orientation of dynamic graphs with arboricity $\leq \alpha$. We show that this explicit lower bound carries over to planar graphs: one cannot explicitly maintain a 3-bounded out-degree orientation of a fully-dynamic planar graph with sublinear update-time.

This immediately implies that one cannot maintain explicit Schnyder woods of a dynamic planar graph even if one gets to choose and change the embedding with respect to which the Schnyder woods are computed.

2 Preliminaries

Fraissex & Mendez and Brehm studied 3-orientations [7, 21]. Brehm showed that a plane triangulation has a unique 3-orientation if and only if it is *stacked* which is equivalent to being 3-degenerate - the property that every subgraph contains a vertex of degree at most 3. In fact this can be slightly generalised - and we will use this slight generalisation later on, when dealing with graphs where no embedding is specified. The proof of the generalisation is similar, so we only provide a sketch:

▶ **Lemma 2.1.** *([7]) Let G be a plane triangulation with a 3-bounded out-degree orientation O. Let x, y, z form a triangle in G, and let H be a component of $G - \{x, y, z\}$, such that $d^+_G(v) = 3$ for all $v \in H$. Then the restriction of any 3-bounded out-degree orientation of G to all edges incident to H is unique if and only if $G[H \cup \{x, y, z\}]$ is 3-degenerate.*

Proof. (Sketch): A counting argument shows that all edges between H and x, y, z are oriented away from H. Indeed, $G[H \cup \{x, y, z\}]$ is planar and so contains at most $3(|H| + 3) - 6$ edges. 3 of these go between x, y, z, so the remaining $3|H|$ edges must be out-edges of vertices in H.

Next, we show that the restriction of any 3-bounded out-degree orientation O to H is unique iff it is acyclic. Indeed, suppose it is not acyclic. Then it has a directed cycle. Flipping the orientation along this cycle creates a new 3-bounded out-degree orientation with a different restriction. The other direction is as follows: suppose that there exists an orientation for which the restriction to H is acyclic, but that this restriction is not unique. Comparing two such restrictions gives an edge which is oriented differently in the two orientations. Now, since every point has out-degree 3 an endpoint cannot be incident to only one such edge, so there is a new edge - oriented differently by the two orientations - that one can follow. Continuing like this eventually gives you a directed cycle in H as, by above, one never reaches x, y, z. This is a contradiction.

The Lemma then follows by noting that having an acyclic 3-bounded out-degree orientation is equivalent to being 3-degenerate. Indeed, beginning at an arbitrary vertex in H and following incoming edges backwards ensures that one ends up in a source in H. This source has degree at most 3. We can remove this vertex and apply induction to see that any subgraph not containing this vertex also has a vertex of degree at most 3. The other direction follows, since the 3-degeneracy implies that in any non-empty subgraph $S \subset H$ one can always find a $v \in S$ of degree 3 in $G[S \cup \{x, y, z\}]$. Beginning from H one can remove such a vertex and orient its incident edges so that it becomes a source. Continuing like this never creates cycles and therefore yields an acyclic 3-bounded out-degree orientation. ◀

3 Explicit lower bounds

Since Schnyder woods are defined with respect to particular embeddings, we shall give a more general explicit lower bound on maintaning 3-bounded out-degree orientations in planar graphs. Since any Schnyder wood trivially gives a 3-bounded out-degree orientation (the outer edges can be oriented arbitrarily), this implies that we cannot maintain a dynamic Schnyder wood - even if the embedding is allowed to change. We give this lower bound by first considering explicit 3-orientations in plane graphs. We have the following Lemma:

▶ **Lemma 3.1.** *Let $\mathcal{A}$ be an algorithm explicitly maintaining a 3-orientation of an n-vertex plane triangulation under diagonal flips. Then the flip operation can be forced to spend $\Omega(n)$ update time, even when considering amortized complexity.*

Proof. Consider the following plane triangulation (see Figure 3) containing a path $s_1, s_2, \cdots, s_k$ of length $k = n - 5 = \Omega(n)$. The plane triangulation is 3-degenerate, and hence by Lemma 2.1, it has a unique 3-orientation: By diagonal flipping the edge uv and subsequently the edge s_1x, one gets a new plane triangulation. It is again 3-degenerate, and hence by Lemma 2.1 it has a unique 3-orientation. (see Figure 4). By diagonally flipping the same edges in the opposite order, one reclaims the original graph. The new 3-orientation has $\Omega(n)$ edges oriented differently compared to the original 3-orientation, but it only requires a constant number of diagonal flips to go between the two graphs. Hence, a constant number of flips forces $\mathcal{A}$ to change $\Omega(n)$ edges, and thus, the update time for the diagonal flip operation must be $\Omega(n)$, even amortized, as one can force this update sequence as many times as desired. ◀

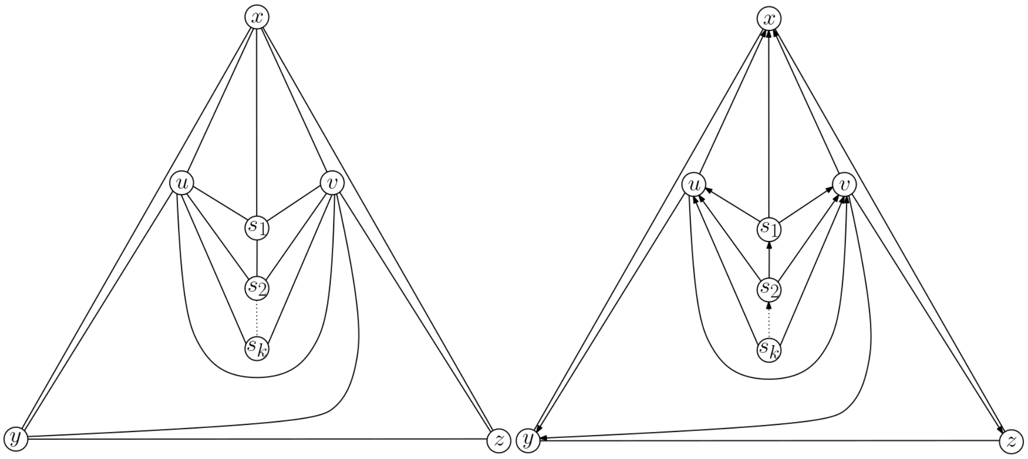


Figure 3 The plane triangulation along with its unique 3-orientation.

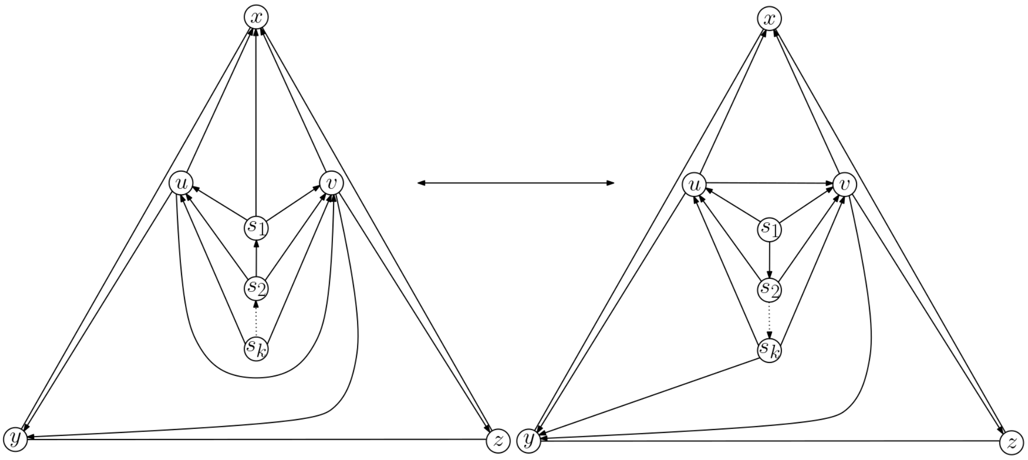


Figure 4 Going between two unique 3-orientations.

As every Schnyder wood is in bijection with the underlying 3-orientation [21], an easy Corollary of this is that one also cannot hope to explicitly and dynamically maintain a

Schnyder wood of a plane triangulation under diagonal flipping in sublinear time.

▶ **Corollary 3.2.** *Let $\mathcal{A}$ be an algorithm explicitly maintaining a Schnyder wood of an n-vertex plane triangulation under inner edge flips. Then the flip operation can be forced to spend $\Omega(n)$ update time, even when considering amortized complexity.*

The goal now is to extend this lower bound to 3-bounded out-degree orientations of planar graphs under insertion/deletion of edges. There are only three things to consider before doing such an extension. Firstly, now we support the operations insertion/deletion of edges and not the diagonal flip. This is however a non-issue since a diagonal flip can be simulated by first deleting the edge and then inserting the other diagonal. Secondly, we now consider 3-bounded out-degree orientations and so outer vertices also have out-edges, and not all inner vertices are required to have out-degree 3. Lastly, the lower bound should apply not only to plane graphs where an embedding is chosen, but also to planar graphs. We deal with the last two points by using at least 13 copies of the graph from above. Then, in at least one of the copies, all inner vertices must have out-degree 3 - both before and after a sequence of updates. Hence, we can do the aforementioned updates in all 13 copies, and this will then ensure that at least one copy has to behave as in Lemma 3.1. Formally, we show:

▶ **Theorem 3.3.** *Let $\mathcal{A}$ be an algorithm explicitly maintaining a 3-bounded out-degree orientation of an n-vertex planar graph under insertion and deletion of edges. Then the insert or the delete operation can be forced to spend $\Omega(n)$ update time, even when considering amortized complexity.*

Proof. Create a planar graph G by placing 13 copies of the graph P from the proof of Lemma 3.1 in the plane, and triangulating arbitrarily. Let $n = |V(G)|$. For each copy P_i of P, we let the set I_i resp. O_i be the set of vertices that are inner resp. outer vertices of P_i, if P_i is embedded as in Lemma 3.1. In particular, for a specific copy of P, say P_i, the corresponding set I_i has size $|I_i| = 2 + \frac{n-5\cdot 13}{13} = \Omega(n)$. Since G is a plane triangulation, it follows from Euler's Theorem that $|E(G)| = 3n - 6$. This implies that at most 6 vertices of G can have out-degree strictly less than 3. Hence, in at least 7 of the 13 copies of P, every vertex in I has out-degree 3. Since the O vertices of each of these specific copies of P form a triangle, it follows from Lemma 2.1 that the edges incident to I must have the same orientation as in the plane embedding in Lemma 3.1 and that this orientation is unique.

Now, we simulate the flip sequence used in Lemma 3.1 in each copy. Doing this in all 13 copies only requires 26 insertions and 26 deletions in total. After these alterations, it is still the case that in at least 7 of the 13 copies of P every vertex in I has out-degree 3. Furthermore, since the O vertices of each of these specific copies of P form a triangle, it follows from Lemma 2.1 that the edges incident to I must have the same orientation as in the altered plane embedding in Lemma 3.1 and that this orientation is unique. At least one copy P_i of P has out-degree 3 at every vertex in I_i in both the orientation before and in the orientation after the update sequence. Consequentially, $\mathcal{A}$ must have flipped at least $|I_i| - 3 = \Omega(n)$ edge-orientations during the update sequence. The update sequence consists of only a constant number of updates, and it can be reversed by first deleting uv and re-inserting it across the opposite diagonal in each copy, and then doing the same for $s_k y$ in every copy of P. Each reversal of the update sequence, requires only a constant number of updates, but forces $\mathcal{A}$ to change at least $\Omega(n)$ edge-orientations. Hence, it follows that either delete or insert must incur an update cost of at least $\Omega(n)$, even when considering amortized cost. ◀

Since any Schnyder wood can be extended to a 3-bounded out-degree orientation, we get:

▶ **Corollary 3.4.** *Let G be a fully-dynamic graph with n vertices subject to edge insertions and deletions, and let $\mathcal{A}$ be an algorithm explicitly maintaining a Schnyder wood of a plane embedding of a plane triangulation of G. Then delete or insert can be forced to spend $\Omega(n)$ update time, even when considering amortized complexity.*

In [8], Brodal & Fagerberg showed that one cannot explicitly maintain an α-bounded out-degree orientation in a dynamic graph with arboricity α with sub-linear update time for either deletetion or insertion. For $\alpha = 3$, the dynamic graph they use is not planar, however, Theorem 3.3 shows that the same is the case for planar graphs.

Acknowledgments. We thank Irene Parada for helpful discussions, and for pointing us to the work of Bhore et al [2].

References

1 Edvin Berglin and Gerth Stolting Brodal. A Simple Greedy Algorithm for Dynamic Graph Orientation. In *28th International Symposium on Algorithms and Computation (ISAAC 2017)*, volume 92 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:12, Dagstuhl, Germany, 2017. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: http://drops.dagstuhl.de/opus/volltexte/2017/8263, doi:10.4230/LIPIcs.ISAAC.2017.12.

2 Sujoy Bhore, Prosenjit Bose, Pilar Cano, Jean Cardinal, and John Iacono. Dynamic schnyder woods. *CoRR*, abs/2106.14451, 2021. URL: https://arxiv.org/abs/2106.14451, arXiv:2106.14451.

3 Nicolas Bonichon, Stefan Felsner, and Mohamed Mosbah. Convex drawings of 3-connected plane graphs. *Algorithmica*, 47(4):399–420, 2007. doi:10.1007/s00453-006-0177-6.

4 Nicolas Bonichon, Cyril Gavoille, Nicolas Hanusse, and Ljubomir Perkovic. Plane spanners of maximum degree six. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part I*, volume 6198 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 2010. doi:10.1007/978-3-642-14165-2_3.

5 Nicolas Bonichon, Bertrand Le Saëc, and Mohamed Mosbah. Wagner's theorem on realizers. In *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 1043–1053. Springer, 2002. doi:10.1007/3-540-45465-9_89.

6 Prosenjit Bose and Ferran Hurtado. Flips in planar graphs. *Computational Geometry*, 42(1):60–80, 2009. URL: https://www.sciencedirect.com/science/article/pii/S0925772108000370, doi:https://doi.org/10.1016/j.comgeo.2008.04.001.

7 Enno Brehm. 3-orientations and schnyder 3-tree-decompositions. Diploma Thesis. FB Mathematik und Informatik, Freie Universität Berlin, 2000.

8 Gerth Stolting Brodal and Rolf Fagerberg. Dynamic representations of sparse graphs. In *In Proc. 6th International Workshop on Algorithms and Data Structures (WADS)*, pages 342–351. Springer-Verlag, 1999.

9 Aleksander B. G. Christiansen. Dynamic algorithms for implicit vertex-colouring of graphs with bounded arboricity. Master's thesis, Technical University of Denmark, Kgs. Lyngby, Denmark, October 2021.

10 Giuseppe Di Battista and Roberto Tamassia. Incremental planarity testing (extended abstract). In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 436–441, 1989. doi:10.1109/SFCS.1989.63515.

11 David Eppstein. Dynamic generators of topologically embedded graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA.*, pages 599–608, 2003. URL: http://dl.acm.org/citation.cfm?id=644108.644208.

12 David Eppstein, Zvi Galil, Giuseppe F. Italiano, and Thomas H. Spencer. Separator based sparsification: I. planarity testing and minimum spanning trees. *Journal of Computer and Systems Sciences*, 52(1):3–27, February 1996. doi:10.1006/jcss.1996.0002.

13 Stefan Felsner. The order dimension of planar maps revisited. *SIAM J. Discret. Math.*, 28(3):1093–1101, 2014. doi:10.1137/130945284.

14 Stefan Felsner and Johan Nilsson. On the order dimension of outerplanar maps. *Order*, 28(3):415–435, 2011. doi:10.1007/s11083-010-9181-1.

15 Zvi Galil, Giuseppe F. Italiano, and Neil Sarnak. Fully dynamic planarity testing with applications. *Journal of the ACM*, 46(1):28–91, 1999. doi:10.1145/300515.300517.

16 Jacob Holm and Eva Rotenberg. Fully-dynamic planarity testing in polylogarithmic time. In *Proccedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, Chicago, IL, USA, June 22-26, 2020*, pages 167–180. ACM, 2020. doi:10.1145/3357713.3384249.

17 Jacob Holm and Eva Rotenberg. Worst-case polylog incremental spqr-trees: Embeddings, planarity, and triconnectivity. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2378–2397. SIAM, 2020. Full version: arXiv.org/1910.09005. doi:10.1137/1.9781611975994.146.

18 Giuseppe F. Italiano, Johannes A. La Poutré, and Monika Rauch. Fully dynamic planarity testing in planar embedded graphs (extended abstract). In *Algorithms - ESA '93, First Annual European Symposium, Bad Honnef, Germany, September 30 - October 2, 1993, Proceedings*, pages 212–223, 1993. doi:10.1007/3-540-57273-2_57.

19 Tsvi Kopelowitz, Robert Krauthgamer, Ely Porat, and Shay Solomon. Orienting fully dynamic graphs with worst-case time bounds. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part II*, volume 8573 of *Lecture Notes in Computer Science*, pages 532–543. Springer, 2014. doi:10.1007/978-3-662-43951-7_45.

20 Łukasz Kowalik. Adjacency queries in dynamic sparse graphs. *Inf. Process. Lett.*, 102(5):191–195, May 2007. doi:10.1016/j.ipl.2006.12.006.

21 Patrice Ossona de Mendez. *Orientations bipolaires.* PhD thesis, 1994. Thašse de doctorat dirigae par Rosenstiehl, Pierre Mathamatiques et informatique appliquaes aux sciences sociales Paris, EHESS 1994. URL: http://www.theses.fr/1994EHES0025.

22 Dominique Poulalhon and Gilles Schaeffer. Optimal coding and sampling of triangulations. In *Automata, Languages and Programming, 30th International Colloquium, ICALP 2003, Eindhoven, The Netherlands, June 30 - July 4, 2003. Proceedings*, volume 2719 of *Lecture Notes in Computer Science*, pages 1080–1094. Springer, 2003. doi:10.1007/3-540-45061-0_83.

23 Johannes A. La Poutré. Alpha-algorithms for incremental planarity testing (preliminary version). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 706–715, 1994. doi:10.1145/195058.195439.

24 Walter Schnyder. Embedding planar graphs on the grid. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, page 138–148, USA, 1990. Society for Industrial and Applied Mathematics.

25 Shay Solomon and Nicole Wein. Improved dynamic graph coloring. *ACM Trans. Algorithms*, 16(3), June 2020. doi:10.1145/3392724.

26 Klaus Wagner. Bemerkungen zum Vierfarbenproblem. *Jahresbericht der Deutschen Mathematiker-Vereinigung*, 46:26–32, 1936. URL: `http://eudml.org/doc/146109`.

27 Jeffery Westbrook. Fast incremental planarity testing. In *Automata, Languages and Programming, 19th International Colloquium, ICALP92, Vienna, Austria, July 13-17, 1992, Proceedings*, pages 342–353, 1992. `doi:10.1007/3-540-55719-9\_86`.

Well-Separation and Hyperplane Transversals in High Dimensions[*]

Helena Bergold[†1], Daniel Bertschinger[2], Nicolas Grelier[3], Wolfgang Mulzer[‡4], and Patrick Schnider[5]

1 Institut für Informatik, Freie Universität Berlin
helena.bergold@fu-berlin.de
2 Department of Computer Science, ETH Zürich
daniel.bertschinger@inf.ethz.ch
3 Department of Computer Science, ETH Zürich
nicolas.grelier@inf.ethz.ch
4 Institut für Informatik, Freie Universität Berlin
mulzer@inf.fu-berlin.de
5 Department of Mathematical Sciences, University of Copenhagen
ps@math.ku.dk

Abstract

A family of k point sets in d dimensions is *well-separated* if the convex hulls of any two disjoint subfamilies can be separated by a hyperplane. This notion is instrumental in showing that certain generalized ham-sandwich cuts exist. But how hard is it to check whether a given family of high-dimensional point sets has this property? Starting from this question, we study several algorithmic aspects of the existence of high-dimensional transversals and separations.

1 Introduction

Given a family of k sets $S_1, \ldots, S_k$ in $\mathbb{R}^d$, we say that the family is *well-separated* if for any proper index set $I \subset [k]$, with $I \neq \emptyset$ and $I \neq [k]$, the convex hulls of S_I and $S_{[k]\setminus I}$ can be separated by a hyperplane, where we define $S_J = \cup_{j \in J} S_j$, for any proper index set $J \subset [k]$. Well-separation is equivalent to the fact that for any proper index set I, the convex hulls of S_I and $S_{[k]\setminus I}$ do not intersect. A hyperplane h is a *transversal* if $S_i \cap h \neq \emptyset$ for all $i \in [k]$. More generally, an *m-flat* (i.e., an affine subspace of dimension m) is an m-transversal if it intersects all the sets of the family. It turns out that well-separation is intimately related to transversals: a family of sets $S_1, \ldots, S_k$ is well-separated if and only if there is no $(k-2)$-transversal of the convex hulls of $S_1, \ldots, S_k$. Observe that for any family of $k \leq d$ sets, there always exists a $(k-1)$-transversal. Indeed choose a point from each of the k sets, and consider a $(k-1)$-flat that contains these k points. Furthermore due to Radon's theorem a family of $d+2$ sets in dimension d cannot be well-separated. Radon's theorem states that any set of $d+2$ points in dimension d can be partitioned into two sets with intersecting convex hulls. Questions related to transversals have been studied extensively, mostly from a combinatorial, but also from a computational perspective. For more background, we refer the interested readers to the relevant surveys [2, 10, 11].

[*] This work was initiated at the 5th DACH Workshop on Arrangements and Drawings, that took place in March 2021 (online) and was funded by Deutsche Forschungsgemeinschaft (DFG), the Austrian Science Fund (FWF) and the Swiss National Science Foundation (SNSF).

[†] Supported by DFG within the Research Training Group GRK 2434 *Facets of Complexity*.

[‡] Supported in part by ERC StG 757609.

Well-separation is a strong assumption on set-families, and it should not be a surprise that for many problems, it leads to stronger results and faster algorithms compared to the general case. One such example concerns *Ham-Sandwich cuts*. Given d point sets $P_1, \dots, P_d$ in $\mathbb{R}^d$, a Ham-Sandwich cut is a hyperplane that simultaneously bisects each point set. While Ham-Sandwich cuts exist for any family of d point sets [16], computing a Ham-Sandwich cut is PPA-complete when the dimension is not fixed [9], meaning that it is unlikely to allow an algorithm that runs in polynomial time in the dimension d. On the other hand, if $P_1, \dots, P_d$ are well-separated, not only do there exist bisecting hyperplanes, but the Ham-Sandwich theorem can be generalized to hyperplanes cutting off arbitrary given fractions from each point set [5,15]. Moreover, the problem of finding such hyperplanes lies in the complexity class UEOPL [8], a subclass of PPA which is believed to allow for significantly faster algorithms.

From an algorithmic perspective, the main focus of work has been on line transversals in dimensions 2 and 3, see, e.g., [1,4,14]. To the authors' knowledge, in higher dimensions only hyperplane transversals have been studied, where the best known algorithm for deciding whether a set of n polyhedra with m edges has a hyperplane transversal, runs in time $O(nm^{d-1})$ [3]. In particular, there is an exponential dependence in the dimension d. This curse of dimensionality appears in many geometric problems. For several problems, it has been shown that there is probably no hope to get rid of the exponential dependence in the dimension. As an example, we mention a result for Ham-sandwich cuts, due to Knauer, Tiwary and Werner [12]: Given d point sets $P_1, \dots, P_d$ in $\mathbb{R}^d$ and a point $p \in \mathbb{R}^d$, where d is part of the input, it is $W[1]$-hard (and thus NP-hard) to decide whether there is a Ham-sandwich cut passing through p.

Our Results. A family of k sets in $\mathbb{R}^d$ is well-separated, if and only if their convex hulls have no $(k-2)$-transversal. This fact seems to be well-known, but we could only find some references without proofs, and some proofs of only one direction, for similar definitions of well-separation [6,7]. Therefore, we present a short proof for sake of completeness in the full version. This immediately implies that testing well-separation is in coNP.

In [8], the authors ask what is the complexity of determining whether a family of point sets is well-separated, when d is not fixed. We present several hardness results for finding $(k-2)$-transversals in a family of k sets in $\mathbb{R}^d$. We consider two cases: a) the sets are finite point sets, and b) the sets are convex.

▶ **Theorem 1.1.** *Given a family of $k > d$ point sets in $\mathbb{R}^d$, each consisting of at most two points, it is strongly NP-hard to check whether there is a $(d-1)$-transversal, even in the special case $k = d+1$.*

Note that this problem is trivial if $k \leq d$, as the answer is always yes. Our result shows that the problem becomes NP-hard for the first value of k for which the problem is non-trivial. We use Theorem 1.1 to show the following:

▶ **Theorem 1.2.** *Given a set of $k > d$ line segments in $\mathbb{R}^d$, it is strongly NP-hard to check whether there is a $(d-1)$-transversal, even in the special case $k = d+1$.*

Theorem 1.2 implies that testing well-separation is coNP-complete even in the case of $d+1$ segments in $\mathbb{R}^d$, answering the question from [8].

As a positive result, we can show the existence of the following approximation algorithm. This can be seen as the special case where each point set consists of a single point.

▶ **Theorem 1.3.** *Given a set P of k points in $\mathbb{R}^d$, it is possible to compute in polynomial time in d and k a hyperplane that contains $\Omega(\frac{OPT \log k}{k \log \log k})$ points of P, where OPT denotes the maximum number of points in P that a hyperplane can contain.*

In Section 3, we study the problem through the lens of parametrized complexity. We show a significant difference depending on whether we consider convex sets or finite point sets.

▶ **Theorem 1.4.** *Checking whether a family of $k \leq d+1$ convex hulls of point sets in $\mathbb{R}^d$ has a $(k-2)$-transversal (or equivalently, whether the point sets are well-separated) is FPT with respect to d.*

▶ **Theorem 1.5.** *Given a set of $k > d$ point sets in $\mathbb{R}^d$, it is W[1]-hard with respect to d to check whether there is a $(d-1)$-transversal, even in the special case $k = d+1$.*

Observe that for finite point sets (and more generally for any sets that are not convex), having no $(k-2)$-transversal does not a priori imply well-separation.

2 Hyperplane Transversals in High Dimensions

Let $S_1, \ldots, S_k \subset \mathbb{R}^d$ be k sets in d dimensions, where d is not fixed. Note that we do not assume the sets to be convex. In particular, the sets can even be finite. We consider the decision problem HypTrans: Given sets $S_1, \ldots, S_k$, decide if there is a $(d-1)$-transversal for them. We consider the finite case and the case of line segments. We also consider the optimisation formulation of HypTrans, that we name MaxHyp: Given the sets $S_1, \ldots, S_k$, find a hyperplane that intersects as many of these sets as possible.

We begin with the case that all S_i are finite point sets. We first assume that every S_i contains a single point, for $i = 1, \ldots, k$. Note that in this situation, HypTrans can be solved greedily. We denote by P the point set that is the union of all S_i. Let us denote by OPT the maximum number of points in P that a hyperplane may contain.

▶ **Theorem 1.3.** *Given a set P of k points in $\mathbb{R}^d$, it is possible to compute in polynomial time in d and k a hyperplane that contains $\Omega(\frac{OPT \log k}{k \log\log k})$ points of P, where OPT denotes the maximum number of points in P that a hyperplane can contain.*

Proof. If $k \leq d$, we just output a hyperplane that contains all points of P. Otherwise, let $f(k) = \log k / \log\log k$. If $f(k) < d$, we pick d points from P, and we output a hyperplane through these points. If $f(k) \geq d$, we partition P into disjoint groups of size $f(k)$. In each group, we compute all hyperplanes that go through some d points from the group. Among all hyperplanes for all groups, we output the hyperplane that contains the most points in P. For each group, we have $O(f(k)^d) = O(f(k)^{f(k)}) = O(k)$ hyperplanes to consider. Thus, the algorithm runs in polynomial time in d and k.

We now analyze the approximation guarantee. If $f(k) < d$, then we output a hyperplane with at least $d > f(k) \geq f(k)\mathrm{OPT}/k$ points, since $\mathrm{OPT} \leq k$. If $f(k) \geq d$, we let h be an optimal hyperplane. If h contains at least d points in a single group, then we output an optimal solution. Otherwise, h contains less than d points in each group, so $\mathrm{OPT} \leq d(k/f(k))$. This means that $d \geq f(k)\mathrm{OPT}/k$, and the claim follows from the fact that our solution contains at least d points. ◀

We now restrict ourselves to the situation that every S_i contains at most two points, for $i = 1, \ldots, k$. We will prove that already in this case HypTrans is strongly NP-hard, by reducing from BinPacking. Our reduction will pass through two intermediate problems EqualBinPacking and FlatTrans. We start by defining all the involved problems.

In BinPacking, we are given as input a set of *items* $I = \{I_1, \ldots, I_n\}$ with weights $w(I_i) := w(i) \in \mathbb{Z}_+$, and a set $B = \{B_1, \ldots, B_k\}$ of *bins*, all with the same *capacity* $b \in \mathbb{Z}_+$.

The goal is to decide whether there is a partition of the items into the bins such that in each bin the total weight of the items does not exceed the capacity. In EqualBinPacking, we are given the same input, but now the goal is to decide whether there exists a partition of the items into the bins such that in each bin the total weight of the items equals exactly the capacity. Note that BinPacking can easily be reduced to EqualBinPacking by adding the appropriate number of elements of weight 1, so EqualBinPacking is strongly NP-hard as well.

Finally, in FlatTrans, we are given m sets $S_0, \dots, S_{m-1}$ in $\mathbb{R}^d$, where m and d are both part of the input, and the goal is to decide whether there is an $(m-2)$-transversal. In other words, the question is whether there exists an $(m-2)$-dimensional affine subspace h such that for each $i \in \{0, \dots, m-1\}$ we have that $S_i \cap h \neq \emptyset$. Note that HypTrans with $k = d+1$ is the same as FlatTrans with $m = d+1$.

▶ **Theorem 2.1.** *FlatTrans is strongly NP-hard even when $S_0 = \{\mathbf{0}\}$ and any other S_i consists of at most two points.*

Sketch of proof. We reduce from EqualBinPacking. Given an input I, B, w, b, where $|I| = n$ and $|B| = k$, to EqualBinPacking, we construct an instance of FlatTrans as follows: First, we set the dimension $d = k + n + kn$ and the number of sets $m = kn + 2$. For any $(i, j) \in [n] \times [k]$ define the vectors

$$v_{i,j}(x) := \begin{cases} w(i), & \text{if } x = j, \\ 1, & \text{if } x = k+i, \\ 1, & \text{if } x = k+n+(i-1)k+j, \\ 0, & \text{else,} \end{cases} \quad \text{and } u_{i,j}(x) := \begin{cases} 0, & \text{if } x = j, \\ 0, & \text{if } x = k+i, \\ 1, & \text{if } x = k+n+(i-1)k+j, \\ 0, & \text{else.} \end{cases}$$

Note that by $x \in \{1, \dots k+n+kn\}$ we describe the entries of the vector. For example the first entry of $v_{i,j}$ is described by $v_{i,j}(1)$. Further, define the vector $c(x)$ whose entries are $-b$ for $1 \leq x \leq k$ and -1 everywhere else. Now set $S_0 = \{\mathbf{0}\}$, $S_l = \{v_{i,j}, u_{i,j}\}$ for each $l = (i-1)k + j$ (note that this choice of l just gives that the order of the l's corresponds to the lexicographic order of the (i,j)'s) and $S_{kn+1} = \{c\}$. Note that all of this can be done in polynomial time.

In the full version, we show that there is a kn-transversal of the sets $S_0, \dots, S_{kn+1}$, if and only if there is a valid partition for the EqualBinPacking instance. ◀

Now, there is only one reduction remaining:

▶ **Theorem 2.2.** *Let $S_0 = \{\mathbf{0}\}$ and let $S_i \subset \mathbb{R}^d$ be finite for $i = 1, \dots, m-1$. Then we can construct in polynomial time sets $S'_0, S'_1, \dots, S'_{d+2} \subset \mathbb{R}^{d+2}$ which can be transversed by a hyperplane if and only if $S_0, S_1, \dots, S_{m-1} \subset \mathbb{R}^d$ have an $(m-2)$-transversal.*

Sketch of proof. We only show the construction of the sets here. For the complete proof, we refer to the full version. First, for each point p in some set S_i we define the point $p' = (p, 0, 0)$ and place it in the set S'_i. For $m \leq i \leq d+2$, define S'_i as the set consisting only of the point $s'_i = (0, \dots, 0, 1, i)$. Additionally, let $S'_0 := \{\mathbf{0}\}$. ◀

Theorem 1.1 now follows from combining Theorems 2.1 and 2.2.

Further, we can now show that deciding whether there is a hyperplane transversal for d line segments and the origin in $\mathbb{R}^d$, where d is not fixed, is NP-hard. We will reduce this to the restricted version of HypTrans where the sets S_i contain at most two points. This is

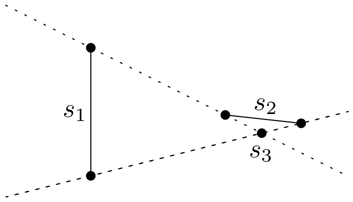


Figure 1 Every hyperplane transversal through s_1, s_2, s_3 must choose an endpoint of s_1 (and of s_2).

done with the help of a gadget that enforces that every hyperplane transversal must use one of the two endpoints of a given line segment. The gadget is shown in Figure 1.

Given a collection of sets of size at most two, for each set we take the line segment formed by its points as s_1, the origin as point s_3, and we construct the corresponding new segment s_2 using the gadget presented in Figure 1. This gives a family S of $2k$ line segments that all lie in a k-dimensional space. In order to prove Theorem 1.2, we need to lift our construction to $\mathbb{R}^{2k}$. This lifting is described in the full version.

3 From the viewpoint of parametrized complexity

Recall that our original motivation comes from determining whether d point sets in $\mathbb{R}^d$ are well-separated. Let us consider those d sets, and let us denote by n the total number of extreme vertices on their respective convex hulls. We say that n is the *convex hull complexity* of the set family. We assume that we are given the extreme points of the convex hull of every set and hence have a finite number of points for every set.

▶ **Theorem 1.4.** *Checking whether a family of $k \leq d+1$ convex hulls of point sets in $\mathbb{R}^d$ has a $(k-2)$-transversal (or equivalently, whether the point sets are well-separated) is FPT with respect to d.*

Sketch of proof. For the $O(2^d)$ choices of index sets $I \subset [k]$, we check with an LP whether the convex hulls of S_I and $S_{[k]\setminus I}$ intersect. ◀

On the other hand, using a framework similar to the one introduced by Marx [13], we show in the full version that

▶ **Theorem 3.1.** *FLATTRANS is $W[1]$-hard with respect to the dimension.*

Combining this with Theorem 2.2, we deduce Theorem 1.5.

References

1 Pankaj K. Agarwal. On stabbling lines for convex polyhedra in 3d. *Computational Geometry*, 4(4):177–189, 1994.

2 Nina Amenta, Jesús A De Loera, and Pablo Soberón. Helly's theorem: new variations and applications. *arXiv preprint arXiv:1508.07606*, 2015.

3 David Avis and Mike Doskas. Algorithms for high dimensional stabbing problems. *Discrete applied mathematics*, 27(1-2):39–48, 1990.

4 David Avis and Rephael Wenger. Algorithms for line transversals in space. In *Proceedings of the third annual symposium on Computational geometry*, pages 300–307, 1987.

5 Imre Bárány, Alfredo Hubard, and Jesús Jerónimo. Slicing convex sets and measures by a hyperplane. *Discrete & Computational Geometry*, 39(1-3):67–75, 2008.

6 Ted Bisztriczky. On separated families of convex bodies. *Archiv der Mathematik*, 54(2):193–199, 1990.

7 Federico Castillo, Joseph Doolittle, and Jose Alejandro Samper. Common tangents to polytopes. *arXiv preprint arXiv:2108.13569*, 2021.

8 Man-Kwun Chiu, Aruni Choudhary, and Wolfgang Mulzer. Computational complexity of the α-ham-sandwich problem. In *Proc. 47th Internat. Colloq. Automata Lang. Program. (ICALP)*, pages 31:1–31:18, 2020. URL: https://doi.org/10.4230/LIPIcs.ICALP.2020.31.

9 Aris Filos-Ratsikas and Paul W. Goldberg. The complexity of splitting necklaces and bisecting ham sandwiches. In *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing*, pages 638–649, 2019.

10 Jacob E. Goodman, Richard Pollack, and Rephael Wenger. Geometric transversal theory. In *New trends in discrete and computational geometry*, pages 163–198. Springer, 1993.

11 Andreas Holmsen and Rephael Wenger. 4 Helly-type theorems and geometric transversals. *Handbook of Discrete and Computational Geometry*, 2017.

12 Christian Knauer, Hans Raj Tiwary, and Daniel Werner. On the computational complexity of ham-sandwich cuts, helly sets, and related problems. In *Symposium on Theoretical Aspects of Computer Science (STACS2011)*, volume 9, pages 649–660, 2011.

13 Dániel Marx. Parameterized complexity of independence and domination on geometric graphs. In *International Workshop on Parameterized and Exact Computation*, pages 154–165. Springer, 2006.

14 Marco Pellegrini and Peter W. Shor. Finding stabbing lines in 3-space. *Discrete & Computational Geometry*, 8(2):191–208, 1992.

15 William Steiger and Jihui Zhao. Generalized ham-sandwich cuts. *Discrete & Computational Geometry*, 44(3):535–545, 2010.

16 Arthur H. Stone and John W. Tukey. Generalized "sandwich" theorems. *Duke Math. J.*, 9(2):356–359, 06 1942.

Planarizing Graphs and their Drawings by Vertex Splitting*

Soeren Nickel[1], **Martin Nöllenburg**[1], **Manuel Sorge**[1], **Anaïs Villedieu**[1], **Hsiang-Yun Wu**[2], **and Jules Wulms**[1]

1 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
{soeren.nickel|noellenburg|manuel.sorge|avilledieu|jwulms}@ac.tuwien.ac.at

2 **St. Pölten University of Applied Sciences, St. Pölten, Austria and Research Unit of Computer Graphics, TU Wien, Vienna, Austria**
hsiang.yun.wu@acm.org

Abstract

The splitting number of a graph $G = (V, E)$ is the minimum number of vertex splits required to turn G into a planar graph, where a vertex split removes a vertex $v \in V$, introduces two new vertices v_1, v_2, and distributes the edges formerly incident to v among its two split copies v_1, v_2. The splitting number problem is known to be NP-complete. In this paper we shift focus to the splitting number of graph drawings in $\mathbb{R}^2$, where the new vertices resulting from vertex splits must be re-embedded into the existing drawing of the remaining graph. We show the NP-completeness of the splitting number problem for graph drawings, even for its two subproblems of (1) selecting a minimum subset of vertices to split and (2) for re-embedding a minimum number of copies of a given set of vertices, which does not need to be a solution to (1). We present an FPT algorithm for the latter subproblem, parameterized by the number of vertex splits, which reduces the instance to bounded outerplanarity and then uses dynamic programming on its sphere-cut decomposition.

Related Version arXiv:2202.12293

1 Introduction

Visualizing dense graphs is a challenging task due to the potentially large number of edge crossings, which make tracing of individual edges harder and create clutter that negatively impacts readability [31]. Several approaches have been proposed to mitigate this issue [20], many aim to achieve readability properties similar to those of crossing-free drawings of planar graphs [30, 32, 34]. One such technique is to apply a sequence of *vertex splitting* operations. This approach has been studied from a theoretical perspective [8, 11, 23, 26], and is used in practice, e.g., by biologists and social scientists [18, 19, 29, 35, 36]. For a given graph $G = (V, E)$ and a vertex $v \in V$, a *vertex split* of v replaces v by two non-adjacent copies v_1, v_2 and distributes the edges formerly incident to v to v_1 and v_2. The minimum number of splits needed to obtain planarity is known as the *splitting number* of a graph and computing it is NP-hard [13]. The splitting numbers of complete graphs, complete bipartite graphs and the 4-cube [12, 16, 17, 22] are known. Similarly, the planar split thickness of a graph G is the minimum k such that G can be turned into a planar graph by applying a k-split (which creates k copies $v_1, \ldots, v_k$) to each vertex v of G. Deciding whether a graph has split thickness k is NP-complete [11].

* Manuel Sorge acknowledges funding by the Alexander von Humboldt Foundation, Anaïs Villedieu acknowledges funding by the Austrian Science Fund (FWF) under grant P31119 and Jules Wulms acknowledges funding partially by the Austrian Science Fund (FWF) under grant P31119 and partially by the Vienna Science and Technology Fund (WWTF) under grant ICT19-035.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.

Contributions. Our focus in this paper is on vertex splitting for topological graph drawings in the plane $\mathbb{R}^2$, where the subgraph induced by the non-split vertices retains its drawing. Similarities can be found with simultaneous embedding problems [5, 14, 15], and planar drawing extension problems [1, 2, 6, 7, 9, 10]. The underlying algorithmic problem for vertex splitting in drawings of graphs is two-fold: firstly, a suitable (minimum) subset of vertices to be split must be selected, and secondly the newly created copies of these vertices must be re-embedded in a crossing-free way together with a partition of the original edges of each split vertex into a subset for each copy. We show that both problems are NP-complete, and present an FPT algorithm for the re-embedding subproblem of the splitting number problem for graph drawings parameterized by the number of splits. We note that the smallest set of vertices as computed for the first subproblem is not necessarily the correct set of vertices to split when solving the complete problem.

Preliminaries. Let $G = (V, E)$ be a graph. We write $G[V']$ to denote the subgraph of G induced by $V' \subseteq V$ and $N_G(v)$ to denote the neighborhood of a vertex v in G.

Let Γ be a *topological drawing* (for simplicity, from now on called a drawing) of G, which maps each vertex to a point in $\mathbb{R}^2$ and each edge to a simple curve (a Jordan arc) connecting the points corresponding to the incident vertices of that edge. We still refer to the points and curves as vertices and edges, respectively, in such a drawing. We assume Γ is a simple drawing, meaning no two edges intersect more than once, no three edges intersect in one point (except common endpoints), and adjacent edges do not cross. A *split* operation of a vertex $v \in V$ into two *copies* $\dot{v}^{(1)}, \dot{v}^{(2)}$ results in a drawing of the graph $G' = (V', E')$ where $V' = V \setminus \{v\} \cup \{\dot{v}^{(1)}, \dot{v}^{(2)}\}$ and E' is obtained from E by distributing the edges incident to v among $\dot{v}^{(1)}, \dot{v}^{(2)}$ such that $N_G(v) = N_{G'}(\dot{v}^{(1)}) \cup N_{G'}(\dot{v}^{(2)})$. It assigns new coordinates $\Gamma(\dot{v}^{(i)})$ to $\dot{v}^{(1)}, \dot{v}^{(2)}$ as well as new curves $\Gamma(e)$ to all edges e incident to any of the split vertices. If a copy $\dot{v}$ of a vertex v is split again, then any copy of $\dot{v}$ is also called a copy of the original vertex v and we use the notation $\dot{v}^{(i)}$ for $i = 1, 2, \ldots$ to denote the different copies of v.

▶ **Problem 1** (Embedded Splitting Number). *Given a graph $G = (V, E)$, a drawing Γ of G and an integer k, can G be transformed into a graph G' by applying at most k splits to G such that G' has a planar drawing that coincides with Γ when restricted to $G'[V(G) \cap V(G')]$?*

Problem 1 includes two interesting subproblems, namely the *candidate selection problem* and the *re-embedding problem.* The candidate selection problem is related to the NP-complete problem of deleting at most k vertices from a non-embedded graph to make it planar [25, 28]. However, here we deal with a given drawing of a graph (with crossings).

▶ **Problem 2** (Candidate Selection). *Given a graph $G = (V, E)$, a drawing Γ of G and an integer k, can we find a candidate set $S_{\text{cdt}} \subset V$ of at most k vertices such that the drawing Γ restricted to $G[V \setminus S_{\text{cdt}}]$ is planar?*

The vertices split in a solution of Problem 1 necessarily form such a candidate set, however, a minimum cardinality candidate set might not be the set that requires the least amount of splits to solve Problem 1, as vertices can be split multiple times and we might have to additionally split vertices whose incident edges are not involved in crossings.

Once a candidate set has been obtained we want to solve the second subproblem:

▶ **Problem 3** (Split Set Re-Embedding). *Given a graph $G = (V, E)$, a candidate set $S_{\text{cdt}} \subset V$, a drawing Γ of the subgraph $G[V \setminus S_{\text{cdt}}]$, and an integer $k \geq |S_{\text{cdt}}|$, can we perform at most k splits, splitting only vertices in S_{cdt} and splitting each vertex in S_{cdt} at least once, such that the resulting graph G' has a planar drawing that coincides with Γ when restricted to $G[V \setminus S_{\text{cdt}}]$?*

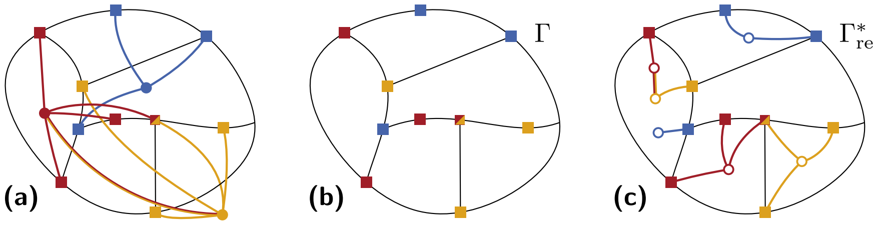


Figure 1 **(a)** An example graph G, **(b)** a planar drawing Γ of G where S_{cdt} has been removed, and **(c)** a solution drawing $\Gamma^\star$. Pistils are squares, copies are circles and vertices in S_{cdt} are disks.

While we find that Split Set Re-Embedding is FPT, the parameterized complexity of Candidate Selection remains open.

2 Embedded Splitting Number Subproblems are NP-Complete

The reduction showing Splitting Number to be NP-complete [13] does not seem to extend to Embedded Splitting Number. Here we show that Candidate Selection is NP-complete using a reduction from planar 3-SAT inspired by Hummel et al. [21].[1]

▶ **Theorem 2.1.** *Candidate Selection is* NP*-complete.*

We then show that Split Set Re-Embedding also is NP-complete. We reduce from Face Cover [4], where we are given a planar graph and a vertex subset S and we ask for the smallest set of faces F such that each vertex in S is incident to a face in F. We construct an instance of Split Set Re-Embedding with the same graph and an extra vertex v, where the candidate set is the vertex v and its neighborhood is $N(v) = S$. A re-embedding of k copies of v uses faces that induce a face cover and vice-versa, a face cover of size k gives the faces in which we can re-embed k copies of v.

▶ **Theorem 2.2.** *Split Set Re-Embedding is* NP*-complete.*

3 Split Set Re-Embedding is Fixed-Parameter Tractable

In this section we propose an FPT algorithm for Problem 3 (Split Set Re-Embedding) and prove the following theorem.

▶ **Theorem 3.1.** *Split Set Re-Embedding can be solved in $2^{O(k^2)} \cdot n^{O(1)}$ time, where k is the number of allowed splits and n is the number of vertices in the input graph G.*

Algorithm outline. We aim to re-embed copies of our candidate vertices with the following setup (Fig. 1). First, from the given set S_{cdt} of candidate vertices (disks in Fig. 1a) we choose how many copies of each vertex we will make. We initialize a set $S_\curlywedge$ with one copy of every candidate vertex, then loop over every possibility of splitting vertices in S_{cdt} $k - |S_{\text{cdt}}|$ times. Note that $|S_{\text{cdt}}| \leq k$, and thus every computed set $S_\curlywedge$ is obtained from k split operations. This creates $2^{O(k^2)}$ different sets. For each such computed set $S_\curlywedge$ of copies we determine the connections among them. Next, we transform our input to be able to compute

[1] Alternatively, one can reduce from Independent Set on segment intersection graphs [24] as suggested by a reviewer.

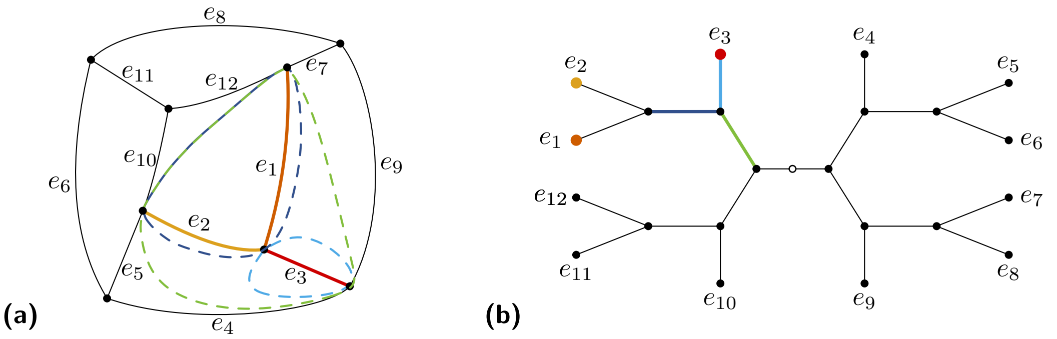


Figure 2 (a) A graph and **(b)** its sphere-cut decomposition. Each labeled leaf corresponds to the same labeled edge of the graph. The middle set of each colored edge in the tree corresponds to the vertices of the corresponding colored dashed noose in the graph.

a sphere-cut decomposition of the new drawing, as explained in Section 3.1. We then use dynamic programming on the tree defined by this decomposition as sketched in Section 3.2. If this algorithm finds that our instance is a yes instance then a solution exists (see Fig. 1c).

We introduce the following terminology. Any vertex v that has a neighbor in S_{cdt} is called a *pistil*. Each face that is incident to a pistil is called a *petal*. Let p be a pistil in the input graph G with neighbors $N(p)$. Let $\dot{v}$ be a copy of some $v \in S_{\text{cdt}}$, where $v \in N(p)$. Given a drawing $\tilde{\Gamma}$ of a subgraph of G, we say $\dot{v}$ *covers* p if $\dot{v}$ is adjacent to p in $\tilde{\Gamma}$.

3.1 Finding a Sphere-Cut Decomposition

Given an instance of Split Set Re-Embedding (SSRE), we transform the induced graph $G[V \setminus S_{\text{cdt}}]$ in the following manner: any vertex $v \in V \setminus S_{\text{cdt}}$ that is not incident to a petal is removed. Then, any bridge in that new drawing is transformed into a multi-edge to obtain G' and its drawing Γ'. We can show that the instance obtained is a yes-instance if and only if the original instance is a yes-instance and the graph G' is $6k$-outerplanar. A graph is ℓ-outerplanar if after ℓ times removing all vertices on the outer face the graph becomes empty. This can be exploited algorithmically in the following..

A *branch decomposition* of a (multi-)graph G is a pair (T, λ) where T is an unrooted binary tree, and λ is a bijection between the leaves of T and $E(G)$. Every edge $e \in E(T)$ defines a bipartition of $E(G)$ into A_e and B_e corresponding to the leaves in the two connected components of $T - e$. We define the *middle set* $\text{mid}(e)$ of an edge $e \in E(T)$ to be the set of vertices incident to an edge in both sets A_e and B_e. The *width* of a branch decomposition is the size of the biggest middle set in that decomposition. The *branchwidth* of G is the minimum width over all branch decompositions of G.

A *sphere-cut decomposition* of a planar (multi-)graph G with a planar embedding Γ on a sphere Σ is a branch decomposition (T, λ) of G such that for each edge $e \in E(T)$ there is a *noose* $\eta(e)$: a closed curve on Σ such that its intersection with Γ is exactly the vertex set $\text{mid}(e)$ (i.e., the curve does not intersect any edge of Γ) and such that the curve visits each face of Γ at most once (see Fig. 2). The removal of e from $E(T)$ partitions T into two subtrees T_1, T_2 whose leaves correspond, respectively, to the noose's partition of Γ into two embedded subgraphs G_1, G_2. Sphere-cut decompositions were introduced by Seymour and Thomas [33], more details can also be found in [27, Section 4.6]. The *length* of the noose $\eta(e)$ for an edge $e \in E(T)$ is the number of vertices on the noose (or the size of $\text{mid}(e)$) and it is at most the branchwidth of the decomposition. We defined drawings in the plane, whereas we need drawings on the sphere for sphere-cut decompositions. However, if we treat the outer face of a planar drawing just as any other face, then spherical and planar drawings are homeomorphic.

An ℓ-outerplanar graph has branchwidth at most 2ℓ [3] and a connected bridgeless planar graph of branchwidth at most b has a sphere-cut decomposition of width at most b that can be computed in $O(n^3)$ time (see [27, Section 4.6]). Since G' is $6k$-outerplanar and bridgeless, we obtain a sphere-cut decomposition of G' of branchwidth $12k$.

3.2 Dynamic Programming on a Sphere-Cut Decomposition Tree

Initialization. The dynamic program works bottom-up in the sphere-cut decomposition tree T from the leaves to an arbitrarily chosen root, considering iteratively larger subgraphs of G'. The algorithm determines how partial solutions look like on the interface between subgraphs and the rest of G'. We first transform T by defining a root vertex and move the information of the middle set from each edge to the child vertex (according to the new parent-child relations). For each vertex t of T, its noose $\eta(t)$ splits the graph into two subgraphs. We define the subgraph whose edges correspond to the leaves of the subtree of T rooted at t to be the graph *inside* the noose. A partial solution on a subgraph G'_t inside noose $\eta(t)$ is a planar drawing of that subgraph, with a subset $S'_\curlywedge \subseteq S_\curlywedge$ of copies embedded in it together with edges to the copies' neighborhood in G'_t such that all pistils not on the noose are covered. To describe those partial solutions we build tuples called *signatures* for each possible solution for each noose. A signature holds the following information (see Fig. 3): (i) the set of copies S_{in} used in faces entirely inside $\eta(t)$ to cover pistils, (ii) the set N_η of sets X_v of neighbors of each vertex $v \in \eta(t)$ that do not cover v, (iii) graphs that represent embeddings of copies for all the faces traversed by the noose, and (iv) for each such graph a pair of pointers $p_{\text{s}}, p_{\text{e}}$ that describe which vertices of that embedding are used to cover pistils in G'_t. We find that the number of signatures is upper bounded by $2^{O(k^2)}$. The embeddings from (iii) are described by a set C_{out} of graphs called *nesting graphs* (see Fig. 4), which are planar graphs C_f, where a set of copies are embedded inside a cycle and each vertex of the cycle has exactly one neighbor that is a copy. The intuition behind a nesting graph is that, when embedded inside a face f, one can simultaneously traverse the cycle of C_f and the face f in the same direction, and draw an edge between each cycle vertex and a corresponding pistil. Then, after removing the cycle edges, and contracting the cycle vertices to pistils edges, we obtain a planar embedding for f, where some pistils strictly inside $\eta(t)$ are covered by the copies in C_f.

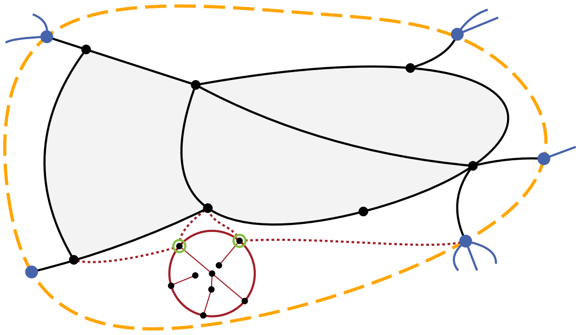


Figure 3 The information stored in a signature of the partial solution inside the orange noose: (grey) copies used in these faces are stored in S_{in}, (blue) noose vertices, for which missing neighbors (outgoing edges outside the noose) are stored in N_η, (red) an example of a nesting graph for a face traversed by the noose, with four (dotted) edges connecting to the cycle, (green) p_{s} and p_{e} pointers.

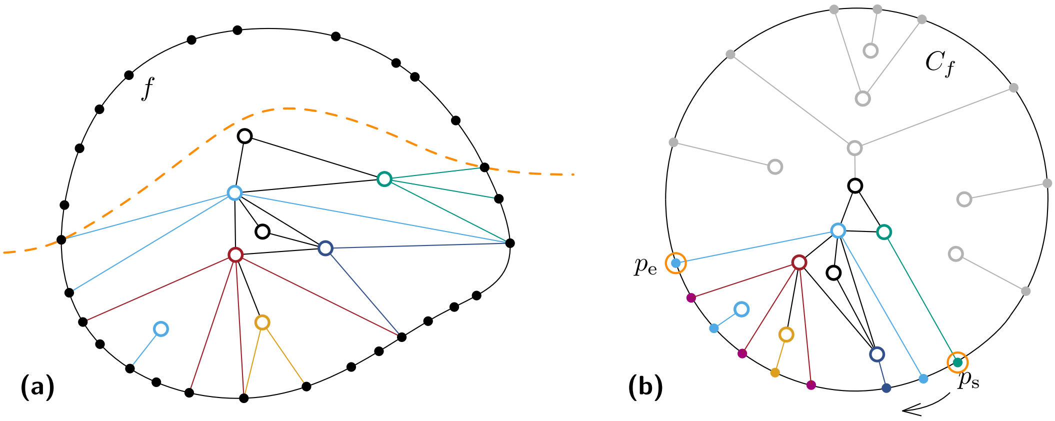


Figure 4 (a) A face f and the copies inside the orange noose. **(b)** The corresponding nesting graph C_f with its p_{s} and p_{e} vertices in orange. The two light blue vertices represent different copies of the same removed vertex.

Traversing the Sphere-Cut Decomposition Tree. To find a solution we perform bottom-up dynamic programming on T. For each node $t \in V(T)$ we compute a table of all signatures with a corresponding partial solution. For a leaf $t \in V(T)$, graph G_t is an edge (u_1, u_2), and we can go over all possible signatures and check whether we can cover all neighbors of u_1 and u_2 not in $N_\eta = \{X_{u_1}, X_{u_2}\}$, using for each incident face f the subgraph of $C_f \in C_{\mathrm{out}}$ that lies between $p_{\mathrm{s}}, p_{\mathrm{e}}$. For internal nodes we merge some pairs of child signatures corresponding to two nooses $\eta(t_1)$ and $\eta(t_2)$. We merge when (1) faces not shared between the nooses do not have copies in common, (2) shared faces use identical nesting graphs and (3) use disjoint subgraphs of those nesting graphs to cover pistils, and (4) noose vertices in $\mathrm{mid}(t_1)$ and $\mathrm{mid}(t_2)$ do not have remaining missing neighbors. Thus we can find valid signatures for all nodes of T and notably for its root. If we find a valid signature for the root, we also have a partial solution. In Γ' all pistils are covered and it is planar, as the nesting graphs are planar and they represent a combinatorial embedding that allowed to cover pistils. We verify that the remaining pistils in $S_\curlywedge \setminus S'_\curlywedge$ form a planar graph which allows us to embed them in a face of Γ' to obtain a solution $\Gamma^\star$.

References

1 Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Vít Jelínek, Jan Kratochvíl, Maurizio Patrignani, and Ignaz Rutter. Testing planarity of partially embedded graphs. *ACM Transactions on Algorithms*, 11(4):32:1–32:42, 2015. doi:10.1145/2629341.

2 Alan Arroyo, Fabian Klute, Irene Parada, Raimund Seidel, Birgit Vogtenhuber, and Tilo Wiedera. Inserting one edge into a simple drawing is hard. In Isolde Adler and Haiko Müller, editors, *Proc. 46th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, volume 12301 of *LNCS*, pages 325–338. Springer, 2020. doi:10.1007/978-3-030-60440-0_26.

3 Therese Biedl. On triangulating k-outerplanar graphs. *Discrete Applied Mathematics*, 181:275–279, 2015. doi:10.1016/j.dam.2014.10.017.

4 Daniel Bienstock and Clyde L. Monma. On the complexity of covering vertices by faces in a planar graph. *SIAM Journal on Computing*, 17(1):53–76, 1988. doi:10.1137/0217004.

5 Peter Braß, Eowyn Cenek, Christian A. Duncan, Alon Efrat, Cesim Erten, Dan Ismailescu, Stephen G. Kobourov, Anna Lubiw, and Joseph S. B. Mitchell. On simultaneous planar

graph embeddings. *Computational Geometry: Theory and Applications*, 36(2):117–130, 2007. doi:10.1016/j.comgeo.2006.05.006.

6 Markus Chimani, Carsten Gutwenger, Petra Mutzel, and Christian Wolf. Inserting a vertex into a planar graph. In Claire Mathieu, editor, *Proc. 20th Symposium on Discrete Algorithms (SODA)*, pages 375–383. SIAM, 2009. doi:10.1137/1.9781611973068.42.

7 Markus Chimani and Petr Hliněný. Inserting multiple edges into a planar graph. In Sándor P. Fekete and Anna Lubiw, editors, *Proc. 32nd International Symposium on Computational Geometry (SoCG)*, volume 51 of *LIPIcs*, pages 30:1–30:15, 2016. doi:10.4230/LIPIcs.SoCG.2016.30.

8 Peter Eades and Candido F. X. de Mendonça N. Vertex splitting and tension-free layout. In Franz-Josef Brandenburg, editor, *Proc. 3rd International Symposium on Graph Drawing (GD)*, volume 1027 of *LNCS*, pages 202–211. Springer, 1995. doi:10.1007/BFb0021804.

9 Eduard Eiben, Robert Ganian, Thekla Hamm, Fabian Klute, and Martin Nöllenburg. Extending nearly complete 1-planar drawings in polynomial time. In Javier Esparza and Daniel Král', editors, *Proc. 45th International Symposium on Mathematical Foundations of Computer Science (MFCS)*, volume 170 of *LIPIcs*, pages 31:1–31:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.MFCS.2020.31.

10 Eduard Eiben, Robert Ganian, Thekla Hamm, Fabian Klute, and Martin Nöllenburg. Extending partial 1-planar drawings. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *Proc. 47th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 168 of *LIPIcs*, pages 43:1–43:19. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ICALP.2020.43.

11 David Eppstein, Philipp Kindermann, Stephen G. Kobourov, Giuseppe Liotta, Anna Lubiw, Aude Maignan, Debajyoti Mondal, Hamideh Vosoughpour, Sue Whitesides, and Stephen K. Wismath. On the planar split thickness of graphs. *Algorithmica*, 80(3):977–994, 2018. doi:10.1007/s00453-017-0328-y.

12 Luérbio Faria, Celina M. H. de Figueiredo, and Candido F. X. de Mendonça N. The splitting number of the 4-cube. In Claudio L. Lucchesi and Arnaldo V. Moura, editors, *Proc. 3rd Latin American Symposium on Theoretical Informatics (LATIN)*, volume 1380 of *LNCS*, pages 141–150. Springer, 1998. doi:10.1007/BFb0054317.

13 Luérbio Faria, Celina M. H. de Figueiredo, and Candido F. X. de Mendonça N. Splitting number is NP-complete. *Discrete Applied Mathematics*, 108(1):65–83, 2001. doi:10.1016/S0166-218X(00)00220-1.

14 Fabrizio Frati, Michael Kaufmann, and Stephen G. Kobourov. Constrained simultaneous and near-simultaneous embeddings. *Journal of Graph Algorithms and Applications*, 13(3):447–465, 2009. doi:10.7155/jgaa.00194.

15 Emilio Di Giacomo, Walter Didimo, Marc J. van Kreveld, Giuseppe Liotta, and Bettina Speckmann. Matched drawings of planar graphs. *Journal of Graph Algorithms and Applications*, 13(3):423–445, 2009. doi:10.7155/jgaa.00193.

16 Nora Hartsfield. The toroidal splitting number of the complete graph $\mathrm{k_n}$. *Discrete Mathematics*, 62(1):35–47, 1986. doi:10.1016/0012-365X(86)90039-7.

17 Nora Hartsfield, Brad Jackson, and Gerhard Ringel. The splitting number of the complete graph. *Graphs and Combinatorics*, 1(1):311–329, 1985. doi:10.1007/BF02582960.

18 Nathalie Henry, Anastasia Bezerianos, and Jean-Daniel Fekete. Improving the readability of clustered social networks using node duplication. *IEEE Transactions on Visualization and Computer Graphics*, 14(6):1317–1324, 2008. doi:10.1109/TVCG.2008.141.

19 Nathalie Henry Riche and Tim Dwyer. Untangling euler diagrams. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1090–1099, 2010. doi:10.1109/TVCG.2010.210.

20 Ivan Herman, Guy Melançon, and M. Scott Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000. doi:10.1109/2945.841119.

21 Matthias Hummel, Fabian Klute, Soeren Nickel, and Martin Nöllenburg. Maximizing ink in partial edge drawings of k-plane graphs. In Daniel Archambault and Csaba D. Tóth, editors, *Proc. 27th International Symposium on Graph Drawing and Network Visualization (GD)*, volume 11904 of *LNCS*, pages 323–336. Springer, 2019. doi:10.1007/978-3-030-35802-0_25.

22 Brad Jackson and Gerhard Ringel. The splitting number of complete bipartite graphs. *Archiv der Mathematik*, 42(2):178–184, 1984. doi:10.1007/BF01772941.

23 Kolja Knauer and Torsten Ueckerdt. Three ways to cover a graph. *Discrete Mathematics*, 339(2):745–758, 2016. doi:10.1016/j.disc.2015.10.023.

24 Jan Kratochvíl and Jaroslav Nešetřil. Independent set and clique problems in intersection-defined classes of graphs. *Commentationes Mathematicae Universitatis Carolinae*, 31(1):85–93, 1990.

25 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980. doi:10.1016/0022-0000(80)90060-4.

26 Annegret Liebers. Planarizing graphs - A survey and annotated bibliography. *Journal of Graph Algorithms and Applications*, 5(1):1–74, 2001. doi:10.7155/jgaa.00032.

27 Dániel Marx and Michal Pilipczuk. Optimal parameterized algorithms for planar facility location problems using voronoi diagrams. *CoRR*, abs/1504.05476, 2015. arXiv:1504.05476.

28 Dániel Marx and Ildikó Schlotter. Obtaining a planar graph by vertex deletion. *Algorithmica*, 62(3-4):807–822, 2012. doi:10.1007/s00453-010-9484-z.

29 Sune S. Nielsen, Marek Ostaszewski, Fintan McGee, David Hoksza, and Simone Zorzan. Machine learning to support the presentation of complex pathway graphs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 18(3):1130–1141, 2019. doi:10.1109/TCBB.2019.2938501.

30 Takao Nishizeki and Md. Saidur Rahman. *Planar Graph Drawing*, volume 12 of *Lecture Notes Series on Computing*. World Scientific, 2004. doi:10.1142/5648.

31 Helen C. Purchase. Which aesthetic has the greatest effect on human understanding? In Giuseppe Di Battista, editor, *Proc. 5th International Symposium on Graph Drawing (GD)*, volume 1353 of *LNCS*, pages 248–261. Springer, 1997. doi:10.1007/3-540-63938-1_67.

32 Helen C. Purchase, Christopher Pilcher, and Beryl Plimmer. Graph drawing aesthetics—created by users, not algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 18(1):81–92, 2012. doi:10.1109/TVCG.2010.269.

33 Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994. doi:10.1007/BF01215352.

34 Luca Vismara. Planar straight-line drawing algorithms. In Roberto Tamassia, editor, *Handbook on Graph Drawing and Visualization*, pages 193–222. Chapman and Hall/CRC, 2013. doi:10.1007/3-540-62495-3_42.

35 Hsiang-Yun Wu, Martin Nöllenburg, Filipa L. Sousa, and Ivan Viola. Metabopolis: Scalable network layout for biological pathway diagrams in urban map style. *BMC Bioinformatics*, 20(1):1–20, 2019. doi:10.1186/s12859-019-2779-4.

36 Hsiang-Yun Wu, Martin Nöllenburg, and Ivan Viola. Multi-level area balancing of clustered graphs. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–15, 2020. doi:10.1109/TVCG.2020.3038154.

A new discrete theory of pseudoconvexity*

Balázs Keszegh[1]

1 **Alfréd Rényi Institute of Mathematics and ELTE Eötvös Loránd University, MTA-ELTE Lendület Combinatorial Geometry Research Group, Budapest, Hungary.**
keszegh@renyi.hu

Abstract

Recently geometric hypergraphs that can be defined by intersections of pseudohalfplanes with a finite point set were defined in a purely combinatorial way. This led to extensions of earlier results about points and halfplanes to pseudohalfplanes, including polychromatic colorings and discrete Helly-type theorems about pseudohalfplanes.

Here we continue this line of research and introduce the notion of convex sets of such pseudohalfplane hypergraphs. In this context we prove several results corresponding to classical results about convexity, namely Helly Theorem, Carathéodory's Theorem, Kirchberger's Theorem, Separation Theorem, Radon's Theorem and the Cup-Cap Theorem. These results imply the respective results about pseudoconvex sets in the plane defined using pseudohalfplanes.

It turns out that most of our results can be also proved using oriented matroids and topological affine planes (TAPs) but our approach is different from both of them. Compared to oriented matroids, our theory is based on a linear ordering of the vertex set which makes our definitions and proofs quite different and perhaps more elementary. Compared to TAPs, which are continuous objects, our proofs are purely combinatorial and again quite different in flavor. Altogether, we believe that our new approach can further our understanding of these fundamental convexity results.

Related Version arXiv:2202.07697

1 Introduction

Given a (finite) point set P and a family of regions $\mathcal{R}$ (e.g., the family of all halfplanes) in the plane (or in higher dimensions), let $\mathcal{H}$ be the hypergraph with vertex set P and for each region of $\mathcal{R}$ having a hyperedge containing exactly the same points of P as this region. There are many interesting problems that can be phrased as a problem about hypergraphs defined this way, which are usually referred to as *geometric hypergraphs*. This topic has a wide literature, researchers considered problems where $\mathcal{R}$ is a family of halfplanes, axis-parallel rectangles, translates or homothets of disks, squares, convex polygons, pseudo-disks and so on. There are many results and open problems about the maximum number of hyperedges of such a hypergraph, coloring questions and other properties. For a survey of some of the most resent results see the introduction of [2] and of [4], for an up-to-date database of such results with references see the webpage [1].[1]

One of the most basic families is the family of halfplanes, about which already many problems are non-trivial. Among others one such problem was considered in [9] where they

* Research supported by the Lendület program of the Hungarian Academy of Sciences (MTA), under the grant LP2017-19/2017, by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences, by the National Research, Development and Innovation Office – NKFIH under the grant K 132696 and FK 132060 and by the ÚNKP-20-5 New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund.

[1] As this paper is in many ways a continuation of [6] by the same author, the first three paragraphs of the introduction rely heavily on its introduction.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

prove that the vertices of every hypergraph defined by halfplanes on a set of points can be k-colored such that every hyperedge of size at least $2k + 1$ contains all colors. In [7] they considered generalizing this result by replacing halfplanes with the family of translates of an unbounded convex region (e.g., an upwards parabola). It turned out that this is true even when halfplanes are replaced by pseudohalfplanes. The main tool of proving this was an equivalent combinatorial definition of so-called *pseudohalfplane hypergraphs* that can be defined on points with pseudohalfplanes.[2] This formulation had the promise that many other statements about halfplane hypergraphs can be generalized to pseudohalfplane hypergraphs in the future. While this combinatorial formulation has the disadvantage of being less visual and thus somehow less intuitive than the geometric setting, it has many advantages, among others covering a much wider range of hypergraphs, also, being purely combinatorial, it might have algorithmic applications as well. One recent application is a similar polychromatic coloring result about disks all containing the origin [4] where after observing that in every quadrant of the plane the disks form a family of pseudohalfplanes they can apply the results from [7].

In [7] the equivalent of the convex hull vertices in the plane (more precisely, the points on the boundary of the convex hull) was defined for pseudohalfplane hypergraphs and called *unskippable vertices* and this made it possible to generalize the proof idea of [9] from halfplanes to pseudohalfplane hypergraphs. To make it more intuitive, we call unskippable vertices as *extremal vertices* from here on. Exact definitions of these notions are postponed to Section 1.1.

We define convex sets of a pseudohalfplane hypergraph $\mathcal{H}$ as sets that are intersections of some hyperedges of $\mathcal{H}$ and we refer to these sets as *pseudoconvex sets.* Notice that this is again in parallel with the geometric definition of convex sets (or more precisely, of the subsets of a base point set P that we get by intersecting P with convex sets). We have seen that already with halfplanes one can phrase many interesting problems, but using the notion of convex sets we can finally phrase many of the formative problems of discrete geometry, like the classical Helly Theorem, Carathéodory's Theorem, Radon's Theorem, Erdős-Szekeres problem and the list goes on. While all of these problems are about discrete point sets, there are two essentially different types among them, in one type the whole statement is about some fixed point set P while in the other type there is a point outside of P that plays a role. E.g., in Carathéodory's theorem the whole statement is about a fix point set, while the classical Helly theorem guarantees the existence of a new point in the plane with some property. The first type of these problems translates immediately to a statement about pseudoconvex sets and it is interesting to see if it remains true in this more general setting. For the second type we can also pose a corresponding problem about pseudoconvex sets, where we want to extend the vertex set of the hypergraph $\mathcal{H}$ with one or more vertices (we can extend the original hyperedges on the new vertices as we like) so that it remains a pseudohalfplane hypergraph and has the required property. Observe that for the first type a result about pseudoconvex sets implies the corresponding geometric result but for the second type such an implication does not immediately follow, although it still implies with a bit of additional work, as we will see later.

Following this approach, we prove results about pseudoconvex sets that correspond to the planar case of some of the most important results of discrete geometry, namely Helly

[2] The definition of pseudohalfplanes can be found in the full version of the paper [8]. The definition of pseudohalfplane hypergraphs can be found in Section 1.1 and its connection to the geometric setting is detailed in the full version [8].

Theorem, Carathéodory Theorem, Radon's Theorem and the Cup-Cap Theorem.

Finally, we discuss the relation of our definitions and results to previous similar results. A careful analysis reveals that we have mostly rediscovered things that were known for a long time about oriented matroids (in particular about rank 3 acyclic oriented matroids) or not so long about topological affine planes (TAPs, in short). As said in [3], in the past several people rediscovered what amounts to an axiom system for oriented matroids (or some special case thereof), without realizing that their work overlapped with already published papers. Our contribution can be regarded as an extension of this sequence of axiom systems by a new and interesting axiomatization of acyclic oriented matroids of rank 3. However, we think that our methods are interesting on their own as they give a completely different approach based on hypergraphs on vertices that have a linear ordering on them. Also, while at the end our particular results are not stronger, formally our approach handles a bigger family of hypergraphs compared to what comes from oriented rank 3 matroids. Overall, the following sentence quoted from I.M. Gel'fand in [3] in relation to rediscoveries of the above mentioned axiom systems certainly applies to our case as well: "If you are not too ambitious, it can be a pleasure to realize that you have rediscovered something previously known, because at least then you know that you were on the right track."

Due to space constraints we state only our Helly theorem result about pseudoconvex sets. The generalizations of Carathéodory's Theorem, Kirchberger's Theorem, Separation Theorem, Radon's Theorem and the Cup-Cap Theorem are stated in the full version [8], along with all the proofs and the connection to geometry and to other abstract notions of convexity.

1.1 Basic definitions

▶ **Definition 1.1.** Given a hypergraph $\mathcal{H}$ on vertex set S and a subset S' of S, the *sub-hypergraph of $\mathcal{H}$ induced by S'* is the hypergraph on vertex set S' with hyperedge set $\{H \cap S' : H \in \mathcal{H}\}$ and it is denoted by $\mathcal{H}[S']$.

We recall the definition of an ABA-free hypergraph and its unskippable vertices from [7].

▶ **Definition 1.2.** A hypergraph $\mathcal{H}$ on an ordered vertex set is called *ABA-free* if $\mathcal{H}$ does not contain two hyperedges A and B for which there are three vertices $x < y < z$ such that $x, z \in A \setminus B$ and $y \in B \setminus A$.[3]

▶ **Definition 1.3.** In a hypergraph $\mathcal{F}$ on an ordered vertex set, a vertex a is *skippable* if there exists an $A \in \mathcal{F}$ such that $\min(A) < a < \max(A)$ and $a \notin A$. In this case we say that *A skips a*. A vertex a is *unskippable* if there is no such A.

▶ **Lemma 1.4.** *[7] If $\mathcal{F}$ is ABA-free, then every $A \in \mathcal{F}$ contains an unskippable vertex.*

Now we recall the definition of pseudohalfplane hypergraphs from [7].

▶ **Definition 1.5.** A hypergraph $\mathcal{H}$ on an ordered vertex set is a pseudohalfplane hypergraph if there exists an ABA-free $\mathcal{F}$ on the same ordered vertex set[4] such that $\mathcal{H} \subseteq \mathcal{F} \cup \bar{\mathcal{F}}$. Call $\mathcal{T} = \mathcal{H} \cap \mathcal{F}$ the topsets and $\mathcal{B} = \mathcal{H} \cap \bar{\mathcal{F}}$ the bottomsets, observe that both $\mathcal{T}$ and

[3] We imagine the vertices on a horizontal line, and thus if $x < y$ then we may say that x is to the left from y and so on.

[4] Let $\bar{\mathcal{F}}$ denote the family of the complements of the hyperedges of $\mathcal{F}$. It is easy to see and was shown in [7] that if $\mathcal{F}$ is ABA-free then $\bar{\mathcal{F}}$ is also ABA-free.

$\mathcal{B}$ are ABA-free. The unskippable vertices of $\mathcal{F}$ (resp. $\bar{\mathcal{F}}$) are called topvertices (resp. bottomvertices).[5]

Now we can proceed by defining the extremal vertices of a pseudohalfplane hypergraph:

▶ **Definition 1.6.** Given a pseudohalfplane hypergraph $\mathcal{H}$, the union of the topvertices and bottomvertices is called the *extremal vertices* of $\mathcal{H}$ and is denoted by $E(\mathcal{H})$ (or simply E when the underlying hypergraph is clear from the context).

In light of the geometric setting the following is a natural way to define convex sets which turns out to be also very fruitful:

▶ **Definition 1.7.** Given a hypergraph $\mathcal{H}$ on an ordered set S of vertices, the family of those subsets which are intersections of hyperedges of $\mathcal{H}$ are called the *convex sets* of $\mathcal{H}$. The *convex hull* of a subset $S' \subseteq S$ of the vertices is the convex set $Conv(S') = \cap\{H : H \in \mathcal{H}, S' \subseteq H\}$ (where we define $\cap\emptyset := S$).

Clearly, given a point set P in the plane, the subsets of P which are defined by (geometric) convex sets are convex sets of the respective (pseudo)halfplane hypergraph.

To state many of our results, we need the slightly technical definition of an extension of a pseudohalfplane hypergraph by new hyperedges or vertices, the definition can be found in the full version [8].

1.2 Helly theorems for pseudohalfplanes

In [7] already some discrete Helly-type theorems were proved about pseudohalfplane hypergraphs:

▶ **Lemma 1.8** (Primal Discrete Helly theorem for pseudohalfplanes, $3 \to +1$)**.** *[7] Given a pseudohalfplane hypergraph $\mathcal{H}$ such that every triple of hyperedges has a common vertex, then we can extend $\mathcal{H}$ to a pseudohalfplane hypergraph by a vertex contained in every hyperedge of the extension.*

Considering if Lemma 1.8 has a dual, dual Helly theorems are meaninglessly true for pseudohalfplane hypergraphs as we can always add a new hyperedge containing all vertices and the hypergraph remains to be a pseudohalfplane hypergraph.

Recently Jensen, Joshi, Ray [5] proved discrete Helly-type theorems which can be formulated in terms of halfplane hypergraphs, their results were extended by the author to pseudohalfplane hypergraphs [6]. In these results while we cannot hit all hyperedges with one vertex, on the other hand we can choose the vertex from the original vertex set.[6] Instead of listing all these results, we mention just one which is closest to Lemma 1.8:

▶ **Theorem 1.9** (Primal Strong Discrete Helly theorem for pseudohalfplanes, $3 \to 2$)**.** *[6] Given a pseudohalfplane hypergraph $\mathcal{H}$ such that every triple of hyperedges has a common vertex, there exists a set of at most* 2 *vertices that hits every hyperedge of $\mathcal{H}$.*

[5] Notice that the top- and bottomvertices depend only on $\mathcal{F}$ and not on $\mathcal{H}$ itself. For a given $\mathcal{H} = \mathcal{T} \cup \mathcal{B}$ multiple $\mathcal{F}$'s can witness that it is a pseudohalfplane hypergraph which can lead to different set of top and bottomvertices. The smallest valid family is $\mathcal{T} \cup \bar{\mathcal{B}}$, which in particular gives the largest set of top and bottomvertices. For these reasons, when given a pseudohalfplane hypergraph $\mathcal{H}$, even when not said explicitly, there is an ABA-free $\mathcal{F}$ corresponding to it.

[6] To distinguish from the rest of the discrete Helly theorems, we will use the word *strong* to refer to the fact that the vertex found is in the original vertex set. This is somewhat similar to the difference between (strong) ϵ-nets and weak ϵ-nets

We are able to generalize Helly Theorem for pseudoconvex sets:

▶ **Theorem 1.10** (Discrete Helly theorem for pseudoconvex sets, $3 \to +1$)**.** *Given a pseudohalfplane hypergraph $\mathcal{H}$ and a $\mathcal{C}$ subfamily of its convex sets such that every triple of convex sets from $\mathcal{C}$ has a common vertex, then we can extend $\mathcal{H}$ to a pseudohalfplane hypergraph by a vertex contained in every convex set which is an extension of a set from $\mathcal{C}$.*

This generalizes Lemma 1.8 from [7] in two ways. As a first step, instead of $\mathcal{H}$ we can consider a subfamily of $\mathcal{H}$ (similar to Theorem **??**, where we considered a subset S' of S instead of the whole S). As a second step, this family is actually not required to be a subfamily of $\mathcal{H}$, instead it has to be only a subfamily $\mathcal{C}$ of the convex sets of $\mathcal{H}$.

2 Discussion

We have presented a method to generalize statements about discrete point sets, halfplanes and convex sets to statements on discrete points sets, pseudohalfplanes and pseudoconvex sets. Our setting is purely combinatorial and built using elementary parts, starting with the notion of a hypergraph being ABA-free everything else is built up step-by-step. This offers a very simple axiomatization of planar (pseudo)convexity. We managed to generalize this way many classical results about planar convexity. This discrete relaxation of planar geometry is significantly more general than the planar setting yet still allows us to prove statements that are exactly like their geometric counterparts. Also, many natural families of regions are pseudohalfplane families, thus our generalizations have immediate geometric consequences (e.g., about translates of an unbounded convex region). We compared our results to other similar results about TAPs, oriented matroids and p-convex hulls, pinpointing the connections and differences between them.

There are many further important results about convex sets, yet it falls beyond the scope of this paper to consider all of them for pseudoconvex sets. We list a few of these possibilities: colorful Carathéodory's Theorem, colorful Helly Theorem, Tverberg's Theorem, colorful Tverberg's Theorem, fractional versions, results about the existence of empty k-holes; are these true for pseudoconvex sets? Further, it is interesting to see if problems open about convex sets can be improved in the context of pseudoconvex sets, in particular finding a maximal subset of points in a convex position, bounding the size of weak epsilon-nets for convex sets or the number of k-sets.

Finally, it would be interesting to develop a similar framework of higher dimensional discrete pseudoconvexity.

Acknowledgement

The author is grateful to D. Pálvölgyi for the many discussions about these results and to A. Holmsen for the several insightful comments (primarily but not exclusively) about the connections to earlier results, especially to oriented matroids.

References

1 Geometric hypergraph zoo. URL: `http://coge.elte.hu/cogezoo.html`.

2 Eyal Ackerman, Balázs Keszegh, and Dömötör Pálvölgyi. Coloring hypergraphs defined by stabbed pseudo-disks and ABAB-free hypergraphs. *SIAM Journal on Discrete Mathematics*, 34(4):2250–2269, 2020.

3 Anders Björner, Michel Las Vergnas, Bernd Sturmfels, Neil White, and Günter M. Ziegler. *Oriented Matroids*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2nd edition, 1999. doi:10.1017/CBO9780511586507.

4 Gábor Damásdi and Dömötör Pálvölgyi. Realizing an m-uniform four-chromatic hypergraph with disks. 2020. arXiv:2011.12187.

5 Frederik Brinck Jensen, Aadi Joshi, and Saurabh Ray. Discrete Helly type theorems. In *Proceedings of the 30th Annual Canadian Conference on Computational Geometry, CCCG 2020, August 5-7, 2020, University of Saskatchewan, Saskatoon, Saskatchewan, Canada*, pages 332–335, 2020.

6 Balázs Keszegh. Discrete Helly-type theorems for pseudohalfplanes. *European Journal of Combinatorics*, 101:103469, 2022.

7 Balázs Keszegh and Dömötör Pálvölgyi. An abstract approach to polychromatic coloring: shallow hitting sets in aba-free hypergraphs and pseudohalfplanes. *J. Comput. Geom.*, 10:1–26, 2019.

8 Balázs Keszegh. A new discrete theory of pseudoconvexity, 2022. arXiv:2202.07697.

9 Shakhar Smorodinsky and Yelena Yuditsky. Polychromatic coloring for half-planes. *Journal of Combinatorial Theory, Series A*, 119(1):146–154, 2012.

The complexity of geodesic spanners

Sarita de Berg[1], Marc van Kreveld[1], and Frank Staals[1]

1 Department of Information and Computing Sciences, Utrecht University, the Netherlands
S.deBerg@uu.nl, M.J.vanKreveld@uu.nl, F.Staals@uu.nl

Abstract

A *geometric t-spanner* for a set S of n point sites in $\mathbb{R}^2$ is an edge-weighted graph for which the (weighted) distance between any two sites $p, q \in S$ is at most t times the original distance between p and q. We introduce a novel spanner property for spanners with non-constant complexity edges: the spanner *complexity*, i.e. the total complexity of all edges in the spanner. Let S be a set of n point sites in a simple polygon P with m vertices. We provide a general construction, for any constant $\varepsilon > 0$ and fixed integer $k \geq 1$, of a $(2k+\varepsilon)$-spanner with complexity $O((mn^{1/k} + n) \log^2 n)$. Additionally, for any constant $\varepsilon > 0$ and integer constant $t \geq 2$, we show a lower bound for the complexity of any $(t-\varepsilon)$-spanner of $\Omega(mn^{1/(t-1)} + n)$.

1 Introduction

In the design of networks on a set of nodes, we often consider two criteria: few connections between the nodes, and small distances. Spanners are geometric networks on point sites that replace the small distance criterion by a small detour criterion. Formally, a *geometric t-spanner* for a set S of n point sites in $\mathbb{R}^2$ is an edge-weighted graph $\mathcal{G} = (S, E)$ for which the (weighted) distance $d_{\mathcal{G}}(p, q)$ between any two sites $p, q \in S$ is at most $t \cdot d(p, q)$, where $d(p, q)$ denotes the distance between p and q in the distance metric we consider. The smallest t for which a graph $\mathcal{G}$ is a t-spanner is called the *spanning ratio* of $\mathcal{G}$. The number of edges in the spanner is called the *size* of the spanner.

For the Euclidean distance, for any fixed $\varepsilon > 0$, there is a $(1+\varepsilon)$-spanner of $O(n)$ edges [8]. For the more general case, namely metric spaces of bounded doubling dimension, we can also construct a $(1+\varepsilon)$-spanner of size $O(n)$ for any fixed $\varepsilon > 0$ [7, 5, 6]. This is no longer the case when the sites lie in a simple polygon P and we measure the distance between two points p, q by their geodesic distance: the length of the shortest path between p and q fully contained within P. Abam et al. [1] show there is a set of n sites in a simple polygon P for which any geodesic $(2-\varepsilon)$-spanner has $\Omega(n^2)$ edges. Recently, Abam et al. [2] showed that a geodesic $(2+\varepsilon)$-spanner with $O(n \log n)$ edges exists for points on a polyhedral terrain, thereby almost closing the gap between the upper and lower bound.

The spanning ratio and size of spanners are not the only properties of spanners that can be optimized. Many different properties have been studied, such as total weight (or lightness), maximum degree, (hop) diameter, and fault-tolerance [8, 4].

When we consider distance metrics for which the edges in the spanner—which are shortest paths—no longer have constant complexity, another interesting property of spanners arises: the spanner *complexity*, i.e. the total complexity of all edges in the spanner. In this paper, we study this novel property in a setting where our sites lie in a simple polygon P with m vertices, and we measure the distance between two sites by their geodesic distance. In this setting, a single shortest path may have complexity $\Theta(m)$. We show that any $(3-\varepsilon)$-spanner may have complexity $\Omega(nm)$, thus implying that the $(2+\varepsilon)$-spanner of Abam et al. [2] may also have complexity $\Omega(nm)$, despite having $O(n \log n)$ edges.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

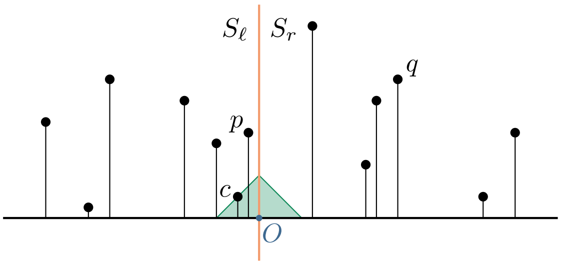


Figure 1 Construction of the 1-dimensional additively weighted spanner.

To improve this complexity, we first introduce a simple 2-spanner with $O(n \log n)$ edges for an additively weighted point set in a 1-dimensional Euclidean space; see Section 2. In Section 3, we use this result to obtain a geodesic $2\sqrt{2}$-spanner with $O(n \log^2 n)$ edges for a point set in a simple polygon. In Section 4, we focus on the complexity of geodesic spanners. We provide a general construction, for any constant $\varepsilon > 0$ and fixed integer $k \geq 1$, of a $(2k+\varepsilon)$-spanner with complexity $O((mn^{1/k}+n)\log^2 n)$. Additionally, for any constant $\varepsilon > 0$ and integer constant $t \geq 2$, we show a lower bound for the complexity of any $(t-\varepsilon)$-spanner of $\Omega(mn^{1/(t-1)}+n)$. Some proofs are omitted due to the space constraints and will be included in a future full version.

2 A 1-dimensional additively weighted 2-spanner

We consider an additively weighted spanner $\mathcal{G}$ in 1-dimensional Euclidean space, where each site $p \in S$ has a non-negative weight $w(p)$. The distance $d_w(p,q)$ between two sites $p, q \in S$ is given by $d_w(p,q) = w(p) + |pq| + w(q)$, where $|pq|$ denotes the Euclidean distance. Without loss of generality, we can map $\mathbb{R}^1$ to the x-axis, and the weights to the y-axis, see Figure 1. This allows us to speak of the sites left (or right) of some site p.

To construct our spanner $\mathcal{G}$, we first partition the points into two sets S_ℓ and S_r of roughly equal size by a point O with $w(O) = 0$. S_ℓ contains all points left of O, and S_r all points right of O. When one or more points lie on the vertical line through O, we simply assign them to S_r or S_ℓ arbitrarily in such a way that the resulting sets are of roughly equal size. We then find a point $c \in S$ for which $d_w(c,O)$ is minimal. For all $p \in S$, $p \neq c$, we add the edge (p,c) to $\mathcal{G}$. Finally, we handle the sets S_ℓ and S_r, excluding the site c, recursively.

▶ **Lemma 1.** *The graph $\mathcal{G}$ is a 2-spanner of size $O(n \log n)$.*

Proof. As we add $O(n)$ edges in each level of the recursion, the total number of edges in $\mathcal{G}$ is $O(n \log n)$. Consider two sites $p, q \in S$. Let c be the chosen center point at the level of the recursion where p and q are assigned to different subsets S_ℓ and S_r. Assume w.l.o.g. that $p \in S_\ell$ and $q \in S_r$. Note that, because $p \in S_\ell$ and $q \in S_r$ we have $d_w(p,q) = d_w(p,O) + d_w(q,O)$. Furthermore, $d_w(c,O) \leq d_w(p,O)$ and $d_w(c,O) \leq d_w(q,O)$, by the choice of c. Because both edges (p,c) and (q,c) are in $\mathcal{G}$, we get for $d_\mathcal{G}(p,q)$:

$$d_\mathcal{G}(p,q) \leq d_w(p,O) + 2d_w(c,O) + d_w(q,O) \leq 2d_w(p,O) + 2d_w(q,O) = 2d_w(p,q). \quad ◀$$

3 A simple geodesic spanner

Just like Abam et al. [2], we use our 1-dimensional spanner to construct a geodesic spanner. We are more interested in the simplicity of the spanner than its spanning ratio, as we base our

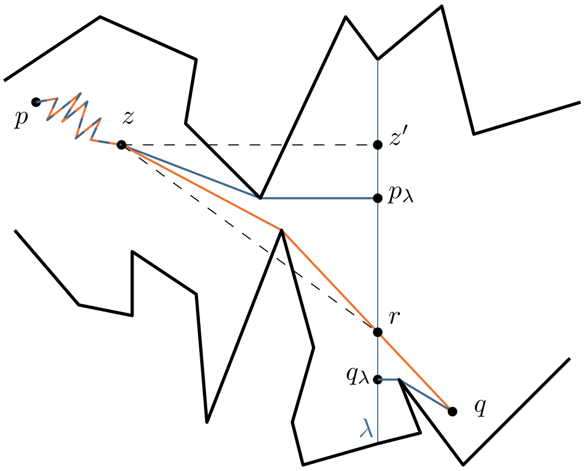


Figure 2 The shortest path $\pi(p,q)$ crosses λ at r. The difference in length between the direct path from z to r and the path through p_λ can be bounded by considering the triangle $\mathcal{T} = (z, z', r)$.

low complexity spanners, discussed in Section 4, on this simple geodesic spanner. We denote by $d(p,q)$ the geodesic distance between $p, q \in P$, and by $\pi(p,q)$ the shortest (geodesic) path from p to q. We analyze the simple construction with respect to any 1-dimensional additively weighted t-spanner of size $O(n \log n)$. We show that restricting the domain to a simple polygon improves the achieved spanning ratio from $3t$ to $\sqrt{2}t$. The construction can be refined to achieve a spanning ratio of $t + \varepsilon$ [2].

As in [2] and [1], we first partition P into two subpolygons P_ℓ and P_r by a line segment λ, such that each subpolygon contains at most two thirds of the sites in S [3]. We denote by S_ℓ and S_r the sites in P_ℓ and P_r, respectively. For each site $p \in S$, we then find the point p_λ on λ geodesically closest to p. As λ is a line segment, the set S_λ, containing all projected points, gives rise to a weighted 1-dimensional Euclidean space, where $w(p_\lambda) := d(p, p_\lambda)$. We compute a t-spanner $\mathcal{G}_\lambda = (S_\lambda, E_\lambda)$ of size $O(n \log n)$ for this set. For each pair $(p_\lambda, q_\lambda) \in E_\lambda$, we add the edge (p,q) to our spanner $\mathcal{G}$. Finally, we recursively compute spanners for S_ℓ and S_r, and add their edges to $\mathcal{G}$ as well.

▶ **Lemma 2.** *Given an algorithm to construct a 1-dimensional additively weighted t-spanner $\mathcal{G}_\lambda$ of size $O(n \log n)$, the graph $\mathcal{G}$ is a geodesic $t\sqrt{2}$-spanner of size $O(n \log^2 n)$.*

Proof. As $\mathcal{G}_\lambda$ has $O(n \log n)$ edges, and these directly correspond to edges in $\mathcal{G}$, we have $O(n \log^2 n)$ edges in total. Let p, q be two sites in S. If both are in S_ℓ (or S_r), then there is a path of length $t\sqrt{2}d(p,q)$ by induction. So, we assume w.l.o.g. that $p \in S_\ell$ and $q \in S_r$. Let r be the intersection point of $\pi(p,q)$ and λ. Observe that p_λ and q_λ must be on opposite sides of r, otherwise r cannot be on the shortest path. We assume w.l.o.g. that λ is a vertical line segment (and S_ℓ is left of λ), and that p_λ is above r and q_λ below r. Because $\mathcal{G}_\lambda$ is a t-spanner, we know that there is a weighted path from p_λ to q_λ of length at most $td_w(p_\lambda, q_\lambda)$. As $w(p_\lambda) = d(p, p_\lambda)$, this directly corresponds to a path in the polygon. So,

$$d_{\mathcal{G}}(p,q) \le d_{\mathcal{G}_\lambda}(p_\lambda, q_\lambda) \le td_w(p_\lambda, q_\lambda) = t(d(p, p_\lambda) + |p_\lambda r| + |r q_\lambda| + d(q_\lambda, q)). \qquad (1)$$

Let z be the point where the shortest paths from p to p_λ and r separate. See Figure 2 for an illustration. The shortest paths $\pi(z, p_\lambda)$ and $\pi(z, r)$ form a funnel $\mathcal{F}(z, p_\lambda, r)$ to the line segment $p_\lambda r$. Consider the right triangle $\mathcal{T} = (z, z', r)$, where z' is the intersection point of the line perpendicular to λ through z and the line containing λ. Note that z' does not necessarily lie within P. For this triangle we have that

$$|zr| \geq \frac{\sqrt{2}}{2}(|zz'| + |z'r|). \tag{2}$$

Next, we show that the path $\pi(z, p_\lambda)$ from z to p_λ is a y-monotone convex polygonal chain ending at or below z'. The path $\pi(z, p_\lambda)$ is bounded from below by $\pi(z, r)$. These paths do not overlap by definition of z, thus the path $\pi(z, p_\lambda)$ can never bend upwards. If z sees z', then $p_\lambda = z'$, otherwise the chain must bend at one or more vertices of the part of the polygon above $\pi(z, p_\lambda)$, and thus lie below z'. It follows that $\pi(z, p_\lambda)$ is a convex y-monotone chain contained within $\mathcal{T}$. Similarly, we conclude that $\pi(z, r)$ is contained within $\mathcal{T}$. Additionally, this gives us that $d(z, p_\lambda) \leq |zz'| + |z'p_\lambda|$, and $d(z, r) \geq |zr|$. Together with Equation 2 this yields $d(z, p_\lambda) + |p_\lambda r| \leq |zz'| + |z'r| \leq \sqrt{2}|zr| \leq \sqrt{2}d(z, r)$. And thus

$$d(p, p_\lambda) + |p_\lambda r| = d(p, z) + d(z, p_\lambda) + |p_\lambda r| \leq d(p, z) + \sqrt{2}d(z, r) \leq \sqrt{2}d(p, r).$$

Symmetrically, we find for q that $d(q, q_\lambda) + |q_\lambda r| \leq \sqrt{2}d(q, r)$. From this, together with Equation 1, we conclude that $d_{\mathcal{G}}(p, q) \leq t\left(\sqrt{2}d(p, r) + \sqrt{2}d(r, q)\right) = t\sqrt{2}d(p, q)$. ◀

Applying Lemma 2 to the spanner of Section 2 yields a $2\sqrt{2}$-spanner of size $O(n \log^2 n)$.

4 Complexity of geodesic spanners

In general, a geodesic spanner $\mathcal{G} = (S, E)$ in a simple polygon P with m vertices may have complexity $O(m|E|)$. It is easy to see that the $2\sqrt{2}$-spanner of Section 2 and 3 can have complexity $\Omega(nm)$, just like the spanners in [2]. As one of the sites, c, is connected to all other sites, the polygon in Figure 3 provides this lower bound. The construction in Figure 3 even shows that the same lower bound holds for the worst-case complexity of any $(3 - \epsilon)$-spanner. Additionally, the following theorem implies a trade-off between the spanning ratio and the spanner complexity.

▶ **Theorem 3.** *For any constant $\varepsilon > 0$ and integer constant $t \geq 2$, there exists a set of n points in a simple polygon P with $m = \Omega(n)$ vertices for which any geodesic $(t - \varepsilon)$-spanner has complexity $\Omega(mn^{1/(t-1)})$.*

The proofs of these lower bounds are omitted here. Next, we present a spanner that almost matches this bound. We first present a $4\sqrt{2}$-spanner of bounded complexity, and then generalize the approach to obtain a $(2k + \varepsilon)$-spanner of complexity $O((mn^{1/k} + n) \log^2 n)$.

4.1 A $4\sqrt{2}$-spanner of complexity $O((m\sqrt{n} + n) \log^2 n)$

To improve the complexity of the geodesic spanner, we adapt our construction for the additively weighted spanner $\mathcal{G}_\lambda$ as follows. After finding the site $c \in S$ for which $d_w(c, O)$ is minimal, we do not add all edges (p, c), $p \in S$, to $\mathcal{G}_\lambda$. Instead, we form groups of sites whose original points (before projection to λ) are 'close' to each other in the polygon. For each group S_i, we add all edges (p, c_i), $p \in S_i$, to $\mathcal{G}_\lambda$, where c_i is the site in S_i for which $d_w(c_i, O)$ is minimal. Finally, we add all edges (c_i, c) to $\mathcal{G}_\lambda$.

To make sure the complexity of our geodesic spanner does not become too large, we must choose the groups in such a way that the edges (p_λ, q_λ) we add to the spanner $\mathcal{G}_\lambda$—these are included in $\mathcal{G}$ as shortest paths $\pi(p, q)$—do not cross 'bad' parts of the polygon too often. To achieve this, we consider the shortest path tree SPT_c of c: the union of all shortest paths

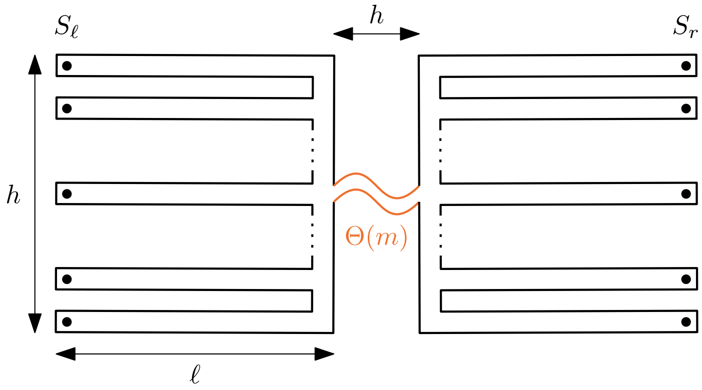


Figure 3 Any $(3-\varepsilon)$-spanner in a simple polygon with m vertices may have complexity $\Omega(nm)$.

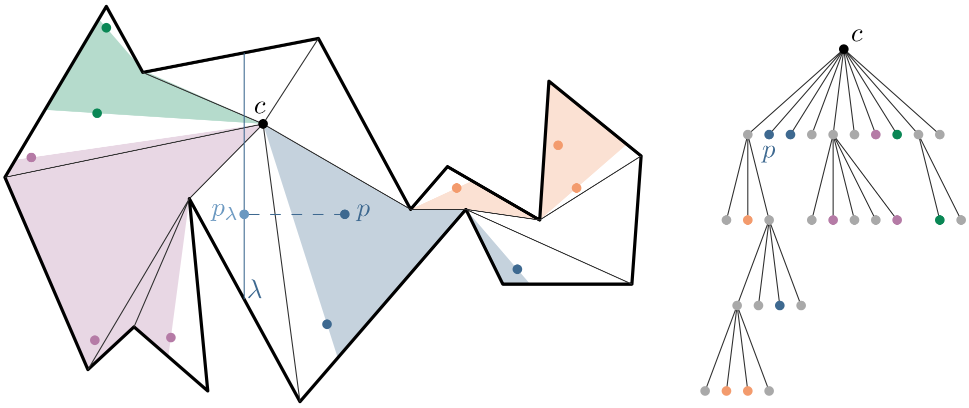


Figure 4 The shortest path tree of c. Each group S_i has an associated polygonal region R_i in P.

from c to the vertices of P. Note that here we associate each site p_λ in the 1-dimensional space with its original site p in the polygon, while constructing the 1-dimensional spanner. We include each site $p \in S \setminus \{c\}$ as a leaf in SPT_c as the child of the last vertex on $\pi(c,p)$. This gives rise to an ordering of the sites, based on the in-order traversal of the tree. We assign the first $\lceil\sqrt{n}\rceil$ sites to S_1, the second $\lceil\sqrt{n}\rceil$ to S_2, etc. See Figure 4.

We will prove the complexity of the edges in one level of the 1-dimensional spanner is $O(m\sqrt{n}+n)$. This implies that the complexity of $\mathcal{G}$ is $O((m\sqrt{n}+n)\log^2 n)$.

Two types of edges are added to the spanner: 1) edges from some c_i to c, and 2) edges from some $p \in S_i$ to c_i. There are $O(\sqrt{n})$ type 1 edges, that each have a complexity of $O(m)$. Thus the total complexity of these edges is $O(m\sqrt{n})$. Analyzing the complexity of the type 2 edges, the edges within each group, is more involved. For each group S_i, consider the minimal subtree $\mathcal{T}_i$ of SPT_c containing all $p \in S_i$. $\mathcal{T}_i$ defines a polygonal region R_i in P as follows. Let v_i be the root of $\mathcal{T}_i$. Consider the shortest path $\pi(v_i, p_\ell)$, where p_ℓ is the leftmost site of S_i in $\mathcal{T}_i$ by the ordering used before. Let π_ℓ be the path obtained from $\pi(v_i, p_\ell)$ by extending the last segment of $\pi(v_i, p_\ell)$ to the boundary of P. Similarly, let π_r be such a path for the rightmost site of S_i in $\mathcal{T}_i$. We take R_i to be the region in P rooted at v_i and bounded by π_ℓ, π_r, and some part of the boundary of P. In case v_i is c, we split R_i into two regions R_j and R_k, such that the angle of each of these regions at c is at most π. The set S_i is then also split into two sets S_j and S_k accordingly. See Figure 4. Note that only vertices of P that are in $\mathcal{T}_i$ can occur in R_i. All shortest paths between sites in S_i are contained within R_i. The following lemma bounds the number of $\mathcal{T}_i$'s a vertex of P can occur in.

▶ **Lemma 4.** *Any vertex $v \in SPT_c$ occurs in at most two trees $\mathcal{T}_i$ and $\mathcal{T}_j$ as a non-root node.*

Note that the root r of $\mathcal{T}_i$ is never used in a shortest path between sites in S_i, because r cannot be a reflex vertex of R_i. Let m_i be the number of non-root nodes in $\mathcal{T}_i$. Lemma 4 implies that $\sum_i m_i = O(m)$. The complexity of all type 2 edges is thus $O(n)+\sum_i m_i O(\sqrt{n}) = O(m\sqrt{n}+n)$.

▶ **Lemma 5.** *The graph $\mathcal{G}$ is a geodesic $4\sqrt{2}$-spanner of size $O(n\log^2 n)$.*

Proof. We prove the 1-dimensional spanner $\mathcal{G}_\lambda$ is a 4-spanner with $O(n\log n)$ edges. Together with Lemma 2, this directly implies $\mathcal{G}$ is a $4\sqrt{2}$-spanner with $O(n\log^2 n)$ edges.

In each level of the recursion, we still add only a single edge for each site. Thus, the total number of edges is $O(n\log n)$. Again, consider two sites $p, q \in S$, and let c be the chosen center point at the level where p and q are separated by O. Let S_i be the group of p and S_j the group of q. Both the edges (p, c_i) and (c_i, c) are in $\mathcal{G}_\lambda$, similarly for q. We thus have a path $p \to c_i \to c \to c_j \to q$ in $\mathcal{G}_\lambda$. Using that $d_w(p, c_i) \leq d_w(p, O) + d_w(c_i, O)$, because of the triangle inequality, and $d_w(c_i, O) \leq d(p, O)$, we find:

$$\begin{aligned} d_{\mathcal{G}_\lambda}(p,q) &= d_w(p,c_i) + d_w(c_i,c) + d_w(c,c_j) + d_w(c_j,q) \\ &\leq d_w(p,O) + 2d_w(c_i,O) + 2d_w(c,O) + 2d_w(c_j,O) + d_w(q,O) \\ &\leq 3d_w(p,O) + 2d_w(c,O) + 3d_w(q,O) \\ &\leq 4d_w(p,O) + 4d_w(q,O)) \\ &= 4d_w(p,q) \end{aligned}$$

◀

4.2 A $(2k+\varepsilon)$-spanner of complexity $O((mn^{1/k}+n)\log^2 n)$

In this section we sketch how to generalize the approach of Section 4.1 to obtain a spanner with a trade-off between the (constant) spanning ratio and complexity. Instead of $O(\sqrt{n})$ groups, we create $O(n^{1/k})$ groups, for some integer constant $k \geq 1$. For each of these groups we select a center and then again partition it into $O(n^{1/k})$ groups, and so on. By connecting each center to its parent center, we obtain a tree of height k. This results in a spanning ratio of $k2\sqrt{2}$. Using the refinement mentioned in Section 3, we even obtain a $(2k+\varepsilon)$-spanner.

▶ **Theorem 6.** *Let S be a set of n point sites in a simple polygon P with m vertices, and let $k \geq 1$ be any integer constant. Then there exists a geodesic $(2k+\varepsilon)$-spanner of size $O(n\log^2 n)$ and complexity $O((c_{\varepsilon,k}mn^{1/k}+n)\log^2 n)$, where $c_{\varepsilon,k}$ is a constant depending on ε and k.*

References

1 Mohammad Ali Abam, Marjan Adeli, Hamid Homapour, and Pooya Zafar Asadollahpoor. Geometric spanners for points inside a polygonal domain. In *31st International Symposium on Computational Geometry, SoCG*, volume 34 of *LIPIcs*, pages 186–197, 2015.

2 Mohammad Ali Abam, Mark de Berg, and Mohammad Javad Rezaei Seraji. Geodesic spanners for points on a polyhedral terrain. *SIAM J. Comput.*, 48(6):1796–1810, 2019.

3 Prosenjit Bose, Jurek Czyzowicz, Evangelos Kranakis, Danny Krizanc, and Anil Maheshwari. Polygon cutting: Revisited. In *Discrete and Computational Geometry, Japanese Conference, JCDCG, Revised Papers*, volume 1763 of *LNCS*, pages 81–92, 1998.

4 Prosenjit Bose and Michiel H. M. Smid. On plane geometric spanners: A survey and open problems. *Comput. Geom.*, 46(7):818–830, 2013.

5 T.-H. Hubert Chan, Anupam Gupta, Bruce M. Maggs, and Shuheng Zhou. On hierarchical routing in doubling metrics. *ACM Trans. Algorithms*, 12(4):55:1–55:22, 2016.

6 Lee-Ad Gottlieb and Liam Roditty. An optimal dynamic spanner for doubling metric spaces. In *16th Annual European Symposium on Algorithms, ESA*, volume 5193 of *Lecture Notes in Computer Science*, pages 478–489, 2008.

7 Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM J. Comput.*, 35(5):1148–1184, 2006.

8 Giri Narasimhan and Michiel H. M. Smid. *Geometric Spanner Networks*. Cambridge University Press, 2007.

Extendability of higher dimensional signotopes*

Helena Bergold[1], Stefan Felsner[2], and Manfred Scheucher[2]

1 **Department of Computer Science,**
Freie Universität Berlin, Germany,
`firstname.lastname@fu-berlin.de`

2 **Institut für Mathematik,**
Technische Universität Berlin, Germany,
`lastname@math.tu-berlin.de`

Abstract

In 1926, Levi showed that, for any pseudoline arrangement $\mathcal{A}$ and two points in the plane, $\mathcal{A}$ can be extended by a pseudoline which contains the two prescribed points. Later extendability was studied for arrangements of pseudohyperplanes in higher dimensions. While the extendability with d prescribed points in an arrangement of proper hyperplanes in $\mathbb{R}^d$ is trivial, Richter-Gebert (1993) found an arrangement of pseudoplanes in $\mathbb{R}^3$ which cannot be extended with two particular prescribed points.

In this article, we investigate the extendability of signotopes, which are a rich subclass of oriented matroids. Our main result is that signotopes of odd rank are extendable with two prescribed crossing points. Moreover, we conjecture that for all even ranks $r \geq 4$ there exist signotopes which are not extendable for two prescribed points. Our conjecture is supported by examples for rank 4, 6, and 8.

1 Introduction

Given a family of hyperplanes $\mathcal{H}$ in $\mathbb{R}^d$, any d points in $\mathbb{R}^d$, not all on a common hyperplane of $\mathcal{H}$, define a hyperplane which is distinct from the hyperplanes in $\mathcal{H}$. For dimension $d = 2$, Levi [10] proved in his pioneering article on pseudoline arrangements that the fundamental extendability of line arrangements also applies to the more general setting of pseudoline arrangements. A *pseudoline* is a Jordan curve in the Euclidean plane such that its removal from the plane results in two unbounded components, and a *pseudoline arrangement* is a family of pseudolines such that each pair of pseudolines intersects in exactly one point, where the two curves cross.

▶ **Theorem 1.1** (Levi's extension lemma for pseudoline arrangements [10]). *Given an arrangement $\mathcal{A}$ of pseudolines and two points in $\mathbb{R}^2$, not lying on a common pseudoline of $\mathcal{A}$. Then $\mathcal{A}$ can be extended by an additional pseudoline which passes through the two prescribed points.*

Several proofs for Levi's extension lemma are known today (besides [10], see also [1, 6, 14]) and generalizations to higher dimensions have been studied in the context of oriented matroids, which are by the representation theorem of Folkman and Lawrence [7] projective pseudohyperplane arrangements. For more about oriented matroids, see [5].

Goodman and Pollack [9] showed that there is an arrangement of 8 pseudoplanes in $\mathbb{R}^3$ such that three particular prescribed points do not determine a pseudoplane which is compatible with the arrangement. Richter-Gebert [13] then investigated a weaker version with only two prescribed points such that the extending pseudohyperplane contains these two points.

[*] H. Bergold was funded by the DFG-Research Training Group 'Facets of Complexity' (DFG-GRK 2434). M. Scheucher was supported by the DFG Grant SCHE 2214/1-1.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

More specifically, he gave an example of a rank 4 oriented matroid on 8 elements such that even the weaker version does not hold. However, the existence of an extension lemma or counterexamples remains open in higher dimensions/ranks.

In this article, we present a proof of Levi's extension lemma in a purely combinatorial setting which generalizes to higher dimensions. Our proof uses the notion of r-signotopes and applies to even dimensions d, that is, when the rank $r = d + 1$ is odd; see Theorem 1.2. However, there are non-extendable examples for the ranks 4, 6, and 8, and we conjecture that there is no extension lemma for any even rank $r \geq 4$; see Conjecture 1.3.

Before we can formulate our extension lemma for r-signotopes, we have to introduce some notation, discuss the relation between pseudoline arrangements and signotopes (in Section 1.1), and find an appropriate reformulation of Levi's extension lemma which can be investigated in the context of signotopes (in Section 1.2).

1.1 Signotopes

Signotopes are a combinatorial structure generalizing permutations and *simple* pseudoline arrangements (i.e., no three pseudolines cross in a common point). An *r-signotope* ($r \geq 1$) on n elements is a mapping σ from r-element subsets (*r-subsets*) of $[n]$ to $+$ or $-$, i.e., $\sigma : \binom{[n]}{r} \to \{+, -\}$ such that for every $(r+1)$-subset $X = \{x_1, \ldots, x_{r+1}\}$ of $[n]$ with $x_1 < x_2 < \ldots < x_{r+1}$ there is at most one sign change in the sequence

$$\sigma(X \backslash \{x_1\}), \sigma(X \backslash \{x_2\}), \ldots, \sigma(X \backslash \{x_{r+1}\}).$$

Note that this sequence lists the signs of all induced r-subsets of X in reverse lexicographic order. For 3-signotopes, the following 8 sign patterns on 4-subsets are allowed:

$$++++,\ +++-,\ ++--,\ +---,\ ----,\ ---+,\ --++,\ -+++\,.$$

It is well-known that every arrangement of pseudolines is isomorphic to an arrangement of x-monotone pseudolines [8]. If we label the pseudolines from top to bottom on the left by $1, \ldots, n$, we can read its corresponding 3-signotope σ. The sign of $\sigma(a, b, c)$ for $a < b < c$ indicates the orientation of the triangle formed by the pseudolines a, b, c (see Figure 1). If $\sigma(a, b, c) = +$ the crossing of a and c is below b and if $\sigma(a, b, c) = -$ the crossing of a and c is above b. Furthermore, σ gives information about the ordering of the crossings from left to right along each pseudoline. If $\sigma(a, b, c) = +$ it holds $bc \succ ac \succ ab$ and if $\sigma(a, b, c) = -$ it is $bc \prec ac \prec ab$.

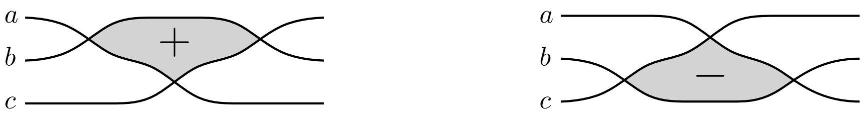


Figure 1 Connection between pseudoline arrangements and 3-signotopes.

Felsner and Weil [6] showed that rank 3 signotopes are in correspondence with simple pseudoline arrangements in $\mathbb{R}^2$ with a special top cell related to the cyclic arrangement. For $r \geq 4$, r-signotopes correspond to special pseudohyperplane arrangements in $\mathbb{R}^{r-1}$, i.e., they are a subclass of oriented matroids of rank r. A geometric representation of r-signotopes in the plane is presented in [11] (see also [3] for the rank 3 case).

1.2 An extension lemma for signotopes

In Levi's extension lemma for pseudoline arrangements, each of the two prescribed points can either lie in a cell of the arrangement, on a pseudoline, or be the crossing point of two pseudolines. To formulate an extension lemma in terms of 3-signotopes, which only captures the combinatorics of an arrangement, we restrict ourselves to simple pseudoline arrangements and to prescribed points, which are crossing points. Crossing points in a pseudoline arrangement are subsets of cardinality 2 given by the two crossed elements. Since the extending pseudoline passes through the two prescribed crossing points, the extension yields a non-simple arrangement. However, by perturbing the extending pseudoline at the non-simple crossing points, we end up with a simple arrangement, see Figure 2.

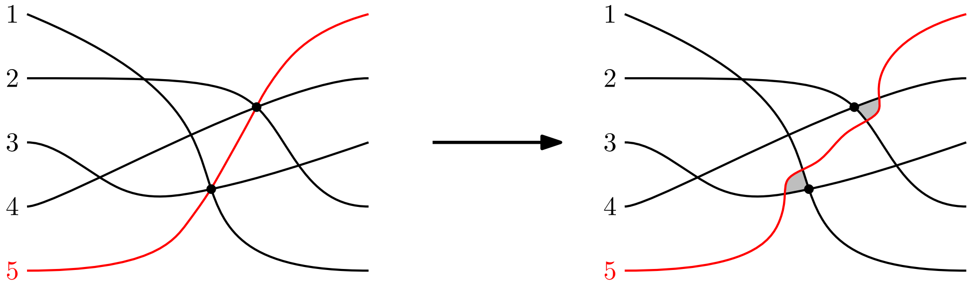


Figure 2 Perturbing an extending pseudoline at the two non-simple crossing points.

A perturbation at a prescribed crossing yields a *triangular cell* incident to the crossing. This cell is bounded by the two pseudolines defining the crossing and the extending pseudoline. Triangular cells play an important role in the study of pseudoline arrangements, since it is possible to change the orientation of a triangle by moving one of its bounding pseudolines over the crossing of the two others. Such a local perturbation is called *triangle flip* and does not change the orientation of any other triangle in the arrangement. For 3-signotopes triangular cells correspond to a 3-subset for which we can exchange the corresponding sign and it remains a signotope. We call such a 3-subset a *fliple*. The notion of fliples generalizes to higher ranks. In an r-signotope σ on $[n]$, an r-subset $X \subseteq [n]$ is a *fliple* if both assignments $+$ and $-$ to $\sigma(X)$ result in a signotope.

When we apply Levi's extension lemma to extend an arrangement of pseudolines, which are ordered from top to bottom on the left, we do not know at which place of the order the new pseudoline will be inserted. In particular, the label of all pseudolines which start below the new one increases by one. To cope with this relabeling-issue in terms of signotopes, we introduce the following notion. For $k \in [n]$ and a subset X of $[n]$, we define

$$X{\downarrow_k} = \{x \mid x \in X, x < k\} \cup \{x-1 \mid x \in X, x > k\}.$$

For an r-signotope σ on the elements $[n]$, we define the k *deletion* $\sigma{\downarrow_k}$ on $[n-1]$ by $\sigma{\downarrow_k}(X{\downarrow_k}) = \sigma(X)$ for all r-sets $X \subseteq [n]$ with $k \notin X$. This is an r-signotope on $[n-1]$.

An r-signotope σ on n elements is *2-extendable* if for each pair of disjoint $(r-1)$-subsets I, J, there is an r-signotope σ^* on $[n+1]$ with fliples I^*, J^* and an *extending* element $k \in [n+1]$ such that $\sigma^*{\downarrow_k} = \sigma$, $I^*{\downarrow_k} = I$, and $J^*{\downarrow_k} = J$.

Using this notion we are now ready to formulate an extension lemma for r-signotopes.

▶ **Theorem 1.2** (Extension lemma for signotopes of odd rank). *For every odd rank $r \geq 3$, every r-signotope is 2-extendable.*

The statement of Theorem 1.2 only applies to signotopes of odd rank. In fact, for ranks 4, 6, and 8, we found signotopes on 8, 12, and 16 elements, respectively, which are not 2-extendable[1]. Based on these examples, we dare the following conjecture:

▶ **Conjecture 1.3.** *For every even $r \geq 4$, there are r-signotopes which are not 2-extendable.*

Despite the restrictions to simple arrangements and crossing points as prescribed points, Theorem 1.2 implies Levi's extension lemma (Theorem 1.1). Details are deferred to the full version; see [4] for a preliminary version.

1.3 Signotopes as a rich subclass of oriented matroids

It is well known that the number of oriented matroids on n elements of rank r is $2^{\Theta(n^{r-1})}$ [5, Corollary 7.4.3]. As shown by Balko [2], r-signotopes are a rich subclass of oriented matroids of rank r.

▶ **Proposition 1.4.** *For $r \geq 3$, the number of r-signotopes on n elements is $2^{\Theta(n^{r-1})}$.*

In ranks 1 and 2 there are 2^n and $n!$ signotopes on $[n]$, respectively. For rank $r \geq 3$, the precise number of r-signotopes on $[n]$ has been computed for small values of r and n; see A6245 (rank 3) and A60595 to A60601 (rank 4 to rank 10) on the OEIS [12].

2 Preliminaries

We now prepare for the proof of Theorem 1.2. In rank 3 the left to right order on each pseudoline yields a partial order of the crossing points of the arrangement. We now define the corresponding partial order on the $(r-1)$-subsets associated with a r-signotope σ. For every r-subset $X = \{x_1, \ldots, x_r\}$ define:

$$\begin{aligned} X\backslash\{x_1\} \succ X\backslash\{x_2\} \succ \cdots \succ X\backslash\{x_r\} \quad &\text{if} \quad \sigma(x_1, \ldots, x_r) = +, \qquad \text{and} \\ X\backslash\{x_1\} \prec X\backslash\{x_2\} \prec \cdots \prec X\backslash\{x_r\} \quad &\text{if} \quad \sigma(x_1, \ldots, x_r) = -. \end{aligned}$$

By taking the transitive closure of all relations obtained from r-subsets, we obtain a partial order on the $(r-1)$-subsets corresponding to σ [6, Lemma 10].

If we rotate an arrangement of pseudolines, i.e., we choose another unbounded cell as the top cell, we get an pseudoline arrangement with the same cell structure. If we only rotate a single pseudoline, then the orientation of the triangle spanned by 3 pseudolines stays the same if and only if the rotated pseudoline is not involved (see for example the triangle spanned by 2,3,4 in the left, resp. 1,2,3 in the right arrangement in Figure 3). In terms of the 3-signotope σ the signs of the rotated signotope σ_{rot} are: $\sigma_{\text{rot}}(a, b, c) = \sigma(a+1, b+1, c+1)$ if $c \neq n$ and $\sigma_{\text{rot}}(a, b, n) = -\sigma(1, a+1, b+1)$.

In general, we define the clockwise rotated signotope σ_{rot} of a given r-signotope σ as:

$$\sigma_{\text{rot}}(x_1, \ldots, x_r) = \begin{cases} -\sigma(1, x_1+1, \ldots, x_{r-1}+1) & \text{if } x_1 < x_2 < \cdots < x_r = n, \\ \sigma(x_1+1, \ldots, x_r+1) & \text{if } x_1 < x_2 < \cdots < x_r < n. \end{cases}$$

Indeed, σ_{rot} is an r-signotope on n elements (see [4] for more details). To keep track of the index shift caused by a clockwise rotation, we define

$$X_{\text{rot}} = \begin{cases} \{x_1 - 1, x_2 - 1, \ldots, x_k - 1\} & \text{if } x_1 \neq 1; \\ \{x_2 - 1, \ldots, x_k - 1, n\} & \text{if } x_1 = 1 \end{cases}$$

[1] The examples and the source code to verify their correctness are available on demand.

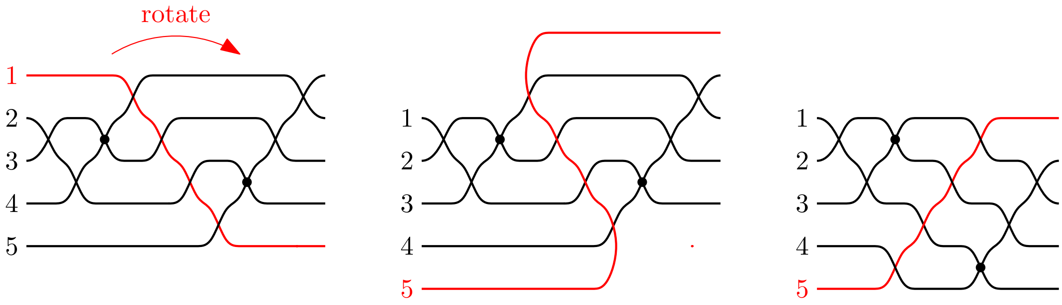


Figure 3 An illustration of a clockwise rotation. The rotated pseudoline is highlighted red.

for any subset $X = \{x_1, \ldots, x_k\}$ of $[n]$ with $x_1 < \ldots < x_k$.

3 Proof of Theorem 1.2

Using these properties we can give a proof for Levi's extension lemma using only the notation of signotopes and the corresponding partial order as introduced in Section 2.

If incomparable elements in the corresponding order are chosen as prescribed points, an arrangement is extendable by an element which we put in the last position, i.e., the $(n+1)$st element, see Figure 2. More abstractly we can extend the arrangement when the prescribed points are maximal elements of a down-set of the partial order. A *down-set* of a partial order $(P, \prec)$ is a subset $D \subset P$ such that for all $p \in P$ and $d \in D$ with $p \preceq d$ it holds $p \in D$.

▶ **Proposition 3.1.** *Let $(P, \prec)$ be the partial order on $(r-1)$-sets corresponding to an r-signotope σ on $[n]$. For every down-set $D \subseteq P$ there is a signotope σ^* on $[n+1]$ such that all r-subsets of the form $m \cup \{n+1\}$ for a maximal element m of D are fliples.*

Proof. Define the extended r-signotope σ^* on $[n+1]$ as follows:

$$\sigma^*(x_1 \ldots, x_r) = \begin{cases} \sigma(x_1, \ldots, x_r) & \text{if } x_1, \ldots, x_r \in [n]; \\ + & \text{if } x_r = n+1 \text{ and } \{x_1, \ldots, x_{r-1}\} \in D; \\ - & \text{if } x_r = n+1 \text{ and } \{x_1, \ldots, x_{r-1}\} \notin D. \end{cases}$$

This is indeed an r-signotope and fulfills the conditions mentioned in the statement. Details are deferred to the full version; see [4] for a preliminary version. ◀

Note that Proposition 3.1 holds for general rank. For odd rank we can always find a rotation of the corresponding signotope such that the two prescribed $(r-1)$-subsets are incomparable and we can use Proposition 3.1 to define an extension.

▶ **Lemma 3.2.** *Let r be an odd integer, let σ be an r-signotope on $[n]$ and let X, Y be two disjoint $(r-1)$-subsets. After at most n clockwise rotations, σ, X, and Y are transformed into σ', X', and Y', resp., such that X' and Y' are incomparable in the partial order $\prec'$ corresponding to σ'.*

Proof. Assume X and Y are comparable in the partial order $\prec$ corresponding to the r-signotope σ with $X \prec Y$. We show that after n clockwise rotations, all signs of σ are reversed. Hence the partial order $\prec_{\text{rot}}$ is the reversed relation to $\prec$.

The sign of an r-subset $\{z_1, \ldots, z_r\}$ changes from $+$ to $-$ or vice versa if and only if the rotated element is contained in $\{z_1, \ldots, z_r\}$, i.e., if we rotate z_1. Hence after rotating n times in general every z_i was rotated and thus the sign of an r-subset changes exactly r times. Since r is odd, the sign after rotating n times is opposite. The obtained signotope is the reverse of the original signotope σ and the corresponding partial order is also reversed.

Since we cannot reverse the order of two disjoint $(r-1)$-sets in one rotation (details are deferred to the full version [4]), there will be a moment where the two disjoint sets are incomparable. ◀

Proposition 3.1 and Lemma 3.2 together imply Theorem 1.2, which completes the proof. The outline is as follows. Using Lemma 3.2, we rotate until the required disjoint $(r-1)$-subsets are incomparable. To extend the signotope we then use the down-set, which consists of all elements smaller than one of the two incomparable $(r-1)$-subsets. In this down-set the two prescribed $(r-1)$-subsets are the maximal elements. Hence we can apply Proposition 3.1 in order to add a new elements as required. Finally, we rotate back so that the original signotope is contained in the new extended signotope. Details are deferred to the full version.

4 Conclusion

Using complete enumeration of small signotopes and a SAT based test of extendability, we found a 4-signotope on 8 elements which is not 2-extendable. Since the number of signotopes explodes as the ranks increases, the complete enumeration was impossible in higher ranks. To still cope with higher ranks, we instead used a SAT based search for signotopes which share structural properties with the rank 4 example. This allowed us to find the examples in rank 6 and 8 which are not 2-extendable.

References

1 A. Arroyo, D. McQuillan, R. B. Richter, and G. Salazar. Levi's lemma, pseudolinear drawings of K_n, and empty triangles. *Journal of Graph Theory*, 87(4):443–459, 2018.

2 M. Balko. Ramsey numbers and monotone colorings. *Journal of Combinatorial Theory, Series A*, 163:34–58, 2019.

3 M. Balko, R. Fulek, and J. Kynčl. Crossing numbers and combinatorial characterization of monotone drawings of K_n. *Discrete & Computational Geometry*, 53(1):107–143, 2015.

4 H. Bergold, S. Felsner, and M. Scheucher. Extendability of higher dimensional signotopes (with Appendix), 2022. `http://page.math.tu-berlin.de/~scheuch/publ/bfs-ehds-eurocg22.pdf`.

5 A. Björner, M. Las Vergnas, B. Sturmfels, N. White, and G. M. Ziegler. *Oriented Matroids*, volume 46 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, second edition, 1999.

6 S. Felsner and H. Weil. Sweeps, Arrangements and Signotopes. *Discrete Applied Mathematics*, 109(1):67–94, 2001.

7 J. Folkman and J. Lawrence. Oriented matroids. *Journal of Combinatorial Theory, Series B*, 25(2):199–236, 1978.

8 J. E. Goodman. Proof of a conjecture of Burr, Grünbaum, and Sloane. *Discrete Mathematics*, 32(1):27–35, 1980.

9 J. E. Goodman and R. Pollack. Three points do not determine a (pseudo-) plane. *Journal of Combinatorial Theory, Series A*, 31(2):215–218, 1981.

10 F. Levi. Die Teilung der projektiven Ebene durch Gerade oder Pseudogerade. *Berichte über die Verhandlungen der Sächsischen Akademie der Wissenschaften zu Leipzig, Mathematisch-Physische Klasse*, 78:256–267, 1926.

11 H. Miyata. On combinatorial properties of points and polynomial curves. arXiv:1703.04963, 2021.

12 OEIS Foundation Inc. The On-Line Encyclopedia of Integer Sequences. Published electronically at `http://oeis.org`.

13 J. Richter-Gebert. Oriented matroids with few mutations. In *Discrete & Computational Geometry*, volume 10, pages 251–269. Springer, 1993.

14 M. Schaefer. A proof of Levi's extension lemma. arXiv:1910.05388, 2019.

Polyline Simplification under the Local Fréchet Distance has Subcubic Complexity

Sabine Storandt[1] **and Johannes Zink**[2]

1 **University of Konstanz, Germany**
`sabine.storandt@uni-konstanz.de`

2 **University of Würzburg, Germany**
`zink@informatik.uni-wuerzburg.de`

Abstract

Given a polyline on n vertices, the polyline simplification problem asks for a minimum size subsequence of these vertices defining a new polyline whose distance to the original polyline is at most a given threshold under some distance measure. We improve the running time bound for the simplification of polylines under the local Fréchet distance. The best algorithm known so far in the literature is using the local Fréchet distance within the Imai-Iri algorithm, which has a cubic running time. We extend the algorithm of Melkman and O'Rourke [Comp. Morph. '88], who consider polyline simplification under the local Hausdorff distance, to be applicable to the local Fréchet distance. The runtime of the algorithm is $\mathcal{O}(n^2 \log n)$ in the Euclidean norm. Moreover, we transfer this principle to other L_p norms. In particular, we can improve the runtime to $\mathcal{O}(n^2)$ in the L_1 and L_∞ norm.

Related Version arXiv:2201.01344

1 Introduction

Polyline simplification is an extensively studied topic due to its relevance to a variety of applications, such as trajectory and shape analysis, data compression or map visualization. The goal of polyline simplification is to replace a given polyline with n vertices with a simpler one while ensuring that the input and the output polyline are sufficiently similar.

The similarity is governed by a given distance threshold ε. We consider only polyline simplifications where the output polyline has to consist of a subsequence of the input polyline vertices. Line segments between these vertices are then called *shortcuts*. The distance measure is applied locally, i.e., the distance between each shortcut and the part of the input polyline it bridges must not exceed ε. To determine the similarity of the input and output polyline, the Hausdorff and the Fréchet distance are the most commonly used measures.

For the problem of polyline simplification under the local Hausdorff distance, the Imai-Iri algorithm [6] from 1988 guarantees a running time of $\mathcal{O}(n^3)$ by reducing the simplification problem to a graph problem. Melkman and O'Rourke [7] showed in 1988 that the running time of the Imai-Iri algorithm can be improved to $\mathcal{O}(n^2 \log n)$ by accelerating the graph construction phase. They exploit the geometric properties using cone-shaped wedges and a wave front, which they call *frontier*. We use both concepts and apply them also to the local Fréchet distance. In 1996, Chan and Chin [3] improved the running once more to $\mathcal{O}(n^2)$ by getting rid of the frontier and, consequently, a log-factor. They only use the cone-shaped wedges – but now in two directions because otherwise they would find false-positive shortcuts.

For the local Fréchet distance, Godau [4] showed in 1991 how to simplify a polyline in $\mathcal{O}(n^3)$ by a straight-forward adjustment of the Imai-Iri algorithm. Since then, no improvements were made; see the article by van Kreveld, Löffler and Wiratma [9] for a recent overview. But the Fréchet distance is often considered the superior distance measure as it takes the

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

order of the vertices along the polyline into account, while the Hausdorff distance ignores this aspect. This, however, makes the computation of the Fréchet distance more intricate.

We show that nevertheless near-quadratic running time bounds can be achieved by adjusting and combining known techniques. Most notably, the concept of a *wave front* data structure, which encodes, for a polyline vertex, the region in which the vertices that are endpoints of shortcuts must lie. This answers an open question by Agarwal, Har-Peled, Mustafa, and Wang [1] whether it is possible to compute an optimal simplification under the local Fréchet distance in subcubic time. Moreover, we investigate the problem not only in the (Euclidean) L_2 norm, but in all L_p ($p \in [1, \infty]$) norms. All of our results refer to the two-dimensional problem, i.e., polyline simplification in the plane.

2 Preliminaries

Our algorithm heavily builds upon the Imai-Iri and the Melkman-O'Rourke algorithm, which are briefly recapped below. In this description we also define our notation before we sketch the main differences to our novel approach.

2.1 Imai-Iri Algorithm for the Local Hausdorff and Fréchet Distance

Given a polyline $\mathcal{L}$ with n vertices $p_1, \ldots, p_n \in \mathbb{R}^2$, the polyline simplification algorithm by Imai and Iri [6] proceeds in two phases. In the first phase, the *shortcut graph* is constructed. This graph has a node for each vertex of $\mathcal{L}$ and it has an edge between two nodes if and only if there is a valid shortcut between the corresponding two vertices of $\mathcal{L}$. For the Hausdorff and the Fréchet distance it can be checked in $\mathcal{O}(n)$ time [2] whether the distance between a line segment and a polyline having $\mathcal{O}(n)$ vertices exceeds ε. Hence, the total running time of the first phase amounts to $\mathcal{O}(n^3)$. In the second phase, a shortest path from the first node p_1 to the last node p_n is computed in the shortcut graph, which can be accomplished in $\mathcal{O}(n^2)$.

2.2 Melkman-O'Rourke Algorithm for the Local Hausdorff Distance

Since in the Imai-Iri algorithm the construction of the shortcut graph dominates the runtime, accelerating this first phase also leads to an overall improvement. Melkman and O'Rourke [7] introduced a faster technique to compute the shortcut graph for the local Hausdorff distance. Starting once at each vertex p_i for $i \in \{1, \ldots, n\}$, they traverse the rest of the polyline vertex by vertex in $\mathcal{O}(n \log n)$ time to determine all valid shortcuts originating at p_i.

To this end, they maintain a cone-shaped region called *wedge* in which all valid shortcuts are required to lie. The wedge is an angular region having its origin at p_i and being the intersection of all angular regions (which we call *local wedges*) that define the areas where valid shortcuts may lie for each intermediate vertex[1]. When traversing the polyline, the wedge iteratively becomes narrower. Moreover, they maintain a *wave front* which is a sequence of circular arcs coming from circles of radius ε, from now on called *unit circles*. The wave front subdivides the wedge into two regions – all valid shortcuts starting at p_i need to have the other end point in the other region not containing p_i. We call this region *valid region*[2]. The

[1] W.l.o.g., we assume hereunder that p_{i+1} has distance at least ε to p_i because otherwise, we could ignore all vertices following p_i and having distance $\leq \varepsilon$ to p_i since they are in ε-distance to any shortcut (p_i, p_j). Moreover, we assume w.l.o.g. that p_i is below p_{i+1} and therefore at the bottom of a wedge.

[2] Note that our notation follows Chan and Chin [3] – like them we call the whole cone-shaped region *wedge*. Melkman and O'Rourke [7] only call the valid region *wedge* and they call the wave front *frontier*.

wave front has size in $\mathcal{O}(n)$ and is stored in a balanced search tree such that querying and updating operations can be made in $\mathcal{O}(\log n)$ time.

When starting at p_i and encountering p_j during the traversal, we denote by D_{ij} the *local wedge* of p_i and p_j, which is the area between the two tangential rays of the unit circle around p_j emanating at p_i. We denote by W_{ij} the (global) wedge where $W_{ij} := \bigcap_{k\in\{i+1,i+2,\ldots,j\}} D_{ik}$. For $\overline{p_ip_j}$ to be a valid shortcut, p_j needs to lie within the valid region, i.e., p_j lies within $W_{i(j-1)}$ and above the wave front. We can check containment in $W_{i(j-1)}$ in constant time and determine the position relative to the wave front in $\mathcal{O}(\log n)$ time. As one can update the wedge from $W_{i(j-1)}$ to W_{ij} in constant time and its wave front in $\mathcal{O}(\log n)$ time, the running time of this phase is $\mathcal{O}(n \log n)$ per starting vertex p_i and $\mathcal{O}(n^2 \log n)$ in total.

3 Novel Approach for the Local Fréchet Distance

In this section, we describe how to obtain an algorithm for polyline simplification running in near-quadratic time by means of the wave front similar to Melkman and O'Rourke [7] and within the framework of Imai and Iri [6]. We first consider the L_2 norm and describe our differences to Melkman and O'Rourke, and then generalize this approach to arbitrary L_p norms ($p \in [1, \infty]$) and exploit special properties of L_1 and L_∞ to obtain better runtimes.

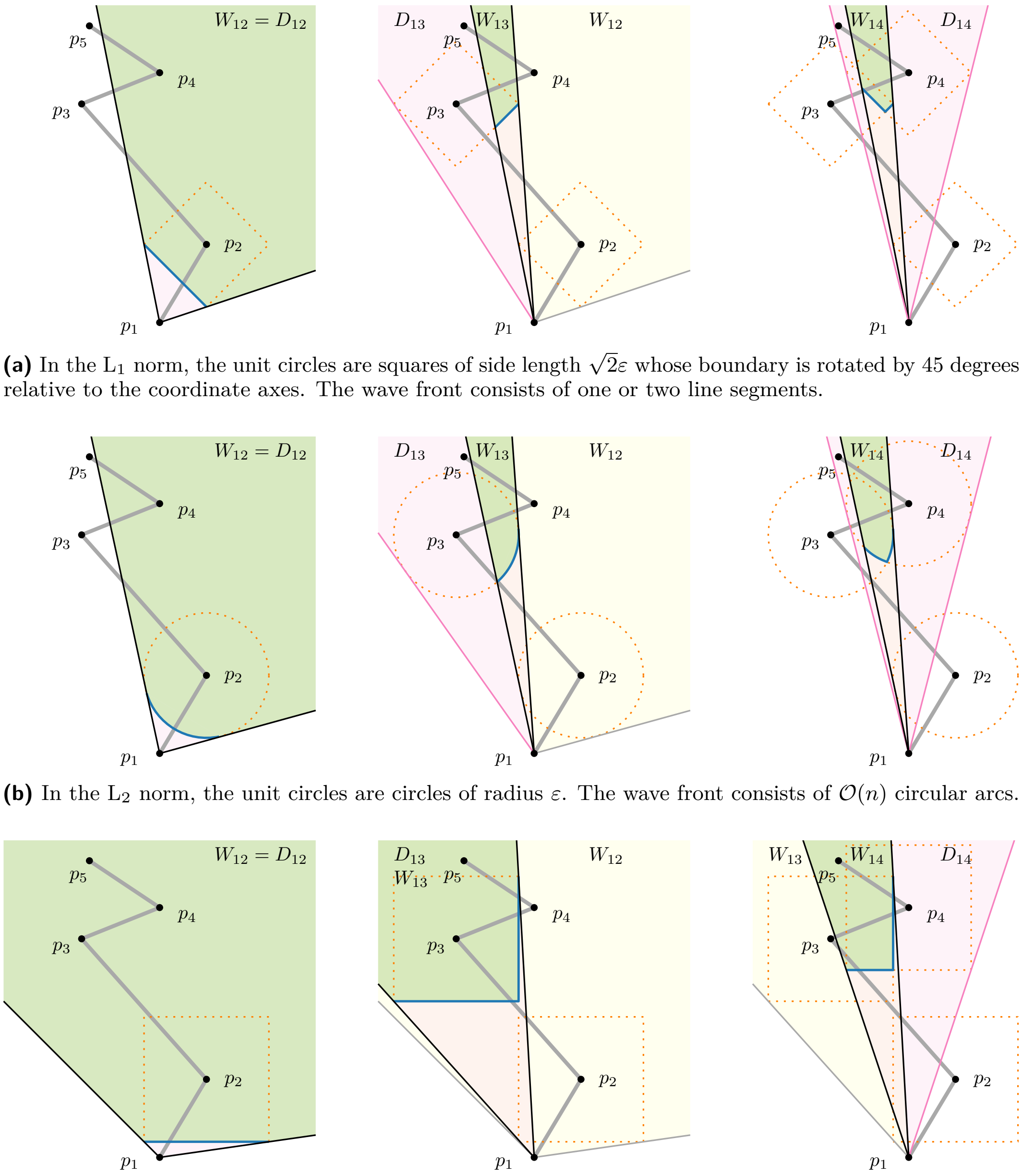
3.1 Outline

Based on Imai and Iri, we build the shortcut graph by traversing the given polyline n times – starting once from each vertex p_i and determining all shortcuts starting at p_i. For each p_i, we construct a wedge with a wave front. The shape of the wave front depends on the shape of the unit circle in the L_p norm. Fig. 1 illustrates the wave front in the L_1, L_2, and L_∞ norm. The properties of the wave front will be discussed in more detail later.

Let us describe the procedure how to determine, for each vertex p_i of the polyline, the set of subsequent vertices to which p_i has a valid shortcut. We traverse the polyline in order $p_{i+1}, p_{i+2}, \ldots, p_n$. During this traversal, we maintain the wedge in which all valid shortcuts need to lie. This would, as in the algorithm by Chan and Chin [3] suffice to assures that the Hausdorff distance threshold is not violated which is a lower bound for the Fréchet distance. To also not exceed the Fréchet distance threshold, we use the wave front. As in the algorithm by Melkman and O'Rourke, the invariant maintained is that for a valid shortcut from p_i to $p_{j>i}$, the point p_j has to be within the valid region of the wedge $W_{i(j-1)}$. Hence, whenever a point p_j lies in the current valid region of p_i, we add the edge (p_i, p_j) to the shortcut graph.

Then, regardless of whether (p_i, p_j) is a valid shortcut or not, we first update the wedge $W_{i(j-1)}$ to an intermediate wedge W'_{ij} by computing the intersection between $W_{i(j-1)}$ and the local wedge D_{ij}. Afterwards, we update the intermediate wedge W'_{ij} and the wave front by the following operations. This update process is illustrated for the L_2 norm in Fig. 2 and for multiple steps and multiple norms in Fig. 1.

A valid shortcut $(p_ip_{k>j})$ in the Fréchet distance needs to go through the intersection region I between the current valid region (i.e., the wedge behind the wave front) and the unit circle c_j around p_j. Otherwise, the vertices of the subpolyline from p_i to p_k would be encountered in the wrong order contradicting the definition of the Fréchet distance. Hence, we narrow the wedge a second time such that the rays R_l and R_r emanating at p_i and enclosing I constitute the wedge W_{ij}; see Fig 2a. Note that this latter step is different to what Melkman and O'Rourke do because for the Hausdorff distance, it is irrelevant in which order the intermediate points skipped by a shortcut are encountered by the shortcut segment.

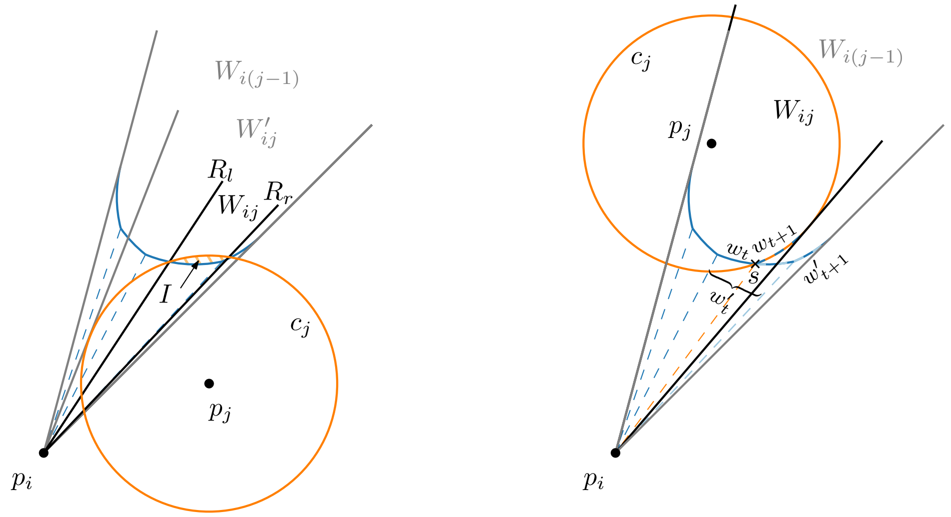


(a) In the L_1 norm, the unit circles are squares of side length $\sqrt{2}\varepsilon$ whose boundary is rotated by 45 degrees relative to the coordinate axes. The wave front consists of one or two line segments.

(b) In the L_2 norm, the unit circles are circles of radius ε. The wave front consists of $\mathcal{O}(n)$ circular arcs.

(c) In the L_∞ norm, the unit circles are squares of side length 2ε whose boundary is parallel to the coordinate axes. The wave front consists of one or two line segments.

Figure 1 Iterative construction of the wedge in the L_1, L_2 and L_∞ norm: From left to right, the local wedges D_{12}, D_{13}, and D_{14} are visualized in pink. Here, the intersection of local wedges defines the wedges W_{12}, W_{13}, and W_{14}, respectively. Additionally, we use the wave front, which is a sequence of unit circle arcs – here depicted in blue. Within the wedge and above the wave front, there is the valid region (depicted in green). This is the area, where a subsequent vertex p_j needs to lie if there is a valid shortcut (p_1, p_j). For example, (p_1, p_5) is a valid shortcut in the L_∞ norm, whereas in the L_1 and L_2 norm it is not.



(a) When encountering p_j, we update the wedge in two steps – even if p_j lies outside the wedge.

(b) Vertex p_j contributes an arc to the new wave front. Here, (p_i, p_j) is also a valid shortcut.

Figure 2 Updating the wedge and its wave front in the L_2 norm.

Guibas et al. [5] also apply this extra narrowing step, but in a different polyline simplification setting than ours.

Then, we update the wave front as Melkman and O'Rourke do. The parts of the bottom arc of the unit circle c_j around p_j within the local wedge D_{ij} that are above the current wave front are included to the new wave front. For an example see Fig 2b. There, we compute the intersection point s between c_j and the wave front and replace the arcs w'_t and w'_{t+1} of the wave front by the arcs w_t (which is a part of w'_t) and w_{t+1} (which is a part of c_j). There can be up to two intersection points between c_j and the wave front. If the valid region becomes empty by one of these update operations, we abort the search for further shortcuts from p_i.

3.2 Correctness

To show correctness, we need to argue that any shortcut (p_i, p_j) found by our algorithm is valid and that all valid shortcuts are identified.

For the former, we argue that we can always find an ordered mapping of the vertices $p_i, p_{i+1}, \dots, p_j$ onto points on $\overline{p_i p_j}$ that are also part of the wave front and hence at most at distance ε by definition. These points are the intersection points of $\overline{p_i p_j}$ with all wave fronts of the previous steps, i.e., the wave fronts of the wedges $W_{i(i+1)}, W_{i(i+2)}, \dots, W_{i(j-1)}$.

For the latter, we show that for each point p_j not in the valid region of $W_{i(j-1)}$, the Fréchet distance of $\overline{p_i p_j}$ to the subpolyline $p_i, p_{i+1}, \dots, p_j$ exceeds ε. If p_j lies outside the wedge $W_{i(j-1)}$, then $\overline{p_i p_j}$ does not intersect a unit circle of some vertex $p_{k \in \{i+1,\dots,j-1\}}$ or encounters the vertices $p_{i+1}, \dots, p_{j-1}$ in the wrong order. If p_j lies inside the wedge $W_{i(j-1)}$ but below the wave front, then there is some arc on the wave front that belongs to a unit circle of some vertex p_k that has distance greater than ε to all points of the line segment $\overline{p_i p_j}$.

3.3 L_2 and $L_{p \in (1,\infty)}$ Norm

In L_2, there are at most $\mathcal{O}(n)$ arcs on the wave front [7]. However, this means we cannot iterate over all arcs in each step since this would again require cubic time in total. Similar to

Melkman and O'Rourke, we use a balanced binary search tree to locate a point p_j relative to the wave front and to update the wave front. Note that we have an additional update operation where we determine the intersection region I and potentially make the wedge narrower. To this end, we need to determine whether there is two, one or zero intersection points between the wave front and a unit circle and where they lie in logarithmic time. We describe how to achieve this in our arXiv version [8][3] and we conclude the following theorem.

▶ **Theorem 3.1.** *An n-vertex polyline can be simplified optimally under the local Fréchet distance in the L_2 norm in $\mathcal{O}(n^2 \log n)$ time.*

We can generalize this approach to $\mathrm{L}_{p\in(1,\infty)}$ under the assumption that we can find the intersection between a line and a unit circle and between two unit circles in constant time.

3.4 L$_1$ and L$_\infty$ Norm

In the L_1 and the L_∞ norm, the unit circles are squares. Thus, the wave front consists of a sequence of orthogonal line segments. Here, we assume that all line segments are horizontal and vertical as in the L_∞ norm. In the L_1 norm it is the same but rotated by 45 degrees.

We inductively show that the wave front consists of at most two line segments – a horizontal and a vertical line segment. When we consider p_{i+1}, the wave front is first constructed as the bottom arc of the unit circle around p_{i+1} within the local wedge $D_{i(i+1)}$. This is just one or two horizontal or vertical line segments. Now assume that the wave front of $W_{i(j-1)}$ is given and we now consider p_j. To construct the wave front of W_{ij}, we need to compute the intersection between the wave front of $W_{i(j-1)}$ and the bottom arc of the unit circle around p_j. Both is just one or two horizontal or vertical line segments and the valid region of W_{ij} is above both vertical and to the same side of both horizontal line segments. Hence, the new wave front is determined by one of the horizontal and one of the vertical line segments – again at most two line segments.

Since the wave front has size ≤ 2, we can check containment in the valid region in constant time and we can update the wave front in constant time. This yields the following theorem.

▶ **Theorem 3.2.** *An n-vertex polyline can be simplified optimally under the local Fréchet distance in the L_1 and L_∞ norm in $\mathcal{O}(n^2)$ time.*

4 Conclusions and Future Work

We proposed a method how to simplify polylines under the local Fréchet distance optimally based on the concept of wave fronts that matches the running time bounds under the (conceptually simpler) local Hausdorff distance up to logarithmic factors. It would be interesting to implement our algorithm and to evaluate its running time in practice. In the L_2 norm, the theoretical analysis implies that the wave front might have linear complexity. However, for a large wave front to arise, the polyline vertices need to form a specific pattern. Since it is unlikely that such patterns occur naturally, we actually expect the wave front to have constant size in practice, resulting in $\mathcal{O}(n^2)$ time also for L_2.

[3] Guibas et al. [5] also do this operation, but they do not specify how to accomplish a runtime of $\mathcal{O}(\log n)$.

References

1 Pankaj K. Agarwal, Sariel Har-Peled, Nabil H. Mustafa, and Yusu Wang. Near-linear time approximation algorithms for curve simplification. *Algorithmica*, 42(3):203–219, 2005. doi:10.1007/s00453-005-1165-y.

2 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry and Applications*, 5:75–91, 1995. doi:10.1142/S0218195995000064.

3 W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *International Journal of Computational Geometry and Applications*, 6(1):59–77, 1996. doi:10.1142/S0218195996000058.

4 Michael Godau. A natural metric for curves – computing the distance for polygonal chains and approximation algorithms. In *Proc. 8th Annual Symposium on Theoretical Aspects of Computer Science (STACS'91)*, pages 127–136, 1991. doi:10.1007/BFb0020793.

5 Leonidas J. Guibas, John Hershberger, Joseph S. B. Mitchell, and Jack Snoeyink. Approximating polygons and subdivisions with minimum link paths. *International Journal of Computational Geometry and Applications*, 3(4):383–415, 1993. doi:10.1142/S0218195993000257.

6 Hiroshi Imai and Masao Iri. Polygonal approximations of a curve – formulations and algorithms. In Godfried T. Toussaint, editor, *Computational Morphology*, volume 6 of *Machine Intelligence and Pattern Recognition*, pages 71–86. North-Holland, 1988. doi:10.1016/B978-0-444-70467-2.50011-4.

7 Avraham Melkman and Joseph O'Rourke. On polygonal chain approximation. In Godfried T. Toussaint, editor, *Computational Morphology*, volume 6 of *Machine Intelligence and Pattern Recognition*, pages 87–95. North-Holland, 1988. doi:10.1016/B978-0-444-70467-2.50012-6.

8 Sabine Storandt and Johannes Zink. Polyline simplification under the local Fréchet distance has subcubic complexity in 2D. *arXiv preprint*, 2022. URL: https://arxiv.org/abs/2201.01344.

9 Marc J. van Kreveld, Maarten Löffler, and Lionov Wiratma. On optimal polyline simplification using the Hausdorff and Fréchet distance. *Journal of Computational Geometry*, 11(1):1–25, 2020. doi:10.20382/jocg.v11i1a1.

Compacting Squares: Input-Sensitive In-Place Reconfiguration of Sliding Squares[*]

Hugo A. Akitaya[1], Erik D. Demaine[2], Matias Korman[3], Irina Kostitsyna[4], Irene Parada[5], Willem Sonke[4], Bettina Speckmann[4], Ryuhei Uehara[6], and Jules Wulms[7]

1 **University of Massachusetts Lowell, USA**
hugo_akitaya@uml.edu

2 **Massachusetts Institute of Technology, USA**
edemaine@mit.edu

3 **Siemens Electronic Design Automation, USA**
matias_korman@mentor.com

4 **TU Eindhoven, The Netherlands**
[i.kostitsyna, w.m.sonke, b.speckmann]@tue.nl

5 **Technical University of Denmark, Denmark**
irmde@dtu.dk

6 **JAIST, Japan**
uehara@jaist.ac.jp

7 **TU Wien, Austria**
jwulms@ac.tuwien.ac.at

Abstract

Edge-connected configurations of square modules, which can reconfigure through so-called sliding moves, are a well-established theoretical model for modular robots in two dimensions. Dumitrescu and Pach [Graphs and Combinatorics, 2006] proved that it is always possible to reconfigure one such configuration of n squares into any other using $O(n^2)$ sliding moves, while maintaining connectivity.

For certain pairs of configurations, reconfiguration may require $\Omega(n^2)$ sliding moves. However, significantly fewer moves may be sufficient. We present Gather&Compact, an input-sensitive in-place algorithm that requires only $O(\bar{P}n)$ sliding moves to transform one configuration into the other, where $\bar{P}$ is the maximum perimeter of the two bounding boxes. Our algorithm is built on the basic principle that well-connected components of modular robots can be transformed efficiently. Hence we iteratively increase the connectivity within a configuration, to finally make it xy-monotone.

We implemented Gather&Compact and compared it experimentally to the in-place modification by Moreno and Sacristán [EuroCG 2020] of the Dumitrescu and Pach algorithm (MSDP). Our experiments show that Gather&Compact consistently outperforms MSDP by a significant margin.

1 Introduction

Self-reconfigurable modular robots [13] promise adaptive, robust, scalable, and cheap solutions in a wide range of technological areas, from aerospace engineering to medicine. Modular robots are envisioned to consist of identical building blocks arranged in a lattice and are intended to be highly versatile, due to their ability to reconfigure into arbitrary forms. An actual realization of this vision depends on fast and reliable reconfiguration algorithms, which hence have become an area of growing interest.

[*] JW was partially supported by the Austrian Science Fund (FWF) under grant P31119 and by the Vienna Science and Technology Fund (WWTF) under grant ICT19-035. IP was supported by Independent Research Fund Denmark grant 2020-2023 (9131-00044B) "Dynamic Network Analysis".

One of the best-studied paradigms of modular robots is the *sliding cube model* [6]. In this model, a robot *configuration* is a face-connected set of cubic modules on the cubic grid. The cubes can perform two types of moves, illustrated in two dimensions in Figure 1.

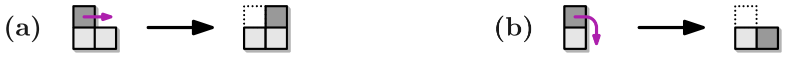


Figure 1 Moves admitted by the sliding cube model: **(a)** slide, **(b)** convex transition.

First, a module can *slide* along two face-adjacent cubes to reach a face-adjacent empty grid cell. Second, a module m can make a *convex transition* around a module m' to end in a vertex-adjacent empty grid cell. For this second move to be feasible, also the grid cell (not occupied by m') face-adjacent to both the starting and the ending positions must be empty. There are several prototypes of modular robots that realize the sliding cube model in 2D [3, 4, 7]. Units of multiple other prototypes, including expandable and contractible units [11, 12] as well as large classes of modular robots [2, 10], can be arranged into cubic *meta-modules* consisting of several units such that the meta-module can perform slide and convex transition moves. Thus, algorithmic solutions in the sliding cube model can be applied to modular robot systems realizing other models.

We study the reconfiguration problem for the 2D sliding cube model (the *sliding square model*): given two configurations of n unlabeled squares (each describing the relative positions of squares), compute a short sequence of moves that transforms one configuration into the other, while preserving edge-connectivity at all times. In Section 3, we present an algorithm for this problem, based on the "compact-and-deploy" approach. Using the basic principle that well-connected components of modular robots can be transformed efficiently, our algorithm iteratively increases the connectivity within a configuration, to arrive at a single solid xy-monotone component, before deploying it into the target configuration. Hence, our algorithm builds the target configuration in such a way that the lower left corner of the bounding boxes of both configurations are aligned. Our algorithm is *input-sensitive*: it requires only $O(\bar{P}n)$ sliding moves to transform one configuration into the other, where $\bar{P}$ is the maximum perimeter of the two bounding boxes. Our algorithm is also *in-place*: only one square at a time is allowed to move through cells edge-adjacent to the respective bounding box.

Comparison with existing algorithms. Dumitrescu and Pach [5] described an algorithm which transforms any two configurations of n squares into each other using $O(n^2)$ moves. This bound is worst-case optimal: there are pairs of configurations (a horizontal and a vertical line) which require $\Omega(n^2)$ moves for any transformation. The algorithm constructs a canonical shape from both input configurations. In the original paper this canonical shape is a strip that grows to the right of a right-most square and hence, necessarily, their algorithm always requires $\Theta(n^2)$ moves. Moreno and Sacristán [8, 9] modify Dumitrescu and Pach to be in-place; their canonical shape is a rectangle within the bounding box of the input. The in-place modification by Moreno and Sacristán of Dumitrescu and Pach (henceforth MSDP) has the potential to use fewer than $\Theta(n^2)$ moves in practice. Though, if configurations are tree-like (such as the spiral illustrated in Figure 2 left), then each square moves along all remaining squares, for a total of $\Omega(n^2)$ moves (see Figure 2 bottom row). However, the width and the height of this spiral configuration is $O(\sqrt{n})$. Our algorithm gathers $\Theta(\sqrt{n})$ squares from the end of the spiral and then compacts in a total of $O(n\sqrt{n})$ moves. In Section 4 we compare our Gather&Compact to MSDP experimentally; Gather&Compact consistently outperforms MSDP by a significant margin, on all types of square configurations.

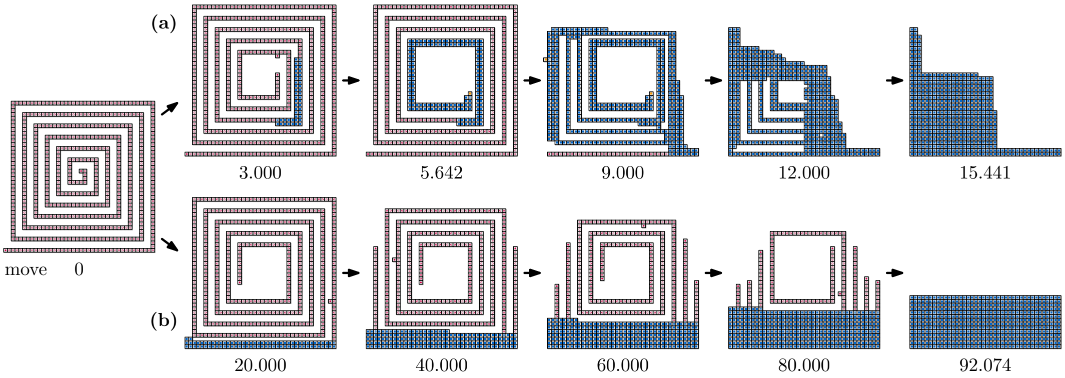


Figure 2 Spiral configuration in 40 × 40 bounding box. **(a)** Gather&Compact: gathering 5.642 moves; total 15.441 moves. **(b)** MSDP: 92.074 moves. Video: ▶ `https://tinyurl.com/algaspiral`

2 Preliminaries

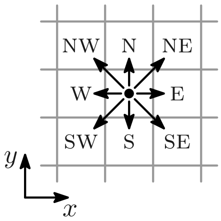
To describe our reconfiguration algorithm, we first need to introduce the following definitions and notations. Let $\mathcal{C}$ be an edge-adjacent configuration of squares on the square grid and let G be the *edge-adjacency graph* of $\mathcal{C}$. In G each node represents a square and two nodes are connected by an edge, if the corresponding squares are edge-adjacent. With slight abuse of notation we identify the squares and the nodes in the graph. A square $s \in \mathcal{C}$ is called a *cut square* if $\mathcal{C} \setminus \{s\}$ is disconnected. Otherwise, s is called a *stable square* (see Figure 3a).

A *chunk* is any inclusion-maximal set of squares in $\mathcal{C}$ bounded by (and including) a simple cycle in G (*boundary cycle*), including edge-adjacent squares of degree-1 in G (*loose squares*).

A *link* is a connected component of squares which are not in any chunk. A *connector* is a chunk square edge-adjacent to a square in a link or in another chunk. By definition a connector is a cut square. The *size* of a chunk C is the number of squares contained in C.

The *component tree* T of $\mathcal{C}$ has a vertex for each chunk or link and an edge (u, v) iff the chunks / links represented by u and v have edge-adjacent squares or share a square. The component tree is rooted at the component that contains the leftmost square in the bottom row, the *root square*, of $\mathcal{C}$. If a chunk is a leaf of T, we call it a *leaf chunk* (see Figure 3b).

A *hole* in $\mathcal{C}$ is a finite maximal vertex-connected set of empty grid cells. The infinite vertex-connected set of empty grid cells is the *outside*. If a chunk C encloses a hole in $\mathcal{C}$, we say that C is *fragile*, otherwise C is *solid*. The *boundary* of a hole H is the set of squares vertex-connected to any grid cell in H. The *boundary* of $\mathcal{C}$ is equal to the boundary of the outside. Note that the boundary of a hole is edge-connected.



Consider now the bounding box B of $\mathcal{C}$ on the square grid. We refer to the bottom-most left-most grid cell inside B as the *origin*. We say that P is the perimeter of B, and hence any square in $\mathcal{C}$ can be connected to the origin by an xy-monotone path of at most $P/2$ squares.

Let $c = (x, y)$ be a grid cell. We use compass directions (N, NE, E, etc.) to indicate neighbors of c. When we use grid coordinates, we assume the usual directions (the x-axis increases towards E and the y-axis increases towards N, so the N-neighbor of c is $(x, y + 1)$). Similarly, we indicate slide moves using compass directions ('a W-move') and convex transitions using a sequence of two compass directions ('a WS-move': a movement towards W followed by movement towards S).

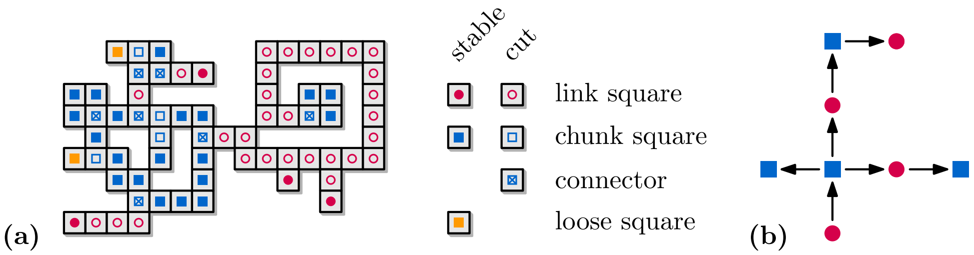


Figure 3 **(a)** A configuration $\mathcal{C}$. **(b)** The component tree T.

3 Input-sensitive in-place algorithm

In the first phase of our algorithm we ensure that the leaves of the component tree T are sufficiently large and well-connected. Specifically, we *gather* squares from the leaves of T until each leaf is a chunk of size at least P. To grow chunks, we use at most $O(P)$ moves for each square that was moved. During this process, the final position of each square is chosen inside bounding box B, but squares can move through the layer of grid cells adjacent to B.

After gathering, all leaves are *heavy chunks* of size at least P. Our goal is now to make each leaf chunk contain the origin, while ensuring that all squares remain part of their chunk (and thus never decreasing connectivity). A heavy chunk C contains a sufficient number of squares to be transformed into a chunk containing both the connector of C and the origin: we can connect the connector with the origin by an xy-monotone path of at most $P/2$ squares; two such paths, which are disjoint, form a new boundary cycle for C. We do not explicitly construct these two paths, but instead we *compact* the configuration by filling holes and using lexicographically monotone movement towards the origin for squares in heavy leaf chunks. Moves in this phase are *valid*, if they ensure that squares do not leave a chunk, and squares stay inside bounding box B (unless they are explicitly allowed to leave B).

During compacting each square in a leaf chunk makes primarily lexicographic monotone (LM-)moves towards the origin while staying inside B: S- and W-moves (slides), as well as SW-, WS-, NW-, and WN-moves (convex transitions). Figure 4 gives an overview of the types of compacting moves. In some cases, a square in the leftmost column or bottom row can exit B, and move along the bounding box to enter the same column/row closer to the origin. This is the only time a non-lexicographic monotone move is used, and every square can perform it at most $O(P)$ times. Hence the total number of moves during compacting is $O(Pn)$.

When compacting, every (heavy) leaf chunk will eventually contain a square at the origin. This means that the whole configuration becomes a single chunk, as all leaves of the component tree have merged into a single component. Therefore, once no valid moves can be applied anymore, we arrive at an xy-monotone configuration that fits inside B. If at any

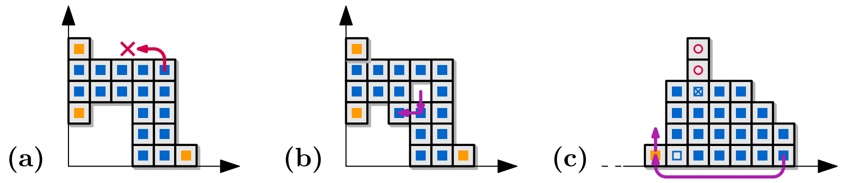


Figure 4 **(a)** An invalid default *LM-move*: this move splits a chunk. **(b)** A *corner move*: two slides extending a concave corner diagonally, performed as one atomic unit. **(c)** A *chain move*: sliding along the bounding box, possibly preceded by moving a single loose square.

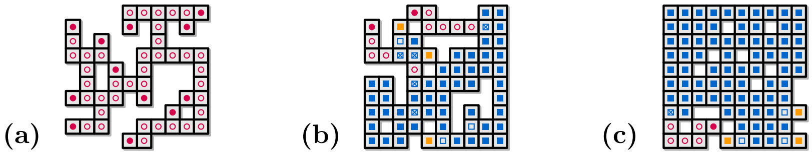


Figure 5 Example input instances on a 10 × 10 grid: density **(a)** 50 %; **(b)** 70 %; **(c)** 85 %.

point during this process the configuration becomes xy-monotone, then we simply stop. In particular, if the configuration is xy-monotone at the start, for example squares in only a single row or column, then we do not have to gather or compact, even though there are no heavy chunks. See the top row of Figure 2 for a visual impression of our algorithm.

In the special case that the input configuration $\mathcal{C}$ contains less than P squares, we first ensure that $\mathcal{C}$ contains the origin and then execute the gathering and compaction steps as before. The number of moves is trivially bounded by $O(Pn) = O(P^2)$.

Finally, we can convert any xy-monotone configuration into a different xy-monotone configuration with at most $O(\bar{P}n)$ moves, where $\bar{P}$ is the maximum perimeter of the bounding boxes of source and target configurations. Thus, since all moves are reversible, we can transform the source into the target configuration via this transformation.

▶ **Theorem 3.1.** *Let $\mathcal{C}$ and $\mathcal{C}'$ be two configurations of n squares each, let P and P' denote the perimeters of their respective bounding boxes, and let $\bar{P} = \max\{P, P'\}$. We can transform $\mathcal{C}$ into $\mathcal{C}'$ using at most $O(\bar{P}n)$ sliding moves while maintaining edge-connectivity at all times.*

Due to space constraints, all omitted proofs and details can be found in [1], along with an asymptotically tight lower bound and NP-completeness for minimizing the number of moves.

4 Experiments

We use square grids of sizes 10 × 10, 32 × 32, 55 × 55, 80 × 80, 100 × 100 for our experiments. The data sets for MSDP were created by hand and are not available.[1] We attempted to create meaningful data sets of the same nature by starting with a fully filled square grid and then removing varying percentages of squares while keeping the configuration connected. We arrived at three densities, namely (50 %, 70 %, 85 %), which arguably capture the different types of inputs for MSDP well [8, 9] (see Figure 5). For both algorithms we count moves

[1] V. Sacristán, personal communication, April 2021.

	Gather&Compact			**MSDP**		
D	50%	70%	85%	50%	70%	85%
10	237 31%	156 16%	95 8%	502 19%	427 21%	233 35%
32	5.395 4%	4.188 5%	2.529 8%	28.759 12%	18.447 13%	10.027 8%
55	25.916 2%	20.024 3%	12.124 4%	193.390 8%	116.431 12%	61.617 8%
80	77.745 2%	60.516 2%	36.395 3%	638.847 12%	344.529 9%	235.413 5%
100	150.666 1%	118.232 2%	69.488 3%	1.318.232 11%	743.133 17%	513.113 7%

Table 1 The number of moves for Gather&Compact and MSDP on various grid sizes ($D \times D$, such that $P = 4D$) and densities (in % of $D \times D$). Averages and standard deviations (in % of average) over 10 randomly generated instances are shown.

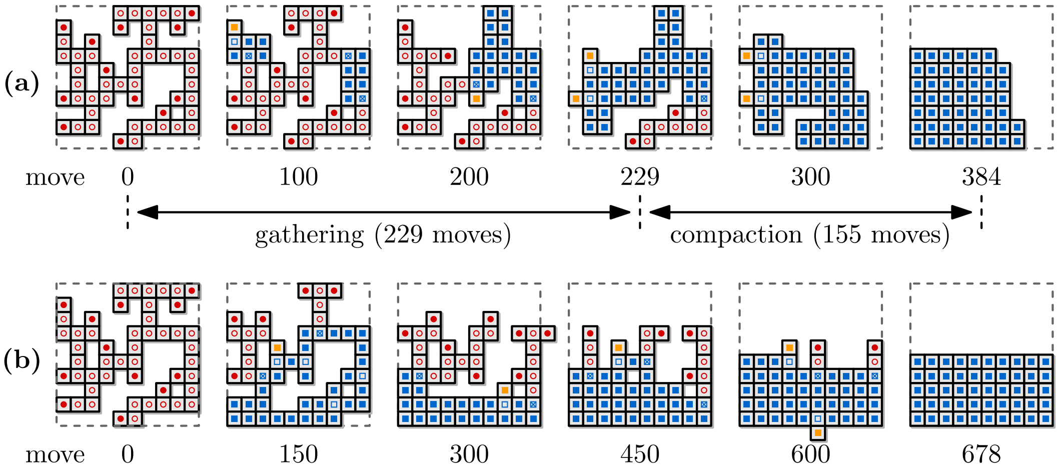


Figure 6 Execution of the two algorithms on one of the input instances for grid size 10 × 10, density 50 %. **(a)** Gather&Compact; **(b)** MSDP. Video: `https://tinyurl.com/alga10x10`

until they reach their respective canonical configurations. Our online material[2] contains our code for Gather&Compact, the input instances, and the adapted version of MSDP.

Table 1 summarizes our results and Figure 6 shows snapshots for both algorithms on a particular instance. We observe that Gather&Compact always uses significantly fewer moves than MSDP, even on high density instances where most squares are already in place. This is likely due to the fact that MSDP walks squares along the boundary of the configuration, while Gather&Compact shifts squares locally into better position. Figure 6b shows this behavior at move 600 where one can observe a square on its way along the bottom boundary.

References

1 Hugo A. Akitaya, Erik D. Demaine, Matias Korman, Irina Kostitsyna, Irene Parada, Willem Sonke, Bettina Speckmann, Ryuhei Uehara, and Jules Wulms. Compacting squares: Input-sensitive in-place reconfiguration of sliding squares. *CoRR*, abs/2105.07997, 2021. URL: `https://arxiv.org/abs/2105.07997`.

2 Greg Aloupis, Nadia Benbernou, Mirela Damian, Erik D. Demaine, Robin Flatland, John Iacono, and Stefanie Wuhrer. Efficient reconfiguration of lattice-based modular robots. *Computational Geometry: Theory and Applications*, 46(8):917–928, 2013. `doi:10.1016/j.comgeo.2013.03.004`.

3 Byoung Kwon An. EM-Cube: Cube-shaped, self-reconfigurable robots sliding on structure surfaces. In *Proc. 2008 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3149–3155, 2008. `doi:10.1109/ROBOT.2008.4543690`.

4 Chih-Jung Chiang and Gregory S. Chirikjian. Modular robot motion planning using similarity metrics. *Autonomous Robots*, 10:91–106, 2001. `doi:10.1023/A:1026552720914`.

5 Adrian Dumitrescu and János Pach. Pushing squares around. *Graphs and Combinatorics*, 22:37–50, 2006. `doi:10.1007/s00373-005-0640-1`.

6 Robert Fitch, Zack Butler, and Daniela Rus. Reconfiguration planning for heterogeneous self-reconfiguring robots. In *Proc. 2003 IEEE/RSJ International Conference on Intelligent Robots and System*, volume 3, pages 2460–2467, 2003. `doi:10.1109/IROS.2003.1249239`.

[2] `https://alga.win.tue.nl/software/compacting-squares/`

7 Kazuo Hosokawa, Takehito Tsujimori, Teruo Fujii, Hayato Kaetsu, Hajime Asama, Yoji Kuroda, and Isao Endo. Self-organizing collective robots with morphogenesis in a vertical plane. In *Proc. 1998 IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 2858–2863, 1998. `doi:10.1109/ROBOT.1998.680616`.

8 Joel Moreno. In-place reconfiguration of lattice-based modular robots. Bachelor's thesis, Universitat Politècnica de Catalunya, 2019.

9 Joel Moreno and Vera Sacristán. Reconfiguring sliding squares in-place by flooding. In *Proc. 36th European Workshop on Computational Geometry (EuroCG)*, pages 32:1–32:7, 2020.

10 Irene Parada, Vera Sacristán, and Rodrigo I. Silveira. A new meta-module design for efficient reconfiguration of modular robots. *Autonomous Robots*, 45(4):457–472, 2021. `doi:10.1007/s10514-021-09977-6`.

11 Daniela Rus and Marsette Vona. A physical implementation of the self-reconfiguring crystalline robot. In *Proc. 2000 IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1726–1733, 2000. `doi:10.1109/ROBOT.2000.844845`.

12 John W. Suh, Samuel B. Homans, and Mark Yim. Telecubes: mechanical design of a module for self-reconfigurable robotics. In *Proc. 2002 IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 4095–4101, 2002. `doi:10.1109/ROBOT.2002.1014385`.

13 Mark Yim, Wei-Min Shen, Behnam Salemi, Daniela Rus, Mark Moll, Hod Lipson, Eric Klavins, and Gregory S. Chirikjian. Modular self-reconfigurable robot systems. *IEEE Robotics & Automation Magazine*, 14(1):43–52, 2007. `doi:10.1109/MRA.2007.339623`.

The k-outlier Fréchet distance [*]

Maike Buchin[1] and Lukas Plätz[1]

1 **Faculty of Computer Science, Ruhr-Universität Bochum**
`{maike.buchin, lukas.plaetz}@rub.de`

Abstract

The Fréchet distance is a popular metric for curves; however, its bottleneck character is a disadvantage in many applications. Here we introduce two variants of the Fréchet distance to cope with this problem and expand the work on shortcut Fréchet distances. We present an efficient algorithm for computing the new distance measure.

1 Introduction

The analysis of curves is a growing field of study in computational geometry. The Fréchet distance is a popular metric between two curves. However, working with real-life data brings errors in measurement with it; and the Fréchet distance is quite sensitive to such outliers as it is defined as the largest distance in a minimal correspondence between the two curves. Hence outliers in the data may determine the Fréchet distance for the entire curves.

There are three different approaches to handle such error: the partially measured [5], the averaged [6, 12] and the shortcut Fréchet distance [10]. In this paper, we want to focus on the last one. Driemel and Har-Peled introduced the shortcut Fréchet distance. It allows replacing parts of one curve through straight line segments, called shortcuts. Deciding the shortcut Frechet distance between two curves was shown NP-hard by Buchin, Driemel and Speckmann in [7]. In the vertex restricted case, these line segments must start and end at vertexes. For this case Driemel and Har-Peled showed that the vertex restricted directed k-shortcut Fréchet distance, k counting the number of shortcuts, can be approximated in near-linear time. Buchin, Driemel and Speckmann gave an algorithm to decide the vertex restricted shortcut distance exactly in $O(n^3 \log n)$ time. An open problem is to extend the shortcut distance to allow shortcuts on both curves without completely short-cutting both curves. Avraham et al. considered the discrete problem [3] and faced the same problem for the two-sided case. To resolve it, they forbid simultaneous movement on the curves. In [1] Alt et al. discuss first how to define and compute the Fréchet distance between a curve and a graph, and then between two graphs. It introduces a free space surface for these, which is similar in spirit to the outlier free space we define here.

Some results on curve simplification are special cases of our problem in the sense that the directed outlier distance of a curve to itself is considered here. In [11] Imai and Iri for simplifying a curve introduce an associated graph to the curve. This is also called the shortcut graph of a curve, which we will later use in our algorithm. Bringmann and Chaudhury [4] considered curve simplification with vertex restricted shortcuts. They showed that this simplification needs $O(n^3)$ time. They also showed that for L_p with $p \neq 2, \infty$ this cannot be done in $O(n^{3-\varepsilon})$ for all $\varepsilon > 0$ unless the $\forall\forall\exists-$OV hypothesis fails. Kerkhof, Kostitsyna, Löffler, Mirzanezhad and Wenk [13] achieved the same runtime for their simplification algorithm.

[*] The work was supported by the PhD School "SecHuman - Security for Humans in Cyberspace" by the federal state of NRW.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

one-sided	shortcut	k-outlier
discrete	$O((n+m)^{6/5+\epsilon})$ [3]	naive $O(nmk \log n)$ [8]
continuous	NP-Hard [7] | vertex restricted $O(n^5 \log n)$ [9]	naive $O((n^2mk + nmk^2) \log n)$ [8]
two-sided	shortcut	k-outlier
discrete	$O((m^{2/3}n^{2/3} + m + n) \log^3(m+n))$ [3]	naive $O(nmk^2 \log n)$ [8]
continuous	–	$O((n^2mk + nmk^3) \log n)$ [Thm. 2.7]

Table 1 Computational complexity of the shortcut and outlier computation problem

We will address the two open problems of allowing shortcuts on both curves and taking into account the length of the shortcuts. For this, we present the k-outlier distances. These distances allow ignoring k outliers on one or both curves and computing the optimal Fréchet distance given the number of vertices to leave out. Switching from counting shortcuts to counting vertices makes it possible to compute a symmetrical shortcut distance. Further we allow other starting and ending points, hence it can also be seen as a partial Fréchet distance. Table 1 compares our result to previous results and naive algorithms (see our full paper [8]).

2 k-outlier Fréchet distance

2.1 Outlier free space cell

We start by defining curves, shortcuts and the Fréchet distance.

▶ **Definition 2.1.** Let $X = \langle p_0, p_1, \ldots, p_n \rangle$ be a polygonal curve. We consider X as a continuous map $X\colon [0,n] \to \mathbb{R}^d$, where $X(i) = p_i$ for $i \in \mathbb{N}$, and the i-th edge is linearly parametrized as $X(i+\lambda) = (1-\lambda)p_i + \lambda p_{i+1}$. We denote the **shortcut** $\langle p_i, p_a \rangle$ for $i < a$ as the straight line segment connecting the points.

A **reparametrisation** (σ, θ) of two curves X and Y is a pair of continuous non-decreasing surjective functions, where σ and θ map from $[0,1]$ to $[0,n]$ and $[0,m]$, respectively. The **Fréchet distance** between two polygonal curves X and Y is the maximum distance attained by optimal reparameterisations, i.e. $\mathrm{d}_{\mathcal{F}}(X,Y) := \inf_{(\sigma,\theta)} \max_t \|X(\sigma(t)) - Y(\theta(t))\|$. A reparametrisation of maximum distance at most ε is called an **ε-realization**.

Now we can define our outlier distances.

▶ **Definition 2.2** (Outlier distance)**.** A curve $\overline{X} := \langle \overline{p_1}, \overline{p_2}, \ldots, \overline{p_\ell} \rangle$ is in the set of **k-outlier curves** $C_k(X)$ if and only if the points $\langle \overline{p_1}, \overline{p_2}, \ldots, \overline{p_\ell} \rangle$ are a subsequence of $\langle p_1, p_2, \ldots, p_n \rangle$ and $n - \ell \leq k$. The **directed k-outlier Fréchet distance** $\mathrm{d}_{\mathcal{O}}(k, X, Y) := \min_{\overline{Y} \in C_k(Y)} \mathrm{d}_{\mathcal{F}}(X, \overline{Y})$.

The set of **k-outlier curve tuples** $T_k(X,Y) := \bigcup_{i=0}^{k} C_i(X) \times C_{k-i}(Y)$ contains those curves, where the number of outliers of both curves is at most k. The **undirected k-outlier Fréchet distance** is $\mathrm{d}_{\mathcal{T}}(k, X, Y) := \min_{(\overline{X}, \overline{Y}) \in T_k(X,Y)} \mathrm{d}_{\mathcal{F}} \left(\overline{X}, \overline{Y}\right)$.

Note that we do not (necessarily) restrict to start or end $\overline{X}$ with the same vertices as X. We will later see that allowing this does not increase the computation time and that the directed case is a specialization of the undirected case. Hence in the remainder of the paper, the k-outlier Fréchet distance will refer to the undirected case.

▶ Remark (Axioms of Metrics). The identity of indiscernibles does not hold for Fréchet curves because with the k-outlier Fréchet distances we cannot distinguish between two curves with k different points. The undirected case is symmetrical because the set of k-outlier curves tuples is symmetrical. The triangle inequality is not satisfied, this can be shown a counter-example. See figure 1. For this example even $\mathrm{d}_{\mathcal{T}}(k, X, Y) + \mathrm{d}_{\mathcal{T}}(k, Y, Z) \geq c\,\mathrm{d}_{\mathcal{T}}(2k, X, Z)$ is not satisfied.

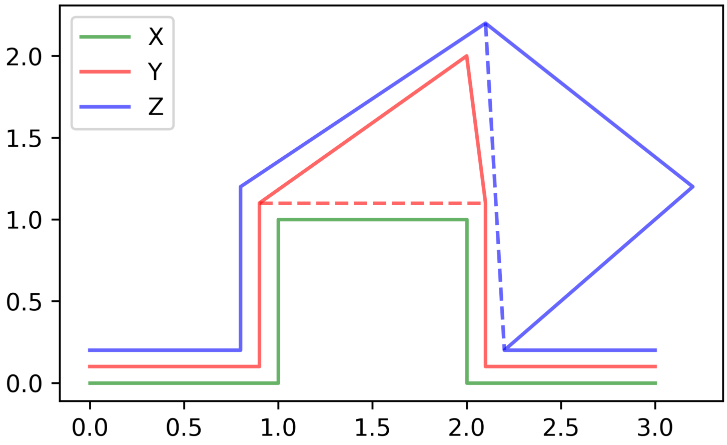


Figure 1 Example of Driemel and Har-Peled in [10]. X and Y are close under $\mathrm{d}_{\mathcal{S}}$ and $\mathrm{d}_{\mathcal{T}}$ for $k = 1$. The same is true for Y and Z. For X and Z they are only close under $\mathrm{d}_{\mathcal{T}}$ for $k \geq 3$.

With the shortcuts, we introduce new edges on our curves and with them, in turn, new cells in our free space by allowing these shortcuts. To distinguish the terminology of the classical and the outlier Fréchet distance, we add the word outlier to the new terms.

We present a dynamic program to decide the k-outlier Fréchet distance. We introduce the height in the outlier free space to keep track of the number of outliers. We start by initialising the outlier free space with the starting points. Here we also allow omitting vertices at the beginning of both curves, with the number of omitted points defining the starting height. Then we compute the reachable intervals for each outlier cell with the starting point and the previously computed reachable outlier intervals. Here every cell defines how much height has to be added to the result. After the computation of an outlier cell, we have the classical reachable free space interval of that cell. But to compute the outlier interval we need all adjacent cells. We combine then all reachable intervals into the reachable outlier intervals. In the end, we check if any ending point is reachable. They are similar to the starting points but allow to use less than k outliers, too.

▶ **Definition 2.3** (Outlier Cell). An **outlier cell** $C[(i,j),(a,b),h]$ is defined as the classical free space cell of the edges $\langle p_i, p_a\rangle$ and $\langle p_j, p_b\rangle$ and with an added **height** h. See 2 for a visualisation. With $R^h(C)$ and $R^v(C)$ we describe the **horizontal** and **vertical reachable interval** of an outlier cell C. The **length** L of a shortcut $\langle p_i, p_a\rangle$ with $a > i$ is defined as $a - i - 1$. It counts the vertices skipped by the shortcut. For the special case $a = i$ we set the length to 0. See 3 for an example. The **horizontal length** $\mathrm{L}_h(C[(i,j),(a,b),h]) := \mathrm{L}(\langle p_i, p_a\rangle)$ and the **vertical length** $\mathrm{L}_v(C[(i,j),(a,b),h]) := \mathrm{L}(\langle p_j, p_b\rangle)$. A **starting point** (i,j) of an outlier curve tuple is the pair of its first vertices. The height of that point in the outlier free space is simply the sum of both indices. An **ending point** $(n-i, m-j)$ is a pair of the last vertices of an outlier curve tuple. The height of this point has to be below $k - i - j$ to allow the last points to be left out. Furthermore a **vertical** and **horizontal reachable outlier interval** $I[(a,j),(a,b),h] := \bigcup_{l\in[k+1]} R^v(C[(a-l,j),(a,b),h])$ and $I[(i,b),(a,b),h] := \bigcup_{l\in[k+1]} R^h(C[(i,b-l),(a,b),h])$. The **horizontal free space interval** $F[(i,b),(a,b)] := \{p \in [i,a] \times \{b\} \mid \|X(x_p) - Y(y_p)\| \leq \varepsilon\}$ only depends on the edge $\langle p_i, p_a \rangle$ and the point p_j. The similar is true for the **vertical free space interval** $F[(a,j),(a,b)] := \{p \in \{a\} \times [j,b] \mid \|X(x_p) - Y(y_p)\| \leq \varepsilon\}$ We indicate by using an index twice that this is defined by a vertex rather than an edge. An **outlier point** in the free space diagram is defined as $P[(i,j),h] := I[(i,j),(a,j),h] \cap I[(i,j),(i,b),h]$.

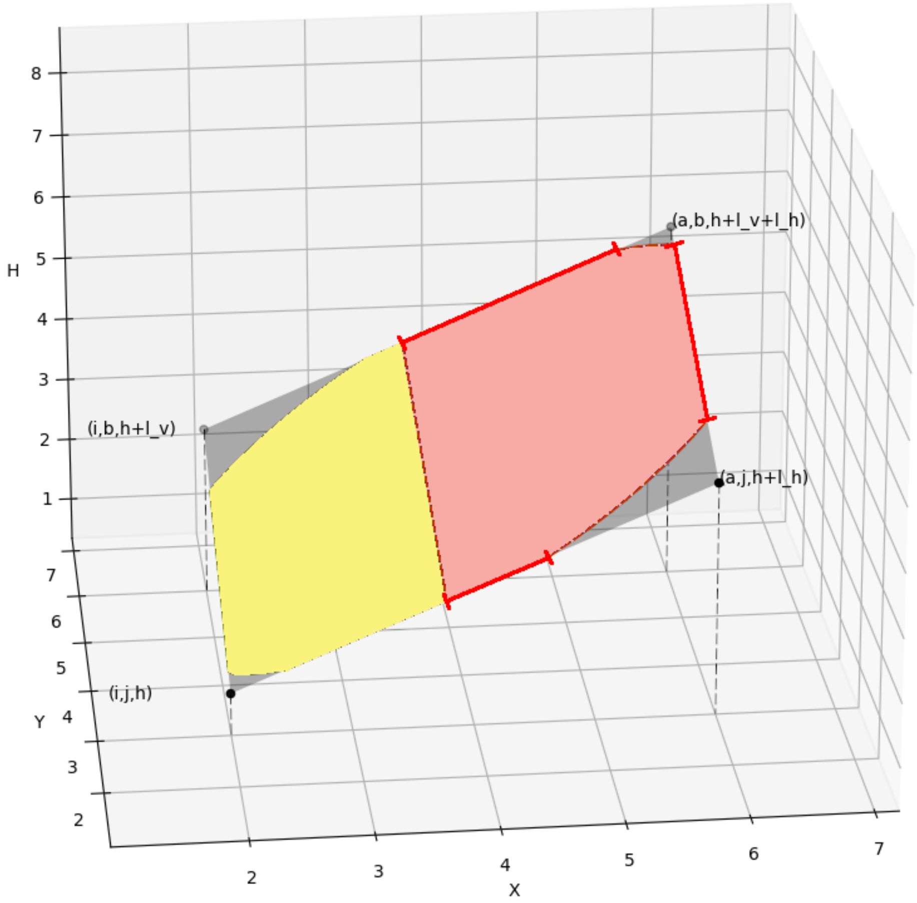


Figure 2 We visualise an outlier free space cell in 3-dimensional space. The four points of the cell $C[(i,j),(a,b),h]$ are $(i,j,h), (i,b,h+\mathrm{L}_v), (a,j,h+\mathrm{L}_h)$ and $(a,b,h+\mathrm{L}_v+\mathrm{L}_h)$. The lower left is at the same height as the cell. The upper left is lifted by the vertical length L_v, the lower right by the horizontal length L_h and the upper right by both lengths. The reachable free space is shown in red.

Figures 4 to 6 show examples of two classical and an outlier free space diagram.

2.2 Algorithm

For an easier description, we will introduce the term predecessor. Here and in the following we use $[n] := \{0, \ldots, n\}$ for $n \in \mathbb{N}$ as a shorthand.

▶ **Definition 2.4.** The **predecessors** of a cell $C[(i,j),(a,b),h]$ are the outlier intervals $I[(i,j),(a,j),h]$ and $I[(i,j),(i,b),h]$. The predecessors of an outlier interval $I[(i,j),(i,b),h]$ and $I[(i,j),(a,j),h]$ are the cells $C[(i-l-1,j),(i,b),h-l]$ and $C[(i,j-l-1),(a,j),h-l]$ for $l \in [k]$ respectively. The predecessors of a point $P[(i,j),h]$ are the outlier intervals $I[(i-l-1,j),(i,j),h-l]$ and $I[(i,j-l-1),(i,j),h-l]$ for $l \in [k]$.

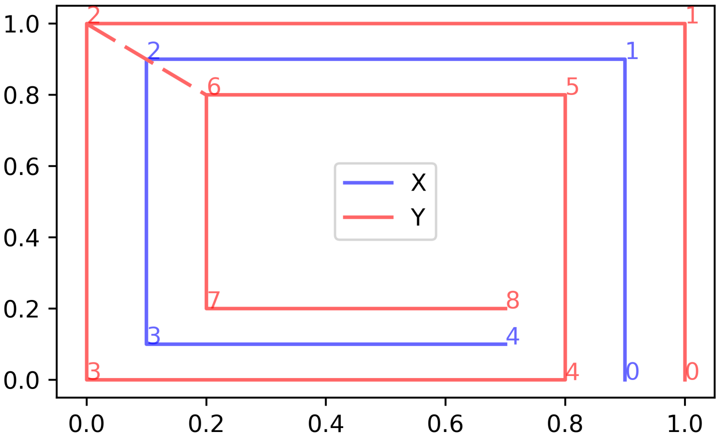


Figure 3 Two curves with a possible shortcut (dashed) on the red curve of length 3.

In the following, we only consider curves with at least one edge and $k \geq 1$. With the terminology introduced in the previous section, we can decide the reachability of a point with the predecessors of the point. To compute an outlier interval applying two times its predecessors gives us a dependence only on $O(k)$ intervals.

We start Algorithm 1 by initializing our reachable free space in the first loop and placing the starting points in the second loop. Then we can enforce the correct ordering of cells with a lexicographical ordering of the cells by their five identifying values. In this ordering, we compute the reachable outlier intervals in the third loop. After computing all cells we have to check in the last loop if any of the ending points are in the reachable free space below the needed height. We test if an ending point is reachable by computing the reachable outlier interval ending with that point and accounting for the length of that interval.

```
Algorithm 1: Algorithm($(k, X, Y, \varepsilon)$)

// Initialize Free Space
foreach $(i, j, l, h) \in [n] \times [m] \times [k+1] \times [k]$ do          // Outlier Intervals
    Compute $F[(i,j),(i+l,j)]$ and $F[(i,j),(i,j+l)]$
    Set $I[(i,j),(i+l,j),h], I[(i,j),(i,j+l),h]$ and $P[(i,j),h]$ as empty
// Adding Reachable Starting Points
for $(i,j) \in [k]^2$ with $i + j \leq k$ do                                 // Starting Points
    if $\|X(x_i) - Y(y_j)\| \leq \varepsilon$ then Set $(i,j) \in P[(i,j), i+j]$
// Computing the Reachable Space
foreach $(i, j, s, t, h) \in [n] \times [m] \times [k+1] \times [k+1] \times [k]$ do // Cell $C[(i,j),(a,b),h]$
    $a = i + s, b = j + t$ and $C = C[(i,j),(a,b),h]$
    Update $I[(i,b),(a,b),h + \mathrm{L}_v]$ and $I[(a,j),(a,b),h + \mathrm{L}_h]$ with $R^h(C)$ and $R^v(C)$
// Testing Ending Points
for $(i,j,h) \in [k]^3$ with $i + j \leq k, i + j + h \leq k$ do             // Ending Points
    if $(n - i, m - j) \in P[(n-i, m-j), h)]$ then return True
return False
```

▶ Remark. There are several possibilities to alter the algorithm to compute different distances. The first would be to set k to 0. Then the algorithm is the same as algorithm one of Alt and Godau in [2] and the runtime collapses to $O(nm)$.

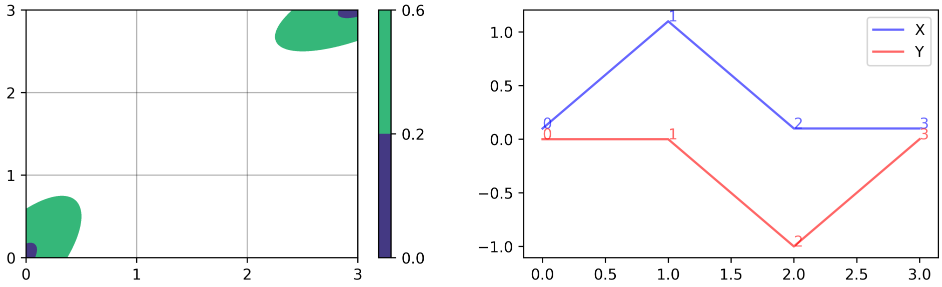


Figure 4 On the right side are the curves and on the left is the free space diagram for two values.

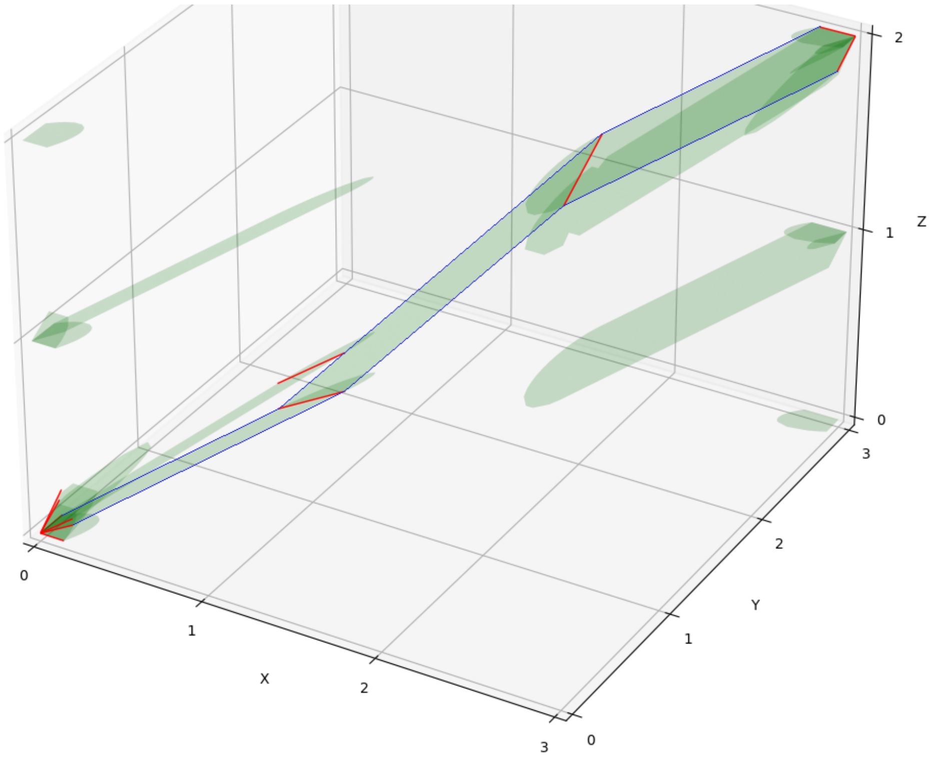


Figure 5 The outlier free space diagram for the same curves as in Figure 4. At the plane $z = 0$, we can see the classical free space diagram. The reachable free space intervals are drawn in red. The free space leading to an ending point of the curve is marked with a blue line.

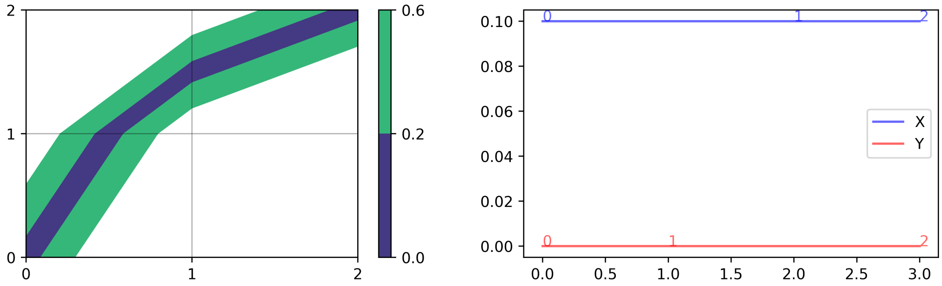


Figure 6 On the right side are the curves realizing the 2-outlier Fréchet distance of the curves in Figure 4. On the left is the classical free space of them.

The second would be to set a to $i+1$ instead of iterating over it in the third loop. With this, we forbid the omission of a vertex on the first curve. The runtime of the third loop becomes $O(nmk^2)$. Setting also the i to 0 in the second and fourth loop we disallow any changes to the first curve and hence we get the directed outlier Fréchet distance.

The third would be to also change the counting from points to shortcuts and only allow the starting and ending points $(0,0,0)$ and (n,m,h) for $h \in [k]$. Then the algorithm would decide the directed vertex restricted k-shortcut Fréchet distance with runtime $O(n^2mk)$.

For the correctness of the algorithm, we first show that reachability decides the k-outlier distance, see the full paper [8] for the proofs. The proof uses two ideas. The first idea is to have for every pair of matched edges in the reparametrisation of an outlier curve tuple an outlier cell representing the free space of it. The second idea is to only use cells below or at the height k. With these outlier cells and the outlier free space diagram we embedded the free space diagram of every outlier curve tuple in the outlier free space diagram and can count the number of outliers.

▶ Lemma 2.5. The k-outlier distance is at most ε if and only if an ending point is reachable in the outlier free space below or on height k.

▶ **Theorem 2.6.** *Algorithm 1 decides correctly if the k-outlier Fréchet distance is at most ε. The runtime of Algorithm 1 is in $O(nmk^3)$. The space usage is in $O(nmk^2)$.*

2.3 Critical values

We assume that $n \geq m$. The three types of critical values, Alt and Godau introduced, also cover all cases of the outlier free space. The first case covers the different starting and ending points in the outlier free space, of which there are $O(k^2)$ many. The second type describes the opening of free space intervals. A vertex and an edge define these. There are $O(nmk)$ critical values of this type for the $O(mk)$ edges and $O(n)$ points. The last type is the monotonicity event or the opening of a passage. Here two vertices and an edge are the defining features. So we get $O(n^2)$ for the vertices, $O(mk)$ for the edge and $O(n^2mk)$ in total of this type.

▶ **Theorem 2.7.** *The k-outlier Fréchet distance can be computed in $O((n^2mk + nmk^3)\log n)$.*

Proof. The argument is the same as in [2] by Alt and Godau. First, compute and sort the $O(n^2mk)$ critical values and then use the decision algorithm with the runtime of $O(nmk^3)$ in a binary search to find the k-outlier Fréchet distance. ◀

3 Conclusion

We introduced a new Fréchet variant to work with real-life data. We can decide the undirected k-outlier Fréchet distance of two curves of complexity m, n in $O(nmk^3)$ time and the directed distance in $O(nmk^2)$ time.

References

1 Helmut Alt, Alon Efrat, Günter Rote, and Carola Wenk. Matching planar maps. *Journal of Algorithms*, 49(2):262–283, 2003. doi:10.1016/S0196-6774(03)00085-3.

2 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 05(01n02):75–91, 1995. doi:10.1142/S0218195995000064.

3 Rinat Ben Avraham, Omrit Filtser, Haim Kaplan, Matthew J. Katz, and Micha Sharir. The discrete and semicontinuous Fréchet distance with shortcuts via approximate distance counting and selection. *ACM Trans. Algorithms*, 11(4), apr 2015. doi:10.1145/2700222.

4 Karl Bringmann and Bhaskar Ray Chaudhury. Polyline Simplification has Cubic Complexity. In *35th International Symposium on Computational Geometry (SoCG 2019)*, volume 129 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 18:1–18:16, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.SoCG.2019.18.

5 Kevin Buchin, Maike Buchin, and Yusu Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, page 645–654, USA, 2009. Society for Industrial and Applied Mathematics. URL: https://dl.acm.org/doi/abs/10.5555/1496770.1496841.

6 Maike Buchin. *On the Computability of the Frechet Distance Between Triangulated Surfaces*. PhD thesis, Freien Universität Berlin, 2007. URL: http://dx.doi.org/10.17169/refubium-6111.

7 Maike Buchin, Anne Driemel, and Bettina Speckmann. Computing the Fréchet distance with shortcuts is np-hard. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, SOCG'14, page 367–376, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2582112.2582144.

8 Maike Buchin and Lukas Plätz. The k-outlier Fréchet distance, 2022. arXiv:2202.12824.

9 A. Driemel. *Realistic analysis for algorithmic problems on geographical data*. PhD thesis, Eindhoven University of Technology, Utrecht University, Netherlands, 2013.

10 Anne Driemel and Sariel Har-Peled. Jaywalking your dog: Computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42(5):1830–1866, 2013. doi:10.1137/120865112.

11 Hiroshi Imai and Masao Iri. Polygonal approximations of a curve — formulations and algorithms. In *Computational Morphology*, volume 6 of *Machine Intelligence and Pattern Recognition*, pages 71–86. North-Holland, 1988. doi:10.1016/B978-0-444-70467-2.50011-4.

12 Anil Maheshwari, Jörg-Rüdiger Sack, and Christian Scheffer. Approximating the integral Fréchet distance. *Computational Geometry*, 70-71:13–30, 2018. doi:10.1016/j.comgeo.2018.01.001.

13 Mees van de Kerkhof, Irina Kostitsyna, Maarten Löffler, Majid Mirzanezhad, and Carola Wenk. Global Curve Simplification. In *27th Annual European Symposium on Algorithms (ESA 2019)*, volume 144 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 67:1–67:14, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ESA.2019.67.

Continuous mean distance of a weighted graph (extended abstract)*

Delia Garijo[1], Alberto Márquez[2], and Rodrigo I. Silveira[3]

1,2 **Universidad de Sevilla, Spain**
`{dgarijo, almar}@us.es`
3 **Universitat Politécnica de Catalunya, Spain**
`rodrigo.silveira@upc.edu`

Abstract

Motivated by the study of geometric graphs, we study the *continuous mean distance* of a weighted graph, defined as the mean of the distances between all pairs of *points* on the edges of the graph. Despite being a natural generalization of the well-studied notion of mean distance, which only considers distances between vertices, this concept has been barely studied. We show that the continuous mean distance can be computed in time quadratic in the number of edges, present structural results allowing a faster computation for several classes of weighted graphs, and study the relation between the (discrete) mean distance and its continuous counterpart.

Related Version arXiv:2103.11676v1

1 Introduction

The *mean distance* of a connected unweighted graph $G = (V(G), E(G))$ was first introduced by March and Steadman [6, Chap.14] in the context of architecture to compare floor plans, although interest in the concept dates back to the work of Wiener in chemistry [10] (after whom the closely related *Wiener index*, the sum of all pairwise distances in the graph, is named). The Wiener index has received extensive attention due to its many applications, most notably in chemistry [7], but also in other areas (e.g., [8]).

The most usual way to define the mean distance $\mu(G)$ is as the arithmetic mean of all nonzero distances between vertices, where distances are taken over all unordered pairs of vertices in the graph. In the context of graph theory, Doyle and Graver [1] were the first to propose the mean distance as a graph parameter. Since then, it has been intensively studied.

In a different direction, Doyle and Graver [2, 3] also introduced the mean distance of a *shape*, defined for any weighted graph embedded in the plane. Each edge of the graph is iteratively subdivided into shorter edges, so that the edge lengths approach zero. The mean distance of the shape is then defined as the limit of the mean distance of such a sequence of refinements. Doyle and Graver managed to compute its exact value for seven specific types of simple graphs (i.e., a path, a Y-shape, an H-shape, a cross, and three more) and six rather specific families of graphs; the most general ones being cycles and stars with k edges of length $1/k$. A summary of these formulas is given in [3].

In this paper we continue in this direction, studying in depth the mean distance of weighted graphs in a *continuous* setting, with the focus on its computational aspects. Our main motivation arises from *geometric graphs*. A geometric graph is an undirected graph where each vertex is a two-dimensional point, and each edge is a straight line segment between

* This work was supported by grants PID2019-104129GB-I00/ AEI/ 10.13039/501100011033, Gen. Cat. 2017SGR1640, and PID2019-103900GB-I00.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

the corresponding two points. Unlike abstract graphs, in geometric graphs distances are not only defined for pairs of vertices, but they exist for any two points on the graph, including points on the interior of the edges. Therefore, the concept of mean distance generalizes naturally to (weighted) geometric graphs, defined as the average distance between all *pairs of points* on the edges of the graph. Our main contributions are:

- We show that the continuous mean distance of a weighted graph with m edges can be computed in $O(m^2)$ time; see Section 3.
- We present structural results that allow a faster computation of the continuous mean distance for several classes of weighted graphs, see also Section 3.
- We study the relation between the discrete mean distance and the continuous counterpart, especially in terms of convergence, to understand when iteratively subdividing edges and computing the discrete mean distance converges to the continuous mean distance. See Section 4.

While all of our results apply to geometric graphs, we present them in the following for weighted graphs, making them slightly more general. Omitted proofs can be found in the full version of the paper [4].

2 Preliminaries

Let $G = (V(G), E(G))$ be a connected graph with n vertices and m edges; when no confusion may arise, we indistinctly write V or $V(G)$ and E or $E(G)$. Consider a function $\omega : E \longrightarrow \mathbb{R}^+$ that assigns a positive weight $\omega(e)$ to each edge $e \in E$. The value $\omega(e)$ is called the *length* of edge e, and is also denoted by $|e|$. In general, given a subset of edges $E' \subseteq E$, its *weight* or *length* is $|E'| = \sum_{e \in E'} \omega(e)$.

Graph G together with function ω is a weighted graph where every edge can be identified with a segment of length $\omega(e)$ in the Euclidean plane. Every point p on an edge $e = uv$ can be expressed as $p = \lambda_p v + (1 - \lambda_p)u$ for some $\lambda_p \in [0, 1]$. Let G_ℓ be the set of all points that are on the edges on G. We point out that all the graphs considered in this work are connected and weighted, although both terms will be in general omitted as it is understood from the context. We also consider *uniform* graphs: graphs where all edges have the same length; we write α-*uniform* to refer to a uniform graph where all edge lengths are α.

The *distance* $d(p, q)$ between p and q on G_ℓ is the length of a shortest path connecting the two points. In this work, we shall assume that the distance between the two endpoints of any edge e is $|e|$. The set of points G_ℓ together with this distance function is a metric space, and it will be treated indistinctly as a graph (with vertex set $V(G_\ell) = V(G)$ and edge set $E(G_\ell) = E(G)$) or as a closed point set. The distance between an edge $e = uv$ and a point $p \notin e$ is $d(p, e) = \min\{d(p, u), d(p, v)\}$, and the distance between two edges e and $e' = ab$ is $d(e, e') = \min\{d(a, e), d(b, e)\}$.

We begin by defining the variant of the *discrete mean distance* that we will consider in the remainder of this work, which differs from the one mentioned in the Introduction in two aspects: (i) it considers *all* pairs of distances, including those that are zero, and (ii) it considers *ordered* pairs of vertices (thus it includes a multiplicative factor 2):

$$\mu_d(G) = \frac{2\sum_{u,v \in V} d(u, v)}{n^2} = \frac{2W(G)}{n^2},$$

where $W(G)$ denotes the Wiener index of G. Observe that $\mu_d(G)$ is the arithmetic mean of the entries of the distance matrix of the graph. This alternative form of mean distance has been considered before [9], and allows us to analyze its convergence to the continuous mean distance when iteratively subdividing the edges of the graph.

To define formally the continuous mean distance of a weighted graph, we start by defining it between two subsets of edges $E', E'' \subseteq E(G_\ell)$ as

$$\mu_c(E', E'') = \frac{1}{|E'||E''|} \int \int_{p \in E' \; q \in E''} d(p,q)\, dp\, dq.$$

With some abuse of notation, we shall write $\mu_c(G', G'')$, where G' and G'' are the graphs with edge sets E' and E'', respectively; when the edge sets are single edges e and e', we use $\mu_c(e, e')$. In addition, $|E'|$ shall also be called the *edge weight* of G' (analogous for G'').

The *continuous mean distance* of the graph G_ℓ can then be defined as

$$\mu_c(G_\ell) = \mu_c(E(G_\ell), E(G_\ell))$$

An example: paths. The discrete mean distance of an α-uniform path P is known to be $\mu_d(P) = \alpha(n^2 - 1)/3n$, for n vertices [4, 9]. For non-uniform paths P, there is no closed formula to compute $\mu_d(P)$. However, $\mu_c(P_\ell)$ can be easily shown to be $\frac{t}{3}$ for any path, where t is the total path length, using the fact that to compute μ_c any path can be considered as a single edge of length t [4].

3 Computation of the continuous mean distance

The continuous nature of the continuous mean distance makes its computation nontrivial. In this section we show that $\mu_c(G_\ell)$ can be computed rather efficiently, in time quadratic in the number of edges of G_ℓ. We have proved this result by two different methods, which apply fundamental concepts in discrete algorithms and computational geometry: that of shortest path trees and that of Voronoi diagrams for the L_1 metric. We sketch here the latter and only mention briefly the other method; the reader may consult the full version of this work [4] for a precise description of both methods.

First we observe that $\mu_c(G_\ell)$ can be obtained as a weighted sum of the continuous mean distances of all *ordered* pairs of edges (this is simply a consequence of elementary properties of integration):

$$\mu_c(G_\ell) = \frac{1}{|E|^2} \left(\sum_{e,e' \in E \times E, e \neq e'} \mu_c(e, e')|e||e'| + \sum_{e \in E} \frac{|e|}{3}|e|^2 \right) \tag{1}$$

Our approach, which is based on well-known geometric tools, shows that the mean distance between two edges and, therefore, of the whole graph can be computed using simple geometric arguments. The *lower envelope* of a set of functions is the function resulting of taking the point-wise minimum of all functions in the set. A first step is to prove that for any two edges $e, e' \in E(G_\ell)$, the distance between a point $p \in e$ and a point $q \in e'$ can be seen as the lower envelope of at most four planes in 3D. This already implies that $\mu_c(e, e')$ can be computed in constant time. However, we can give a direct way to compute it considering the volume of a three-dimensional body with a rectangular base (one side with the length of e and the other with the length of e'), four vertical faces from each of the four base edges, and a *roof* that is the lower envelope mentioned above. This allows us to show that the function $\mu_c(e, e')$ can be expressed as a weighted volume of at most eight truncated rectangular prisms. This proves the following result.

▶ **Theorem 3.1.** *The continuous mean distance of a weighted graph G_ℓ with m edges can be computed in $O(m^2)$ time.*

In the full version of this work [4], we present an alternative proof of this theorem based on shortest path trees. Very briefly, we define a continuous version of the shortest path tree from one point, which can be computed within the same running time as the well-known discrete shortest path tree. Using this, a careful case analysis allows us to derive expressions for the mean distance between any two edges, which depend solely on distances stored in continuous shortest path trees.

In addition to the general methods presented above, we can compute the continuous mean distance faster for several special cases. This includes complete graphs and graphs that have cut vertices. In particular, for graph families that have a cut vertex, the continuous mean distance can be computed faster by solving each block recursively, and combining the mean distance of each block with weights proportional to the relative edge weight of each block with respect to the total edge weight of the graph.

▶ **Proposition 3.2.** *The continuous mean distance can be computed in $O(n)$ time for weighted trees and weighted cactus graphs with n vertices.*

Using the geometric method described earlier, we can give an exact expression for the continuous mean distance of the α-uniform complete graph K_n. While it is easy to prove $\mu_d(K_n) = (n-1)/n$, this is much harder in the continuous case.

▶ **Proposition 3.3.** *The continuous mean distance of the α-uniform complete graph K_n is given by*

$$\mu_c(K_n) = \frac{\alpha(9\,n^2 - 22\,n + 12)}{6\,(n^2 - n)}$$

Finally, we can prove that a relation between the continuous mean distance of stars, trees, and paths, known for the Wiener index [5] for the unweighted case (so, by definition, for the discrete mean distance), also holds in the continuous case when the corresponding graphs are uniform.

▶ **Proposition 3.4.** *Let S_ℓ and P_ℓ be an α-uniform star and α-uniform path, respectively, on n vertices. Then, $\mu_c(S_\ell) \le \mu_c(T_\ell) \le \mu_c(P_\ell)$ for every α-uniform tree T_ℓ with n vertices.*

4 Discrete versus continuous mean distances

There is no obvious relation between the discrete and the continuous mean distances, in the sense that for different graphs, any of these two values can be larger. For instance, μ_c is larger than μ_d for complete graphs (Proposition 3.3) and cycles[1] but it is smaller for paths (see the related example in Section 2). However, we can give bounds on the continuous mean distance of two edges in terms of discrete distances (and these bounds are tight).

▶ **Proposition 4.1.** *Let e and e' be two distinct edges in a weighted graph G. Then,*

$$d(e,e') + \frac{|e| + |e'|}{4} \le \mu_c(e,e') \le d(e,e') + \frac{|e| + |e'|}{2}$$

By means of equation (1), the previous result leads to bounds for $\mu_c(G_\ell)$ whenever G_ℓ is uniform, in terms of the discrete mean distance of a weighted version of its line graph [4, Corollary 5.1], but this method cannot be extrapolated to a graph in general. Thus, a natural

[1] The continuous mean distance of a 1-uniform cycle C_n of n vertices is $n/4$ [3], as well as $\mu_d(C_n)$ for n even; otherwise $\mu_d(C_n) = \frac{n}{4} - \frac{1}{4n}$ [9].

question is whether the discrete mean distance is convergent to its continuous counterpart when iteratively subdividing the edges.

One may propose different subdivision schemes, but not all of them guarantee convergence. By definition of continuous mean distance, the convergence happens for uniform graphs by simply adding, at each step, a new vertex—anywhere—on each edge. With the same scheme, we can show that the discrete mean distance is also convergent for nonuniform trees, although not necessarily to the continuous counterpart. Further, the convergence of μ_d to μ_c can be guaranteed for all graphs where the applied edge subdivision system satisfies that the ratio between the longest and the shortest edge, at an arbitrary step k, tends to 1. For instance, we may subdivide at each step k only the longest edges, say of length t, and those whose length is larger than $(1-1/k)t$. The problem is, however, that such a scheme completely depends on the original structure of the graph.

Next we present an edge subdivision scheme that does not depend on the graph structure, and allows us to obtain bounds on the discrete mean distance of its k-th edge subdivision, and on its limit when k tends to infinity. For a graph $G=(V,E)$ with n vertices and m edges, let $G^1=(V^1,E^1)$ be the graph that results from subdividing each edge of G by inserting a new vertex on its midpoint. Furthermore, we subdivide each edge of G^1 into 2^{k-1} new edges of the same length by inserting $2^{k-1}-1$ equidistant vertices; the resulting graph $G^k=(V^k,E^k)$ is called the *k-th subdivision of G*. See Figure 1 for a small example.

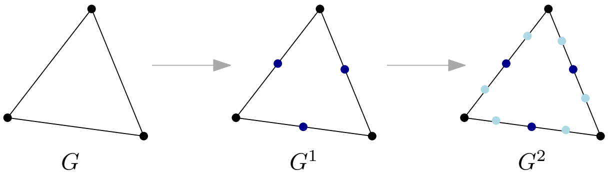


Figure 1 Example of the first two steps of our subdivision scheme.

By definition,

$$\mu_d(G^k)=\frac{2W(G^k)}{(n+m(2^k-1))^2}$$

where $W(G^k)=\sum_{u,v\in V^k} d(u,v)$. With some abuse of notation we write, for sets $A,B\subseteq V^k$, $W(A;B)=\sum_{u\in A,v\in B} d(u,v)$ and $W(A)=W(A;A)$. For this subdivision scheme we can show the following relation.

▶ **Theorem 4.2.** *Let $G=(V,E)$ be a weighted graph with n vertices and m edges, and let $G^k=(V^k,E^k)$ be its k-th subdivision. Let $\mathcal{B}$ be the set of vertices inserted in G to obtain G^1. For $k>1$ it holds that:*

$$\frac{2\left[\Omega(G,G^1)-\rho\left(3\binom{m}{2}+m(n-2)\right)\left(2^{k-2}-\frac{1}{2}\right)\right]}{(n+m(2^k-1))^2}<\mu_d(G^k)\leq\frac{2\,\Omega(G,G^1)}{(n+m(2^k-1))^2}$$

where $\rho=\max\{|e|\,:\,e\in E\}$, and

$$\Omega(G,G^1)=W(V^1)+(2^k-2)\left(2^kW(\mathcal{B})+W(\mathcal{B};V)\right)+|E|\left(\frac{2^{2k-1}}{3}-2^{k-1}+\frac{1}{3}\right)$$

Moreover, the upper bound is tight.

By simply taking limits in Theorem 4.2, we obtain the following bounds.

▶ **Corollary 4.3.** *Let $G = (V, E)$ be a weighted graph with m edges, and let $G^k = (V^k, E^k)$ be its k-th subdivision. Let $\mathcal{B}$ be the set of vertices inserted in G to obtain G^1. Then,*

$$\lim_{k\to\infty} \mu_d(G^k) \leq \mu_d(\mathcal{B}) + \frac{|E|}{3m^2}$$

Moreover, if G is α-uniform then, $\lim_{k\to\infty} \mu_d(G^k) = \mu_c(G_\ell) \leq \mu_d(\mathcal{B}) + \dfrac{\alpha}{3m}$.

We want to highlight that all trees attain the preceding upper bounds. However, for nonuniform graphs it is easy to construct cases where the subdivision of edges does not converge to the continuous counterpart. Consider, for example, a path P with 4 vertices and edge lengths $2, 1, 1$; we have $\mu_c(P_\ell) = 4/3 \approx 1.33$, whilst $\lim_{k\to\infty} \mu_d(P^k) = 5/8 + 4/27 \approx 0.77$.

Acknowledgments. We thank Julian Pfeifle for proposing the topic of this work, and for stimulating discussions.

References

1 J. K. Doyle and J. E. Graver. Mean distance in a graph. *Discrete Math.*, 17:147–154, 1977.

2 J. K. Doyle and J. E. Graver. Mean distance for shapes. *J. Graph Theory*, 6(4):453–471, 1982.

3 J. K. Doyle and J. E. Graver. A summary of results on mean distance in shapes. *Environment and Planning B: Planning and Design*, 9:177–179, 01 1982.

4 D. Garijo, A. Márquez, and R. I. Silveira. Continuous mean distance of a weighted graph, 2021. `arXiv:2103.11676`.

5 I. Gutman. A property of the Wiener number and its modifications. *Indian J. Chem.*, 36(A):128–132, 1997.

6 L. March and P. Steadman. *The geometry of environment.* Royal Institute of British Architects, London, 1971.

7 S. Nikolić, N. Trinajstić, and Z. Mihalić. The Wiener index: Development and applications. *Croat. Chem. Acta*, 68:105–129, 1995.

8 E. Otte and R. Rousseau. Social network analysis: A powerful strategy, also for the information sciences. *J. Inf. Sci.*, 28:441–453, 12 2002.

9 E. W. Weisstein. Mean distance. From MathWorld—A Wolfram Web Resource. Last visited on 16/11/2020. URL: `https://mathworld.wolfram.com/MeanDistance.html`.

10 H. Wiener. Structural determination of paraffin boiling points. *J. Am. Chem. Soc.*, 69(1):17–20, 1947.

Approximation of Minimum Convex Partition

Nicolas Grelier[1]

1 Department of Computer Science, ETH Zürich, Switzerland
nicolas.grelier@inf.ethz.ch

Abstract

We consider the Minimum Convex Partition problem: Given a set P of n points in the plane, draw a plane graph G on P, with positive minimum degree, such that G partitions the convex hull of P into a minimum number of convex faces. We present an $\mathcal{O}(\log OPT)$-approximation algorithm running in $\mathcal{O}(n^8)$-time, where OPT denotes the minimum number of convex faces needed. This result is obtained by relating the problem to the Covering Points with Non-Crossing Segments problem.

Related Version arXiv:1911.07697

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.

1 Introduction

The CG Challenge 2020 organised by Demaine, Fekete, Keldenich, Krupke and Mitchell [5], was about solving instances of *Minimum Convex Partition* (MCP).

▶ **Definition 1.1** (Demaine *et al.* [5]: Minimum Convex Partition problem)**.** Given a set P of n points in the plane. The objective is to compute a plane graph with vertex set P (with each point in P having positive degree) that partitions the convex hull of P into the smallest possible number of convex faces. Note that collinear points are allowed on face boundaries, so all internal angles of a face are at most π.

As explained by Bose *et al.*, this problem has applications in routing [3]. They showed that a routing algorithm named *Random-Compass* that works for triangulations can be extended to convex partitions. Having a convex partition with few faces reduces the amount of data to store. From now on, we denote by P a set of n points in the plane.

In this paper, we present an approximation algorithm for MCP. We obtain this approximation algorithm by relating the MCP problem to the *Covering Points with Non-Crossing Segments* (CPNCS) problem. First, we define what *non-crossing segments* are.

▶ **Definition 1.2** (Non-Crossing Segments)**.** We call a part of a (straight) line bounded by two points a *segment*. The two points are referred to as *endpoints* of the segment. Note that we do not force the endpoints to be distinct, therefore we consider a point p as being a segment. The endpoint of p is p itself. Two segments are *non-crossing* if the intersection of their relative interior is empty.

▶ **Definition 1.3** (Covering Points with Non-Crossing Segments)**.** Given a set P of n points, find a minimum number of non-crossing segments whose endpoints are in P such that each point of P is contained in at least one segment.

The condition that the endpoints of the segments must be in P has no effect on the number of segments required. We add it as it simplifies some arguments. Note that CPNCS is not a so-called *set cover problem* nor an *exact cover problem*. We believe that CPNCS is interesting in itself. Even though it is a very natural problem, to the best of our knowledge it had not been introduced before.

In the full version of the paper, we show that MCP is NP-hard [7]. This result was also presented at EuroCG 2020. Under the assumptions that the points lie on the boundaries of a fixed number h of nested convex hulls, and that no three points lie on a line, Fevens, Meijer and Rappaport gave an algorithm for solving MCP in time $\mathcal{O}(n^{3h+3})$ [6]. Some integer linear programming formulations of the problem have been recently introduced [2, 11, 4].

For the related problem *Minimum Convex Partition of Polygons with Holes*, Bandyapadhyay, Bhowmick and Varadarajan showed the existence of a $(1+\varepsilon)$-approximation algorithm running in time $n^{\mathcal{O}((\log n/\varepsilon)^4)}$ [1]. Although they only consider holes with non empty interior, one can observe that their proof extends to the case of point holes. This is an even more general setting than MCP for point sets, so their algorithm also applies in our setting. This implies that MCP is not APX-hard unless $NP \subseteq DTIME(2^{\text{polylog } n})$.

Under the assumption that no three points are collinear, Knauer and Spillner have shown a $\frac{30}{11}$-approximation algorithm [8] for MCP in 2006. As a lower bound on the number of convex faces for one particular point set, they rely on the observation that each inner point has degree at least 3. The *inner points* of P are the points not on the boundary of the convex hull. This gives a lower bound on the number of edges, and therefore on the number of faces, by Euler's formula. Note that the restriction that no three points are on a line is necessary, as shown in Figure 1. There are only two faces in a minimum convex partition of this point set, and all the inner points have degree 2.

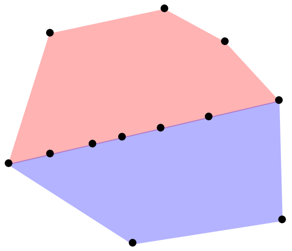


Figure 1 The number of inner points can be arbitrarily much larger than the number of convex faces required.

Additionally, Knauer and Spillner showed how to adapt any constructive upper bound on the number of faces into an approximation algorithm. More explicitly, they showed that if one can compute in polynomial time a convex partition with at most λn convex faces, then there exists a 2λ-approximation algorithm running in polynomial time. The best result to date is a proof by Sakai and Urrutia that one can partition a point set in quadratic time using at most $\frac{4}{3}n$ convex faces (the result was presented at the 7th JCCGG in 2009, the paper appeared on arXiv in 2019) [10]. Although they do not mention it, combining this result with the one by Knauer and Spillner gives a quadratic time $\frac{8}{3}$-approximation algorithm.

The lower bound used by Knauer and Spillner does not extend to our setting, where we consider all point sets. They say that a constant-approximation algorithm would be desirable for unrestricted point sets, but so far not even an $\mathcal{O}(n^{1-\varepsilon})$-approximation is known. In Section 3, we prove the following:

▶ **Theorem 1.4.** *There exist $\mathcal{O}(\log OPT)$-approximation algorithms for MCP and CPNCS running in $\mathcal{O}(n^8)$-time.*

Allowing several points to be on a line does not simply create tedious technicalities to deal with. The crux of the matter is to find, for a fixed point set, an exploitable lower bound on the number of faces in a minimum convex partition. When no three points are on a line, the number of inner points in P gives a linear lower bound on the number of faces in a convex partition [8]. In this paper, we consider point sets with no restriction. We introduce the CPNCS problem as it pinpoints where the difficulty of finding a constant-approximation algorithm for MCP is and makes the problem easier to study. We show in Section 2 the following:

▶ **Theorem 1.5.** *Let P be a set of n points with at least one inner point, and let $\lambda \geq 1$ be a real number. Let f_m denote the minimum number of faces in a convex partition of P. Let s_m denote the minimum number of non-crossing segments in a covering of the inner points of P, denoted by P_i.*

1. *It holds that $\frac{s_m}{6} \leq f_m \leq 8s_m$.*
2. *Given a covering of P_i with at most λs_m non-crossing segments, it is possible to compute in $\mathcal{O}(n^2)$-time a convex partition of P with at most $24\lambda f_m$ convex faces.*
3. *Given a convex partition of P with at most λf_m convex faces, it is possible to compute in $\mathcal{O}(n)$-time a covering of P_i with at most $44\lambda s_m$ non-crossing segments.*

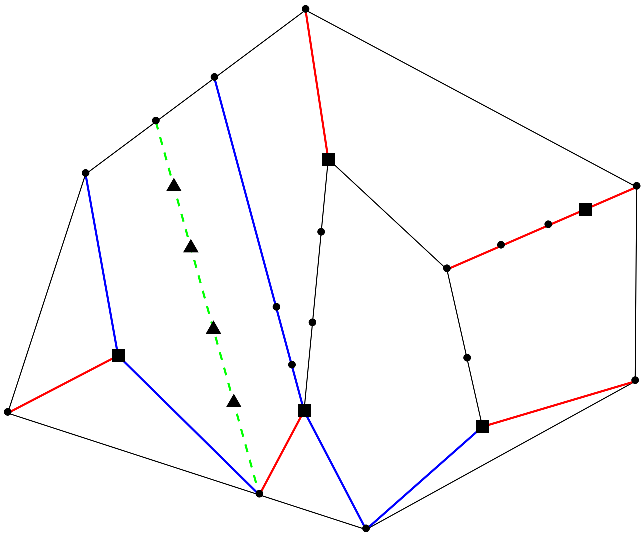


Figure 2 Illustration of Lemma 2.2. The green dashed edge and the triangle points are removed at the beginning for the analysis, and added back at the end. The extreme points in P'' are represented as square points. The edges in E' are in red. The other edges from P'' to the boundary of the convex hull are in blue.

2 The relation between MCP and CPNCS

Throughout this section, we denote by P a point set in the plane. We denote by P_i the set of inner points of P. Let p be in P. If P and $P \setminus \{p\}$ do not have the same convex hull, we say that p is an *extreme point*. We denote by P' the extreme points in P_i. Note that a point might lie on the boundary of the convex hull of a point set without being an extreme point. We say that P is *special* if $|P'| \leq 2$. The proof of Lemma 2.1 can be found in the full version of the paper [7].

▶ **Lemma 2.1.** *Let P be a set of n points that is not special. Given a covering of P_i with s non-crossing segments, one can compute in $\mathcal{O}(n^2)$-time a convex partition of P with at most $4s + 2|P'|$ faces.*

▶ **Lemma 2.2.** *Let P be a set of n points. Given a convex partition of P with f faces, one can compute in $\mathcal{O}(n)$-time a covering of P_i with at most $6f - 2|P'|$ non-crossing segments.*

Proof. The proof is illustrated in Figure 2. Let us denote by $G_0 = (V_0, E_0)$ the plane graph corresponding to the convex partition. Observe that the relative interior of an edge in E_0 might overlap with points in P. We assume that G_0 is given with a doubly connected edge list (DCEL) structure. If there is an edge between two points on the boundary of the convex hull of V_0, but not consecutive, we remove this edge. Note that this decreases the number of faces by 1, and does not break the convexity property. We denote by m the number of such edges that we have removed. We also remove from P all points contained in the relative interior of an edge between two points on the boundary of the convex hull. We denote by P'' the extreme points in P_i that we have not removed. As an edge contains at most two points in P', we have $|P''| \geq |P'| - 2m$. Using the DCEL structure, this can be done in $\mathcal{O}(n)$-time. We have obtained a new graph $G = (V, E)$, and there are $f - m$ convex faces in G. We denote by Q the set of inner points that are of degree at least 3 in G. We set $k := |Q|$. Now observe that for each point p in P'', there exists at least one edge e in E with one endpoint in Q, one endpoint on the boundary of the convex hull, such that e overlaps with p. This is because if we consider p and the two lines going through p and one of the two consecutive vertices in P'' (the one before p and the one after p when going around P'' in clockwise order), they define a wedge in which one edge must lie because of convexity. The point p can be an endpoint of e or in its relative interior. If for a point $p \in P''$ there are several edges that satisfy the conditions, we choose one arbitrarily. We denote these edges by E'. An edge in E' overlaps with exactly one point in P'', thus $|E'| = |P''|$. We denote by E_b the edges not in E' that have a point on the boundary of the convex hull and the other in Q, and we denote $|E_b|$ by m'. The vertices on the boundary of the convex hull are adjacent to two other vertices on the boundary of the convex hull. Moreover, those vertices are incident to $|P''| + m'$ additional edges. We have $2|E| = \sum_{v \in V} \deg(v) \geq 3k + 2(n-k) + |P''| + m' = k + 2n + |P''| + m'$. By Euler's formula, we have $f - m = |E| - n + 1 \geq \frac{k+|P''|+m'}{2} + 1$.

Now, the solution consists of the union of all edges in E incident to two points in Q, with the m edges in E_0 that we have removed, and with the $|P''| + m'$ edges in $E' \cup E_b$. We may need those edges as they might overlap with points in P_i. Note that there are at most $3k$ edges in E incident to two points in Q as G is plane. Moreover, all points in P_i are indeed covered by the edges in our solution. Thus, we obtain a covering of P_i with s segments, where $s \leq 3k + m + m' + |P''| \leq 3(2(f-m) - |P''| - m') + m + m' + |P''| \leq 6f - 5m - 2|P''| \leq 6f - 5m - 2(|P'| - 2m) \leq 6f - 2|P'|$. ◀

We combine Lemmas 2.1 and 2.2 to prove Theorem 1.5 in the full version of the paper [7].

3 Approximation algorithm for CPNCS

We present an $\mathcal{O}(\log OPT)$-approximation algorithm running in $\mathcal{O}(n^8)$ for CPNCS, where OPT denotes the minimum number of segments need to cover the points. The entire proof can be found in the full version of the paper [7]. The existence of an algorithm with the same approximation ratio and running time for MCP follows from Theorem 1.5.

Mitchell presented an algorithm for a related covering problem [9]. We adapt his algorithm to our setting of CPNCS. Let P be a set of n points. By doing a rotation if necessary, we can assume that no two points in P have the same x-coordinate. We say that a trapezoid is *constrained* if 1) it has two disjoint vertical sides, each lying on a line that contains a point in P, and 2) the two remaining sides are lying on lines that contain each at least two points in P. Note that there are $\mathcal{O}(n^6)$ constrained trapezoids.

We also allow for some degeneracies. Let us consider a triangle with vertices a, b and c, not all three on a line. If a is in P, the segment with endpoints b, c is vertical and lies on a line that contains a point in P, and the segments with endpoints a, b and a, c respectively are contained in some lines ℓ and ℓ' such that ℓ and ℓ' contains at least two points in P, then we say that the triangle is a constrained trapezoid. If a constrained trapezoid is split into two halves by a vertical line ℓ going through its interior, with ℓ containing a point in P, we obtain two constrained trapezoids. Likewise, if a segment s is in a constrained trapezoid τ, such that s lies on a line that contains at least two points in P, s intersects the interior of τ, and the endpoints of s are contained in the vertical sides of τ, then s splits τ into two constrained trapezoids.

Now we are ready to describe the algorithm. We give an exhaustive description of the algorithm, along with the proof of correctness, in the full version of the paper [7]. The algorithm uses dynamic programming. We first compute how to cover the points in the thinnest constrained trapezoids, with respect to their width on the x-axis. Now, for some larger constrained trapezoid, we consider the $\mathcal{O}(n^2)$ ways of splitting it non-vertically to obtain two new constrained trapezoids. We recurse on each of them, and store the splitting that minimises the number of segments needed. Likewise, we consider the $\mathcal{O}(n)$ ways of splitting vertically, recurse on the two new constrained trapezoids, and store the best solution. Finally, we take the best solution between splitting non-vertically and splitting vertically. As we spend quadratic time for each constrained trapezoid, the total running time is in $\mathcal{O}(n^8)$.

Acknowledgments. Research supported by the Swiss National Science Foundation within the collaborative DACH project *Arrangements and Drawings* as SNSF Project 200021E-171681. The author thanks Michael Hoffmann for his helpful advice.

References

1 Sayan Bandyapadhyay, Santanu Bhowmick, and Kasturi Varadarajan. Approximation schemes for partitioning: Convex decomposition and surface approximation. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1457–1470. SIAM, 2014. doi:10.1137/1.9781611973730.96.

2 Allan S. Barboza, Cid C. de Souza, and Pedro J. de Rezende. Minimum convex partition of point sets. In *Proceedings of International Conference on Algorithms and Complexity*, pages 25–37. Springer, 2019. doi:/10.1007/978-3-030-17402-6_3.

3 Prosenjit Bose, Andrej Brodnik, Svante Carlsson, Erik D Demaine, Rudolf Fleischer, Alejandro López-Ortiz, Pat Morin, and J Ian Munro. Online routing in convex subdivisions. *International Journal of Computational Geometry & Applications*, 12(04):283–295, 2002. doi:10.1142/S021819590200089X.

4 Hadrien Cambazard and Nicolas Catusse. An integer programming formulation using convex polygons for the convex partition problem. In *37th International Symposium on Computational Geometry (SoCG 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.SoCG.2021.20.

5 Erik Demaine, Sándor Fekete, Phillip Keldenich, Dominik Krupke, and Joseph S. B. Mitchell. CG:SHOP 2020. https://cgshop.ibr.cs.tu-bs.de/competition/cg-shop-2020. Accessed: 12/02/2020.

6 Thomas Fevens, Henk Meijer, and David Rappaport. Minimum convex partition of a constrained point set. *Discrete Applied Mathematics*, 109(1-2):95–107, 2001. doi:10.1016/S0166-218X(00)00237-7.

7 Nicolas Grelier. Hardness and approximation of minimum convex partition. *arXiv preprint arXiv:1911.07697*, 2019.

8 Christian Knauer and Andreas Spillner. Approximation algorithms for the minimum convex partition problem. In *Proceedings of Scandinavian Workshop on Algorithm Theory*, pages 232–241. Springer, 2006. doi:10.1007/11785293_23.

9 Joseph S. B. Mitchell. Approximation algorithms for geometric separation problems. *Technical report, Dept. of Applied Math. and Statistics, State U. of New York at Stony Brook*, 1993. Available at https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.50.7089&rep=rep1&type=pdf.

10 Toshinori Sakai and Jorge Urrutia. Convex decompositions of point sets in the plane. *arXiv preprint arXiv:1909.06105*, 2019.

11 Allan Sapucaia, Pedro J. de Rezende, and Cid C. de Souza. Solving the minimum convex partition of point sets with integer programming. *Computational Geometry*, page 101794, 2021. doi:10.1016/j.comgeo.2021.101794.

Curvature variation based adaptive sampling for Delaunay triangulations of Riemannian manifolds

Hana Dal Poz Kouřimská[1] **and Mathijs Wintraecken**[2]

1 **IST Austria**
hana.kourimska@ist.ac.at

2 **IST Austria**
m.h.m.j.wintraecken@gmail.com

Abstract

In recent years there has been an increase of interest in developing algorithms that triangulate Riemannian manifolds. One of the popular methods to tackle this task is to construct a Delaunay triangulation from a point sample on the manifold. Algorithms based on this method are guaranteed to succeed if certain global geometric assumptions on the point sample are fulfilled: besides being sufficiently dense and sparse, the point sample needs to be Delaunay protected, meaning that there are no foreign vertices near a Delaunay ball. Recent results show that if the protection of the point sample is large compared to the density and bounds on the absolute value of the sectional curvature, the Delaunay triangulation arising from this point sample is homeomorphic to the manifold.

However, it is well known that we need no protection to construct Delaunay triangulations in spaces of constant curvature. A bound on the variability of the sectional curvature would thus be a more natural choice for the protection than its absolute value. This paper addresses this issue and shows that the protection of a point sample only needs to be large compared to how far the manifold is (locally) from a space of constant curvature. This makes a far better adaptive sampling on Riemannian manifolds possible.

1 Introduction

Adaptive sampling for Delaunay triangulations has always been an important part of the work on the triangulations of manifolds. Its goal is to Delaunay triangulate a given manifold using as little vertices as possible. Triangulating manifolds with this method significantly improves the efficiency of algorithms running on these manifolds, since both their runtime and space complexities are tied to the number of vertices of the triangulation.

Computational geometers strive to define bounds on a point sample of a manifold such that the manifold can be Delaunay triangulated using these points as vertices.

The majority of work on triangulations of submanifolds of the Euclidean space formulates these bounds in terms of the *local feature size* (see Figure 1). In dimensions two and three see for example [1, 12]; for higher dimensional submanifolds we cite [11, 7].

The work on Delaunay triangulations of Riemannian manifolds [14, 5, 8, 6], on the other hand, assumes bounds on the *geometry of the manifold.* More precisely, the triangulation criteria demand that the density and protection (see Figure 2 on the right) of the point sample is large compared to the absolute value of the sectional curvatures and the injectivity radius (illustrated in Figure 2 on the left) on the manifold.

However, the bound on the absolute value of the sectional curvatures is from a certain perspective unnatural. Indeed, to Delaunay triangulate a space of any constant curvature it is sufficient to take a point sample that is generic; no extra protection is required. One therefore expects that the quality bounds are formulated in terms of *how far a space is from having constant curvature*, as supposed to its absolute value.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

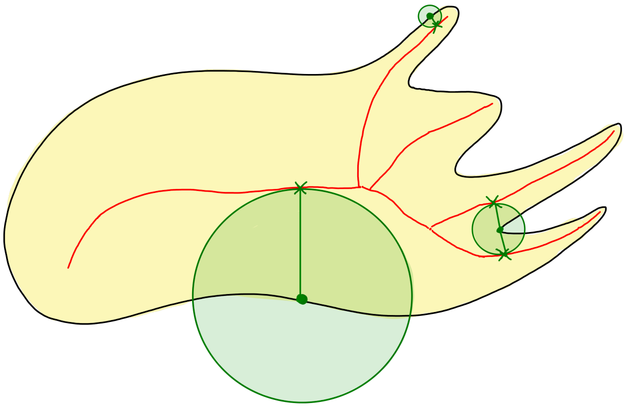


Figure 1 A manifold (in black) and its medial axis (in red). The local feature size of any point in the manifold is its distance (in green) to the medial axis.

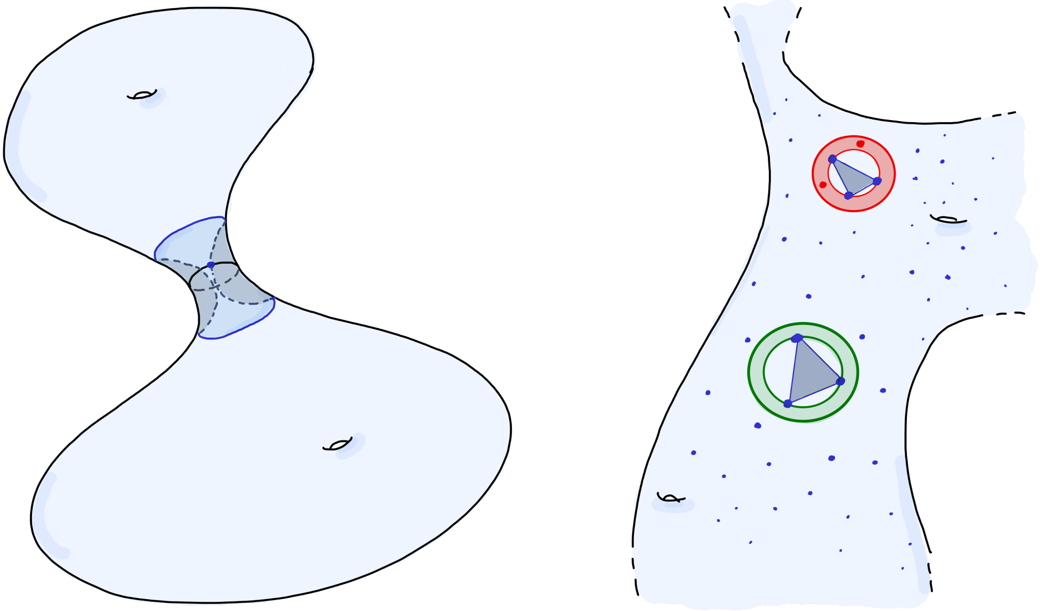


Figure 2 On the left: the injectivity radius is the largest radius such that no disc on the manifold with this radius self-intersects. On the right: the triangle with the green circumcircle is protected, since no other points lie in a green belt around it. The triangle with the red circumcircle is not protected, since its red belt contains two points.

This paper is the second to address this disparity. In [15] we discussed non-degeneracy criteria for simplices on spaces of nearly constant curvature. In this paper we use these results to derive quality bounds that guarantee a successful construction of Delaunay triangulations in spaces of nearly constant curvature. In contrast to Boissonnat, Dyer, and Ghosh [4], who focus in their work on Delaunay balls, we focus on the Voronoi diagram. Our investigation leads to surprising connections with *quadrics*, which we believe are of independent interest.

2 Background

Simplices in space forms A *space form* is a complete, simply connected Riemannian manifold with constant sectional curvature K. We denote n-dimensional space forms by $\mathbb{H}^n(K)$, or $\mathbb{H}(K)$. In short: If $K < 0$, $\mathbb{H}(K)$ is a hyperbolic space, if $K = 0$, $\mathbb{H}(K)$ is the Euclidean space, and if $K > 0$, $\mathbb{H}(K)$ is a sphere. An n-dimensional *simplex* σ in $\mathbb{H}^n(K)$ is a convex hull of a set of $n + 1$ points $v_0, \dots, v_n \in \mathbb{H}^n(K)$. We recall that a *convex hull* of a set of points is the smallest convex set containing these points. If σ is non-degenerate, there exists a unique[1] n-sphere in $\mathbb{H}^n(K)$ containing the vertices of σ. We call this sphere the *circumsphere* of σ, and its centre the *circumcentre* of σ.

In this paper we study how the positions of circumcentres of simplices are influenced by small perturbations of the metric. To this end we recall that the circumcentre of σ is the intersection of bisectors of pairs of its vertices v_i and v_j, where a *bisector* of v_i and v_j is an $n-1$-dimensional space form in which all points are equidistant to v_i and v_j.

The *quality* of a simplex quantifies how far a simplex is from being degenerate. The standard measures in Euclidean space are the thickness (the height[2] divided by the longest edge length) and the fatness (a normalized volume). In [15], the notion of quality has been extended to spaces of constant curvature to be able to formulate non-degeneracy bounds on Riemannian simplices on spaces of *almost* constant curvature.

Riemannian simplices

Throughout this paper, $\mathcal{M}$ denotes a Riemannian manifold with sectional curvatures K bounded by $\Lambda_\ell \leq K \leq \Lambda_u$.

We call a set $A \subseteq \mathcal{M}$ *convex* if for any two distinct points $a, b \in A$ there exists a minimizing geodesic in $\mathcal{M}$ connecting a and b, this geodesic is unique, it is contained in A, and no other geodesic between a and b is contained in A. We denote the injectivity radius of $\mathcal{M}$ by $\iota_{\mathcal{M}}$. Due to [10, Theorem IX.6.1] we then know that any closed ball of radius r in $\mathcal{M}$ with

$$r < r_C = \min\left\{\frac{\iota_{\mathcal{M}}}{2}, \frac{\pi}{2\sqrt{\Lambda_u}}\right\},$$

is convex. (If $\Lambda_u \leq 0$, we define $1/\sqrt{\Lambda_u} = \infty$.) The radius r_C is called the *convexity radius*.

The construction of a *Riemannian simplex* from a set of points in $\mathcal{M}$ is more involved than taking the convex full if $\mathcal{M}$ is not a space form. This is because the convex hull of 3 points in a generic manifold of dimension at least 3 is generally full dimensional [2, Section 6.1.3]. We rely on the Riemannian centre of mass construction (an example of the Fréchet means), defined by Karcher [16].

Metric distortion and the associated simplex In order to assess the quality of triangulations of Riemannian manifolds we develop means of comparing Riemannian simplices to 'similar' simplices in space forms. We use the concatenation $E_{q,K}$ of exponential maps (illustrated in Figure 3) to map a simplex σ from the manifold to the space form $\mathbb{H}(K)$, and work with its metric distortion. We recall [13] that the exponential map $\exp_{q,\mathcal{M}}$ at a point $q \in \mathcal{M}$ maps tangent vectors $v \in T_q\mathcal{M}$ to points on $\mathcal{M}$. Intuitively, it tells you what point

[1] The uniqueness follows from the reduction to the Euclidean case using stereographic projection or e.g. the Poincaré model.

[2] The height is the minimal altitude, where the altitude is defined as the distance from a vertex to the affine hull of the opposite face.

in $\mathcal{M}$ you would reach if you were to walk from q for a distance $\|v\|$ in the direction of $\frac{v}{\|v\|}$. For a small enough neighbourhood of $0 \in T_q\mathcal{M}$, the exponential map is a homeomorphism onto its image U_q. If $\mathcal{M} = \mathbb{H}(K)$, $\exp_{q,\mathcal{M}=\mathbb{H}(K)}$ is independent of the base point q, and we denote it by $\exp_{\mathbb{H}(K)}$ for simplicity.

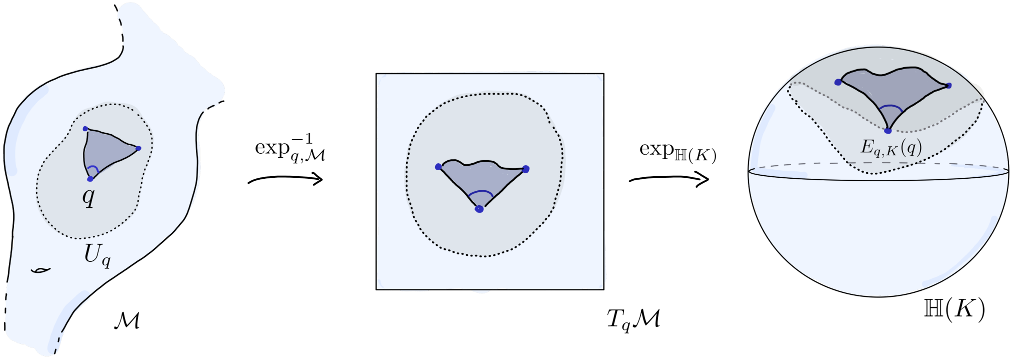


Figure 3 An illustration of the map $E_{q,K}$.

The *associated simplex* $\sigma_K(q)$ of σ is then the convex hull of the vertices of $E_{q,K}(\sigma)$ in $\mathbb{H}(K)$, as illustrated in Figure 4. It is the key object in assessing non-degeneracy of simplices.

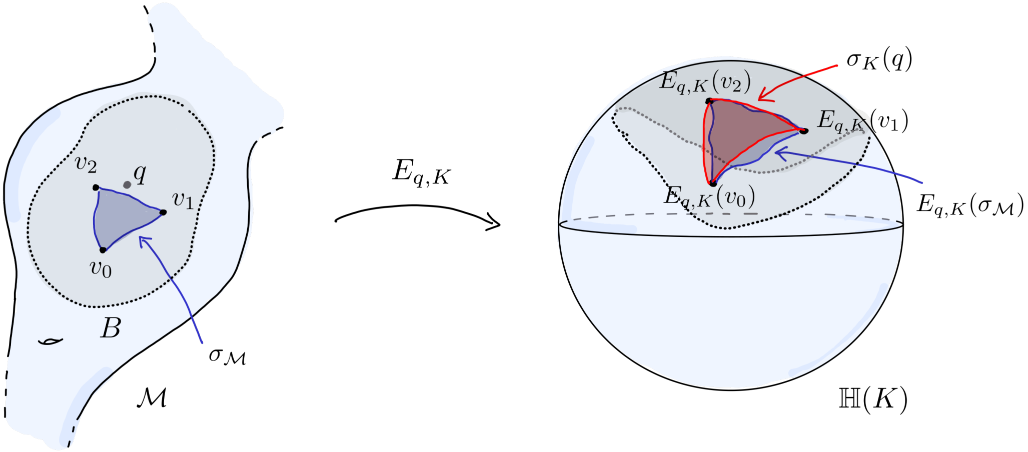


Figure 4 A Riemannian simplex (in blue) and its associated simplex (in red).

From a point sample to the Delaunay triangulation Next, we patch non-degenerate simplices together to form a triangulation. To derive the necessary sampling conditions we borrow techniques from [5, 6, 8, 14]. First, we need a point sample that covers our manifold well enough. We use the notion of an (ε, μ)*-net*, where ε and μ are the covering and the packing radius, respectively. An (ε, μ)-net of a planar region is illustrated in Figure 5.

The triangulation of the manifold $\mathcal{M}$ is built from its (ε, μ)-net P by the Voronoi-Delaunay construction. Recall that a (full-dimensional) *Voronoi cell* $\mathrm{Vor}_{\mathcal{M}}(p)$ of a point $p \in P$ is the locus of all points in $\mathcal{M}$ as close or closer to p than any other point in P. Lower-dimensional Voronoi cells are the intersections of the full-dimensional cells. The *Delaunay*

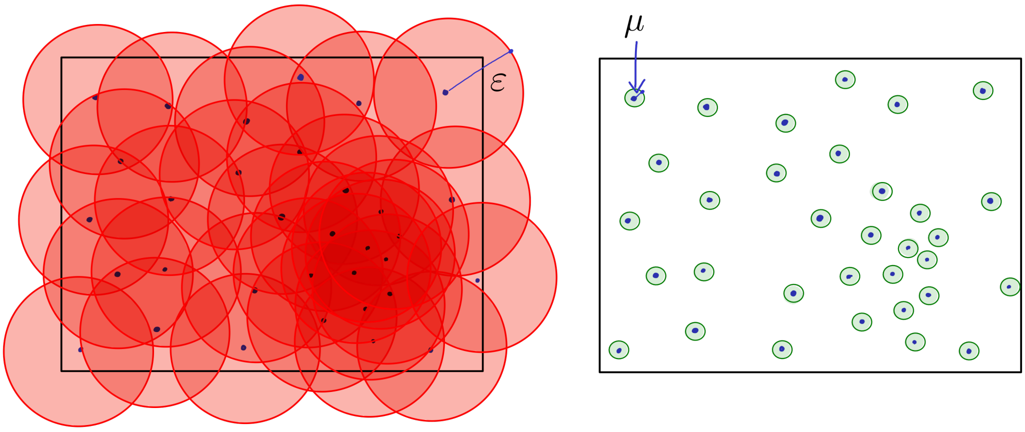


Figure 5 The point sample (in black) is an (ε, μ)-net of the planar region $\mathcal{M}$ since $\mathcal{M}$ is covered by balls of radius ε centred at the points of the point sample (on the left), and no two balls of radius μ centred at the points of the point sample intersect (on the right).

complex $\mathrm{Del}(P)$ of $\mathcal{M}$ is the nerve of the Voronoi diagram. That is, points $p_0, \ldots, p_j \in P$ form a simplex σ in $\mathrm{Del}(P)$ if and only if the intersection of their Voronoi cells is non-empty. This implies that for every $\sigma \in \mathrm{Del}(P)$ there exists a point in $\mathcal{M}$ that is equidistant to all points in σ. Let $v \in \bigcap_{i=0}^{j} \mathrm{Vor}(p_i)$ be such that the distance $d(v, p_0) = \cdots = d(v, p_j) =: r$ is minimal. We call the ball with centre v and radius r a *Delaunay ball* of σ.
The set $\mathrm{Del}(P)$ is always a simplicial complex, but its dimension can be arbitrarily high. Next to conditions on ε and μ we thus need to impose an additional condition on P to ensure that $\mathrm{Del}(P)$ is of the right dimension — the so-called δ*-protection*. In our paper we assume δ-protection not on the manifold, but on the space form. As in [5, 7, 3], protection on the manifold can be achieved using the Lovász Local Lemma [17]. The perturbation algorithm for our manifolds of almost constant curvature would be essentially the same as in [7] and we refer to that paper for details.

Whitney's lemma The core of this paper consists of showing that if the (ε, μ)-net P is sufficiently protected, we can construct parts of the Riemannian Delaunay complex by constructing non-degenerate Riemannian Delaunay simplices using the combinatorial structure from $\mathrm{Del}(P)$ and the results from previous sections. These local Delaunay complexes then combine to give a triangulation of the whole manifold thanks to Whitney's Lemma [18] (see also [9]). To apply this lemma simplices need to be glued together properly along their facets, and there must exist a 'safe' point in each simplex that is not covered by any other simplex.

3 Our results

The main theorem Our paper provides conditions on an (ε, μ)-net P of our manifold $\mathcal{M}$ that assure that the Delaunay complex $\mathrm{Del}_{\mathcal{M}}(P)$ is a piecewise smooth triangulation of $\mathcal{M}$, geometrically realized by Riemannian simplices on $\mathcal{M}$. We do not state these conditions explicitly in this extract, since they contain many technical constants. We note however, that these conditions are satisfied if $|\Lambda_\ell - \Lambda_u|$ is small enough compared to ε, μ, and the protection.

The Delaunay complex of a generic sample in a space form is always homeomorphic to the space form. We show that this homeomorphism is preserved under small distortions of the metric. We first bound the geometry of these distorted Voronoi cells, and study how protection ensures the quality of Delaunay simplices. Combined, these quality bounds yield Hausdorff stability of the Voronoi vertices. We also show that protection can be used to bound non-adjacent faces away from each other. We then generalize these results to spaces of almost constant curvature. Our bounds on the metric distortion yield conditions under which we can guarantee both the combinatorial stability and the existence of a so-called safe point, which together with Whitney's lemma yields the final result.

3.1 Results in space forms

Geometric interpretation of thickened bisectors Let $\nu \geq 0$. The *thickened bisector* of two distinct points $p, q \in \mathbb{H}^n(K = \pm 1)$ is the set $\mathfrak{B}_\nu(p,q) = \{x \in \mathbb{H}^n(\pm 1) \mid |d(x,p) - d(x,q)| \leq 2\nu\}$. One such thickened bisector is illustrated in Figure 6.

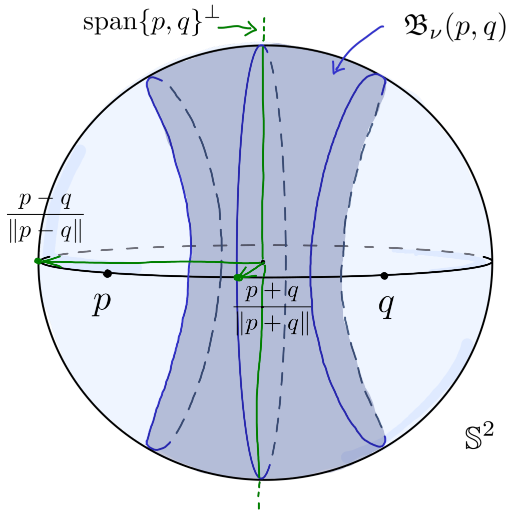


Figure 6 The thickened bisector of p and q.

We discovered that thickened bisectors have a straightforward geometric interpretation. They are the intersection of $\mathbb{H}^n(\pm 1)$ with a rigid body in $\mathbb{R}^{n+1}$. This body can be constructed as a Cartesian product of the orthogonal complement of $\operatorname{span}\{p,q\}$ and the union of ellipses (if $K = 1$) or hyperbolas (if $K = -1$) depicted in Figure 7.

Distortion of circumcentres Recall that a circumcentre of a simplex is the intersection of the bisectors of its vertices. After a small distortion, controlled by a factor ν, the circumcentre lies in the intersection of the thickened bisectors of pairs of vertices of the simplex. We bound the distance between the circumcentre C and its distorted image $\tilde{C}$. More precisely,

$$d\left(C, \tilde{C}\right) \leq \begin{cases} \frac{(n+1)\sin\left(\sqrt{K}\nu\right)}{K \operatorname{height}(\overline{\sigma})} & \text{if } K > 0, \\ \frac{(n+1)\sinh\left(\sqrt{|K|}\nu\right)}{|K| \operatorname{height}(\overline{\sigma})} & \text{if } K < 0, \end{cases}$$

where $\operatorname{height}(\overline{\sigma})$ denotes the height of the convex hull in $\mathbb{R}^{n+1}$ of the vertices of the considered simplex. Assuming further that the simplex forms a part of the Delaunay triangulation of

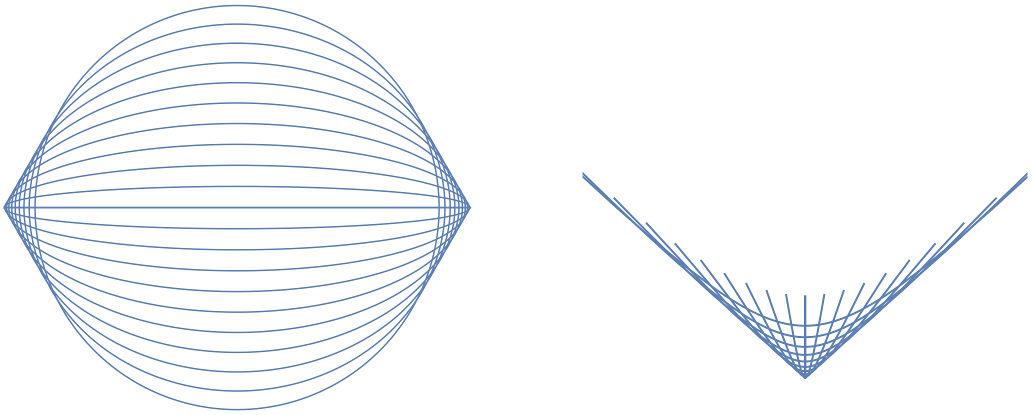


Figure 7 Ellipses and hyperbolas used in the construction of thickened bisectors.

a δ-protected (ε, μ)-net we lower bound height$(\overline{\sigma})$ to obtain an upper bound for $d\left(C, \tilde{C}\right)$ expressed only in terms of the curvature K, and the parameters δ, μ, and ε.

Voronoi objects lie in union of the Delaunay balls of their vertices For a generic point sample P in $\mathbb{H}^n(K)$, we consider a face F of the Voronoi diagram of P, $F = \bigcap_{j=0}^{k} \operatorname{Vor}(p_j)$. We denote the vertices of F by w_i. Each vertex w_i is a centre of a Delaunay ball, which we denote by $B(w_i, r_i)$ (see Figure 8). Then for each $x \in F$ and each j,

$$B(x, d(x, p_j)) \subseteq \bigcup_i B(w_i, d(w_i, p_j)) = \bigcup_i B(w_i, r_i).$$

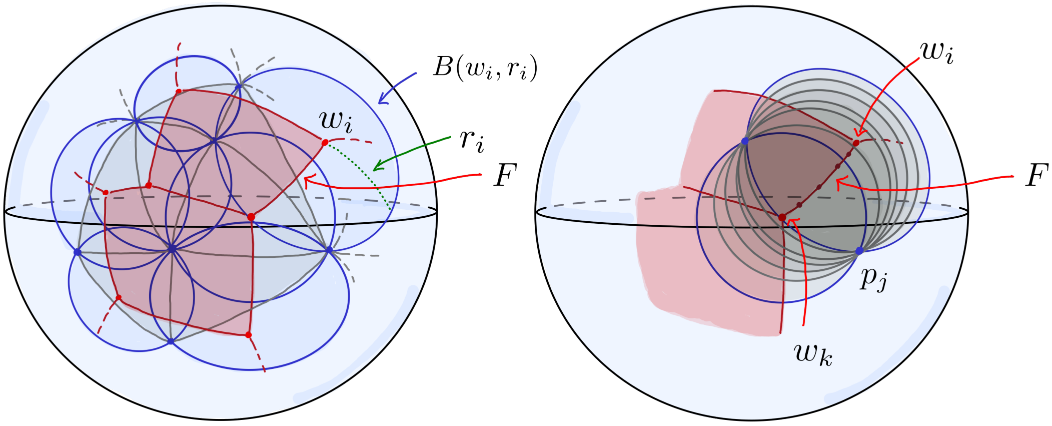


Figure 8 On the left: The Delaunay (in gray) and Voronoi (in red) cell decomposition. In blue the Delaunay balls. On the right: Illustration of our first result.

Furthermore: A Voronoi cell is called *foreign* to F if the two sets are disjoint. If P is δ-protected, then the minimal distance between any Voronoi face and any foreign Voronoi cell is lower bounded by $\delta/2$.

3.2 Results in spaces of almost constant curvature

For any small enough simplex in $\mathcal{M}$ we establish a lower bound on the height of the associated simplex in the space form $\mathbb{H}(\Lambda_{\text{mid}})$, with $\Lambda_\ell \leq \Lambda_{\text{mid}} \leq \Lambda_u$, as in [15].

We use this result to upper bound the parameter ν that controls how far the bisectors get distorted. This bound, in turn, allows us to control the images of the Voronoi faces under the map $E_{p,\Lambda_{\text{mid}}}$.

Combining these results we finally show that under certain conditions on the three parameters δ, μ, and ε the star of any vertex q in the Delaunay triangulation of $P \subseteq \mathcal{M}$ is combinatorially equivalent to the star of $E_{p,\Lambda_{\text{mid}}}(q)$ of the Delaunay triangulation of $E_{p,\Lambda_{\text{mid}}}(P)$. In addition, we prove the existence of a safe point for each simplex of the triangulation. This proof is based on a distortion bound between the image of the Riemannian simplex under the map $E_{p,\Lambda_{\text{mid}}}(q)$ and the associated simplex (where points with the same barycentric coordinates are identified).

Acknowledgments We are greatly indebted to Jean-Daniel Boissonnat, Ramsay Dyer, and Mael Rouxel-Labbé for discussion. We thank all members of the Edelsbrunner group for the atmosphere in which this research was conducted.

References

1 Nina Amenta and Marshall Bern. Surface reconstruction by Voronoi filtering. *Discrete & Computational Geometry*, 22(4):481–504, 1999. doi:10.1007/PL00009475.

2 Marcel Berger. *A panoramic view of Riemannian geometry.* Springer-Verlag, Berlin, 2003. doi:10.1007/978-3-642-18245-7.

3 Jean-Daniel Boissonnat, Frédéric Chazal, and Mariette Yvinec. *Geometric and topological inference.* Cambridge Texts in Applied Mathematics. Cambridge University Press, Cambridge, 2018. doi:10.1017/9781108297806.

4 Jean-Daniel Boissonnat, Ramsay Dyer, and Arijit Ghosh. Delaunay stability via perturbations. *International Journal of Computational Geometry & Applications*, 24(02):125–152, 2013. doi:10.1142/S021819591450006X.

5 Jean-Daniel Boissonnat, Ramsay Dyer, and Arijit Ghosh. Delaunay triangulation of manifolds. *Found. Comput. Math.*, 18(2):399–431, 2018. doi:10.1007/s10208-017-9344-1.

6 Jean-Daniel Boissonnat, Ramsay Dyer, Arijit Ghosh, and Mathijs Wintraecken. Local criteria for triangulation of manifolds. In *34th International Symposium on Computational Geometry (SoCG 2018)*, volume 99, pages 9:1–9:14, 2018. doi:10.4230/LIPIcs.SoCG.2018.9.

7 Jean-Daniel Boissonnat and Arijit Ghosh. Manifold reconstruction using tangential Delaunay complexes. *Discrete & Computational Geometry*, 51(1):221–267, 2014. doi:10.1007/s00454-013-9557-2.

8 Jean-Daniel Boissonnat, Mael Rouxel-Labbé, and Mathijs Wintraecken. Anisotropic triangulations via discrete Riemannian Voronoi diagrams. *SIAM Journal on Computing*, 48(3):1046–1097, 2019. doi:10.1137/17M1152292.

9 Jean-Daniel Boissonnat and Mariette Yvinec. *Algorithmic geometry.* Cambridge University Press, Cambridge, 1998. Translated from the 1995 French original by Hervé Brönnimann. doi:10.1017/CBO9781139172998.

10 Isaac Chavel. *Riemannian geometry—a modern introduction*, volume 108 of *Cambridge Tracts in Mathematics.* Cambridge University Press, Cambridge, 1993. doi:10.1017/CBO9780511616822.

11 Siu-Wing Cheng, Tamal Dey, and Edgar Ramos. Manifold reconstruction from point samples. pages 1018–1027, 01 2005. doi:10.1145/1070432.1070579.

12 Siu-Wing Cheng, Tamal Krishna Dey, and Jonathan Richard Shewchuk. *Delaunay mesh generation*. Chapman & Hall/CRC Computer and Information Science Series. Chapman & Hall/CRC, Boca Raton, FL, 2013. doi:10.1201/b12987.

13 Manfredo Perdigão do Carmo. *Riemannian geometry*. Mathematics: Theory & Applications. Birkhäuser Boston, Inc., Boston, MA, 1992. Translated from the second Portuguese edition by Francis Flaherty. doi:10.1007/978-1-4757-2201-7.

14 Ramsay Dyer, Gert Vegter, and Mathijs Wintraecken. Riemannian simplices and triangulations. *Geometriae Dedicata*, 2015. doi:10.1007/s10711-015-0069-5.

15 Ramsay Dyer, Gert Vegter, and Mathijs Wintraecken. Simplices modelled on spaces of constant curvature. *Journal of Computational Geometry*, 10(1):223–256, Jul. 2019. doi:10.20382/jocg.v10i1a9.

16 H. Karcher. Riemannian center of mass and mollifier smoothing. *Communications on Pure and Applied Mathematics*, 30(5):509–541, 1977. doi:10.1002/cpa.3160300502.

17 Robin A. Moser and Gábor Tardos. A constructive proof of the general Lovász local lemma. *J. ACM*, 57(2):Art. 11, 15, 2010. doi:10.1145/1667053.1667060.

18 H. Whitney. *Geometric Integration Theory*. Princeton University Press, 1957. doi:10.1515/9781400877577.

Arrangements of Pseudocircles: On Digons and Triangles[*]

Stefan Felsner[1], Sandro Roch[1], and Manfred Scheucher[1]

1 **Institut für Mathematik,**
Technische Universität Berlin, Germany,
`lastname@math.tu-berlin.de`

Abstract

The investigation of arrangements of pseudolines and their cell structure goes back to Levi in the 1920's. In Grünbaum's monograph from the 1970's, he started the investigation of arrangements of pseudocircles and posed several interesting problems and conjectures, some of which are still open. Here we discuss the cell-structure of arrangements of pairwise intersecting pseudocircles.

First, we discuss the maximum number of digons or touching points. Grünbaum conjectured that every arrangement of n pairwise intersecting pseudocircles has at most $2n-2$ digons or equivalently at most $2n-2$ touchings. Using a result from Agarwal et al. (2004), who proved the conjecture for cylindrical arrangements, we show that the conjecture holds for any arrangement, where a triple of pseudocircles is pairwise touching. Even though the general conjecture remains open, this substantially narrows the options for potential counter-examples.

Second, we discuss the minimum number of triangular cells (triangles) in an arrangement of n pairwise intersecting pseudocircles without digons and touchings. While Snoeyink and Hershberger (1991) showed that there are at least $p_3 \geq \frac{4}{3}n$ triangles, Felsner and Scheucher (2017) showed that there exist arrangements on $n \geq 6$ pseudocircles with $p_3 < \lceil \frac{16}{11}n \rceil$ triangles, which disproved a long-standing conjecture of Grünbaum. Here we provide a construction for $n \geq 6$ with only $p_3 = \lceil \frac{4}{3}n \rceil$ triangles, showing that the lower bound of Snoeyink and Hershberger is tight.

1 Introduction

An *intersecting arrangement of pseudocircles* is a collection of simple closed curves on the sphere or plane such that any two of the curves either touch in a single point or intersect in exactly two points where they cross. Throughout this article, we consider all arrangements to be *simple*, that is, no three pseudocircles meet in a common point. An arrangement $\mathcal{A}$ partitions the plane into cells. Cells which have k crossings on their boundary are *k-cells* and we denote their number by $p_k(\mathcal{A})$. We also call 2-cells *digons* and 3-cells *triangles*.

The investigation of cells in arrangements started about 100 years ago with the study of *arrangements of (pairwise intersecting) pseudolines* by Levi [8], who showed that in the projective plane every pseudoline is incident to at least 3 triangles and proved the famous extension lemma. In the 1970's, Grünbaum [7] intensively investigated arrangements of pseudolines and initiated the study of arrangements of pseudocircles.

[*] A part of this work was initiated at a workshop of the collaborative DACH project *Arrangements and Drawings* in Gathertown. We thank the organizers and all the participants for the inspiring atmosphere. S. Roch was funded by the DFG-Research Training Group 'Facets of Complexity' (DFG-GRK 2434). M. Scheucher was supported by the DFG Grant SCHE 2214/1-1.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

1.1 Digons and touchings

Concerning digons in intersecting arrangements of pseudocircles, Grünbaum [7, Conjecture 3.6][1] posed the following conjecture:

▶ **Conjecture 1.1** (Grünbaum's digon conjecture [7]). *Every intersecting arrangement of n pseudocircles has at most $2n - 2$ digons.*

An intersecting arrangement of pseudocircles is called *cylindrical*, if there is a pair of cells which are separated by each pseudocircle of the arrangement. It was shown by Agarwal et al. [1, Corollary 2.12] that Conjecture 1.1 holds for simple cylindrical arrangements.

Moreover, Agarwal et al. show for intersecting arrangements of pseudocircles that the number of digons is at most linear in n. The proof of this linear bound is based on the fact that every arrangement of intersecting pseudocircles can be stabbed by constantly many points. That is, there exists an absolute constant k, called the *stabbing number*, such that, for every arrangement of n pseudocircles in the plane, there exists a set of k points with the property that each pseudocircle contains at least one such point in its interior. In the literature, the stabbing number is also often referred to as piercing number or transversal number. Hence the arrangement can be decomposed into constantly many cylindrical subarrangements. The multiplicative constant of the linear term however remains unknown. In [6] we verified the conjecture for up to $n = 7$ pseudocircles.

Here we show that Grünbaum's digon conjecture (Conjecture 1.1) holds for simple arrangements with three pseudocircles that pairwise form a digon; see Section 2. Before we state the result, let us introduce some notation which will be used extensively. Any arrangement $\mathcal{A}$ of pseudocircles can be perturbed so that any selection of its digons become touching points. Figure 1 gives an illustration. It is therefore sufficient to find an upper bound on the number of touchings. The *touching graph* $T(\mathcal{A})$ consists of the pseudocircles as vertices, and two of them share an edge if they have a touching.

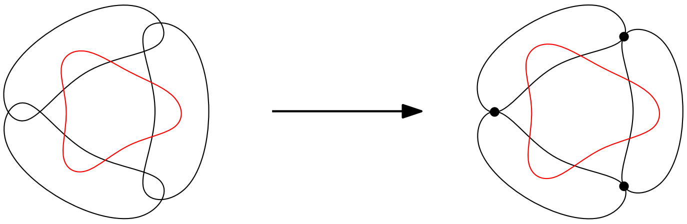


Figure 1 Contracting some of the digons to touchings.

▶ **Theorem 1.2.** *Let $\mathcal{A}$ be an arrangement of n pairwise intersecting pseudocircles. If the touching graph $T(\mathcal{A})$ contains a triangle, then there are at most $2n - 2$ touchings.*

1.2 Triangles in digon- and touching-free arrangements

The study of triangles in arrangements goes back to Levi [8], who showed that every arrangement of n pseudolines in the projective plane contains at least n triangles. Since pseudoline arrangements are in correspondence with arrangements of *great-pseudocircles* (see

[1] Originally the conjecture extends to non-simple arrangements which are *non-trivial*, i.e., arrangements with at least 3 crossing points.

e.g. [5, Section 4]), it directly follows that an arrangement of n great-pseudocircles contains at least $p_3 \geq 2n$ triangles.

Grünbaum conjectured that every digon- and touching-free intersecting arrangement on n pseudocircles contains at least $p_3 \geq 2n - 4$ triangles [7, Conjecture 3.7]. Snoeyink and Hershberger [10] proved a sweeping lemma for arrangements of pseudocircles. Using this powerful tool, they concluded that in every digon- and touching-free intersecting arrangement every pseudocircle has two triangles on each of its two sides (interior and exterior) and derived the lower bound $p_3(\mathcal{A}) \geq 4n/3$; see Section 4.2 in [10].

In [6] we constructed an infinite family of arrangements with $p_3 < \frac{16}{11}n$ which shows that Grünbaum's conjecture is wrong and verified that the lower bound $p_3 \geq 4n/3$ by Snoeyink and Hershberger is tight for $6 \leq n \leq 14$. We now have:

▶ **Theorem 1.3.** *For every $n \geq 6$, there exists a digon- and touching-free arrangement $\mathcal{A}_n$ of n pairwise intersecting pseudocircles with $p_3 = \lceil \frac{4}{3}n \rceil$ triangles.*

All arrangements constructed in Section 3 contain $\mathcal{A}_6$ (depicted on the left of Figure 7) as a subarrangement. This remarkable arrangement has been studied as the arrangement $\mathcal{N}_6^\Delta$ in [5] where it was shown that $\mathcal{N}_6^\Delta$ is *non-circularizable*, i.e., $\mathcal{N}_6^\Delta$ cannot be represented by an arrangement of proper circles. As a consequence, all arrangements constructed in Section 3 are as well non-circularizable. In fact, all known counter-examples to Grünbaum's triangle conjecture contain $\mathcal{N}_6^\Delta$ and are therefore non-circularizable. Hence, Grünbaum's conjecture may still be true when restricted to arrangements of proper circles.

▶ **Conjecture 1.4** (Weak Grünbaum triangle conjecture, [6, Conjecture 2.2]). *Every intersecting digon- and touching-free arrangement of n circles has at least $2n - 4$ triangles.*

1.3 Discussion

For intersecting arrangements of unit-circles, Pinchasi showed an upper bound of $p_2 \leq n + 3$ [9, Lemma 3.4 and Corollary 3.10]. For arrangements of unit circles there is a classical construction of Erdős [3] with n not necessarily pairwise intersecting circles and $\Omega(n^{1+c/\log\log n})$ touchings. An upper bound of $O(n^{3/2+\epsilon})$ on the number of digons in circle arrangements was shown by Aronov and Sharir [2]. We are not aware of upper bounds on the number of digons in the case of not necessarily intersecting pseudocircles.

Concerning intersecting arrangements with digons, the number of triangles behaves slightly different. While our best lower bound so far is $p_3 \geq 2n/3$, we have used computer assistance to verify that $p_3 \geq n - 1$ is a tight lower bound for $3 \leq n \leq 7$ [6]. It remains open, whether $p_3 \geq n - 1$ is a tight lower bound for every $n \geq 3$ [6, Conjecture 2.10]. For the maximum number of triangles in intersecting arrangements in [6], we have shown an upper bound $p_3 \leq \frac{4}{3}\binom{n}{2} + O(n)$ which is optimal up to a linear error term.

2 Sketch of the proof of Theorem 1.2

We outline the proof of Theorem 1.2. A complete proof is deferred to the full version; see [4] for a preliminary version.

Since the touching graph $T(\mathcal{A})$ contains a triangle, there are three pseudocircles in $\mathcal{A}$ that pairwise touch. Let $\mathcal{K}$ be the subarrangement induced by these three pseudocircles and let $\triangle$ and $\triangle'$ denote the two triangle cells in $\mathcal{K}$. We label the three touching points, which are also the corners of $\triangle$ and $\triangle'$, as a, b, c. Furthermore, we label the three boundary arcs of $\triangle$ (resp. $\triangle'$) as α, β, γ (resp. α', β', γ'), as shown in Figure 2(a).

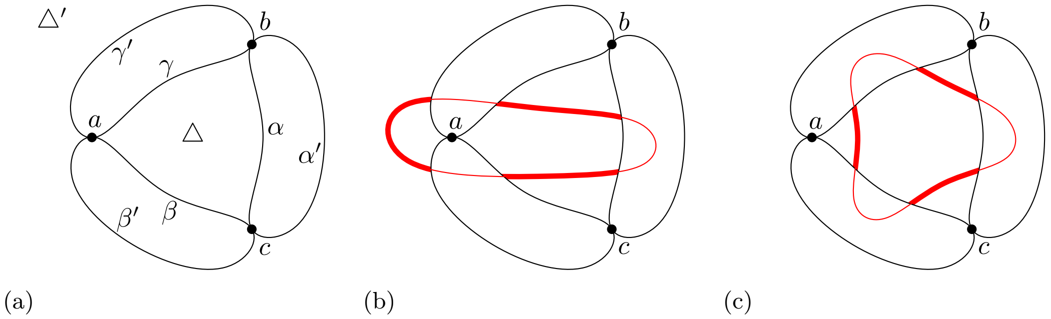


Figure 2 (a) An illustration of the subarrangement $\mathcal{K}$. (b) and (c) illustrate an additional pseudocircle C (red). The pc-arcs inside both $\triangle$ and $\triangle'$ are highlighted.

Assume that all digons in $\mathcal{A}$ are contracted to touchings.

The intersection of a pseudocircle $C \in \mathcal{A} \setminus \mathcal{K}$ with $\triangle \cup \triangle'$ results in three connected segments, which we denote as the three *pc-arcs* of C, see Figures 2(b) and 2(c). Note that each pc-arc in $\triangle$ connects two of α, β or γ while a pc-arc in $\triangle'$ connects two of α', β' and γ'. Depending on the boundary arcs on which they start and end, they belong to one of the types $\alpha\beta$, $\beta\gamma$, $\alpha\gamma$, $\alpha'\beta'$, $\beta'\gamma'$ or $\alpha'\gamma'$.

▶ **Claim 2.1.** *If two pc-arcs inside $\triangle$ or $\triangle'$ have a touching or cross twice, then they are of the same type.*

Proof of Claim 2.1. Suppose towards a contradiction that two distinct pseudocircles C, C' from $\mathcal{A} \setminus \mathcal{K}$ contain pc-arcs $A \subset C \cap \triangle$ and $A' \subset C' \cap \triangle$ of different types that have a touching or cross twice. One needs to check the four cases depicted in Figure 3. In none of these cases, pc-arc A' can be completed to a pseudocircle extending the intersecting arrangement of the four given pseudocircles. This is a contradiction. $\triangle$

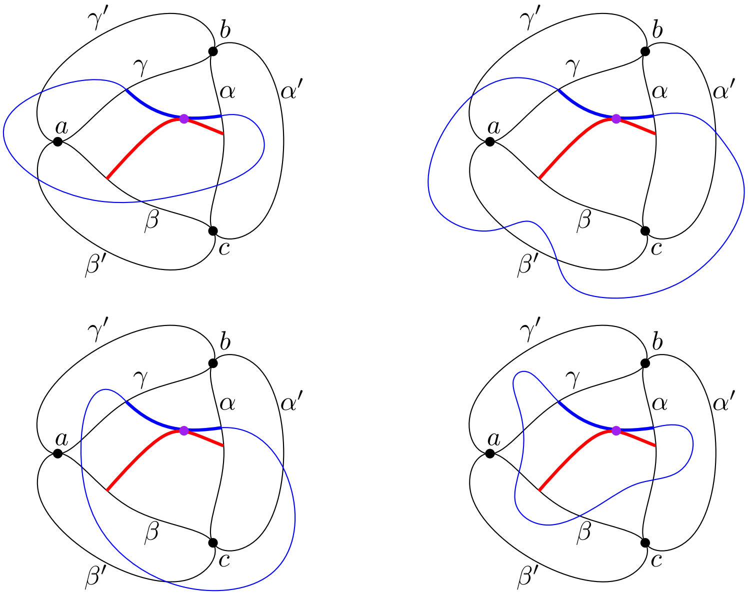


Figure 3 An illustration of the proof of Claim 2.1. The pseudocircles C and C' are highlighted blue and red, respectively. The pc-arcs A and A' are emphasized.

Next we explain how to transform $\mathcal{A}$ into another intersecting arrangement $\mathcal{A}'$ by changing the intersection pattern of pc-arcs within $\triangle$ and $\triangle'$. This transformation will ensure that the touching graphs of $\mathcal{A}$ and $\mathcal{A}'$ are identical and the arrangement $\mathcal{A}' \setminus \mathcal{K}$ will turn out to be cylindrical.

In both triangles, $\triangle$ and $\triangle'$, we concentrate all crossings and touchings of each arc type in a narrow region as depicted in Figure 4. For example, all the crossings of $\alpha\beta$ pc-arcs are in a region close to c and none of the crossings or touchings of these arcs is separated from c by an arc of type $\alpha\gamma$ or $\beta\gamma$. This is done in a way such that for each type of pc-arcs the arrangement of these arcs stays the same and all the endpoints of all pc-arcs stay at their original position.

By applying Claim 2.1, one can check that this transformation preserves the crossing and touching relations between any pair of pseudocircles. Hence we obtain again a valid intersecting pseudocircle arrangement $\mathcal{A}'$ with the same number of touchings.

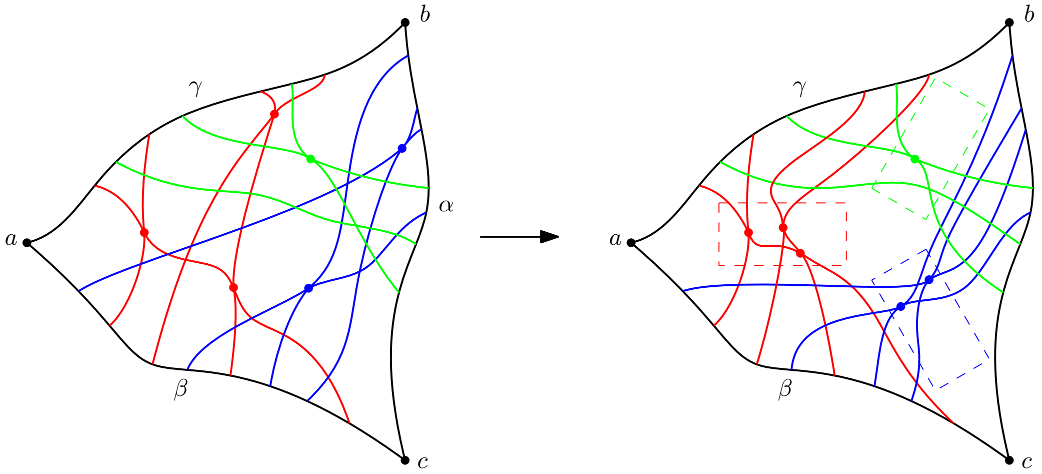


Figure 4 Concentrate all crossings and touchings of one arc type in a narrow region. The narrow regions are indicated by dashed rectangles.

Moreover, one can verify that $\mathcal{A}'$ can always be drawn as in Figure 5 on a cylinder, so that all pseudocircles except the three pseudocircles of $\mathcal{K}$ wrap around the cylinder. This means that the following claim holds:

▶ **Claim 2.2.** *The arrangement induced by $\mathcal{A}' \setminus \mathcal{K}$ is cylindrical.*

Next we replace the three pseudocircles of $\mathcal{K}$ by six pseudocircles as illustrated in Figure 6, so that the resulting arrangement $\mathcal{A}''$ is cylindrical. Each of the three touching points a, b, c in $\mathcal{K}$ is replaced by two new touching points and altogether we obtain touchings $a', a'', b', b'', c', c''$. Hence, when transforming $\mathcal{A}$ into $\mathcal{A}''$, the number of pseudocircles is increased by 3 and the number of touchings is also increased by 3.

An *intersecting arrangement of pseudoparabolas* is a collection of infinite x-monotone curves, called *pseudoparabolas*, where each two of them either have a single touching or intersect in exactly two points where they cross. As every cylindrical pseudocircle arrangement can be represented as an arrangement of pseudoparabolas and vice versa, Agarwal et al. [1] proved the $p_2(\mathcal{A}) \leq 2n-2$ upper bound on the number of touchings in arrangements of cylindrical intersecting arrangements by bounding the number of touchings in an intersecting arrangement of pseudoparabolas. They show that their touching graph is planar and bipartite [1, Theorem 2.4]. In fact, the drawing of $\mathcal{A}''$ in Figure 6 can be seen as an

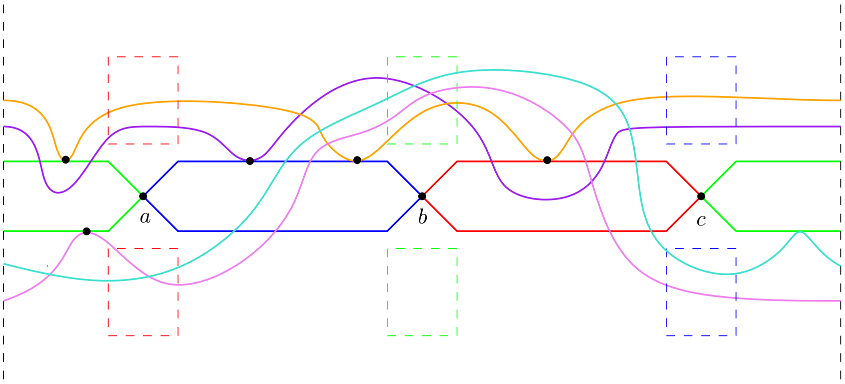


Figure 5 A cylindrical drawing of $\mathcal{A}' \setminus \mathcal{K}$.

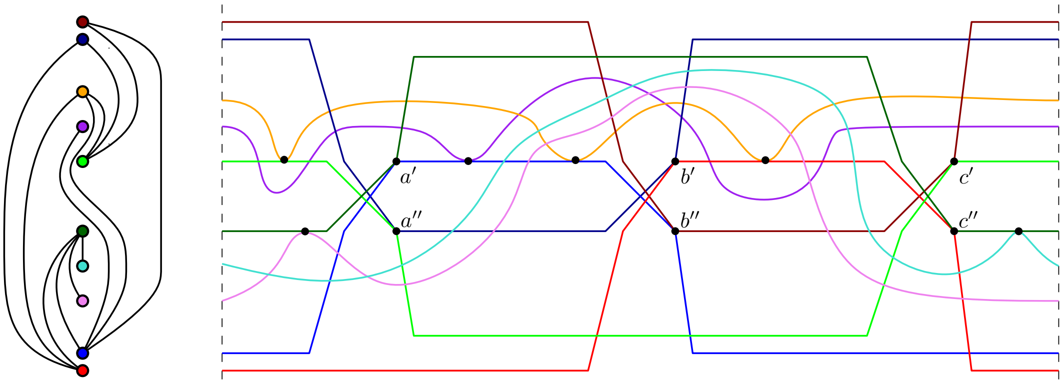


Figure 6 Replace each of the three pseudocircles of $\mathcal{K}$ by two new pseudocircles so that the entire arrangement is now cylindrical. On the left: the touching graph $T(\mathcal{A}'')$ of the arrangement.

intersecting arrangement of pseudoparabolas. We review their proof to prove the following claim.

▶ **Claim 2.3.** *$T(\mathcal{A}'')$ remains planar and bipartite after adding a certain edge.*

Since $T(\mathcal{A}'')$ remains planar and bipartite after adding an edge, and since planar bipartite n-vertex graphs have at most $2n-4$ edges, we obtain

$$p_2(\mathcal{A}) + 3 = p_2(\mathcal{A}'') \leq 2(n+3) - 5 \quad \Longrightarrow \quad p_2(\mathcal{A}) \leq 2n - 2.$$

This completes the sketch of the proof of Theorem 1.2.

3 Proof of Theorem 1.3

We denote by $\mathcal{A}_6$, $\mathcal{A}_7$, and $\mathcal{A}_8$ the three arrangements shown in Figure 7. These three arrangements on 6, 7, and 8 pseudocircles, respectively, are digon- and touching-free and contain 8, 10, and 11 triangles, respectively. In each of the three arrangements, there is a pseudocircle C and four incident triangles which are alternatingly inside and outside of C in the cyclic order around C. In fact, this *alternation property* holds for all pseudocircles of these three arrangements.

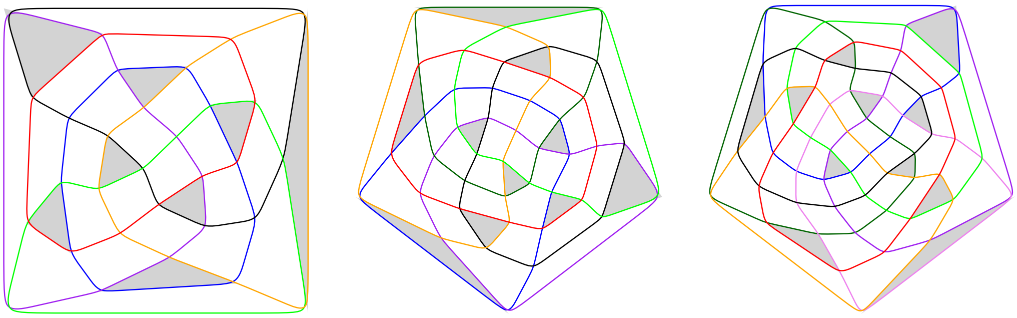
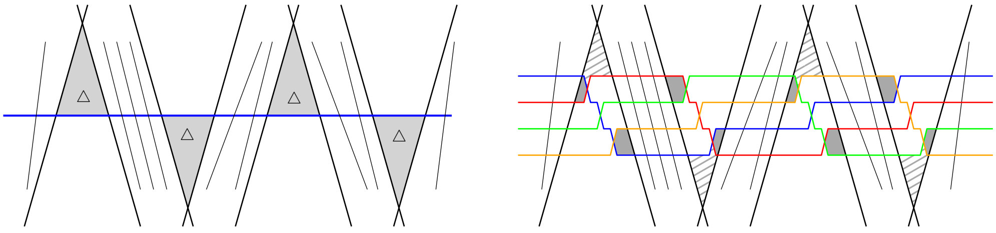


Figure 7 Digon- and touching-free intersecting arrangements of $n = 6, 7, 8$ pseudocircles with 8, 10, 11 triangles, respectively. Triangular cells are highlighted gray. [6, Fig. 2]



(a)

(b)

Figure 8 Replacing one pseudocircle with the alternation property (i.e., four triangles on alternating sides) by a particular arrangement of four pseudocircles.

To recursively construct $\mathcal{A}_n$ for $n \geq 9$, we replace a pseudocircle C with the alternation property from $\mathcal{A}_{n-3}$ by a particular arrangement of four pseudocircles as depicted in Figure 8.

With this replacement we destroy 4 triangles incident to C in the original arrangement, and in total the four new pseudocircles are incident to eight new triangles. Hence, we have $p_3(\mathcal{A}_n) = p_3(\mathcal{A}_{n-3}) + 4 = \lceil \frac{4}{3}(n-3) \rceil + 4 = \lceil \frac{4}{3}n \rceil$.

Moreover, for each of the four new pseudocircles, there are four new triangles (among the eight new triangles) that lie on alternating sides. This allow us to recurse by using one of the four new pseudocircles in the role of C for the next iteration. This completes the proof.

It is worth noting that $\mathcal{A}_6$ can be created as illustrated in Figure 9 by extending the Krupp arrangement of three pseudocircles, in which all cells are triangles.

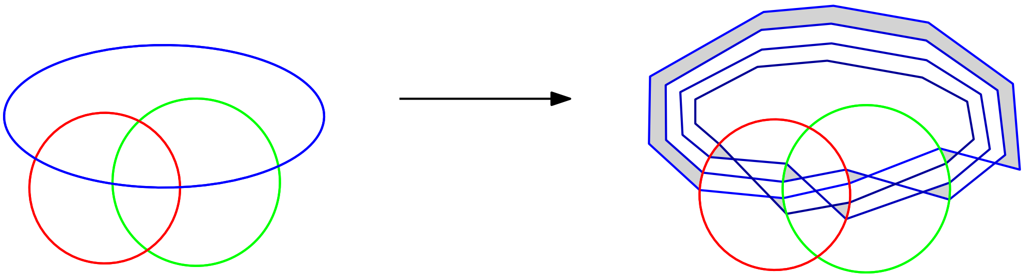


Figure 9 Extending the Krupp arrangement (left) to the arrangement $\mathcal{A}_6$ (right).

References

1 P. K. Agarwal, E. Nevo, J. Pach, R. Pinchasi, M. Sharir, and S. Smorodinsky. Lenses in Arrangements of Pseudo-circles and Their Applications. *Journal of the ACM*, 51(2):139–186, 2004.

2 B. Aronov and M. Sharir. Cutting circles into pseudo-segments and improved bounds for incidences. *Discrete & Computational Geometry*, 28(4):475–490, 2002.

3 P. Erdős. On sets of distances of n points. *The American Mathematical Monthly*, 53(5):248–250, 1946.

4 S. Felsner, S. Roch, and M. Scheucher. Arrangements of Pseudocircles: On Digons and Triangles (with Appendix), 2022. `http://page.math.tu-berlin.de/~roch/publ/eurocg22_full.pdf`.

5 S. Felsner and M. Scheucher. Arrangements of Pseudocircles: On Circularizability. *Discrete & Computational Geometry, Ricky Pollack Memorial Issue*, 64:776–813, 2020.

6 S. Felsner and M. Scheucher. Arrangements of Pseudocircles: Triangles and Drawings. *Discrete & Computational Geometry*, 65:261–278, 2021.

7 B. Grünbaum. *Arrangements and Spreads*, volume 10 of *CBMS Regional Conference Series in Mathematics*. AMS, 1972.

8 F. Levi. Die Teilung der projektiven Ebene durch Gerade oder Pseudogerade. *Berichte über die Verhandlungen der Sächsischen Akademie der Wissenschaften zu Leipzig, Mathematisch-Physische Klasse*, 78:256–267, 1926.

9 R. Pinchasi. Gallai—Sylvester theorem for pairwise intersecting unit circles. *Discrete & Computational Geometry*, 28(4):607–624, 2002.

10 J. Snoeyink and J. Hershberger. Sweeping arrangements of curves. In *Discrete & Computational Geometry: Papers from the DIMACS Special Year*, volume 6 of *DIMACS*, pages 309–349. AMS, 1991.

Unweighted Shortest Path in Disk Graphs

Katharina Klost[1]

[1] **Institut für Informatik, Freie Universität Berlin, Germany**
kathklost@inf.fu-berlin.de

Abstract

Shortest path problems are among the fundamental problems in graph theory. The unweighted single source shortest path problem (SSSP) in general graphs can be solved optimally with breadth first search (BFS) in $O(m+n)$ time. A *disk graph* $D(S)$ is a graph that is defined on a set S of sites, where each site $s \in S$ has an associated radius r_s. The vertex set of $D(S)$ is S and two sites s, t are connected by an edge st in $D(S)$ if and only if $\|st\| \leq r_s + r_t$, or equivalently, if the disks induced by s and t intersect. In this paper, we use a proxy graph defined by Kaplan et al. to solve the unweighted SSSP problem in disk graphs in $O(n \log^2 n)$ time. This significantly improves the previous best bound of $O(n \log^7 n \lambda_6(\log n))$[8, 9].

1 Introduction

The unweighted single source shortest path problem (SSSP) is a fundamental graph theoretic problem. To be precise the problem is *Given an unweighted graph G and a starting vertex v, find the shortest path distances from s to all other vertices, together with a shortest path tree.* For a general graph with n vertices and m edges it is widely known that it can be optimally solved by using BFS in $O(m+n)$ time.

Given a set S of n point sites, where each site $s \in S$ has an associated radius r_s, the disk graph is the intersection graph of the disks induced by these sites. To be precise, the disk graph $D(S)$ has a vertex for each site and an edge between two sites, if and only if the associated disks intersect. While the input size for a general graph is $m + n$, the input for problems on disk graphs consist of the sites and the associated radii and has size $O(n)$. Furthermore, as every complete graph can be realized as a disk graph, such a graph can have $\Theta(n^2)$ edges. Thus, explicitly constructing $D(S)$ and running BFS on the resulting graph, could lead to a $O(n^2)$ running time. This stands in contrast to the $\Omega(n \log n)$ lower bound for SSSP in disk graphs [1].

For unit disk graph, that is disk graphs where all sites have the same radius, there are multiple algorithms that achieve the optimal $O(n \log n)$ running time by using the underlying geometry [1, 2, 4]. On a very high level, these three algorithms all use a BFS in multiple rounds, where in round i all sites with distance $i + 1$ to the starting vertex are discovered. One should also mention the weighted case, where an edge uv is weighted with the Euclidean distance $\|uv\|$ between the sites. Here the currently best know algorithm by Wang and Xue [11] achieves a running time of $O(n \log^2 n)$ leaving an $O(\log n)$ gap to the lower bound. Where the optimal algorithms in the unweighted case can be seen as variants of BFS, the algorithm by Wang and Xue is based on Dijkstra's algorithm. They use (semi-dynamic) additively weighted nearest neighbor structures to identify a batch of edges that can be relaxed.

For unit disk graphs, there also is a variety of related problems that was studied. When considering the L_1 metric, the unweighted algorithms can be easily adapted, while for the weighted case there is a specialized optimal algorithm [12]. There is also the reverse version of the problem, where the input is the length of a shortest path between two vertices, and one

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

asks for the minimal radius of the disks, such that the shortest path has at least this length. The problem was considered for the L_1 and L_2 metrics in the weighted and unweighted case. In the L_1 metric both variants can be solved in $O(n \log^3 n)$ time, where in the L_2 metric the currently best know algorithm for the unweighted case takes $O(n^{5/4} \log^{7/4} n)$ time, where as the weighted case can be solved in $O(n^{5/4} \log^{5/2} n)$ time [13].

For unweighted general disk graphs, the best known SSSP algorithm so far takes $O(n \log^7 n \lambda_6(\log n))$ time [8, 9], where λ_6 is the length of a Davenport-Schinzel sequence. This algorithm directly implements a BFS by using repeated queries and deletions to a dynamic additively weighted nearest neighbor data structures to discover new edges.

In this paper, we use a proxy graph developed in the context of dynamic connectivity by Kaplan et al.[6] to perform a batched BFS on general disk graphs. We first compute the proxy graph of Kaplan et al. and use it to batch together sites on which we perform additively weighted nearest neighbor queries. By the choice of these batches, static additively weighted nearest neighbor data structures suffices to find the relevant edges of the SSSP tree, leading to the improved running time of $O(n \log^2 n)$.

2 The Proxy Graph

We briefly describe the proxy graph as defined by Kaplan et al.[6]. This proxy graph will be the base for our algorithm. We will focus our description on the parts of the proxy graph that are relevant for the purposes of this paper. For the remaining details, refer to the original paper. The proxy graph H is a bipartite graph on $O(n)$ vertices and $O(n \log n)$ edges that accurately represents the connectivity of a given disk graph. The vertex set consists of $O(n)$ *region* vertices in addition to one vertex for each site. Region vertices are used to represent cliques and sites connected to these cliques in a sparse fashion.

The regions are defined on subsets of cells of an *extended compressed quadtree* $\mathcal{Q}$ on S. The quadtree is extended to contain for each site $s \in S$ the special cell σ_s with $s \in \sigma_s$ and $|\sigma_s| \leq r_s \leq 2|\sigma_s|$ and a constant sized neighborhood of σ_s of cells with the same size.

On the cells of $\mathcal{Q}$, Kaplan et al. define a set of $O(n)$ *canonical paths*, such that every path starting at the root of $\mathcal{Q}$ can be uniquely represented by $O(\log n)$ disjoint canonical paths.[1] Each canonical path can be interpreted as the union of cells from the hierarchical grid that underlies the compressed quadtree.

For each canonical path with smallest cell σ and largest cell τ, a set of $O(1)$ disjoint regions is defined. One of these regions is a disk centered at the center of σ with radius $|\sigma|$. For the remaining regions, let c_1, c_2 be constants and let $\mathcal{C}_{c_1}$ and $\mathcal{C}_{c_2}$ be two sets of c_1 or c_2 respectively cones that partition the plane and have their apex at the center of σ. Then there are c_2 regions that are the intersection of the cones in $\mathcal{C}_{c_2}$ with an annulus of inner radius $|\sigma|$ and outer radius $\frac{5}{2}|\sigma|$ centered at the center of σ. Furthermore there are c_1 regions that are the intersection of the cones in $\mathcal{C}_{c_1}$ with an annulus of inner radius $\frac{5}{2}|\sigma|$ and outer radius $\frac{5}{2}|\sigma| + 2|\tau|$, see Figure 1.

To define the edges of the proxy graph, there are two sets $S_1(A)$ and $S_2(A)$ for each of these regions defined as follows. A site t is in $S_1(A)$, if $t \in A, |\sigma| \leq r_t \leq 2|\tau|$ and if the distance from t to the center of σ is at most $r_t + \frac{5}{2}|\sigma|$. Kaplan et al. show that when c_1 and c_2 are chosen appropriately the induced disk graph on $S_1(A)$ forms a clique.

[1] In the arXiv version of their paper[6], Kaplan et al. claim $O(\log^2 n)$ canonical paths. They recently improved this to $O(\log n)$, but the better bound is not published yet[7].

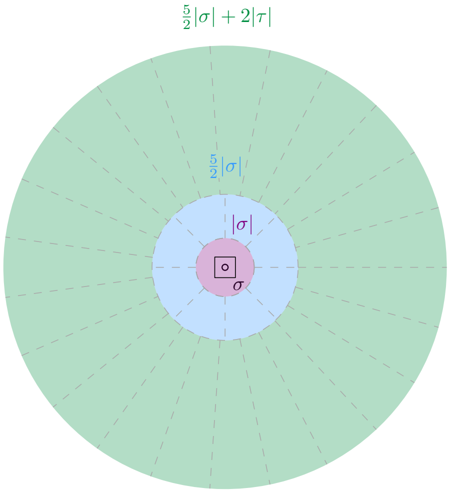


Figure 1 The regions in the plane

For s to lie in $S_2(A)$ for a region A, the region has to have been defined by a canonical path that is part of the unique representation of the path from the root of the quadtree to the special cell σ_s of s as defined above. Furthermore s has to intersect at least one site in $S_1(A)$.

A region A is connected to a site s with an edge in the proxy graph, if and only if $s \in S_1(A) \cup S_2(A)$. The set of all regions, together with the sets $S_1(A)$ and $S_2(A)$ can be found in $O(n \log^2 n)$ time (implied by Lemma 7.6 [6] and the improvement mentioned above).

We will need the following two properties of the proxy graph that Kaplan et al. did not explicitly state. In the following, we assume the starting vertex s for the SSSP problem to be fixed, and we denote by d_u the unweighted shortest path distance from s to a vertex u.

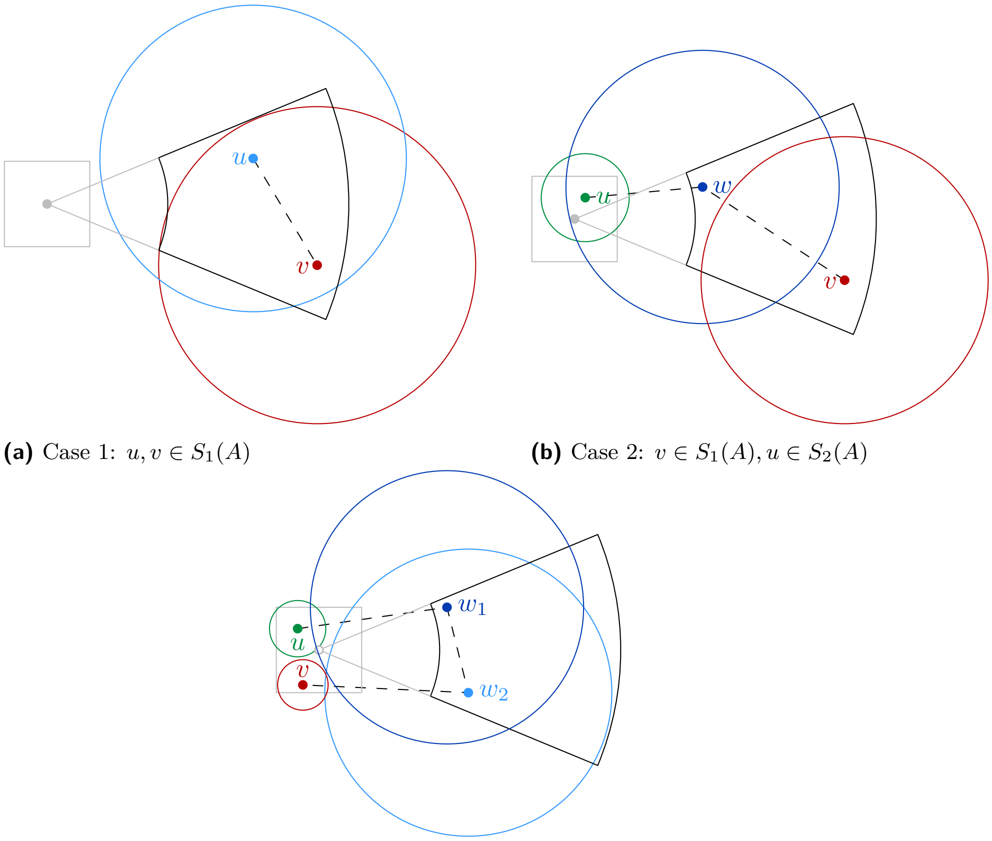
▶ **Lemma 2.1.** *If u and v are connected to the same region vertex A in the proxy graph H, then $|d_u - d_v| \leq 3$.*

Proof. The proof is an extension of the proof of Kaplan et al. (Lemma 6.3, Lemma 7.3 [6]) which shows that u and v are connected in $D(S)$, if they are connected to the same region vertex. Similar to their argument, this proof is based on the fact, that all sites that lie in the same set $S_1(A)$ form a clique (Lemma 6.2, Lemma 7.2 [6]). We consider three cases, see Figure 2 for an illustration.

1. $u, v \in S_1(A)$ If both sites lie in $S_1(A)$, they are part of the same clique. Thus $|d_u - d_v| \leq 1$.

2. $u \in S_1(A)$ and $v \in S_2(A)$, or $u \in S_2(A)$ and $v \in S_1(A)$ Without loss of generality, let $v \in S_2(A)$. This implies by the definition of $S_2(A)$, that there is a site $w \in S_1(A)$ such that v intersects w. Thus $|d_w - d_v| \leq 1$. Furthermore w lies in the same clique as u, implying that $|d_u - d_v| \leq 2$. The other case is symmetric.

3. $u, v \in S_2(A)$ Again by the definition of $S_2(A)$, we have sites $w_1, w_2 \in S_1(A)$ such that u intersects w_1 and v intersects w_2. As w_1 and w_2 lie in the same clique, the path u, w_1, w_2, v exists in $D(S)$ and thus $|d_u - d_v| \leq 3$. ◀



(a) Case 1: $u, v \in S_1(A)$

(b) Case 2: $v \in S_1(A), u \in S_2(A)$

(c) Case 3: $u, v \in S_2(A)$

Figure 2 Illustration of Lemma 2.1. The dashed lines correspond to the paths connecting u and v in $D(S)$

▶ **Observation 2.2** (Follows from Lemma 7.3 in Kaplan et al.[6])**.** *For every edge $st \in D(S)$ there is a region A such that $sA \in H$ and $At \in H$.*

3 Batched BFS

We efficiently implement a batched BFS on the proxy graph described in section 2. The batched BFS follows a similar idea to that in the unweighted shortest paths algorithm for unit disk graphs by Efrat et al.[4]. We first describe the algorithm and argue its correctness, then we analyze the running time.

The algorithm works on the proxy graph H in up to n rounds. In round i it identifies all sites with unweighted distance exactly $i+1$ to the starting vertex s. During the algorithm, we fill out a table $\mathit{dist}[\ ]$ for all $v \in S$ and later show that $\mathit{dist}[v] = d_v$. Let $W_i = \{u \in S \mid \mathit{dist}[u] = i\}$ be the set of all sites discovered in the previous round. We build a static additively weighted nearest neighbor data structure (AWNN) on the sites in W_i, where each site has weight $-r_s$. Let $\mathcal{A}_i$ be the set of regions that are adjacent to at least one site in W_i and let T_i be the set of all sites v adjacent to at least one region in $\mathcal{A}_i$ that have $\mathit{dist}[v] = \infty$. Then we query the AWNN one by one with the sites $v \in T_i$. Let u' be the result of the nearest neighbor query for a site $v \in T_i$. If $\|u'v\| \leq r_v$, we set $\mathit{dist}[v] = i+1$ and add v to W_{i+1}. See Algorithm 1 for a pseudocode of the algorithm.

Algorithm 1 The batched BFS algorithm for general disk graphs (with starting vertex s)

```
Compute the proxy graph $H$
$\mathit{dist}[v] = \infty \quad \forall v \in S$
$\mathit{dist}[s] = 0$
$i = 0$
$W_0 = \{s\}$
while $W_i \neq \emptyset$ do
    $W_{i+1} \leftarrow \emptyset$
    Build AWNN on $W_i$ with weights $-r_s$
    $\mathcal{A}_i \leftarrow \{R \mid uR \in E_H$ and $u \in W_i\}$
    $T_i \leftarrow \{v \mid vR \in E_H, R \in \mathcal{A}_i$ and $\mathit{dist}[v] = \infty\}$
    for $v \in T_i$ do
        $u' \leftarrow$ result of query of AWNN with $v$
        if $u'v$ is an edge in $D(S)$ then
            $\mathit{dist}[v] \leftarrow i+1$
            $W_{i+1} \leftarrow W_{i+1} \cup \{v\}$
    $i \leftarrow i+1$
```

Before we analyze the running time of the algorithm, we argue it's correctness.

▶ **Lemma 3.1.** *Algorithm 1 correctly computes the unweighted single source shortest path distances in a disk graph.*

Proof. To be precise, we show that at the end of the algorithm, $\mathit{dist}[v] = i$ if and only if $d_v = i$. First note, that once $\mathit{dist}[v]$ is changed from ∞ for a site $v \in S$, it will never change its value again. Now we can show the statement by induction. For $i = 0$ the statement holds by the preprocessing step.

For arbitrary i we first show, that if $\mathit{dist}[v] = i$ then $d_v = i$. Assume for the sake of contradiction that $\mathit{dist}[v] = i$ but $d_v \neq i$. If $\mathit{dist}[v]$ is set to i, this was done in iteration $i-1$

and only after a site u with $dist[u] = i - 1$ and $uv \in D(S)$ was discovered. By induction hypothesis, this implies that $d_u = i - 1$ and thus $d_v \leq i$. Now if $d_v = j < i$, by induction hypothesis, we would have already set $dist[v] = j$, a contradiction.

For the other direction, let π be a shortest s to v path in $D(S)$ and let u be the predecessor of v on this path. Then $d_u = i - 1$ and by induction hypothesis, $dist[u] = d_u$. This implies that $u \in W_{i-1}$ and thus u is contained in the AWNN in iteration $i - 1$.

At the beginning of iteration $i-1$ we have $dist[v] = \infty$, as by the induction hypothesis only the $dist$ values of sites with $d_v \leq i - 1$ are set. As uv is an edge in $D(S)$, by Observation 2.2 there is a region A such that vA and uA are edges in H. As u is incident to A, $A \in \mathcal{A}_i$ and $v \in T_i$. Let u' be the weighted nearest neighbor returned by the query with v. As uv is an edge in $D(S)$, we have $\|uv\| \leq r_u + r_v$. Furthermore, as the query returned u' it holds that $\|u'v\| - r_{u'} \leq \|uv\| - r_u$. Combining these two inequalities we get $\|u'v\| \leq r_{u'} + r_v$, implying that $u'v$ is also an edge in $D(S)$. Thus $dist[v]$ is set to i when the AWNN containing u is considered, finishing the proof. ◀

▶ **Lemma 3.2.** *Algorithm 1 has a running time of $O(n \log^2 n)$.*

Proof. The proxy graph can be build in $O(n \log^2 n)$ time [6, 7]. Now we consider the running time needed to preprocess the AWNN. As each site can only be in an AWNN in the iteration after its $dist[\,]$ value was set, each site is in at most one AWNN. An AWNN on m sites that allows a query time of $O(\log m)$ can be build in $O(m \log m)$ time [3, 5, 10], summing up over all AWNN gives a time of $O(n \log n)$ to build all AWNN.

Now consider the queries. By Lemma 2.1, the shortest path distances of sites that are adjacent to the same region differ by at most 3. Thus each site can be contained in the set T_i in at most 3 rounds. In each of these rounds, on query with a running time of $O(\log n)$ is performed, for an overall $O(n \log n)$ query time. As the preprocessing time for the proxy graph dominates, the lemma follows. ◀

▶ **Theorem 3.3.** *Algorithm 1 correctly finds all unweighted single source shortest paths distances in a disk graph in $O(n \log^2 n)$ time.*

Proof. The claim follows from Lemma 3.1 and Lemma 3.2. ◀

▶ Remark. The algorithm and all proofs can be straightforwardly extended to also produce the SSSP tree, by setting appropriate pointers in line 14 of the pseudocode.

4 Conclusion

By using an approach that is more tailored towards disk graphs, we are able to significantly improve the running time for solving the unweighted SSSP in general disk graphs. We conjecture that our result carries over to the case of the L_1-metric by slightly adapting the definition of the regions and using a matching additively weighted nearest neighbor data structure.

As the $O(n \log^2 n)$ time bound is confided to the construction of the proxy graph, finding a proxy graph that can be constructed in $O(n \log n)$ time and that satisfies Lemma 2.1 with an arbitrary constant d and some variant of Observation 2.2 would give an optimal algorithm.

Acknowledgements: I would like to thank Wolfgang Mulzer for proofreading a first draft. Special thanks goes to Günter Rote for pointing out how to confine the $\log n$-factor to the construction of the proxy graph.

References

1 Sergio Cabello and Miha Jejčič. Shortest paths in intersection graphs of unit disks. *Computational Geometry*, 48(4):360–367, 2015.

2 Timothy M. Chan and Dimitrios Skrepetos. All-pairs shortest paths in unit-disk graphs in slightly subquadratic time. In Seok-Hee Hong, editor, *27th International Symposium on Algorithms and Computation (ISAAC 2016)*, volume 64 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:13, Dagstuhl, Germany, 2016. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. doi:10.4230/LIPIcs.ISAAC.2016.24.

3 Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 3rd edition, 2008. doi:10.1007/978-3-540-77974-2.

4 A. Efrat, A. Itai, and M. J. Katz. Geometry Helps in Bottleneck Matching and Related Problems. *Algorithmica*, 31(1):1–28, September 2001. doi:10.1007/s00453-001-0016-8.

5 Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2(1):153, November 1987. doi:10.1007/BF01840357.

6 Haim Kaplan, Alexander Kauer, Katharina Klost, Kristin Knorr, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Dynamic Connectivity in Disk Graphs. *arXiv:2106.14935 [cs]*, June 2021. arXiv:2106.14935.

7 Haim Kaplan, Alexander Kauer, Katharina Klost, Kristin Knorr, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Personal communication, 2021.

8 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic Planar Voronoi Diagrams for General Distance Functions and Their Algorithmic Applications. *Discrete & Computational Geometry*, 64(3):838–904, October 2020. doi:10.1007/s00454-020-00243-7.

9 Chih-Hung Liu. Nearly Optimal Planar k Nearest Neighbors Queries under General Distance Functions. In *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Proceedings, pages 2842–2859. Society for Industrial and Applied Mathematics, December 2019. doi:10.1137/1.9781611975994.173.

10 Micha Sharir. Intersection and Closest-Pair Problems for a Set of Planar Discs. *SIAM Journal on Computing*, 14(2):448–468, May 1985. doi:10.1137/0214034.

11 Haitao Wang and Jie Xue. Near-Optimal Algorithms for Shortest Paths in Weighted Unit-Disk Graphs. *Discrete & Computational Geometry*, 64(4):1141–1166, December 2020. doi:10.1007/s00454-020-00219-7.

12 Haitao Wang and Yiming Zhao. An Optimal Algorithm for L1 Shortest Paths in Unit-Disk Graphs. In Meng He and Don Sheehy, editors, *Proceedings of the 33rd Canadian Conference on Computational Geometry, CCCG 2021, August 10-12, 2021, Dalhousie University, Halifax, Nova Scotia, Canada*, pages 211–218, 2021.

13 Haitao Wang and Yiming Zhao. Reverse Shortest Path Problem for Unit-Disk Graphs. *arXiv:2104.14476 [cs]*, April 2021. arXiv:2104.14476.

An Insertion Strategy for Motorcycle Graphs*

Franz Aurenhammer[1] **and Michael Steinkogler**[2]

1 **Institute for Theoretical Computer Science, University of Technology, Graz, Austria,** auren@igi.tugraz.at

2 **Institute for Theoretical Computer Science, University of Technology, Graz, Austria,** michael_steinkogler@gmx.net

Abstract

An insertion method is proposed that leads to a simple and experimentally fast algorithm for constructing the motorcycle graph of a planar polygon.

1 Introduction

In [1] the *straight skeleton* was introduced to computational geometry as a new skeletal structure for polygons. In fact, the concept was already discussed as early as 1877 [11], as an alternative to the medial axis of a polygon. Unlike the medial axis, the straight skeleton is not (and cannot be, in a certain sense) defined via distances from the polygon boundary, but rather results from moving inwards the polygon edges in a self-parallel way. Thereby, the polygon undergoes certain combinatorial and topological changes: Edges might shrink to length zero, and the polygon might even split multiple times. The vertices of the shrinking polygon trace out a skeletal structure in the interior of the polygon that consists of straight-line segments and is therefore called its straight skeleton.

The complexity of computing the straight skeleton mainly stems from the interaction of the reflex vertices, which move at individual speeds during the shrinking process (depending on the interior polygon angles), and are responsible for possible polygon splits. Eppstein et al. abstracted this interaction into a separate problem and called it the *motorcycle graph* problem in [6]. See Figure 1 and Figure 2 for examples. In its general form, n 'motorcycles' start from n given points in the plane, at individual but constant velocities and in different directions. Each motorcycle leaves a trace where other motorcycles, when happening to run into it, crash and stop their travel. The union of the traces of all motorcycles constitutes the so-called motorcycle graph.

Over the years, various methods for computing both the straight skeleton and the motorcycle graph have been proposed; see e.g. [3, 4, 6, 10]. Notably, most known fast (i.e., subquadratic) straight skeleton algorithms build upon precomputing the motorcycle graph. They typically simulate the behaviour of the motorcycles over time, handling all motorcycles simultaneously and in chronological order—the simplest algorithm just computing all intersections between traces and placing them in a priority queue ordered by time.

In the present note we deviate from this pattern and apply randomized insertion to the motorcycles. A simple and practical algorithm is obtained, and our empirical results indicate competitiveness to other implemented algorithms like in [8].

* Supported by Project I 5270-N, Austrian Science Fund (FWF).

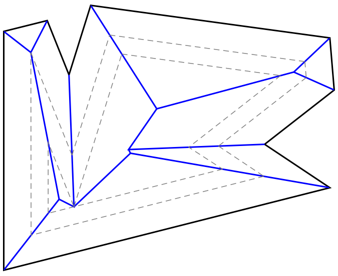


Figure 1 Straight skeleton of a polygon. The dashed polygon offsets indicate its shrinking process.

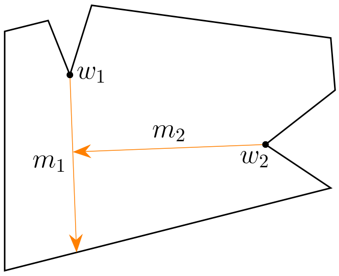


Figure 2 The corresponding motorcycle graph is defined by two motorcycles m_1 and m_2 that start at the reflex vertices w_1 and w_2, respectively.

2 Motorcycle Insertion

Let $\mathcal{P}$ be a simple polygon with n vertices, and denote with $W = \{w_1, \ldots, w_r\}$ the set of its $r < n$ reflex vertices, in an arbitrary but fixed order. During the shrinking process, where each edge of $\mathcal{P}$ is assumed to move inwards at unit speed, a reflex vertex $w_i \in W$ moves at speed $\frac{1}{\sin \alpha/2}$, where α is the interior angle of $\mathcal{P}$ at w_i. Note that w_i moves along the angle bisector of α and thus has a straight trajectory. We associate with w_i a motorcycle m_i, which starts at w_i and drives with its speed and direction, leaving behind a straight trace. We say that a motorcycle m_i crashes when it runs into the trace of another motorcycle or when it hits the boundary of $\mathcal{P}$. Our interest is in the resulting motorcycle graph for $\mathcal{P}$.

For a subset $W_k = \{w_1, \ldots w_k\}$ of reflex vertices of $\mathcal{P}$, let us denote with $\mathcal{M}(W_k)$ the resulting partial motorcycle graph, where only the motorcycles associated with vertices in W_k move. We now investigate what happens when we 'insert' the next motorcycle, m_{k+1}, and how $\mathcal{M}(W_k)$ has to be updated to obtain $\mathcal{M}(W_{k+1})$.

In the simplest case we have $\mathcal{M}(W_k) \subset \mathcal{M}(W_{k+1})$, that is, the structure of $\mathcal{M}(W_k)$ does not change, because m_{k+1} either crashes at the boundary of $\mathcal{P}$, or at some edge of $\mathcal{M}(W_k)$ whose associated motorcycle m_j, $j \leq k$, has reached the crossing point first. If, however, m_{k+1} is the motorcycle which reaches there first (let us denote the respective point with q), then there are potentially significant structural changes from $\mathcal{M}(W_k)$ to $\mathcal{M}(W_{k+1})$ that need to be incorporated.

In the latter case, up to the time t when m_{k+1} reaches the point q, all crashes that happen in $\mathcal{M}(W_k)$ obviously also happen in $\mathcal{M}(W_{k+1})$. Thus, up to that time the graphs $\mathcal{M}(W_k)$ and $\mathcal{M}(W_{k+1})$ are structurally identical, apart from the trace of m_{k+1}.

Let us assume that $\mathcal{M}(W_k)$ has already been constructed. To compute the changes from $\mathcal{M}(W_k)$ to $\mathcal{M}(W_{k+1})$, we adapt the motorcycle graph algorithm introduced by Cheng and Vigneron in [4]. They use a certain partition of the plane to keep track of the motorcycles over time, and search for crashes of motorcycles only within the cells of the partition. Their algorithm has three kinds of events: *boundary events* where a motorcycle crashes into the boundary of $\mathcal{P}$, *collision events* when a motorcycle runs into the trace of another motorcycle, and *switch events* when a motorcycle crosses over some cell boundary into the next cell.

In contrast to [4], we simply use the already available partial motorcycle graph $\mathcal{M}(W_k)$ as the underlying partition of $\mathcal{P}$, and limit the events to those needed to compute the changes from $\mathcal{M}(W_k)$ to $\mathcal{M}(W_{k+1})$. This simplifies the algorithm and gives hope for a speed-up. We keep the necessary events in a time-ordered queue $\mathcal{Q}$. Also, for each cell C of $\mathcal{M}(W_k)$ we maintain a set $A(C)$ of active motorcycles, i.e., those that are currently passing, or have already passed, through C. Finally, for each motorcycle m we keep its current death time $d(m)$, that is, the time when m crashes in the motorcycle graph constructed so far. Note that the edges of $\mathcal{M}(W_k)$ act as both cell boundaries and motorcycle traces, so collision and switch events can happen at the same time. In such a case the collision event is processed first.

We start our event handling algorithm with only one moving motorcycle, m_{k+1}. We initialize $\mathcal{Q}$ with a switch event for m_{k+1} at time 0, set $d(m_{k+1}) = \infty$, and put $A(C) = \emptyset$ for all cells C of $\mathcal{M}(W_k)$. As long as $\mathcal{Q}$ contains events, we remove the next event and process it as is described below.

2.1 Event handling

Boundary event of motorcycle m at time t. If m is alive at time t, that is, if $d(m) \geq t$, then we report the motorcycle edge $\overline{wq}$, where w is the reflex vertex that m started from, and q is the point where m hits the polygon boundary. In addition, we put $d(m) = t$. If m was already dead at time t there is nothing to do.

Collision event of motorcycle m at time t, with the trace of motorcycle m' at point q. Let m' reach q at time t', and observe that $t' < t$ has to hold, because the collision event would not affect m otherwise. If $d(m) < t$ or $d(m') < t'$, then there is nothing to do: Either both motorcycles crashed already, or m can just drive on. If both motorcycles are alive when they reach q, then m crashes at q, and we report the (now shortened) motorcycle edge $\overline{wq}$ and set $d(m) = t$. If $m \neq m_{k+1}$ (that is, if $m \in W_k$), then we additionally need to activate all motorcycles $\widetilde{m}$ that got blocked by m (after m passed q) and that have $d(\widetilde{m}) > t$. See Figures 3 and 4 for illustrations. For all such motorcycles $\widetilde{m}$ we do the following: Put $d(\widetilde{m}) = \infty$, and insert into $\mathcal{Q}$ a switch event of $\widetilde{m}$ crossing the edge defined by m to reach the next cell, say C'. (This switch event will then add $\widetilde{m}$ to $A(C')$.)

Switch event of motorcycle m at time t, from cell C to cell C' at q. Let e be the edge of C that q lies on. This edge e stems from some motorcycle m_j for some $j \leq k$. If m has crashed already then there is nothing to do, so let us suppose $d(m) > t$. Then m drives on to C' (the collision event with m_j was already handled and would have caused $d(m) \leq t$, otherwise), so we add m to $A(C')$, compute collision events of m with all other motorcycles in $A(C')$ (in a straight-forward manner), and determine the next switch event in case m leaves C'. All these events are inserted into $\mathcal{Q}$.

2.2 Correctness

Correctness of our approach mainly follows from the correctness of the Cheng-Vigneron algorithm [4]. The only difference is that our algorithm avoids creating and processing certain events because $\mathcal{M}(W_k)$ is used as the underlying polygon partition.

Consider all collision events for the motorcycle graph $\mathcal{M}(W_k)$, as well as all the switch events for the motorcycles $m_1 \ldots, m_{k+1}$. These events are precomputed in [4], as opposed to our on-demand approach. Our algorithm processes the same events, except that we need not process the collision events for $\mathcal{M}(W_k)$ because we already know the result of these events (namely, $\mathcal{M}(W_k)$ itself)—if these events are relevant at all (motorcycles may crash earlier in $\mathcal{M}(W_{k+1})$). Also, computing switch events on demand—when a motorcycle enters a new

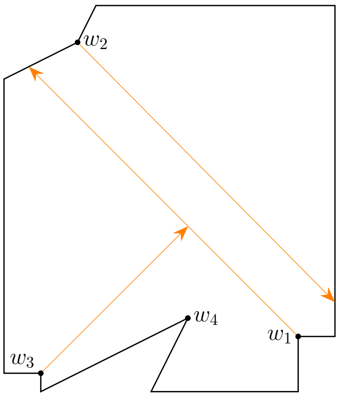


Figure 3 Partial motorcycle graph before the insertion of motorcycle m_4.

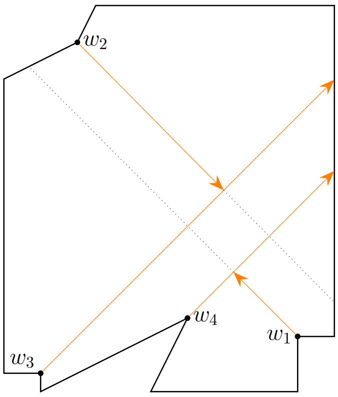


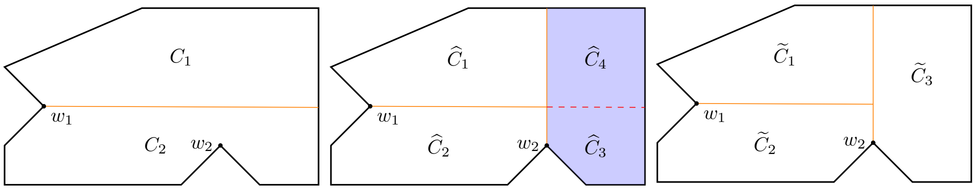
Figure 4 Insertion of m_4 makes m_1 crash. This activates m_3 which is now blocking m_2.

cell—for m_{k+1} and for the unblocked motorcycles of $\mathcal{M}(W_k)$ changes neither the events nor their processing. Thus, our algorithm and the algorithm of [4] produce the same result.

2.3 Partition representation

After processing all the events caused by the insertion of the motorcycle m_{k+1}, we have to update the polygonal partition given by $\mathcal{M}(W_k)$. When a new motorcycle graph edge is reported (in a boundary event or in a collision event) then the respective cell of the partition needs to be split into two cells. Also, when a motorcycle gets blocked and its motorcycle graph edge gets shortened (in a collision event) then the respective two cells per motorcycle have to be merged into one cell. Figure 5 gives an illustration. In addition, in order to determine switch events and boundary events, ray shooting queries in the interior of cells have to be performed. Finally, to get started at all, we need to locate the cell that m_{k+1} starts from.

If we aim for a theoretically efficient algorithm, the data structure of Goodrich and Tamassia [7] is the right choice. They organize a polygonal partition of the plane into an $O(n)$-space dynamic data structure that supports insertion and deletion of edges, as well as point location and ray shooting queries, in $O(\log^2 n)$ time. From the practical point of view, assuming that cell sizes will decrease quickly with progressing motorcycle insertion, it may suffice to store $\mathcal{M}(W_k)$ in a doubly-connected edge list data structure, and to simply scan cell boundaries to perform the necessary operations.



(a) Before inserting m_2.

(b) C_1 and C_2 are split.

(c) $\widehat{C}_3$ and $\widehat{C}_4$ are merged.

Figure 5 Updating the cell structure of the motorcycle graph.

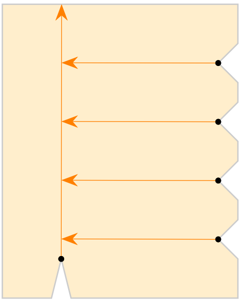


Figure 6 The insertion of one motorcycle can block all the others.

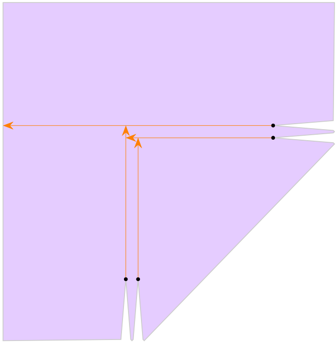


Figure 7 Polygon with $\Theta(n^2)$ motorcycle intersections.

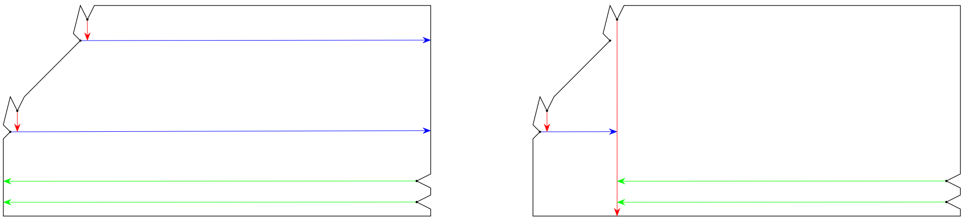


Figure 8 Polygon angles can be adjusted such that removal of any of the $\Theta(n)$ blue motorcycles unblocks its associated red motorcycle, which in turn blocks the $\Theta(n)$ green motorcycles.

3 Complexity Considerations

Insertion strategies commonly suffer from the fact that a single insertion step can be costly. So does ours, where the insertion of a single motorcycle m_{k+1} can cause $\Theta(k)$ structural changes in the graph $\mathcal{M}(W_k)$. The polygon in Figure 6 is an example, when we insert the motorcycle starting from the bottommost reflex vertex in the end. Even worse, there are n-vertex polygons where the insertion of any motorcycle out of a set of size $\Theta(n)$ leads to a structural change of complexity $\Theta(n)$; see the construction in Figure 8. This buries the hope of proving a *randomized* insertion strategy efficient by means of backwards analysis [9]. On the other hand, experiments indicate that randomized insertion *is* efficient in our case when summing up over all insertion steps, and leads to a favorable runtime.

To test the performance of our algorithm in practice, we counted the total number of structural changes in the motorcycle graph during the randomized incremental construction. Both generic random polygons (provided by the Salzburg Database of Geometric Inputs [5]) and specifically designed polygons were used. Interestingly, the number of structural changes per insertion step can on average be bounded by a very small constant. For example, none of the tested polygons from the Salzburg dataset with up to 50.000 reflex vertices caused more than 3 structural changes per motorcycle insertion on average. See Figures 9 and 10. Among our many specifically designed polygons are the types as shown in Figures 7 and 8. The latter exhibited the worst behavior, but still with less than 7 structural changes on average; see Figure 11.

In summary, the insertion strategy enables a quick and simple construction of motorcycle graphs in practice. Together with the simple motorcycle-graph-based skeleton merging algorithm in [2], we obtain a new practical method for computing straight skeletons.

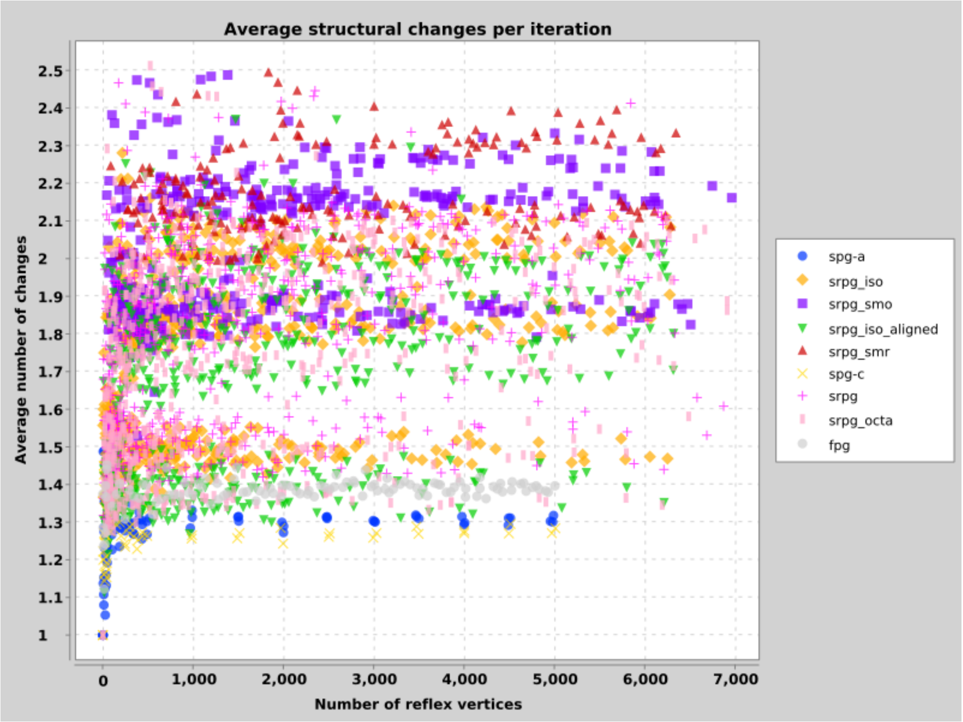


Figure 9 Average number of motorcycle graph edges that changed per insertion step. (For polygons from [5] with up to 7000 reflex vertices).

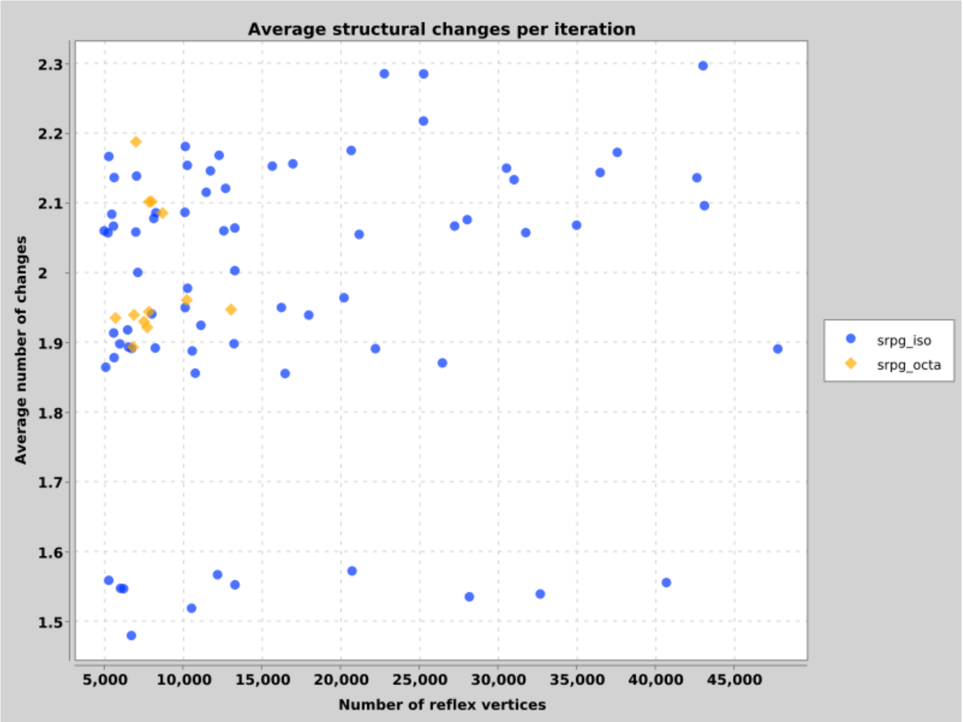


Figure 10 Average number of changing motorcycle graph edges per insertion step, for larger polygons from [5].

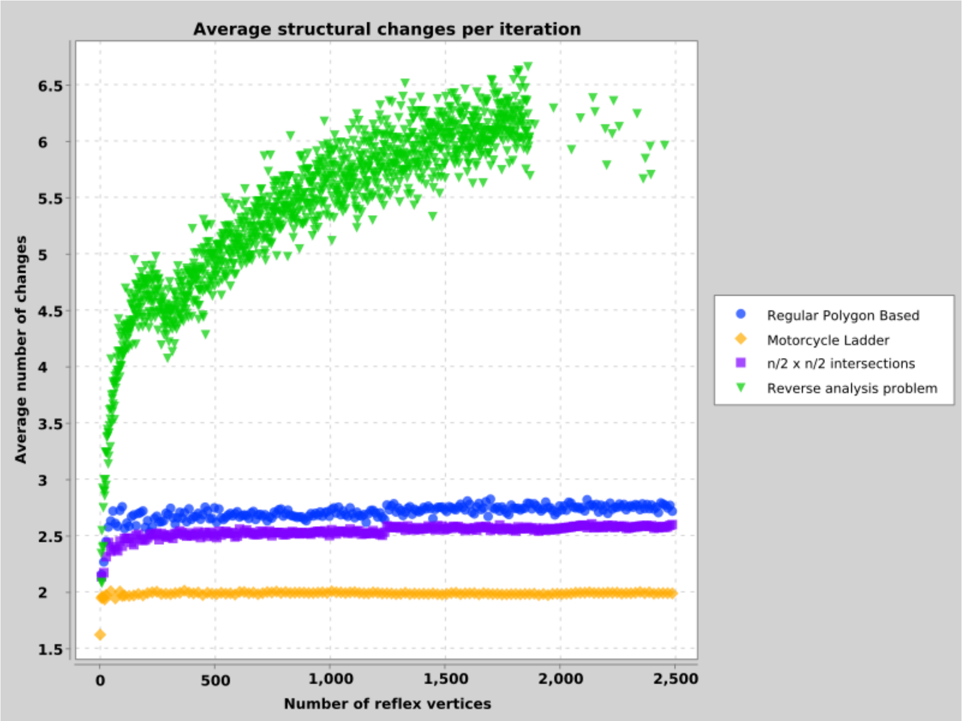


Figure 11 Average number of changing motorcycle graph edges per insertion step, for specifically constructed polygons.

References

1 Oswin Aichholzer, Franz Aurenhammer, David Alberts, and Bernd Gärtner. A novel type of skeleton for polygons. *Journal of Universal Computer Science*, 1:752–761, 1996.

2 Franz Aurenhammer and Michael Steinkogler. On merging straight skeletons. In *Proc. 34th European Workshop on Computational Geometry*, 2018.

3 Siu-Wing Cheng, Liam Mencel, and Antoine Vigneron. A faster algorithm for computing straight skeletons. *ACM Transactions on Algorithms*, 12(3):44:1–44:21, 2016.

4 Siu-Wing Cheng and Antoine Vigneron. Motorcycle graphs and straight skeletons. *Algorithmica*, 47(2):159–182, 2007.

5 Günther Eder, Martin Held, Steinþór Jasonarson, Philipp Mayer, and Peter Palfrader. Salzburg Database of Polygonal Data: Polygons and Their Generators. *Data in Brief*, 31:105984, August 2020. doi:10.1016/j.dib.2020.105984.

6 David Eppstein and Jeff Erickson. Raising roofs, crashing cycles, and playing pool: Applications of a data structure for finding pairwise interactions. *Discrete & Computational Geometry*, 22(4):569–592, 1999.

7 Michael Goodrich and Roberto Tamassia. Dynamic ray shooting and shortest paths in planar subdivisions via balanced geodesic triangulations. *Journal of Algorithms*, 23:51–73, 1997.

8 Stefan Huber and Martin Held. Motorcycle graphs: stochastic properties motivate an efficient yet simple implementation. *Journal of Experimental Algorithmics*, 16:1–1, 2011.

9 Raimund Seidel. Backwards analysis of randomized geometric algorithms. In *J.Pach (ed.) New Trends in Discrete and Computational Geometry, Algorithms and Combinatorics*, pages 37–67. Springer, 1993.

10 Antoine Vigneron and Lie Yan. A faster algorithm for computing motorcycle graphs. *Discrete & Computational Geometry*, 52(3):492–514, 2014.

11 Gustav von Peschka. *Kotirte Ebenen und deren Anwendung.* Buschak & Irrgang, 1877.

k-Transmitter Watchman Routes*

Bengt J. Nilsson[1] **and Christiane Schmidt**[2]

1 **Department of Computer Science and Media Technology, Malmö University, Sweden**
`bengt.nilsson.TS@mau.se`

2 **Communications and Transport Systems, ITN, Linköping University, Sweden**
`christiane.schmidt@liu.se`

Abstract

We show that even in simple polygons the shortest k-transmitter watchman route problem for a discrete set of points $S \subset P$ is NP-complete and cannot be approximated to within a logarithmic factor (unless P=NP), both with and without a given starting point. Moreover, we present a polylogarithmic approximation for the k-transmitter watchman route problem for a given starting point and $S \subset P$ with approximation ratio $O\big(\log^2(|S| \cdot n) \log\log(|S| \cdot n) \log |S|\big)$ with $|P| = n$.

1 Introduction

In the classical *Watchman Route Problem* (WRP)–introduced by Chin and Ntafos [6], we ask for the shortest (closed) route in an environment (usually a polygon, P), such that a mobile guard traveling along this route sees all points of the environment. Classically, visibility means that $p \in P$ sees $q \in P$ if the direct line connection $\overline{pq}$ is fully contained in P. The WRP is solvable in polynomial time in simple polygons with [7, 12, 8] and without [5, 11] a given boundary start point. In polygons with holes, the WRP is NP-hard [6].

Another type of visibility is motivated by modems: while one wall may allow connection to a modem, many walls separating our location from the modem result in a failed connection. This is captured by so-called k-transmitters: $p \in P$ sees $q \in P$ if the direct line connection $\overline{pq}$ intersects P's boundary at most k times. Thus, k is always an even integer. Aichholzer and others [1, 2, 4] present so-called Art Gallery theorems for k-transmitters in different polygon classes; Cannon et al. [4] established NP-hardness of minimum k-transmitter cover, for $k \geq 2$; Biedl et al. [3] considered so-called sliding k-transmitters.

Of course, k-transmitters do not have to be stationary: we might have to find a shortest tour such that a mobile k-transmitter traveling along this route can establish a connection with all (or a discrete subset of the) points of an environment, the *WRP with a k-transmitter*.

2 Notation and Preliminaries

We let P be a polygon, $\partial(P)$ is the boundary of P, and let n denote the number of vertices of P. A point $q \in P$ is *k-visible* to a k-transmitter $p \in \mathbb{R}^2$ if $\overline{qp}$ intersects P's boundary in at most k connected components. For a point $p \in P$, we define the *k-visibility region* of p, $k\mathrm{VR}(p)$, as the set of points in P that are k-visible from p. For a point set $S \subseteq P$: $k\mathrm{VR}(S) = \bigcup_{p \in S} k\mathrm{VR}(p)$. A k-visibility region can have $O(n)$ connected components (CCs), see [4], we denote these components by $k\mathrm{VR}^j(p), j = 1, \ldots, J_p$, with $J_p \in O(n)$.

* Supported by grant 2021-03810 (Illuminate: bevisbart goda algoritmer för bevakningsproblem) from the Swedish Research Council (Vetenskapsrådet).

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

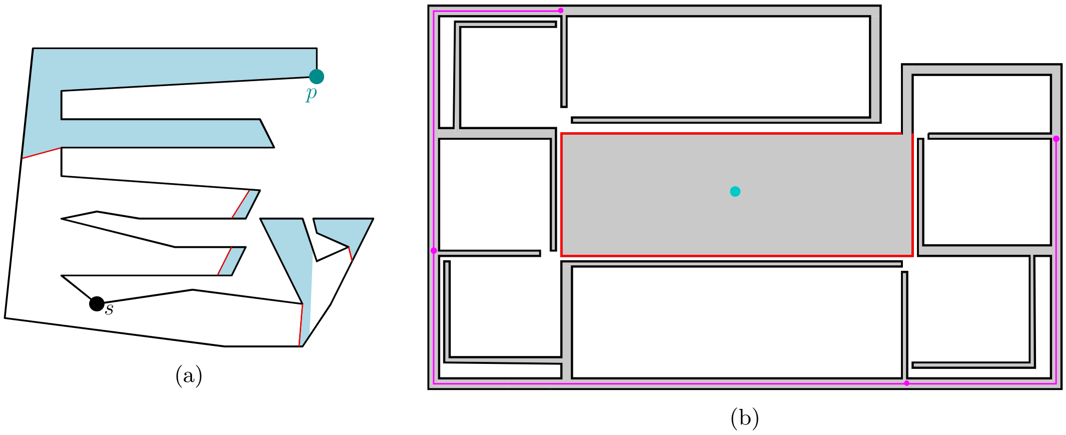


Figure 1 (a): Point p with its 2-visibility region shown in light blue, 2VR(p) has five CCs. The cuts of these CCs w.r.t. s are shown in red. (b): Polygon P: $\partial(P)$ is 2-visible from the pink route (e.g., red edges from the pink points); the aqua point is not 2-visible from that route. Thus, not all of P is 2-visible from that route.

The boundary of each CC of kVR(p) has straight edges that connect (parts of) edges of $\partial(P)$ but where the interior of the edge lies in the interior of P. These are called *windows*. For each window w, denote by $P_s(w)$ the subpolygon of P that contains a given point s and the window w itself. A window w_1 *dominates* another window w_2 if $P_s(w_2) \subset P_s(w_1)$. A window w is *essential* if it is not dominated by any other window. For a given point s in a simple polygon P not in kVR(p) for any $p \in S$, there exists one window w per CC $k\text{VR}^j(p)$, such that any path from s to a window $w' \neq w, w' \in k\text{VR}^j(p)$ intersects w, that is, $P_s(w) \subseteq P_s(w')$. We denote this window as the *cut* of $k\text{VR}^j(p)$; see Figure 1(a).

We aim to find shortest watchman routes for k-transmitters. In particular, we aim to find a shortest route R, such that either all points of a polygon P or a set of points $S \subset P$ is k-visible for the k-transmitter watchman following R, that is, either $k\text{VR}(R) = P$ or $k\text{VR}(R) \supseteq S$. We define the *$k$-Transmitter WRP for S and P*, k-TrWRP(S, P), possibly with a given starting point s, k-TrWRP(S, P, s), as the problem of finding the shortest route for a k-transmitter within P, starting at s, from which every point in S is k-visible.

In Section 4, we use an approximation algorithm by Garg et al. [10] for the group Steiner tree problem to solve the k-TrWRP(S, P, s) problem: given a graph $G = (V, E)$ with cost function $c : E \to \mathbb{R}^+$ and subsets of vertices $\gamma_1, \gamma_2, \ldots, \gamma_Q \subseteq V$—so-called *groups*, we aim to find a connected subgraph $T = (V', E')$ that minimizes $\sum_{e \in E'} c_e$ such that $V' \cap \gamma_q \neq \emptyset$, $\forall q \in \{1, \ldots, Q\}$. For $|V| = m$, Garg et al. [10] obtained a randomized algorithm with an approximation ratio of $O(\log^2 m \log\log m \log Q)$.

▶ Observation 1. For $k \geq 2$ and a simple polygon P, k-visibility of $\partial(P)$ by a watchman does not imply P being k-visible to the watchman, see Figure 1(b).

3 Computational Complexity

▶ **Theorem 3.1.** *For a discrete set of points S and a simple polygon P, k-TrWRP(S, P) does not admit a polynomial-time approximation algorithm with approximation ratio $c \cdot \ln |S|$, for $k \geq 2$, unless* P=NP.

Proof sketch. We give a gap-preserving reduction from Set Cover (SC): given a set system $(\mathcal{U}, \mathcal{C})$, with $\cup_{C \in \mathcal{C}} C = \mathcal{U}$, find a minimum cardinality sub-family $\mathcal{B} \subseteq \mathcal{C}$ that covers $\mathcal{U}$.

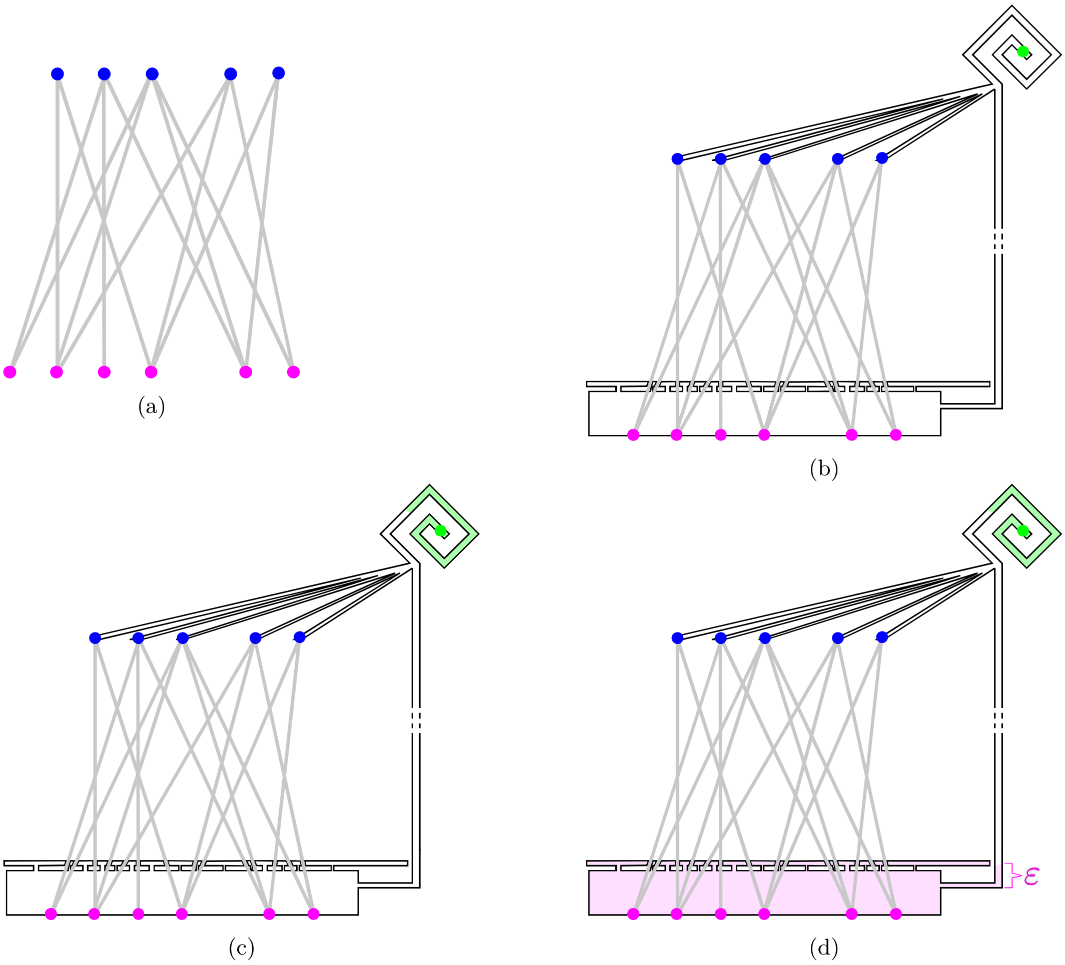


Figure 2 Example construction for the SC instance $(\mathcal{U},\mathcal{C})$ with $\mathcal{U} = \{1,2,3,4,5,6\}$, $\mathcal{C} = \{\{2,4\},\{1,3,5\},\{1,2,5,6\},\{2,4,6\},\{4,5\}\}$. (a) Graph G, (b) polygon P with v shown in green, (c) $k\text{VR}(v)$ in light green, (d) set of points that are not located in the $|\mathcal{C}|$ spikes and see all $u \in \mathcal{U}$.

Given an instance of the SC problem, we construct a polygon P with $S = \mathcal{U} \cup \{v\}$. For the construction, we build a bipartite graph G with vertex set $V(G) = \mathcal{U} \cup \mathcal{C}$ and edge set $E(G) = \{e = (u, c) \mid u \in \mathcal{U}, c \in \mathcal{C}\}$, see Figure 2(a). We construct a spiral with $v \in S$ located in its center, to its end we attach $|\mathcal{C}|$ spikes, each ending at the same y-coordinate. Let the length of the longest spike be $\ell_{\mathcal{C}}$, and let the length of the spikes differ by $\varepsilon' \ll \ell_{\mathcal{C}}$ only. All points $u \in \mathcal{U}$ are located in a long horizontal box to which T-shaped structures are attached, such that the crossbeams leave gaps only where an edge in $E(G)$ connects a c from a spike to a u in the horizontal box. These two polygon parts are connected by a very long vertical polygonal corridor of length $\ell_{\text{vert}} = 4 \cdot |\mathcal{C}| \cdot \ell_{\mathcal{C}} + \varepsilon$ ensuring that any intersections between edges from $E(G)$ occur outside the polygon. See Figure 2 for the construction. By Feige [9], SC cannot be approximated to within a factor $(1 - o(1)) \ln |\mathcal{U}|$, with $|S| = |\mathcal{U}| + 1$.

▶ **Corollary 3.2.** *For S and P as in Theorem 3.1, $s \in P$, k-*TrWRP(S, P, s) *does not admit a polynomial-time approximation algorithm with approximation ratio $c \cdot \ln |S|$, for $k \geq 2$, unless* P=NP.

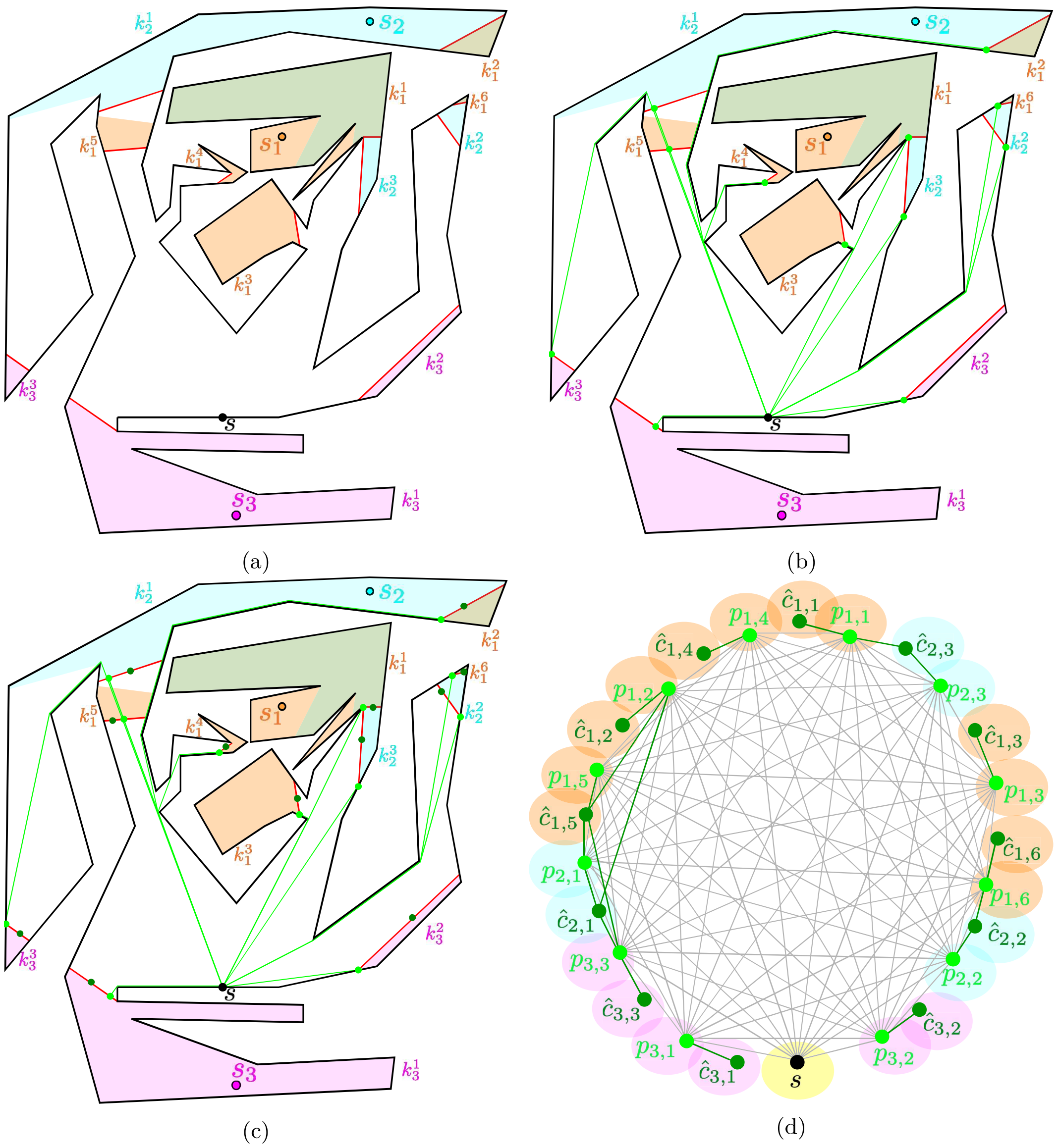
4 Approximation Algorithm for k-TrWRP(S, P, s)

▶ **Theorem 4.1.** *Let P be a simple polygon with n vertices. Let* OPT(S, P, s) *be the optimal solution for the k-*TrWRP(S, P, s) *and let R be the solution output by our algorithm* ALG(S, P, s)*. Then R yields an approximation ratio of $O\big(\log^2(|S| \cdot n) \log\log(|S| \cdot n) \log |S|\big)$.*

Our approximation algorithm ALG(S, P, s) performs several steps:

1. For each $s_i \in S$, we compute the k-visibility region within P, kVR(s_i)—and say that all CCs $k\text{VR}^j(s_i)$ have *color* s_i. We denote $k\text{VR}^j(s_i)$ as k_i^j. Let the cut of each k_i^j be denoted by $c_{i,j}$, and let $\mathcal{C}^{all}$ denote the set of all cuts in P. See Figure 3(a).
2. We compute a geodesic $g_{i,j}$ from s to each cut $c_{i,j}$. Let $p_{i,j}$ be the point where $g_{i,j}$ intersects $c_{i,j}$. See Figure 3(b), the $p_{i,j}$ are shown in light green.
3. We build a complete graph on the $p_{i,j}$ and s: for an edge $\{x, y\}$, we have $\text{cost}(\{x, y\}) = \text{geodesic}_P(x, y)$. We introduce further vertices and edges: one vertex $\hat{c}_{i,j}$ per cut $c_{i,j} \in \mathcal{C}^{all}$. We add edges $\{p_{i,j}, \hat{c}_{i,j}\}$ with edge cost 0, and edges $\{p_{i,j}, \hat{c}_{i',j'}\}$ with edge cost 0 for all cuts $c_{i',j'}$ that $g_{i,j}$ intersects. (Rationale: any path or tour visiting $p_{i,j}$ must visit $c_{i',j'}/c_{i,j}$.) Let the resulting graph be denoted as $G = (V, E)$. See Figure 3(c)/(d): the points of type $\hat{c}_{i,j}$ are shown in green. We have $|V(G)| = O(n \cdot |S|)$.
4. With $\gamma_i = \bigcup_{j=1}^{J_i} p_{i,j} \cup \bigcup_{j=1}^{J_i} \hat{c}_{i,j}$, $\gamma_0 = s$, $Q = |S| + 1$, that is, each group γ_i contains all vertices in $V(G)$ of the color s_i, γ_0 contains the starting point that we must visit, we approximate the group Steiner tree problem on G, using the approximation by Garg et al. [10], the approximation ratio is $O\big(\log^2 |V(G)| \log\log |V(G)| \log |S|\big)$.
5. We double the resulting tree to make it a Eulerian graph and obtain a route R by taking a Eulerian tour of the graph shortcutting any repeated visits to nodes. R visits at least one vertex per color (one point in each kVR(s_i)). Thus, R is a feasible solution for k-TrWRP(S, P, s) visiting one point per γ_i. R is a polylog-approximation to the best tour that is feasible for k-TrWRP(S, P, s), visiting one point per γ_i using edges in G (denoted by $\text{OPT}_G(S, P, s)$).

To prove that R is indeed an approximation with the claimed approximation factor, we alter the optimum k-transmitter watchman route, OPT(S, P, s), (which we of course do not know in reality) to pass points that represent vertices of $V(G)$, and show that this new tour is at most 3 times as long as the optimum route. The basic idea is:



(a) (b) (c) (d)

Figure 3 Example for the idea of our approximation algorithm, $S = \{s_1, s_2, s_3\}$. (a)-(c): The cuts $c_{i,j}$ are shown in red, geodesics and all $p_{i,j}$ in light green, all $\hat{c}_{i,j}$ in green. The CCs of the visibility region of a point s_i are colored in a lighter shade of the same color as the point itself (orange for s_1, turquoise for s_2, and pink for s_3). Line segments are slightly offset to enhance visibility in case they coincide with polygon boundary. (d) Resulting graph G. Edges with edge cost 0 and length of the geodesic between two points are shown in green and in gray, respectively. We highlight each vertex representing a point in k_i^j in s_i—vertices of one color constitute the set γ_i.

a. We identify all cuts of the $k\mathrm{VR}(s_i)$ that $\mathrm{OPT}(S,P,s)$ visits, let these be the set $\mathcal{C}$ ($\mathcal{C} \subseteq \mathcal{C}^{all}$). Let $o_{i,j}$ denote the point where $\mathrm{OPT}(S,P,s)$ visits $c_{i,j}$ (for the first time).
b. We identify the subset of essential cuts $\mathcal{C}' \subseteq \mathcal{C}$.
c. We order the geodesics to the essential cuts $\mathcal{C}'$ by decreasing length: $\|g_1\| \geq \|g_2\| \geq \cdots \geq \|g_{|\mathcal{C}'|}\|$, to perform the next step so that the properties of Lemma 4.2 hold.
d. $\mathcal{C}'' \leftarrow \mathcal{C}'$; FOR $t = 1$ TO $|\mathcal{C}'|$, we identify all $\mathcal{C}_t \subset \mathcal{C}'$ that g_t intersects, and set $\mathcal{C}'' \leftarrow \mathcal{C}'' \backslash \mathcal{C}_t$. $\mathcal{C}'' \subseteq \mathcal{C}'$. We denote the set of these final geodesics as $\mathcal{G}_{\mathcal{C}''}$.
e. The geodesics in $\mathcal{G}_{\mathcal{C}''}$ constitute a set of *independent* geodesics (no essential cut is visited by two of these geodesics). Moreover, each essential cut visited by $\mathrm{OPT}(S,P,s)$, i.e., every cut in $\mathcal{C}'$, is touched by exactly one of the geodesics.
f. The geodesics in $\mathcal{G}_{\mathcal{C}''}$ intersect the cuts in $\mathcal{C}''$ in points of the type $p_{i,j}$, points that represent vertices of $V(G)$. We denote the set of all these points as $\mathcal{P}_{\mathcal{C}''}$ ($\mathcal{P}_{\mathcal{C}''} \subseteq \{p_{i,j} \mid i = 1,\ldots,|S|, j = 1,\ldots,J_i\}$).
g. We build the relative convex hull of all $o_{i,j}$ and all points in $\mathcal{P}_{\mathcal{C}''}$ (relative w.r.t. the polygon P). We denote this relative convex hull by $\mathrm{CH}_P(\mathrm{OPT},\mathcal{P}_{\mathcal{C}''})$.
h. Because we have a set of independent geodesics, no geodesic can intersect $\mathrm{CH}_P(\mathrm{OPT},\mathcal{P}_{\mathcal{C}''})$ between a point $o_{i,j}$ and a point $p_{i,j}$ on the same cut. Thus, between any pair of points of the type $o_{i,j}$ on $\mathrm{CH}_P(\mathrm{OPT},\mathcal{P}_{\mathcal{C}''})$, we have at most two points of $\mathcal{P}_{\mathcal{C}''}$. We show that $\mathrm{CH}_P(\mathrm{OPT},\mathcal{P}_{\mathcal{C}''})$ has length of at most three times $|\mathrm{OPT}(S,P,s)|$.
i. The relative convex hull of the points in $\mathcal{P}_{\mathcal{C}''}$, $\mathrm{CH}_P(\mathcal{P}_{\mathcal{C}''})$, is not longer than $\mathrm{CH}_P(\mathrm{OPT},\mathcal{P}_{\mathcal{C}''})$, and we show that $\mathrm{CH}_P(\mathcal{P}_{\mathcal{C}''})$ visits at least one point per γ_i (except for γ_0).
j. Because s ($= \gamma_0$) might be located in the interior of $\mathrm{CH}_P(\mathcal{P}_{\mathcal{C}''})$, we need to connect s to $\mathrm{CH}_P(\mathcal{P}_{\mathcal{C}''})$. This costs at most $\|\mathrm{OPT}(S,P,s)\|$.
k. Thus, we obtain:

$$\begin{aligned}\|R\| &\in O\big(f(|V(G)|,|S|)\big)\cdot\|\mathrm{OPT}_G(S,P,s)\| \subseteq O\big(f(n|S|,|S|)\big)\cdot\|\mathrm{CH}_P(\mathcal{P}_{\mathcal{C}''})\|\\ &\subseteq O\big(f(n|S|,|S|)\big)\cdot\|\mathrm{CH}_P(\mathrm{OPT},\mathcal{P}_{\mathcal{C}''})\| \subseteq O\big(f(n|S|,|S|)\big)\cdot\|\mathrm{OPT}(S,P,s)\|,\end{aligned}$$

where $f(N,M) = \log^2 N \log\log N \log M$.

Hence, to show Theorem 4.1, we need to prove steps e, h, and i. We show step e using Lemma 4.2; step h using Lemmas 4.3, 4.4, and 4.5; and step i using Lemmas 4.6, 4.7, 4.8, and 4.9. (We omit proofs for Lemmas 4.2, 4.5, 4.6, 4.7 and 4.9.)

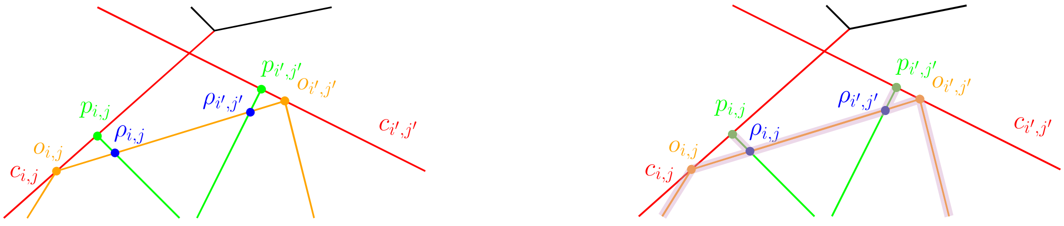
▶ **Lemma 4.2.** *For the geodesics in $\mathcal{G}_{\mathcal{C}''}$, we have:*

I. The geodesics in $\mathcal{G}_{\mathcal{C}''}$ are independent, i.e., no cut in $\mathcal{C}'$ is visited by two of these geodesics.
II. Each cut in $\mathcal{C}'$ is visited by a geodesic in $\mathcal{G}_{\mathcal{C}''}$.

▶ **Lemma 4.3.** *Consider a cut $c \in \mathcal{C}''$ (of CC j of a k-visibility region for $s_i \in S$, $k\,VR^j(s_i)$) for which both the point $o_{i,j}$ and the point $p_{i,j}$ are on $\mathrm{CH}_P(\mathrm{OPT},\mathcal{P}_{\mathcal{C}''})$. No geodesic in $\mathcal{G}_{\mathcal{C}''}$ intersects c between $o_{i,j}$ and $p_{i,j}$.*

Proof. Assume that there exists a geodesic $g_{c'} \in \mathcal{G}_{\mathcal{C}''}$ to a cut $c' \neq c, c' \in \mathcal{C}''$ that intersects c between $o_{i,j}$ and $p_{i,j}$. Let c' be the cut of $k\mathrm{VR}^{j'}(s_{i'})$. Let p_c denote the point in which $g_{c'}$ intersects c. If $\|g_{c'}\| > \|g_c\|$, we would have deleted g_c in step d, hence $c \notin \mathcal{C}''$. If $\|g_{c'}\| < \|g_c\|$, the geodesic to c' restricted to the part between s and p_c, $g_{c'[s;p_c]}$, is shorter than g_c, a contradiction to g_c being the geodesic to c. If $\|g_{c'}\| = \|g_c\|$, either $\|g_{c'[s;p_c]}\| < \|g_{c'}\| = \|g_c\|$ or (if p_c on c') $p_{i,j} = p_c$ and the claim holds. ◀

▶ **Lemma 4.4.** *Between any pair of points of the type $o_{i,j}$ on $\mathrm{CH}_P(\mathrm{OPT},\mathcal{P}_{\mathcal{C}''})$, we have at most two points in $\mathcal{P}_{\mathcal{C}''}$.*



(a) (b)

Figure 4 The cuts $c_{i,j}$ and $c_{i',j'}$ are shown in red; $o_{i,j}$, $o_{i',j'}$ and $\text{OPT}(S, P, s)$ are shown in orange; $p_{i,j}$, $p_{i',j'}$, $g_{i,j}$, and $g_{i',j'}$ are shown in green; $\rho_{i,j}$ and $\rho_{i',j'}$ are shown in blue; and T is shown in pink. A part of P's boundary is shown in black.

Proof. Let $o_{i,j}$ and $o_{i',j'}$ bet two consecutive points from OPT on $\text{CH}_P(\text{OPT}, \mathcal{P}_{\mathcal{C}''})$. By Lemma 4.3, $p_{i,j}$ and $p_{i',j'}$ can lie between $o_{i,j}$ and $o_{i',j'}$, but we can have no point $p_{\kappa,\lambda}$ between $o_{i,j}$ and $p_{i,j}$ or between $o_{i',j'}$ and $p_{i',j'}$. Assume that there exists a point $p_{\kappa,\lambda}$ between $p_{i,j}$ and $p_{i',j'}$ on $\text{CH}_P(\text{OPT}, \mathcal{P}_{\mathcal{C}''})$. Moreover, let $p_{i,j}$, $p_{i',j'}$, and $p_{\kappa,\lambda}$ be on cuts c, c' and c'', respectively. OPT visits $o_{\kappa,\lambda}$ on c''. As $o_{i,j}$ and $o_{i',j'}$ are consecutive points from OPT on $\text{CH}_P(\text{OPT}, \mathcal{P}_{\mathcal{C}''})$, OPT visits the three points either in order $o_{i,j}, o_{i',j'}, o_{\kappa,\lambda}$ or $o_{\kappa,\lambda}, o_{i,j}, o_{i',j'}$. W.l.o.g., assume the order $o_{i,j}, o_{i',j'}, o_{\kappa,\lambda}$. The cut c'' is a straight-line segment. Consider the convex polygon $P_\triangle$ with vertices $o_{i,j}, p_{i,j}, p_{\kappa,\lambda}, o_{\kappa,\lambda}, o_{i',j'}, o_{i,j}$. The point $p_{i',j'}$ must lie in $P_\triangle$'s interior. Moreover, $o_{i',j'}$ cannot lie on $\text{CH}_P(\text{OPT}, \mathcal{P}_{\mathcal{C}''})$; a contradiction. ◀

▶ **Lemma 4.5.** $\|\text{CH}_P(\text{OPT}, \mathcal{P}_{\mathcal{C}''})\| \leq 3 \cdot \|\text{OPT}(S, P, s)\|$

See Figure 4 for the proof idea.

▶ **Lemma 4.6.** $\|\text{CH}_P(\mathcal{P}_{\mathcal{C}''})\| \leq \|\text{CH}_P(\text{OPT}, \mathcal{P}_{\mathcal{C}''})\|$

▶ **Lemma 4.7.** *All points in $\mathcal{P}_{\mathcal{C}''}$ lie on their relative convex hull $\text{CH}_P(\mathcal{P}_{\mathcal{C}''})$.*

▶ **Lemma 4.8.** *$\text{CH}_P(\mathcal{P}_{\mathcal{C}''})$ visits all cuts in $\mathcal{C}$.*

Proof. We have $\mathcal{C} = \mathcal{C}'' \cup \{\mathcal{C}' \setminus \mathcal{C}''\} \cup \{\mathcal{C} \setminus \mathcal{C}'\}$. Cuts in $\{\mathcal{C} \setminus \mathcal{C}'\}$ are dominated by cuts in $\mathcal{C}'$. Thus, any tour visiting all cuts in $\mathcal{C}'$ must visit all cuts in $\{\mathcal{C} \setminus \mathcal{C}'\}$. Cuts in $\mathcal{C}''$ are visited by Lemma 4.7. Assume that there is a cut $c \in \{\mathcal{C}' \setminus \mathcal{C}''\}$ not visited by $\text{CH}_P(\mathcal{P}_{\mathcal{C}''})$. The cut c is not in $\mathcal{C}''$ (we filtered g_c out in step d): There exists a geodesic $g_{c_{i,j}} \in \mathcal{G}_{\mathcal{C}''}$ with $\|g_{c_{i,j}}\| \geq \|g_c\|$ that intersects c; $g_{c_{i,j}}$ visits $c_{i,j}$ in the point $p_{i,j}$. Thus, any tour that visits both s and $p_{i,j}$ must intersect c. By Lemma 4.7, $\text{CH}_P(\mathcal{P}_{\mathcal{C}''})$ is such a tour; a contradiction. ◀

▶ **Lemma 4.9.** *$\text{CH}_P(\mathcal{P}_{\mathcal{C}''})$ visits at least one point per γ_i (except for γ_0).*

References

1 Oswin Aichholzer, Ruy Fabila-Monroy, David Flores-Peñaloza, Thomas Hackl, Clemens Huemer, Jorge Urrutia, and Birgit Vogtenhuber. Modem illumination of monotone polygons. *Computational Geometry: Theory and Applications, Special Issue: Ferran Hurtado, in memoriam*, 2018.

2 Brad Ballinger, Nadia Benbernou, Prosenjit Bose, Mirela Damian, ErikD. Demaine, Vida Dujmović, Robin Flatland, Ferran Hurtado, John Iacono, Anna Lubiw, Pat Morin, Vera Sacristán, Diane Souvaine, and Ryuhei Uehara. Coverage with k-transmitters in the presence of obstacles. In Weili Wu and Ovidiu Daescu, editors, *Comb. Opt. and Appl.*, volume 6509 of *Lecture Notes in Computer Science*, pages 1–15. Springer Berlin Heidelberg, 2010. doi:10.1007/978-3-642-17461-2_1.

3 Therese Biedl, Timothy M. Chan, Stephanie Lee, Saeed Mehrabi, Fabrizio Montecchiani, Hamideh Vosoughpour, and Ziting Yu. Guarding orthogonal art galleries with sliding k-transmitters: Hardness and approximation. *Algorithmica*, 81(1):69–97, 2019. URL: https://doi.org/10.1007/s00453-018-0433-6, doi:10.1007/s00453-018-0433-6.

4 Sarah Cannon, Thomas G. Fai, Justin Iwerks, Undine Leopold, and Christiane Schmidt. Combinatorics and complexity of guarding polygons with edge and point 2-transmitters. *Computational Geometry*, 68:89 – 100, 2018. Special Issue in Memory of Ferran Hurtado. URL: http://www.sciencedirect.com/science/article/pii/S0925772117300470, doi:https://doi.org/10.1016/j.comgeo.2017.06.004.

5 Svante Carlsson, Håkan Jonsson, and Bengt J. Nilsson. Finding the shortest watchman route in a simple polygon. In *Algorithms and Computation, 4th International Symposium, ISAAC '93, Proceedings*, pages 58–67, 1993.

6 Wei-Pang Chin and Simeon Ntafos. Optimum watchman routes. In *SCG '86: Proceedings of the second annual symposium on Computational geometry*, pages 24–33, New York, NY, USA, 1986. ACM.

7 Wei-Pang Chin and Simeon C. Ntafos. Shortest watchman routes in simple polygons. *Discrete & Computational Geometry*, 6:9–31, 1991.

8 Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph S. B. Mitchell. Touring a sequence of polygons. In *In Proc. 35th Annu. ACM Sympos. Theory Comput*, pages 473–482. ACM Press, 2003.

9 Uriel Feige. A threshold of ln n for approximating set cover. *JOURNAL OF THE ACM*, 45:314–318, 1998.

10 Naveen Garg, Goran Konjevod, and R. Ravi. A polylogarithmic approximation algorithm for the group Steiner tree problem. *Journal of Algorithms*, 37(1):66–84, 2000. URL: https://www.sciencedirect.com/science/article/pii/S0196677400910964, doi:https://doi.org/10.1006/jagm.2000.1096.

11 Xuehou Tan. Fast computation of shortest watchman routes in simple polygons. *Inf. Process. Lett.*, 77(1):27–33, 2001. URL: https://doi.org/10.1016/S0020-0190(00)00146-0, doi:10.1016/S0020-0190(00)00146-0.

12 Xuehou Tan, Tomio Hirata, and Yasuyoshi Inagaki. Corrigendum to "an incremental algorithm for constructing shortest watchman routes". *International Journal of Computational Geometry and Applications*, 9(3):319–323, 1999.

An algorithm for the convex hull computation of rational plane curves*

Christina Katsamaki[1], Fabrice Rouillier[2], and Elias Tsigaridas[3]

1 **INRIA Paris, Sorbonne Université and Paris Université, Paris, France**
`christina.katsamaki@inria.fr`

2 **INRIA Paris, Sorbonne Université, Paris Université and CNRS, Paris, France**
`fabrice.rouillier@inria.fr`

3 **INRIA Paris, Sorbonne Université and Paris Université, Paris, France**
`elias.tsigaridas@inria.fr`

Abstract

We consider the problem of the convex hull computation of a plane parametric curve on a given interval, over which the image of the parametrization is a compact subset of $\mathbb{R}^2$. We design an exact and complete algorithm that computes a boundary description of the convex hull, as a sequence of line segments connecting two points on the curve and parametric arcs, all of them described by the parameters of their endpoints. When the parametrization involves polynomials of degree at most d and maximum bitsize of coefficients τ, we employ randomized algorithms for polynomial system solving and we achieve an expected complexity of $\widetilde{\mathcal{O}}_B(d^9 + d^8\tau)$.

1 Introduction

Convex hull computation is one of the fundamental problems of computational geometry; given a set of geometric objects in $\mathbb{R}^d$, one is interested in the minimal convex set that includes all of them. Convex objects facilitate taking geometric decisions, such as intersection. For example, interference tests among two non-convex objects can examine the relative position of their convex hulls at a preprocessing step; when the convex hulls do not intersect, the objects do not intersect. Thus, there exist direct applications in motion planning [20, 19], computer vision [11] or in geometric modelling systems [5, 9].

Convex hulls of linear objects are well-studied in literature [4]. We focus on non-linear convex hulls, and in particular on the convex hull of plane curves; there exist several algorithms (e.g. [12, 10, 14, 7, 1, 16, 8]) that perform even an optimal number of arithmetic operations. However, precise bit-complexity estimates lack from literature and the required predicates involve expensive algebraic operations.

We design an algorithm that computes a boundary description of the convex hull of a parametric curve in $\mathbb{R}^2$. Let $\phi(t) = \big(\phi_1(t), \phi_2(t)\big) = \big(\frac{p_1(t)}{q_1(t)}, \frac{p_2(t)}{q_2(t)}\big)$, where $p_i, q_i \in \mathbb{Z}[t]$ for $i = 1, 2$, be a rational parametrization of a real algebraic curve $\mathcal{C}$. We consider $I \subseteq \mathbb{R}$, for whom $\phi(I)$ is a compact subset of $\mathbb{R}^2$. We denote by $\mathrm{conv}(\phi(I))$ the convex hull of $\phi(I)$, i.e., the set

$$\mathrm{conv}(\phi(I)) = \{\mathbf{p} \in \mathbb{R}^2 \mid \mathbf{p} = \sum_{i=1}^{k} \lambda_i \phi(t_i) \text{ for } t_i \in I, \lambda_i \in \mathbb{R}_+ \text{ with } \sum_{i=1}^{k} \lambda_i = 1, k \in \mathbb{Z}\}.$$

[*] This project has received funding from the European Union's Horizon 2020 research and innovation program under the Marie Skłodowska-Curie grant agreement No 754362.
The third author is supported by ANR JCJC GALOP (ANR-17-CE40-000).

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

The boundary of the convex hull consists of a combination of smooth curved arcs and segments joining two points on the curve.

We follow closely the algorithm presented in [14]; it was designed for plane curves given in implicit form by an irreducible polynomial. We adapt it to the parametric case. We assume that the parametrization ϕ is *proper*, i.e., it is injective for almost all points on $\mathcal{C}$ [17, Ch. 4]. If it is not proper, reparametrization algorithms do exist, e.g. [15] (see [13] for an analysis of its complexity). We assume also that it is in *reduced form*, i.e., $\gcd(p_i(t), q_i(t)) = 1$, $i = 1, 2$.

Before introducing the theorem that summarizes our contribution, we fix the following notation: For a polynomial $f \in \mathbb{Z}[x]$, we call bitsize the logarithm (of base 2) of its infinity norm, that is equal to the logarithm of the maximum absolute value of its coefficients. A univariate polynomial is of size (d, τ) when its degree is at most d and has bitsize τ. The bitsize of a rational function is the maximum of the bitsizes of the numerator and the denominator. We denote by $\mathcal{O}$, resp. $\mathcal{O}_B$, the arithmetic, resp. bit, complexity and we use $\widetilde{\mathcal{O}}$, resp. $\widetilde{\mathcal{O}}_B$, to ignore (poly-)logarithmic factors.

▶ **Theorem 1.1.** *Let $\mathcal{C}$ be a curve with a proper parametrization $\phi(t) \in \mathbb{Z}^2(t)$, of size (d, τ), that is also in reduced form. Let $I \subset \mathbb{R}$ such that $\phi(I)$ is compact in $\mathbb{R}^2$. There exists a Las-Vegas algorithm to compute the convex hull of $\phi(I)$ in $\widetilde{\mathcal{O}}_B(d^9 + d^8\tau)$ expected complexity. The output of the algorithm is an ordered list $\mathcal{L}$ of parameter intervals and the corresponding parametrizations; the image of the interval over the parametrization is a parametric arc or a segment on the boundary of conv$(\phi(I))$ in a CCW traversal:*

$$\mathcal{L} = \{\{I_1, \psi_1\}, \ldots, \{I_N, \psi_N\}\}, \tag{1}$$

where I_i are bounded intervals of $\mathbb{R}$ (except from at most one I_i which may be of the form $(-\infty, a] \cup [b, +\infty)$) and $\psi_i : \mathbb{R} \dashrightarrow \mathbb{C}^2$ is either the parametrization ϕ or the parametrization of the segment with endpoints the image of the boundary points of I_i, and $N \in \mathcal{O}(d)$.

The parameters that correspond to the endpoints of the segments and the curve branches on the boundary of the convex hull are algebraic numbers. We represent an algebraic number $\alpha \in \mathbb{R}$ by the *isolating interval representation.* When $\alpha \in \mathbb{R}$, it includes a square-free polynomial which vanishes at α and a (rational) interval containing α and no other root of this polynomial. We also note that for $\phi(I)$ to be compact, I has to be either a union of closed intervals not containing any poles of ϕ (i.e., roots of $q_1(t)$, or $q_2(t)$) or a union of such closed intervals and intervals of the form $(-\infty, a], [a', +\infty)$, for $a \leq a'$, $a, a' \in \mathbb{R}$, where $\lim_{t \to -\infty} \phi(t) = \lim_{t \to +\infty} \phi(t) < \infty$. Throughout our manuscript, we assume that the boundary of I, which we denote by $\mathrm{bd}(I)$, consists of a constant number of points $\{a_1, \ldots, a_k\}$, where $k \in \mathbb{N}$ is a small constant and all the a_i's are rationals of bitsize $\mathcal{O}(1)$.

2 Algorithm description

We sketch the different steps of our algorithm, which are also summarized in Fig.2.

▶ **Definition 2.1** (Types of line segments)**.** We distinguish the following *types of line segments* (I-IV) on the boundary of conv$(\phi(I))$ according to the 'character' of their endpoints:

I. *tangent segments*: segments on the common tangent line of two or more points of $\phi(I)$.

II. *cusp-cusp segments*: segments whose endpoints are both cusps.

III. *cusp-curve segments*: segments connecting a cusp and a point on the curve (not cusp), lying on the latter point's tangent line.

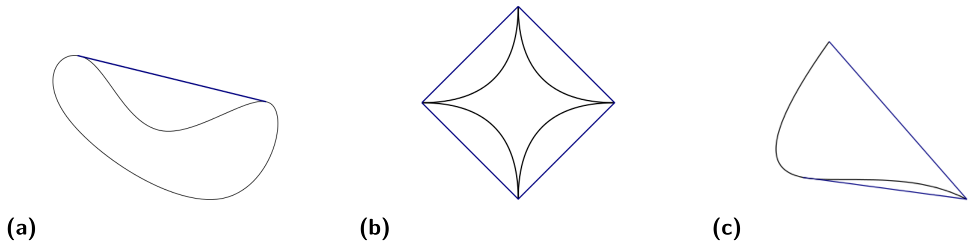


Figure 1 *Segments of type (a) I, (b) II and (c) IV (in blue).*

IV. *endpoint segments*: segments connecting a point corresponding to a parameter on $\text{bd}(I)$ and a point on the curve that is a cusp, or it lies on the latter point's tangent line, or its corresponding parameter is also in $\text{bd}(I)$.

The first step is to compute the set $\mathcal{S}$ of all segments of the types of Def. 2.1 (see Fig.1). However, not all of them are on the boundary of $\text{conv}(\phi(I))$. At the second step, we refine $\mathcal{S}$ so that it contains only the segments that are on the boundary of the convex hull. Let $\tilde{\mathcal{S}}$ be this set of segments. The last step consists of computing a description of the convex hull's boundary through a "*carrier polygon approach*"; the carrier polygon is a convex polygon contained in $\text{conv}(\phi(I))$, whose edges are in $1-1$ correspondence with the segments or the curved arcs on the boundary of the convex hull. Finding this polygon essentially allows to describe the boundary of the convex hull through a sequence of segments and curved arcs.

Algorithm to find a description of the boundary of conv($\phi(I)$):
STEP 1: COMPUTATION
find set $\mathcal{S}$ of candidate segments on the boundary of $\text{conv}(\phi(I))$.
STEP 2: REFINEMENT
find set $\tilde{\mathcal{S}} \subset \mathcal{S}$ of segments that are on the boundary of $\text{conv}(\phi(I))$.
STEP 3: DESCRIPTION
find a carrier polygon $\subset \text{conv}(\phi(I))$ and describe the boundary of $\text{conv}(\phi(I))$.

Figure 2 Summary of the algorithm.

Step 1: Computation. Each segment is determined by two parameter values, say a and b, such that $\phi(a)$ and $\phi(b)$ are the segment's endpoints respectively. We find the parameters that correspond to each type of segments by solving a polynomial system.

For tangent segments (type I) we consider the dual curve $\mathcal{C}^*$ of $\mathcal{C}$, which is in rational parametric form $\phi^* \in \mathbb{Z}^2(t)$ with polynomials of size $(d, d+\tau)$ [18, Sect.1.2.1], directly computable from the input parametrization. We compute the parameters that correspond to multiple points on $\mathcal{C}^*$ and then identify the ones that correspond to the same point on $\mathcal{C}^*$. For any two such parameters $s \neq t$ such that $\phi^*(s) = \phi^*(t)$, the tangent line to $\mathcal{C}$ at $\phi(s)$ and $\phi(t)$ coincides. To find the multiple points of $\mathcal{C}^*$, given its parametrization ϕ^*, we solve a bivariate polynomial system (see [13] for details).

For segments of type II, we consider the polynomials $h_i(t) = \phi_i'(t)q_i^2(t)$ for $i = 1, 2$. A parameter that gives a cusp of $\mathcal{C}$ is a root of $h_1^2(t) + h_2^2(t) = 0$ [13, Lem. 4.1].

For segments of type III, by vector orthogonality, the parameters s that give points on $\mathcal{C}$ whose tangents pass from $\phi(t)$ satisfy the equation $\langle \phi(t) - \phi(s), (-\phi_2'(s), \phi_1'(s)) \rangle = 0$. We denote by $H(s,t)$ the polynomial that occurs after clearing denominators in the previous equation. The system that we need to solve is $\{h_1^2(t) + h_2^2(t) = H(s,t) = 0\}$. For $a \in \mathrm{bd}(I)$ let $H_a(s) = H(s,a)$.

We find the segments of type IV by solving $H_a(s) = 0$, for every $a \in \mathrm{bd}(I)$, and by considering all the pairs of parameters in $\mathrm{bd}(I)$. It holds that $|\mathcal{S}| \in \mathcal{O}(d^2)$.

Step 2: Refinement. Among the segments that we found in Step 1, we keep the ones that contribute to the boundary of the convex hull. Let (s,t) a pair of parameters corresponding to a segment in $\mathcal{S}$. We consider the implicit equation of the line that passes through $\phi(s), \phi(t)$, say $\ell_{s,t}(x,y) = 0$ with $\ell_{s,t} \in (\mathbb{Q}(s,t))[x,y]$. The segment with endpoints $\phi(s), \phi(t)$ is on the boundary of $\mathrm{conv}(\phi(I))$ if and only if $\ell_{s,t}(\phi(\lambda)) \geq 0$, for all $\lambda \in I$ (see Fig.3). Let $\tilde{\mathcal{S}} \subset \mathcal{S}$ the elements of $\mathcal{S}$ that satisfy this condition. We describe the general procedure of refining $\mathcal{S}$: the pairs of parameters corresponding to segments of a certain type (Def.2.1) are solutions of a polynomial system of the form $\{F_1(s,t) = F_2(s,t) = 0\}$. Let $L(s,t,\lambda)$ denote $\ell_{s,t}(\phi(\lambda))$ when considered as a polynomial in $\mathbb{Q}[s,t,\lambda]$ (after clearing denominators). We find the isolated roots of the system $\{F_1(s,t) = F_2(s,t) = L(s,t,\lambda) = 0\}$. Then, for every pair (s_0,t_0) corresponding to a segment (and so $F_1(s_0,t_0) = F_2(s_0,t_0) = 0$), we can determine the sign of $L(s_0,t_0,\lambda)$ over I. $L(s_0,t_0,\lambda)$ being sign-invariant in I geometrically means that $\phi(I)$ does not cross the line $\ell_{s_0,t_0}(x,y) = 0$ and therefore the segment with endpoints $\phi(s_0), \phi(t_0)$ is on the boundary of $\mathrm{conv}(\phi(I))$. We can prove that $|\tilde{\mathcal{S}}| \in \mathcal{O}(d)$, since, eventhough there exist $\mathcal{O}(d^2)$ tangent segments, only $\mathcal{O}(d)$ are on the boundary of the convex hull [8].

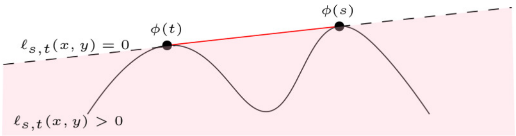


Figure 3 A candidate segment $\overline{\phi(s)\phi(t)}$ is on the boundary of $\mathrm{conv}(\phi(I))$ if and only if $\ell_{s,t}(\phi_1(\lambda), \phi_2(\lambda)) \geq 0$ for all $\lambda \in I$.

Step 3: Description. We describe the boundary of $\mathrm{conv}(\phi(I))$ by means of an ordered sequence of arcs of the curve and segments joining two points on the curve. The convex hull of the segments in $\tilde{\mathcal{S}}$, which are on the boundary of the convex hull of $\phi(I)$, defines a polygon $\mathcal{P}_c$, which we call *carrier polygon*, and is contained in $\mathrm{conv}(\phi(I))$ (see Fig.4). Every segment of $\tilde{\mathcal{S}}$ is an edge on the boundary of $\mathcal{P}_c$. Every edge of $\mathcal{P}_c$ that is not a segment in $\tilde{\mathcal{S}}$ corresponds to an arc of the curve. To describe of the boundary of $\mathrm{conv}(\phi(I))$, first, we order the vertices of $\mathcal{P}_c$ in a clockwise or counter-clockwise order. Then, for the edges of $\mathcal{P}_c$ that correspond to an arc of $\mathcal{C}$, we find the associated parameter interval J for whom $\phi(J)$ gives the arc. This is a delicate algebraic operation whose implementation is described in detail in the proof of Lem. 3.2 which can be found in the appendix.

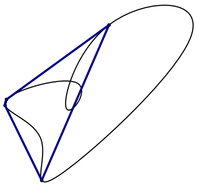


Figure 4 A curve and its carrier polygon (in blue)

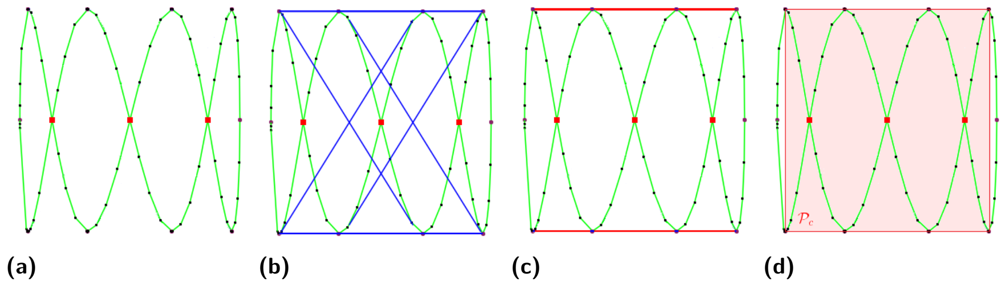


Figure 5 *The graph of the curve* $\left(\frac{-t^2+1}{t^2+1}, \frac{-8t(t^2-1)(t^4-6t^2+1)}{t^8+4t^6+6t^4+4t^2+1}\right)$ *and the three steps of the algorithm.*

3 Complexity Analysis: Sketch of the proof of Theorem 1.1

The first two steps of the algorithm dominate the complexity. As we demonstrated in the previous section, these two steps amount to solving polynomial systems of the form $\{F_1(s,t) = F_2(s,t) = L(s,t,\lambda) = 0\}$. The systems are not always zero-dimensional, which means that they may have infinitely many solutions. Nevertheless, the solutions (s,t) of $\{F_1(s,t) = F_2(s,t) = 0\}$ that corresspond to a segment on the curve, extend to isolated solutions of the system. For each type of segments, the specialization of the system $\{F_1(s,t) = F_2(s,t) = L(s,t,\lambda) = 0\}$ has a"special structure" that enables to extract the isolated roots easily. We will employ the Las-Vegas algorithm of [3] for isolating the roots of the occuring zero-dimensional systems in $\widetilde{\mathcal{O}}_B(d^9 + d^8\tau)$ expected complexity.

We exploit the fact that when one of the endpoints corresponds to a smooth point on the curve, the equation of the line containing the segment, coincides with the tangent line to this point. Then, the polynomial that expresses the intersection of the line with the curve depends only on two parameters s and λ. Therefore, the system corresponding to segments of type I, is of the form $\{F_1(s,t) = F_2(s,t) = L(s,\lambda) = 0\}$. To find its isolated solutions, we compute $R(s) := \mathtt{res}_t(F_1(s,t), F_2(s,t))$ and consider the system $\{R(s) = L(s,\lambda) = 0\}$. The triangular structure of this system makes it then easy to extract its isolated solutions through gcd computations: Let $L(s,\lambda) = l_D(s)\lambda^D + \cdots + l_1(s)\lambda + l_0(s)$, where $D = \mathcal{O}(d)$. We compute the gcd of $l_D(s), \ldots, l_0(s)$, say $g(s)$, in $\widetilde{\mathcal{O}}_B(d^3 + d^2\tau)$ in worst case [13, Lem. 2]. Then we compute the gcd of $R(s)$ and $g(s)$ in $\widetilde{\mathcal{O}}_B(d^5 + d^4\tau)$ [2, Lem. 4]. The gcd free part of R, say $\tilde{R}$, has bitsize $\widetilde{\mathcal{O}}(d^2 + d\tau)$ and is computed in the same complexity [2, Lem. 4]. The system $\{\tilde{R}(s) = L(s,\lambda) = 0\}$ is zero-dimensional and its solutions are the isolated solutions of $\{R(s) = L(s,\lambda) = 0\}$. Isolating intervals for the roots of this system and the corresponding multiplicities are computed in $\widetilde{\mathcal{O}}_B(d^6 + d^5\tau)$ [6, Thm. 2].

For segments of type II we have the system $\{h_1^2(s)+h_2^2(s) = h_1^2(t)+h_2^2(t) = L(s,t,\lambda) = 0\}$. To keep only the zero-dimensional part of the solutions, we work as follows: Let $L(s,t,\lambda) = l_D(s,t)\lambda^D + \cdots + l_1(s,t)\lambda + l_0(s,t)$, where $D = \mathcal{O}(d)$. Then, let $R_i(s) := \mathtt{res}_t(l_i(s,t), h_1^2(t,t) + h_2^2(t,t))$, $i = 1, \ldots, D$. Since the polynomial $L(s,t,\lambda)$ is symmetric with respect to s and t, the polynomials $l_i(s,t)$ are also symmetric and thus, $\mathtt{res}_s(l_i(s,t), h_1^2(s,s) + h_2^2(s,s)) = R_i(t)$. We compute the gcd of $R_0(s), \ldots, R_D(s)$ in $\widetilde{\mathcal{O}}_B(d^7 + d^6\tau)$ [13, Lem. 2] and then the gcd of the latter with $h_1^2(s,s) + h_2^2(s,s)$ in $\widetilde{\mathcal{O}}_B(d^3\tau)$ [2, Lem. 4]. Let $\tilde{h}(s)$ the gcd-free part of $h_1^2(s,s) + h_2^2(s,s)$. Then, the system $\{\tilde{h}(s) = \tilde{h}(t) = L(s,t,\lambda) = 0\}$ is zero-dimensional and gives the isolated solutions of the initial system. We isolate its roots using [3] in $\widetilde{\mathcal{O}}_B(d^9 + d^8\tau)$.

Segments of type III can be treated similarly as segments of type I. For segments of type IV, there is no computational difficulty. We arrive to the following:

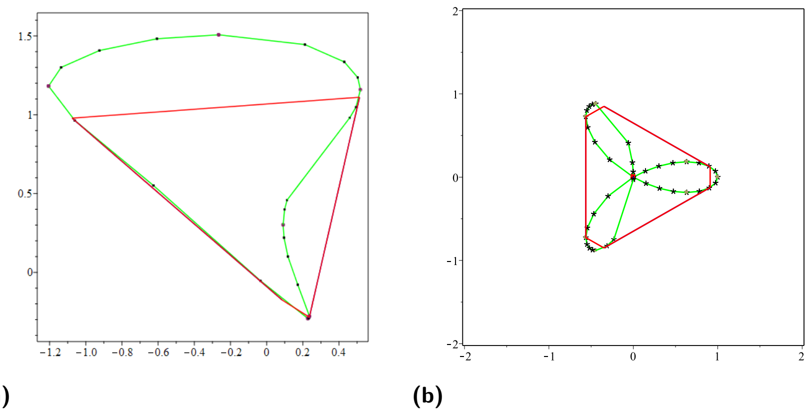


Figure 6 *Output of our implementation for the parametric curves (a)* $\left(\frac{-7t^4+22t^3-55t^2-94t+87}{-56t^4-62t^2+97t-73}, \frac{-4t^4-83t^3-10t^2+62t-82}{-56t^4-62t^2+97t-73}\right)$ *and (b)* $\left(\frac{-3t^2+1}{(t^2+1)^2}, \frac{-3t^2+1)t}{(t^2+1)^2}\right)$.

▶ **Lemma 3.1** (Bit-complexity of steps 1-2). *We compute the set* $\widetilde{S}$ *of the pairs of parameters that give the segments on the boundary of the convex hull in* $\widetilde{\mathcal{O}}_B(d^9 + d^8\tau)$ *expected complexity.*

The computational requirement of the third step of the algorithm, is sorting of the coordinates of the endpoints of the segments on the boundary of the convex hull and of the parameters that give these points. We give more details in the Appendix.

▶ **Lemma 3.2** (Bit-complexity of Step 3). *Given the set of segments* $\tilde{S}$ *on the boundary of the convex hull, Step 3 computes a description of it in* $\widetilde{\mathcal{O}}_B(d^6\tau)$.

4 Conclusion: Future work and Implementation

In the full version of this manuscript (which will be available online soon), we give details on our implementation of the algorithm (Fig.6) and we also compute the diameter of the convex hull and approximate its area with no additional cost. We are currently working on improving the complexity of solving a zero-dimensional polynomial system of the form $\{F_1(s) = F_2(t) = L(s,t,\lambda)\}$, which is dominant, using amortized complexity bounds.

References

1 Chandrajit Bajaj and Myung-Soo Kim. Convex hulls of objects bounded by algebraic curves. *Algorithmica*, 6:533–553, 06 1991. `doi:10.1007/BF01759058`.

2 Yacine Bouzidi, Sylvain Lazard, Guillaume Moroz, Marc Pouget, Fabrice Rouillier, and Michael Sagraloff. Improved algorithms for solving bivariate systems via Rational Univariate Representations. page 51.

3 Cornelius Brand and Michael Sagraloff. On the Complexity of Solving Zero-Dimensional Polynomial Systems via Projection. In *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation - ISSAC '16*, pages 151–158, Waterloo, ON, Canada, 2016. ACM Press. URL: `http://dl.acm.org/citation.cfm?doid=2930889.2930934`, `doi:10.1145/2930889.2930934`.

4 Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational geometry: algorithms and applications, 3rd Edition.* Springer, 2008. URL: `https://www.worldcat.org/oclc/227584184`.

5 Jiansong Deng. Algebraic geometry and geometric modeling. 2013.

6 Daouda Niang Diatta, Sény Diatta, Fabrice Rouillier, Marie-Françoise Roy, and Michael Sagraloff. Bounds for polynomials on algebraic numbers and application to curve topology. working paper or preprint, October 2018. URL: `https://hal.inria.fr/hal-01891417`.

7 David P. Dobkin and Diane L. Souvaine. Computational geometry in a curved world. *Algorithmica*, 5:421–457, 1990.

8 Gershon Elber, Myung-Soo Kim, and Hee-Seok Heo. The convex hull of rational plane curves. *Graphical Models*, 63:151–162, 05 2001. `doi:10.1006/gmod.2001.0546`.

9 Laureano González-Vega, Ioana Necula, Sonia Pérez-Díaz, Juana Sendra, and Juan Sendra. Algebraic methods in computer aided geometric design: Theoretical and practical applications. *Geometric Computation*, 11, 03 2004. `doi:10.1142/9789812794833_0001`.

10 Douglas Ierardi. Convexhull of curved objects via duality – a general framework and an optimal 2-d algorithm. 02 1970.

11 Rui J. Defigueiredo and Hemant D. Tagare. Curves and surfaces in computer vision. 08 1990. `doi:10.1117/12.19726`.

12 J. K. Johnstone. Giftwrapping a curve with the convex hull. In *Proceedings of the 42nd Annual Southeast Regional Conference*, ACM-SE 42, page 224–227, New York, NY, USA, 2004. Association for Computing Machinery. `doi:10.1145/986537.986590`.

13 Christina Katsamaki, Fabrice Rouillier, Elias Tsigaridas, and Zafeirakis Zafeirakopoulos. On the geometry and the topology of parametric curves. In *Proceedings of the 45th International Symposium on Symbolic and Algebraic Computation*, ISSAC '20, page 281–288, New York, NY, USA, 2020. Association for Computing Machinery. `doi:10.1145/3373207.3404062`.

14 David Kriegman, Erliang Yeh, and J Ponce. Convex hulls of algebraic curves. pages 118–127, 11 1992. `doi:10.1117/12.131738`.

15 Sonia Pérez-Díaz. On the problem of proper reparametrization for rational curves and surfaces. *CAGD*, 23(4):307–323, 2006.

16 Alejandro Schaffer and Christopher J Van Wyk. Convex hulls of piecewise-smooth jordan curves. *Journal of Algorithms*, 8:66–94, 03 1987. `doi:10.1016/0196-6774(87)90028-9`.

17 J Rafael Sendra, Franz Winkler, and Sonia Pérez-Díaz. Rational algebraic curves. *Algorithms and Computation in Mathematics*, 22, 2008.

18 E. A. Tevelev. *Projective duality and homogeneous spaces*. Number v. 133. 4 in Encyclopaedia of mathematical sciences, Invariant theory and algebraic transformation groups. Springer, Berlin ; New York, 2005. OCLC: ocm57170635.

19 Y Yang, Y.-C Liu, M.-Y Liu, and M.-Y Fu. A path planning algorithm based on convex hull for autonomous service robot. 31:54–58+63, 01 2011.

20 F Zhou, Baoye Song, and Guohui Tian. Bézier curve based smooth path planning for mobile robot. *Journal of Information and Computational Science*, 8:2441–2450, 12 2011.

Properties for Voronoi Diagrams of Arbitrary Order on the Sphere

Mercè Claverol[1], Andrea de las Heras Parrilla[1], and Clemens Huemer[1]

1 **Department of Mathematics, Universitat Politècnica de Catalunya**
merce.claverol@upc.edu andrea.de.las.heras@estudiantat.upc.edu
clemens.huemer@upc.edu

Abstract

For a given set of points U on a sphere S, the order k spherical Voronoi diagram $SV_k(U)$ decomposes the surface of S into regions whose points have the same k nearest points of U. We study properties for $SV_k(U)$, using different tools: the geometry of the sphere, a labeling for the edges of $SV_k(U)$, and the inversion transformation. Hyeon-Suk Na, Chung-Nim Lee, and Otfried Cheong (Comput. Geom., 2002) applied inversions to construct $SV_1(U)$. We generalize their construction for spherical Voronoi diagrams from order 1 to any order k. We use that construction to prove formulas for the numbers of vertices, edges, and faces in $SV_k(U)$. Among the properties of $SV_k(U)$, we also show that $SV_k(U)$ has a small orientable cycle double cover.

1 Introduction

Let U be a set of n points on a sphere $S \subset \mathbb{R}^3$ such that no three of them lie in the same great circumference and no four of them are cocircular, i.e. U is in general position, and let $1 \leq k \leq n-1$ be an integer. The order k spherical Voronoi diagram $SV_k(U)$ decomposes the surface of S into regions whose points have the same k nearest points of U. Then, each of these regions is a face $f(P_k)$ of $SV_k(U)$ associated with a subset $P_k \subset U$ of size k: Each point in the interior of $f(P_k)$ has P_k as its k nearest neighbors from U.

Many researchers studied the nearest ($k = 1$) and the farthest ($k = n-1$) spherical Voronoi diagrams [2, 11, 10]. For these two diagrams it was seen that practically all algorithms in the plane can be adapted to the sphere. Spherical Voronoi diagrams of order different from $k = 1$ and $k = n-1$ have barely been studied. In this work we deepen in these diagrams and the properties and algorithms that we present are for Voronoi diagrams of arbitrary order k on the sphere. This abstract summarizes our main results on $SV_k(U)$; we refer the reader to the thesis of the second author [5] for more details and more properties. One of the most important tools that we use in our proofs is an edge labeling. This labeling is an extension to the sphere of the already defined edge labeling for Voronoi diagrams in the plane [4]. An edge that delimits a face of $SV_k(U)$ is a spherical segment of the perpendicular bisector (on the sphere) of two points i and j of U. This observation induces a natural labeling of the edges of $SV_k(U)$ with the following rule:

- **Edge rule:** An edge of $SV_k(U)$ which belongs to the perpendicular bisector of points $i, j \in U$ has labels i and j, where we put the label i on the side (half-sphere) of the edge that contains point i and we put label j on the other side. See Figure 1.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Also, from this rule, we deduce two more rules of the labeling of $SV_k(U)$: one rule for the vertices and one rule for the faces. Vertices can be of type I, if they are centers of circles on the sphere passing through three points of U and enclosing $k-1$ points of U, or type II, if they are centers of circles on the sphere passing through three points of U and enclosing $k-2$ points of U. In the literature, vertices of type I (type II) are also called *new* (*old*) [7].

• **Vertex rule:** Let v be a vertex of $SV_k(U)$ and let $\{i, j, \ell\} \subset U$ be the set of labels of the edges incident to v. The cyclic order of the labels of the edges around v is i, i, j, j, ℓ, ℓ if v is of type I, and it is i, j, ℓ, i, j, ℓ if v is of type II.

• **Face rule:** In each face of $SV_k(U)$, the edges that have the same label i are consecutive, and these labels i are either all in the interior of the face, or are all in the exterior of the face.

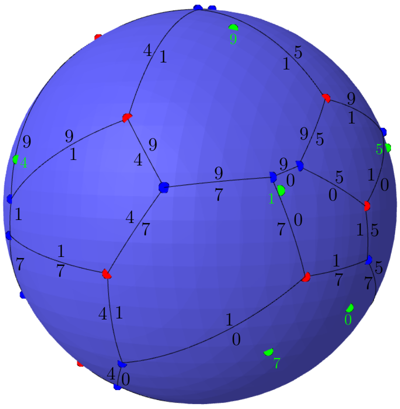


Figure 1 The edge labeling of $SV_2(U)$ for a set U of ten points $\{0, 1, \ldots, 9\}$ in general position (the visible ones are drawn in green color). Vertices of type I are drawn in blue, and vertices of type II in red.

Note that when walking along the boundary of a face, in its interior (exterior), a change in the labels of its edges appears whenever we reach a vertex of type II (type I), see Figure 1.

From this edge labeling, we observe that edges with same label i always form a cycle in $SV_k(i)$; see Figure 2. These edges with the same label i enclose a region $R_k(i)$ that consists of all the points of the sphere that have point $i \in U$ as one of their k nearest neighbors from U. We observe that $R_1(i)$ is contained in the kernel of this star-shaped set $R_k(i)$, and we identify the reflex (convex) vertices on the boundary $B_k(i)$ of $R_k(i)$ as vertices of type II (type I). See [4, 5] for details.

A *cycle double cover* [6] of a graph G is a collection of cycles $\mathcal{C}$ such that every edge of G belongs to precisely two cycles of $\mathcal{C}$. A double cover $\mathcal{C}$ is orientable if an orientation can be assigned to each element of $\mathcal{C}$ such that for every edge e of G, the two cycles that cover e are oriented in opposite directions.

Much research was done on finding small cycle double covers for several classes of graphs, see for instance [1, 12]. We show that every higher-order Voronoi diagram on the sphere admits an orientable double cover of its edges, using, precisely, the n cycles $B_k(i)$ in for $i = 1, \ldots, n$. We refer to [4] for related results on double covers of the edges of higher order Voronoi diagrams in the plane.

As one of our main results, we generalize to any order the construction of spherical

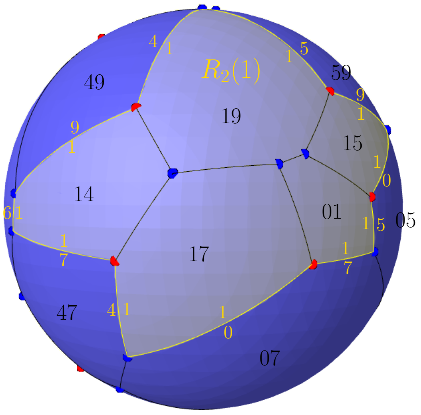


Figure 2 $SV_2(U)$ for the point set U of Figure 1; in each face, its two nearest neighbors are indicated. In yellow, the region $R_2(1)$ formed by all the faces of $SV_2(U)$ that have point 1 as one of their two nearest neighbors. The boundary $B_2(1)$ of $R_2(1)$ is formed by all the edges which have the label 1 and this label is always inside $R_2(1)$. The boundary vertices of $R_2(1)$ with an incident edge lying in the interior of $R_2(1)$ are of type II in $SV_2(U)$ and the remaining boundary vertices are of type I in $SV_2(U)$.

Voronoi diagrams defined by Hyeon-Suk Na, Chung-Nim Lee and Otfried Cheong [11], using precisely the regions $R_k(i)$ and the inversion transformation. Inversions for Voronoi diagrams were already applied in the classical work of Brown [2, 3]. In [11], $SV_1(U)$ is computed from two planar Voronoi diagrams after applying inversions to map U to the plane; two different inversion centers are used. In [11] it is also shown that $SV_1(U)$ is homeomorphic to the union of a nearest and a farthest Voronoi diagram, when glued together. We generalize this to $SV_k(U)$ being homeomorphic to the union of a planar Voronoi diagram of order k, and one planar Voronoi diagram of order $n-k$. Furthermore, these diagrams are linked via $R_k(i)$ in $SV_k(U \cup \{i\})$, with i the center of inversion, where the unbounded edges in the two planar Voronoi diagrams correspond to edges of $SV_k(U)$ intersected by $B_k(i)$. We further derive formulas for the numbers of vertices, edges and faces of $SV_k(U)$. The proof is based on the construction of $SV_k(U)$. Surprisingly, the obtained formulas seem to be new. We also obtain formulas for the number of vertices of type I and for the number of vertices of type II in $SV_k(U)$. The proof of Theorem 3.2 is omitted in this abstract, but also see [5].

2 Properties of $SV_k(U)$

▶ **Property 2.1.** Let u^* be the antipodal point of a point u on a sphere S. Then $SV_k(U) = SV_{n-k}(U^*)$, where $U^* = \{u^* | u \in U\}$.

The proof of this property is essentially the same as the one for the case $k = 1$ given in [2, 11].

Proof. The spherical distance for points $x, y \in S$ is $d(x, y) = \pi r - d(x, y^*)$ where r is the radius of the sphere. It follows that the k nearest neighbors of a point x must be the k farthest neighbors of x^*. Therefore, $x \in f(P_k)$ if and only $x \in f(U^* \setminus P_k^*)$ where $P_k^* = \{p^* | p \in P_k\}$, and the property follows. ◀

▶ **Property 2.2.** Let v be a vertex of type I of $SV_k(U)$. Then v^* is a vertex of type II of $SV_{n-k}(U)$. Similarly, if v is a vertex of type II of $SV_k(U)$ then v^* is a vertex of type I of $SV_{n-k}(U)$. See Figure 3.

Proof. If v is a vertex of type I of $SV_k(U)$, then it is the center of a disk D that passes through three points of U and contains $(k-1)$ points of U. From this, by the geometry of the sphere S, the remaining $(n-k-2)$ points are contained in the complementary disk $S \setminus D$ whose center is v^*. Therefore, v^* must be a vertex of type II of $SV_{n-k}(U)$. The symmetric argument works for v of type II. ◀

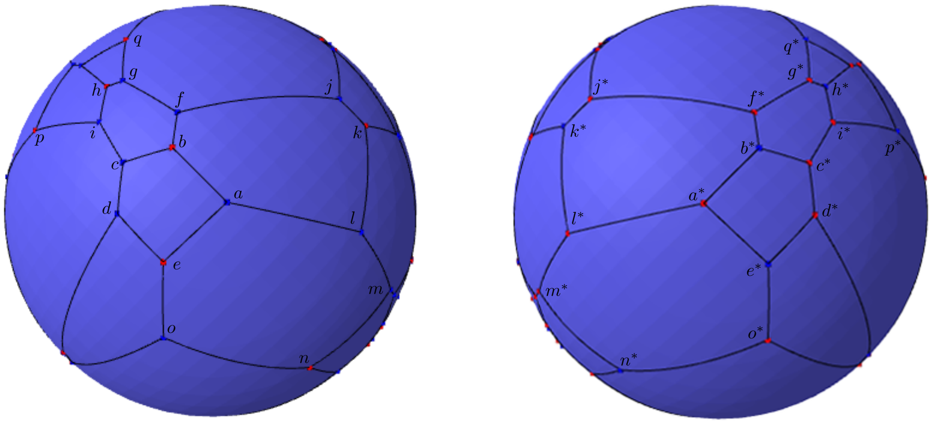


Figure 3 Two complementary Voronoi diagrams on an sphere $SV_k(U)$ and $SV_{n-k}(U)$, showing the homothetic relation between them and their corresponding antipodal points types. Type I vertices are blue and type II vertices are red.

▶ **Property 2.3.** Let $f(P_k)$ be a face of $SV_k(U)$ and let $f(U \setminus P_k)$ be its corresponding antipodal face in $SV_{n-k}(U)$. $f(P_k)$ and $f(U \setminus P_k)$ use the same labels but in opposite sides, i.e., if i is an interior label of an edge of $f(P_k)$ then it is an exterior label for the corresponding antipodal edge in $SV_{n-k}(U)$. See Figure 4.

Proof. It follows from Property 2.1 that $f(P_k)$ and $f(U \setminus P_k)$ are antipodal polygons. Then we just need to observe that antipodal polygons are defined by the complementary half-spheres defined by the same bisector, i.e, their edges are from the same bisectors but the antipodal polygons lie in opposite sides of those bisectors, see Figure 4. Therefore, by the edge rule, the statement is clear. ◀

▶ **Theorem 2.4.** *$SV_k(U)$ has an orientable double cover consisting of $|U| = n$ cycles.*

Proof. It is not difficult to see that for every $1 \leq i \leq n$, all the edges that have the label i in $SV_k(U)$ form one cycle (also see Property 6.1 in [5]). Since each label i, corresponding to a point $i \in U$, is inside the corresponding region $R_k(i)$, we can orient all the edges of a cycle with label i clockwise around point i; note that point i is also contained in $R_k(i)$. This shows that the cycle cover is orientable. Finally, as there is one cycle for each point of U, it follows that $SV_k(U)$ has an orientable double cover of n cycles. ◀

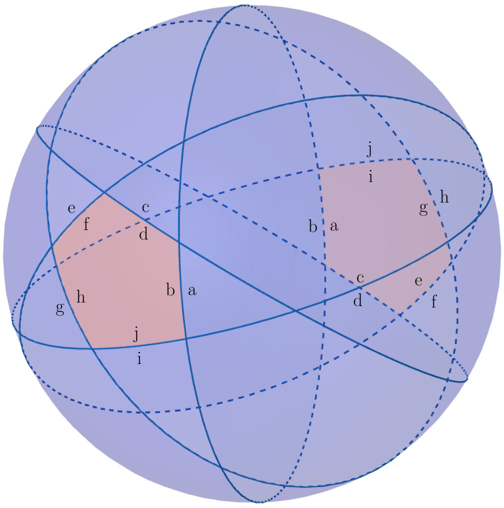


Figure 4 Two antipodal polygons, one has labels b, d, f, h, j in its interior, the other one has these labels in its exterior.

3 Relations between Planar and Spherical Voronoi Diagrams

In this section we generalize to Voronoi diagrams of arbitrary order k the construction given in [11] for the nearest and farthest Voronoi diagrams. We then prove some more properties using this construction.

First, we need to define the inversion transformation, as it is the basis of the relation between Voronoi diagrams on the sphere and on the plane.

▶ **Definition 3.1.** The inversion transformation is determined by two parameters: The center of inversion O and the radius of inversion R. Two points P and P' in $\mathbb{R}^3$ are said to be inverses of each other if:

1. The points P and P' lie in the same half-line with origin in O.
2. The Euclidean distances $|\overline{OP}|$ and $|\overline{OP'}|$ in $\mathbb{R}^3$ satisfy $R^2 = |\overline{OP}||\overline{OP'}|$.

Now, we can proceed in a similar way to [3] to prove the construction for Voronoi diagrams on the sphere of arbitrary order, $SV_k(U)$. From now on, we denote by S' the plane inverse of the sphere S, by U' the set of points on the plane S' that are inverses of the points of $U \subset S$, and by $V_k(U')$ the Voronoi diagram of order k in the plane for the set of points U'.

▶ **Theorem 3.2.** *Let $i \notin U$ be a point on the sphere S such that $U \cup \{i\}$ is in general position. Let U' be the set of inverse points of U for a chosen inversion radius r and i the center of inversion. Then $SV_k(U)$ is homeomorphic to the union of $V_k(U')$ and $V_{n-k}(U')$, joined by the unbounded edges common to $V_k(U')$ and $V_{n-k}(U')$ (unbounded edges from the same bisector are glued together). Moreover, $R_k(i)$ in $SV_k(U \cup \{i\})$ partitions $SV_k(U)$ into two subgraphs that are homeomorphic to $V_k(U')$ and $V_{n-k}(U')$. The vertices of type I (type II) in $V_k(U')$ correspond to the vertices of type I (type II) in $SV_k(U)$ and the vertices of type I (type II) in $V_{n-k}(U')$ correspond to the vertices of type II (type I) in $SV_k(U)$. See Figures 5 and 6.*

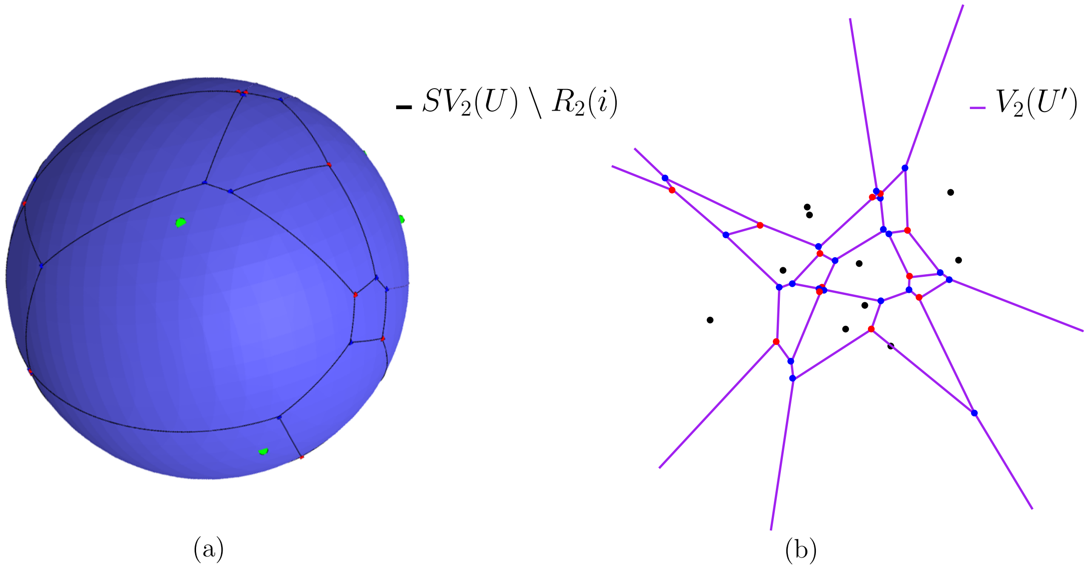


Figure 5 For a set U of ten points on the sphere (the visible ones are drawn in green color): The picture shows the homeomorphism between: (a) The induced graph by $SV_2(U)$ at the exterior of $R_2(i)$ in $SV_2(U \cup \{i\})$. (b) The planar Voronoi diagram of order 2 for the points of U' (black color).

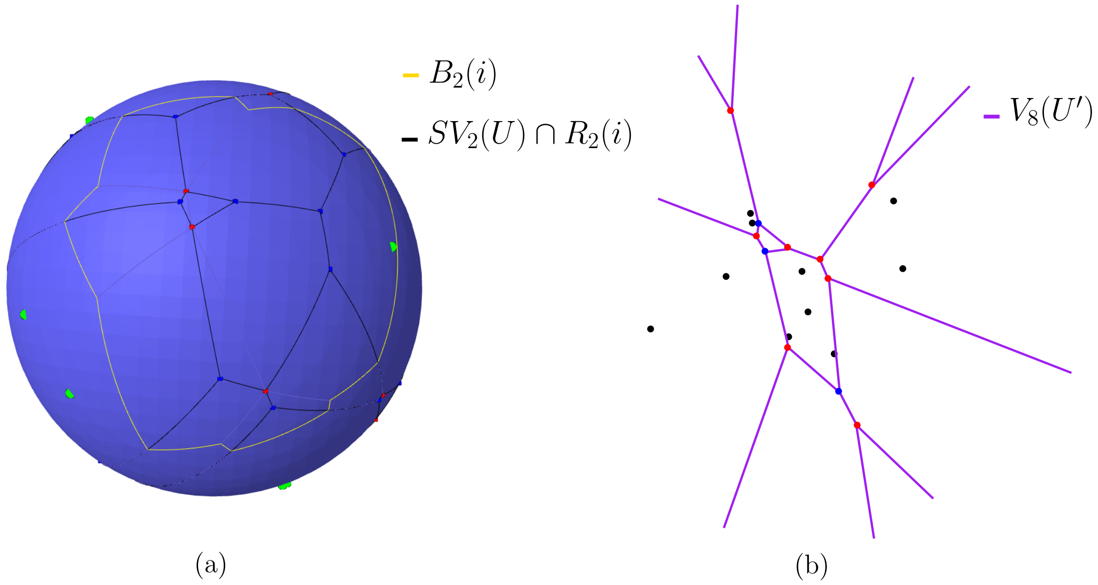


Figure 6 For a set U of ten points on the sphere (the visible ones are drawn in green color): The picture shows the homeomorphism between: (a) The induced graph by $SV_2(U)$ at the interior of $R_2(i)$ in $SV_2(U \cup \{i\})$. (b) The planar Voronoi diagram of order 8 for the points of U' (black color).

Theorem 3.2 tells us how to construct $SV_k(U)$: we just have to invert the points of U, compute planar Voronoi diagrams $V_k(U')$ and $V_{n-k}(U')$, and map them to the sphere as follows: each vertex $a'b'c'$ of either $V_k(U')$ or $V_{n-k}(U')$ corresponds to a vertex abc of $SV_k(U)$ (abc is center of the circle that passes through a, b and c on the sphere); vertices in $SV_k(U)$ are adjacent whenever the correspongin vertices in $V_k(U')$ or in $V_{n-k}(U')$ are adjacent. Finally, the vertices of $SV_k(U)$ corresponding to vertices incident to an unbounded edge from the same bisector in $V_k(U')$ and $V_{n-k}(U')$ get connected.

Let us shortly also comment on the computational complexity of constructing higher order Voronoi diagrams on the sphere. The inversion is a linear time transformation and, once we have the planar Voronoi diagrams, mapping them to the sphere also only requires linear time. Therefore, the computational time for constructing the spherical Voronoi diagrams is bounded by the computational time for the planar ones. See [8] for a discussion on the several algorithms for higher order Voronoi diagrams.

Now, from these constructions, it is easy to see that properties proved for the plane [4] must be true for the sphere. We can prove easily some properties on the sphere using results from the plane, but also we can prove properties in the plane using the sphere. Next, we show that the number of vertices of type I (type II) in $SV_k(U)$ only depends on the number n of points of U, but not on their positions on the sphere.

▶ **Theorem 3.3.** *For a set U of n points on the sphere, the number of vertices of type I in $SV_k(U)$ is $2k(n-k-1)$ and the number of vertices of type II is $2(k-1)(n-k)$.*

Proof. By Theorem 3.2, we can define an inversion transformation such that there is a one-to-one correspondence between the vertices of $SV_k(U)$ and the vertices of $V_k(U')$ and $V_{n-k}(U')$. Vertices of type I of $SV_k(U')$ and vertices of type II of $V_{n-k}(U')$ correspond to the vertices of type I in $SV_k(U)$. Then, the number of vertices of type I in $SV_k(U)$ is the sum of type I vertices of $V_k(U')$ and type II vertices of $V_{n-k}(U')$ which correspond to the circles enclosing $k-1$ points of U' and circles enclosing $n-k-2$ points of U', respectively. We denote the number of such circles with c_{k-1} and c_{n-k-2}. By Theorem 5.3 of [9], we have

$$c_{k-1} + c_{n-k-2} = 2(k-1+1)(n-2-k+1) = 2k(n-k-1). \tag{1}$$

Then, the number of vertices of type I in $SV_k(U)$ is $2k(n-k-1)$. Similarly, we can compute the number of vertices of type II as the sum of vertices of type II in $V_k(U')$ and type I in $V_{n-k}(U')$, i.e., the number of the circles enclosing $k-2$ points of U', c_{k-2}, and enclosing $n-k-1$ points of U', c_{n-k-1}. Again, using Theorem 5.3 of [9], we have

$$c_{k-2} + c_{n-k-1} = 2(k-2+1)(n-2-k+2) = 2(k-1)(n-k). \tag{2}$$

Then, the number of vertices of type II in $SV_k(U)$ is $2(k-1)(n-k)$. ◀

▶ **Theorem 3.4.** *For a set U of n points on the sphere, the order k Voronoi diagram $SV_k(U)$ has $4kn-4k^2-2n$ vertices, $6kn-6k^2-3n$ edges and $2kn-2k^2-n+2$ faces.*

Proof. Vertices of spherical Voronoi diagrams are either of type I or type II, so the total number of vertices is the sum of vertices of the two types. Then, by Theorem 3.3, the number of vertices $|V|$ is

$$|V| = 2k(n-k-1) + 2(k-1)(n-k) = 4kn - 4k^2 - 2n. \tag{3}$$

Now, as each vertex has degree three in $SV_k(U)$, we can count the total number of edges. Since each edge is incident to two vertices, by double counting, the number of edges $|E|$ is

$$|E| = \frac{3}{2}\left(-4k^2 + 4kn - 2n\right) = 6kn - 6k^2 - 3n. \tag{4}$$

Finally, as $SV_k(U)$ is a planar graph, we can apply Euler's Formula to count the number of faces $|F|$, and we have

$$|F| = 2 - (-4k^2 + 4kn - 2n) + (-6k^2 + 6kn - 3n) = 2kn - 2k^2 - n + 2. \tag{5}$$

◀

Acknowledgments. Research of C. Huemer and of M. Claverol was supported by project PID2019-104129GB-I00/ MCIN/ AEI/ 10.13039/501100011033.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 734922.

References

1 J. A. Bondy. *Cycles and Rays.* Nato Science Series C, 1990. ISBN 978-9401067195.

2 K. Q. Brown. *Geometric Transforms To Fast Geometric Algorithms.* Ph.D. Dissertation, Carnegie-Mellon Univ., 1979. ISBN 9798607356866.

3 K. Q. Brown. Voronoi diagrams from convex hulls. *Inform. Process. Lett.*, 9:223–228, 1979. `doi:10.1016/0020-0190(79)90074-7`.

4 M. Claverol, A. de las Heras, C. Huemer, and A. Martínez-Moraian. The edge labeling of higher order Voronoi diagrams. *Proc. of Spanish meeting on Computational Geometry 2021*, pages 23–26. `https://arxiv.org/abs/2109.13002`.

5 A. de las Heras Parrilla. *Properties for Voronoi diagrams of arbitrary order in the sphere.* Master Thesis. Universitat Politècnica de Catalunya, 2021. `https://upcommons.upc.edu/handle/2117/354664`.

6 F. Jaeger. *A survey on the cycle double cover conjecture.* North-Holland, 1985. `doi:10.1016/S0304-0208(08)72993-1`.

7 D. T. Lee. On k-nearest neighbor Voronoi diagrams in the plane. *IEEE Trans. Comput.*, pages 478–487, 1982. `doi:10.1109/TC.1982.1676031`.

8 D.T. Lee, C.H. Liu, and E. Papadopoulou. The k-nearest-neighbor Voronoi diagram revisited. *Algorithmica*, 71:429–449, 2015. `doi:10.1007/s00453-013-9809-9`.

9 R. C. Lindenbergh. A Voronoi poset. *J. Geom. Graph.*, 7:41–52, 2003. ISSN 1433-8157.

10 R. E. Miles. Random points, sets and tessellations on the surface of a sphere. *The Indian Journal of Statistics Series A*, 33(2):145–174, 1971. ISSN 0581572x.

11 H. Na, C.-N. Lee, and O. Cheong. Voronoi diagrams on the sphere. *Computational Geometry*, 23:183–194, 2002. `doi:10.1016/S0925-7721(02)00077-9`.

12 K. Seyffarth. Small cycle double covers of 4-connected planar graphs. *Combinatorica*, 13:477–482, 1993. `doi:10.1016/0012-365X(92)90610-R`.

The Mixed Page Number of Graphs

Jawaherul Md. Alam[1], Michael A. Bekos[2], Martin Gronemann[3], Michael Kaufmann[4], and Sergey Pupyrev[5]

1 **Amazon Inc., Tempe, AZ, USA**
`jawaherul@gmail.com`

2 **University of Ioannina, Ioannina, Greece**
`bekos@uoi.gr`

3 **Algorithms and Complexity Group, TU Wien, Vienna, Austria**
`mgronemann@ac.tuwien.ac.at`

4 **Institut für Informatik, Universität Tübingen, Tübingen, Germany**
`mk@informatik.uni-tuebingen.de`

5 **Facebook, Inc., Menlo Park, CA, USA**
`spupyrev@gmail.com`

Abstract

A linear layout of a graph typically consists of a total vertex-order, and a partition of the edges into sets of either non-crossing edges, called stacks, or non-nested edges, called queues. The stack (queue) number of a graph is the minimum number of required stacks (queues). Mixed linear layouts combine both by allowing each set of edges to form either a stack or a queue. Here we initiate the study of the *mixed page number* of a graph which corresponds to the minimum number of such sets.

Related Version arXiv:2107.04993

1 Introduction

In this work, we focus on linear layouts of graphs in which the edges must be partitioned into a minimum number of sets, called *pages*, such that each page has a certain property in the underlying linear order of the vertices. The most prominent of such representatives are the *stack layouts* (also known as *book embeddings*) and the *queue layouts*. The former do not allow two edges in the same page (called *stack*) to cross [4], while in the latter no two edges of the same page (called *queue*) are allowed to nest [12]; see Fig. 1. The *stack* (*queue*) *number* of a graph is the minimum number of stacks (queues) over all its stack (queue) layouts. Both graph parameters have been extensively studied; see [7, 11, 12, 17, 18].

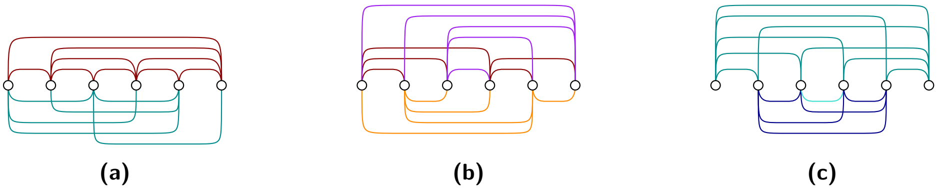


Figure 1 Different layouts of K_6: (a) 1-stack 1-queue, (b) 3-stack, and (c) 3-queue.

A natural generalization of stack and queue layouts was introduced back in 1992 by Heath and Rosenberg [13], who proposed the study of s-stack q-queue layouts that consist of s stacks and q queues; see also [2, 5, 9, 10, 15]. In this context, one expects a reduction on the total number of pages (stacks or queues) required in a mixed layout with respect to the corresponding ones required in pure stack or queue layouts, which, however, has not been confirmed so far. In this work, we substantiate this expectation. To achieve this, we

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

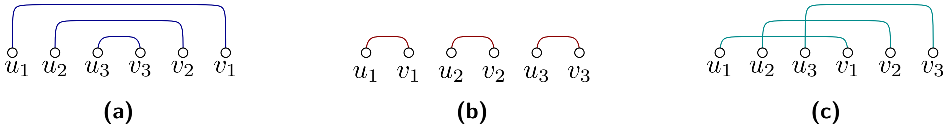

u_1 u_2 u_3 v_3 v_2 v_1
(a)
u_1 v_1 u_2 v_2 u_3 v_3
(b)
u_1 u_2 u_3 v_1 v_2 v_3
(c)

Figure 2 Illustration of: (a) a 3-rainbow, (b) a 3-necklace, and (c) a 3-twist.

introduce and study a new graph parameter, called *mixed page number*, which equals to the minimum value of $s+q$ for which a graph admits an s-stack q-queue layout.

2 Preliminaries

A *vertex order* $\prec$ of a graph is a total order of its vertices; we write $u \prec v$ to denote that vertex u precedes vertex v. A *k-twist* (*k-rainbow*) is a set of k independent edges that pairwise cross (nest) in $\prec$, while a *k-necklace* is a set of k independent edges that pairwise form neither a 2-twist nor a 2-rainbow; see Fig. 2. A stack (queue) is a set of pairwise non-crossings (non-nested) edges in $\prec$. A mixed *s-stack q-queue layout* of a graph consists of a vertex order $\prec$, called *linear order*, and a partition of its edges into s stacks and q queues. Hence, an s-stack (q-queue) layout is a mixed s-stack 0-queue (0-stack q-queue) layout.

3 Basic Properties

We start with an upper bound on the mixed page number of a graph similar to the ones in [14] and [8], which notably holds for every fixed linear order of its vertices.

▶ **Theorem 3.1.** *The mixed page number of a graph G with m edges is at most $\lfloor\sqrt{2m}\rfloor$ for every fixed linear order of the vertices.*

Proof. Consider the maximum rainbow r in a vertex order $\prec$ of G. If its size is at most $\sqrt{2m}$, then all edges of G can be assigned to at most $\lfloor\sqrt{2m}\rfloor$ queues [13]. Otherwise, we assign the edges of r to a stack, and proceed recursively with $G \setminus r$. The total number $T(m)$ of stacks and queues is given by: $T(m) \leq T(m - \lceil\sqrt{2m}\rceil) + 1$, if $m > 1$, and $T(m) \leq 1$, otherwise. ◀

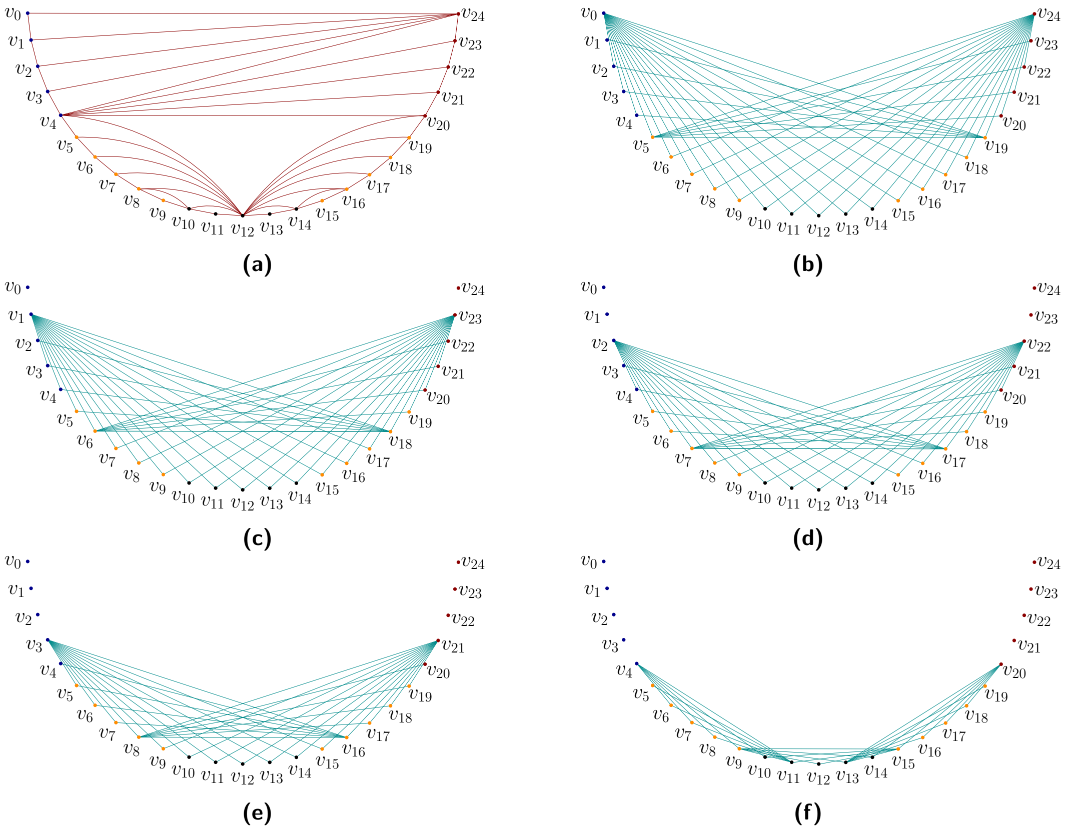
▶ **Theorem 3.2.** *It is* NP*-hard to find the mixed page number of a graph with a given order.*

Proof. Given a permutation $\pi = \langle \pi_1, \dots, \pi_n \rangle$, we construct a graph G on $2n$ vertices, whose vertex order $\prec$ is $1, \dots, n, \pi_1, \dots, \pi_n$, and whose edges are $(1, \pi_1), \dots, (n, \pi_n)$. Finding a mixed layout with k pages for G in $\prec$ is equivalent to deciding if π is k-coverable (i.e., it can be partitioned in k monotone subsequences), which is NP-complete [16]. ◀

▶ **Theorem 3.3.** *An n-vertex graph with mixed page number k has $\leq f(n,k)$ edges, where:*

$$f(n,k) = \begin{cases} 2kn - 2k^2 + k - 2 & \text{if } k \leq \frac{n}{4} + 2 \\ \frac{n^2}{8} + (k+1)n - 3k - 2 & \text{otherwise} \end{cases}$$

Proof. Let $\mathcal{L}$ be a mixed layout of an n-vertex graph G with s stacks and q queues, such that $s + q = k$. W.l.o.g., we assume that the number of edges on each page in $\mathcal{L}$ is maximum. In this regard, the first stack of $\mathcal{L}$ has $2n-3$ edges, while each subsequent one has $n-3$ edges [4]. Similarly, for $1 \leq i \leq q$, the i-th queue of $\mathcal{L}$ has at most $2n - 4i + 1$ edges [8, Lemma 8]. Thus, the first stack is part of $\mathcal{L}$. Since the sparsest queue contains $2n - 4(k-1) + 1$ edges,



(a) **(b)** **(c)** **(d)** **(e)** **(f)**

Figure 3 Illustration for the lower bound of $2nk - 2k^2 + k - 2$ of Note 1 with $n = 25$ and $k = 6$ which yields a 1-stack 5-queue layout with 232 edges in total.

while each of the remaining stacks contains $n - 3$ edges, it follows that, if $k \leq \frac{n}{4} + 2$, then $\mathcal{L}$ is a 1-stack $(k-1)$-queue layout. Otherwise, $\mathcal{L}$ contains $k - \frac{n}{4} - 1$ stacks and $\frac{n}{4} + 1$ queues.

First assume $k \leq \frac{n}{4} + 2$. Let $v_0, \ldots, v_{n-1}$ be the vertices of G as ordered in $\mathcal{L}$. We assume w.l.o.g. that G contains edges $\{(v_i, v_{i+1}), 0 \leq i \leq n-2\}$ assigned to the first stack in $\mathcal{L}$. For $0 \leq i < j \leq n-1$, let the *midpoint* be $\frac{1}{2}(i+j)$. Two edges with the same midpoint cannot belong to the same queue, as they form a 2-rainbow [8]. For $1 \leq i \leq k-1$, at most $i - 1$ edges have a midpoint of $i - 1$, and at most $i - 1$ edges have a midpoint of $i - \frac{1}{2}$. Also, for $1 \leq i \leq k-1$, at most $i - 1$ edges have a midpoint of $n - i$, and at most $i - 1$ edges have a midpoint of $n - i - \frac{1}{2}$. Since $n \geq 2k$, we avoid double-counting. Hence, the number of edges of G, including those in the stack, is at most: $2n - 3 \ + \ 4\sum_{i=1}^{k-1}(i-1) \ + \ (2n - 1 - 4(k-1))(k-1) \ = 2kn - 2k^2 + k - 2$.

Assume now $k > \frac{n}{4} + 2$, i.e., $\mathcal{L}$ contains $k - \frac{n}{4} - 1$ stacks and $\frac{n}{4} + 1$ queues. By our discussion above, $\mathcal{L}$ contains at most $\frac{3n^2}{8} + \frac{9n}{4} - 8$ edges in total in its first stack and in its $\frac{n}{4} + 1$ queues. $\mathcal{L}$ contains at most $kn - 3k - \frac{n^2}{4} - \frac{5n}{4} + 6$ edges in its remaining $k - \frac{n}{4} - 2$ stacks, as each of them has at most $n - 3$ edges. Hence, the number of edges in G is at most $\frac{n^2}{8} + (k+1)n - 3k - 2$. ◀

▶ **Note 1.** *For every n and k with $n \geq 4k + 1$, there is an n-vertex graph with mixed page number k and $2kn - 2k^2 + k - 2$ edges.*

Sketch of proof. For an illustration see Fig. 3; for details refer to [1]. ◀

4 Complete Graphs

In this section, we give bounds on the mixed page number of the complete graph K_n. Note that the stack and the queue number of K_n is $\lceil \frac{n}{2} \rceil$ and $\lfloor \frac{n}{2} \rfloor$, respectively, see [4, 13].

▶ **Theorem 4.1.** *The mixed page number of K_n is at least $\left\lceil \frac{3(n-4)}{8} \right\rceil$ and at most $2 \lceil \frac{n}{5} \rceil$.*

Proof. The lower bound follows from Theorem 3.3. For the upper bound, we prove that K_n admits a $\frac{n}{5}$-stack $\frac{n}{5}$-queue layout $\mathcal{L}_n$ for n being a multiple of 5. Let $v_0, \ldots, v_{n-1}$ be the order of the vertices of K_n in $\mathcal{L}_n$. First, we assign edges to queues $\mathcal{Q}_0, \ldots, \mathcal{Q}_{\frac{n}{5}-1}$ of $\mathcal{L}_n$, such that $\mathcal{Q}_i$, $0 \leq i \leq \frac{n}{5} - 1$, contains the following $2n - 4i - 5$ edges (see Figs. 4a to 4e):

- (v_i, v_j) $\quad i+2 \leq j \leq \frac{4n}{5} - i - 2$
- (v_j, v_{n-1-i}) $\quad \frac{n}{5} + 2i \leq j \leq n - i - 3$
- $(v_{\frac{n}{5}+2i}, v_j)$ $\quad \frac{4n}{5} - i - 1 \leq j \leq n - i - 2$
- $(v_j, v_{\frac{4n}{5}-i-2})$ $\quad i+1 \leq j \leq \frac{n}{5} + 2i$

Next, we assign edges to stacks $\mathcal{S}_0, \ldots, \mathcal{S}_{\frac{n}{5}-1}$ of $\mathcal{L}_n$, such that $\mathcal{S}_i$, $0 \leq i \leq \frac{n}{5} - 1$, contains the following $\frac{4n}{5} + i - 4$ edges (see Figs. 4f to 4j):

● $(v_{\frac{n}{5}+2i+1}, v_j)$ $\quad \frac{4n}{5} - i - 1 \leq j \leq n - i - 2,$ $\quad$ for $(i,j) \neq (\frac{n}{5} - 1, n - 1)$
● (v_{n-i-1}, v_j) $\quad \frac{n}{5} - i \leq j \leq \frac{n}{5} - 1$
● $(v_{\frac{n}{5}-i-1}, v_j)$ $\quad n - i - 1 \leq j \leq n - 1,$ $\quad$ for $i \neq \frac{n}{5} - 1$
● $(v_{\frac{4n}{5}-i-1}, v_j)$ $\quad \frac{3n}{5} \leq j \leq \frac{4n}{5} - i - 3,$ $\quad$ for $(i,j) \neq (\frac{n}{5} - 1, \frac{n}{5} - 1)$
● $(v_{2i+j+1}, v_{\frac{2n}{5}-j-1})$ $\quad \frac{n}{5} \leq j \leq \frac{2n}{5} - i - 2$
● $(v_{2i+j+2}, v_{\frac{2n}{5}-j-1})$ $\quad \frac{n}{5} \leq j \leq \frac{2n}{5} - i - 3,$ $\quad$ for $(i,j) \neq (\frac{n}{5} - 1, \frac{n}{5} - 1)$
● (v_j, v_{2i-j+1}) $\quad \frac{n}{5} \leq j \leq \frac{n}{5} + i - 1$
● (v_j, v_{2i-j}) $\quad \frac{n}{5} \leq j \leq \frac{n}{5} + i - 1$

By construction, no two edges in the same stack (queue) cross (nest), and no edge is assigned to two distinct pages. In total, $\mathcal{L}_n$ has $\frac{1}{2}(n^2 - 3n)$ edges, i.e., the number of edges of K_n neglecting the edges (v_0, v_{n-1}) and (v_i, v_{i+1}), $i = 0, \ldots, n-2$, which have been skipped in the above assignment scheme, since they can be accommodated to any of the stacks of $\mathcal{L}_n$. ◀

5 Complete Bipartite Graphs

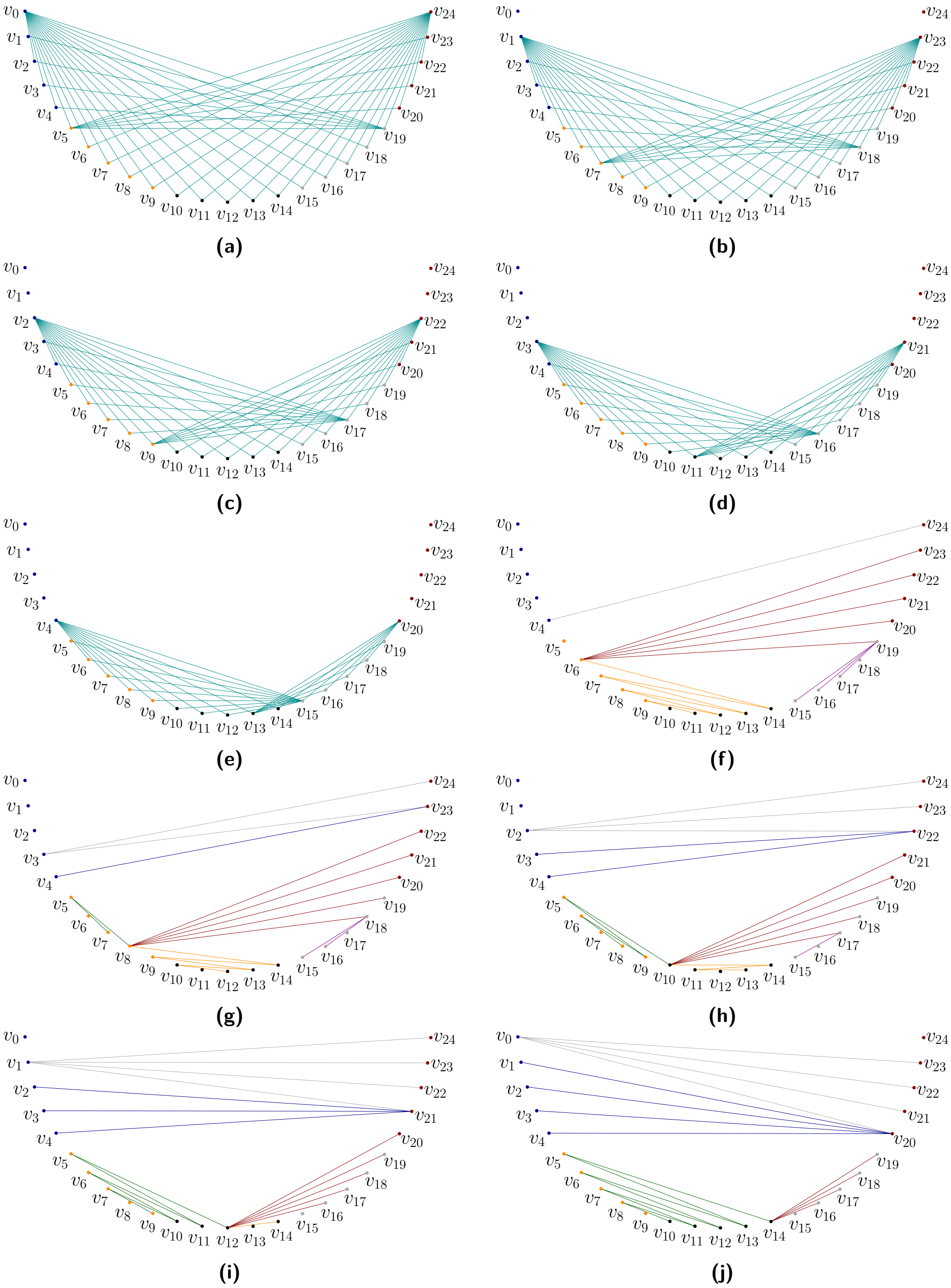
In this section, we study the mixed page number of $K_{n,n} = \{u_0, \ldots, u_{n-1}\} \times \{v_0, \ldots, v_{n-1}\}$.

5.1 The separated setting

In the separated setting, $u_0 \prec \ldots \prec u_{n-1} \prec v_0 \prec \ldots \prec v_{n-1}$ holds; see, e.g., the work by Da Lozzo et al. [3]. Here, two crossing (nesting) edges are nesting (crossing), when one reverses the order of the vertices of one of the two parts of $K_{n,n}$, i.e., stacks and queues are *interchangeable*; see Fig. 5. We map the edge (u_i, v_j) of $K_{n,n}$ to the point (i, j) of the $n \times n$ integer grid $H = [0, n-1] \times [0, n-1]$.

▶ **Proposition 1.** *The edges assigned to the same queue (stack) of a mixed layout $\mathcal{L}$ of $K_{n,n}$ form a not necessarily strict monotonically increasing (decreasing) path in H.*

Proof. Consider (by interchangeability) the edges assigned to the same queue of $\mathcal{L}$ in their lexicographic order, i.e., (u_i, v_j) *precedes* (u_k, v_ℓ) if $i = k$ and $j < \ell$ holds, or $i < k$ and $j \leq \ell$ holds. In this order, for any edge (u_i, v_j) any subsequent edge (u_k, v_ℓ) is mapped to a point (k, ℓ) in the upper-right quadrant of (i, j) in H (see Fig. 5), which proves the property. ◀



(a) **(b)** **(c)** **(d)** **(e)** **(f)** **(g)** **(h)** **(i)** **(j)**

Figure 4 Illustration for the upper bound of Theorem 4.1 with $n = 25$, which yields a 5-stack 5-queue layout.

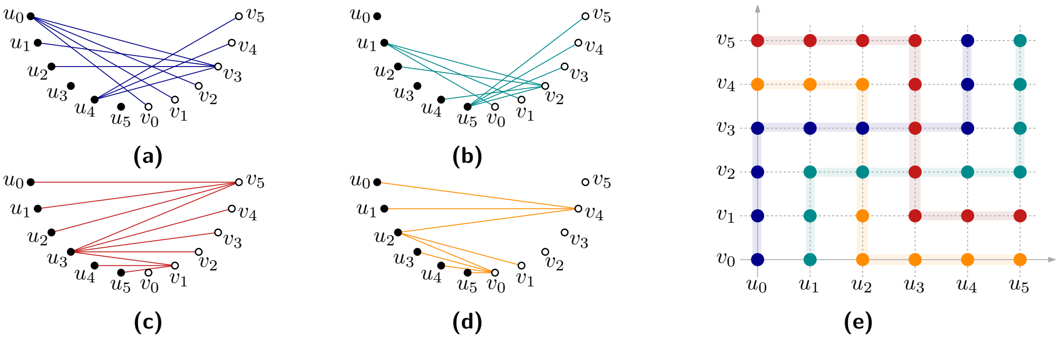

(a)
(b)
(c)
(d)
(e)
u_0 u_1 u_2 u_3 u_4 u_5
v_0 v_1 v_2 v_3 v_4 v_5

Figure 5 (a)–(d) A 2-stack 2-queue layout of $K_{6,6}$ in the separated setting, and (e) the grid corresponding to this layout.

We call a monotonically decreasing (increasing) path of H a *stack-path* (*queue-path*).

▶ **Proposition 2.** *A stack-path starting at (i, j) and ending at (k, ℓ) covers at most $k - i + j - \ell + 1$ grid points; a corresponding queue-path at most $k - i + \ell - j + 1$.*

Proof. It holds since the Manhattan distance between (i, j) and (k, ℓ) is $|k - i| + |\ell - j|$. ◀

▶ **Theorem 5.1.** *The mixed page number of $K_{n,n}$ in the separated setting is at most $\lceil \frac{2n}{3} \rceil$.*

Proof. By Proposition 1, $K_{n,n}$ admits a $\frac{n}{3}$-stack $\frac{n}{3}$-queue layout $\mathcal{L}_n$ with n being a multiple of 3, if and only if, the n^2 points of the integer grid H can be covered with a set of $\frac{n}{3}$ queue-paths and a set of $\frac{n}{3}$ stack-paths, which is possible to be done as in Fig. 6. ◀

Next, we focus on a lower bound for $K_{n,n}$. For this, we need two more properties given in Propositions 3 and 4. The former is proved in [1]; the latter one is symmetric.

▶ **Proposition 3.** *Let Q be a set of q queue-paths covering a largest set $P(Q)$ of points of H. Then, $P(Q)$ can be covered by q queue-paths $p_0 \dots, p_{q-1}$, where p_i has length $2n - 1 - 2i$, starts at $(0, i)$ and ends at $(n - 1 - i, n - 1)$; $i = 0, \dots, q - 1$.*

▶ **Proposition 4.** *Let S be a set of s stack-paths covering a largest set $P(S)$ of points of H. Then, $P(S)$ can be covered by s stack-paths $p_0 \dots, p_{s-1}$, where p_i has length $2n - 1 - 2i$, starts at $(0, n - 1 - i)$ and ends at $(n - 1, i)$; $i = 0, \dots, s - 1$.*

The next lemma provides an estimation on the maximum number edges of an s-stack q-queue layout of a subgraph of $K_{n,n}$ in the separated setting.

▶ **Lemma 5.2.** *Let S and Q be two sets of s stack-paths and q queue-paths covering a largest set of points of H. Then, $S \cup Q$ covers $\sum_{i=0}^{s-1}(2n - 1 - 2i) + \sum_{i=0}^{q-1}(2n - 1 - 2i) - sq$ grid points of H.*

Proof. W.l.o.g., we assume that S and Q are maximal, as otherwise we can extend them to maximal. Then, the sets $P(S)$ and $P(Q)$ of the points of H that are covered by the paths in S and Q can be covered by s stack-paths and q queue-paths that have the properties of the last two propositions. Clearly, each such stack-path and each such queue-path have at least one point of H in common. Therefore, $|P(S) \cap P(Q)| \geq sq$ holds, and thus $|P(S) \cup P(Q)| \leq |P(S)| + |P(Q)| - |P(S) \cap P(Q)| = \sum_{i=0}^{s-1}(2n - 1 - 2i) + \sum_{i=0}^{q-1}(2n - 1 - 2i) - sq$ holds. ◀

▶ **Corollary 5.3.** *In a mixed layout of $K_{n,n}$ in the separated case, any collection of s-stacks and q-queues contains at most $2n(q + s) - q^2 - s^2 - sq$ edges.*

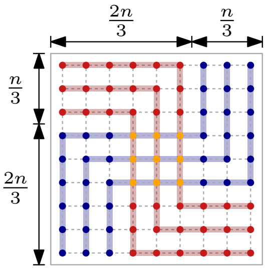


Figure 6 Illustration for the proof of Theorem 5.1 in which every stack-path shares exactly one point with every queue-path.

Next we prove that the upper bound of Theorem 5.1 is tight.

▶ **Theorem 5.4.** *The mixed page number of $K_{n,n}$ in the separated setting is $\lceil \frac{2n}{3} \rceil$.*

Proof. The upper bound follows from Theorem 5.1. For the lower bound, denote by k the total number of pages in an s-stack, q-queue layout $\mathcal{L}_n$ of $K_{n,n}$. By Theorem 5.3, we obtain $2n(q+s) - q^2 - s^2 - sq \geq n^2$. Since $k = s + q$, it follows that $2nk - q^2 - k^2 + kq \geq n^2$. To determine the maximum for the left-hand side of the inequality with respect to q, we compute the roots of its first derivative taken over q, which is $\frac{\partial}{\partial q}\left(2nk - q^2 - k^2 + kq\right) = 0$. This yields $k = 2q$, i.e., $s = q = \frac{k}{2}$. Thus, $2nk - \frac{3k^2}{4} - n^2 \geq 0$ holds. Hence, $k \geq \lceil \frac{2n}{3} \rceil$. ◀

5.2 Non-separated setting

We now study the mixed page number of $K_{n,n}$ in the general (non-separated) setting. Here, we are not able to provide an upper bound that is better than $\lfloor \frac{n}{2} \rfloor$, i.e., the queue number of $K_{n,n}$ [13]. We conjecture that this bound is tight, but we do not have a proof for this. However, we are able to provide a lower bound using the techniques of the previous section.

▶ **Theorem 5.5.** *The mixed page number of $K_{n,n}$ is at least $\lceil \frac{n}{3} \rceil$.*

Proof. By the pigeonhole principle, in any vertex order of $K_{n,n}$ at least $\lceil \frac{n}{2} \rceil$ of the first n vertices belong to one part. Then at least $\lceil \frac{n}{2} \rceil$ of the remaining n vertices belong to the other part. In other words, every vertex order induces a $K_{\lceil \frac{n}{2} \rceil, \lceil \frac{n}{2} \rceil}$ in the separated setting, which by Theorem 5.4 requires at least $\lceil \frac{n}{3} \rceil$ pages. ◀

6 Conclusions

In this work, we shed some light on the mixed page number of some basic graph classes. We want to emphasize the following open questions: (i) Can the gaps of the bounds on the mixed page numbers of K_n and $K_{n,n}$ be closed? We believe that space for improvement is on the side of the lower bounds. (ii) Do graphs with bounded mixed page number have bounded stack or queue number? By a recent result of Dujmović et al. [6], this question can be answered affirmatively, only if queue number is bounded by stack number for every graph. The question is interesting even for separated layouts. (iii) Finally, we suggest to investigate the mixed page number of other classes of graphs such as k-planar graphs.

References

1 Alam, J.M., Bekos, M.A., Gronemann, M., Kaufmann, M., Pupyrev, S.: The mixed page number of graphs. CoRR abs/2107.04993 (2021), https://arxiv.org/abs/2107.04993

2 Angelini, P., Bekos, M.A., Kindermann, P., Mchedlidze, T.: On mixed linear layouts of series-parallel graphs. In: Auber, D., Valtr, P. (eds.) Graph Drawing and Network Visualization. LNCS, vol. 12590, pp. 151–159. Springer (2020), https://doi.org/10.1007/978-3-030-68766-3_12

3 Angelini, P., Da Lozzo, G., Di Battista, G., Frati, F., Patrignani, M.: 2-level quasi-planarity or how caterpillars climb (spqr-)trees. In: Marx, D. (ed.) Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021. pp. 2779–2798. SIAM (2021), https://doi.org/10.1137/1.9781611976465.165

4 Bernhart, F., Kainen, P.C.: The book thickness of a graph. J. Comb. Theory, Ser. B 27(3), 320–331 (1979), https://doi.org/10.1016/0095-8956(79)90021-2

5 de Col, P., Klute, F., Nöllenburg, M.: Mixed linear layouts: Complexity, heuristics, and experiments. In: Archambault, D., Tóth, C.D. (eds.) Graph Drawing and Network Visualization. LNCS, vol. 11904, pp. 460–467. Springer (2019), https://doi.org/10.1007/978-3-030-35802-0_35

6 Dujmović, V., Eppstein, D., Hickingbotham, R., Morin, P., Wood, D.R.: Stack-number is not bounded by queue-number. arXiv:2011.04195 (2020)

7 Dujmović, V., Joret, G., Micek, P., Morin, P., Ueckerdt, T., Wood, D.R.: Planar graphs have bounded queue-number. J. ACM 67(4), 22:1–22:38 (2020), https://dl.acm.org/doi/10.1145/3385731

8 Dujmović, V., Wood, D.R.: On linear layouts of graphs. Discrete Mathematics & Theoretical Computer Science 6(2), 339–358 (2004), http://dmtcs.episciences.org/317

9 Dujmović, V., Wood, D.R.: Stacks, queues and tracks: Layouts of graph subdivisions. Discrete Mathematics & Theoretical Computer Science 7(1), 155–202 (2005), http://dmtcs.episciences.org/346

10 Enomoto, H., Miyauchi, M.: Stack-queue mixed layouts of graph subdivisions. In: Forum on Information Technology. pp. 47–56 (2014)

11 Ganley, J.L., Heath, L.S.: The pagenumber of k-trees is $O(k)$. Discrete Applied Mathematics 109(3), 215–221 (2001), https://doi.org/10.1016/S0166-218X(00)00178-5

12 Heath, L.S., Leighton, F.T., Rosenberg, A.L.: Comparing queues and stacks as mechanisms for laying out graphs. SIAM J. Discrete Math. 5(3), 398–412 (1992), https://doi.org/10.1137/0405031

13 Heath, L.S., Rosenberg, A.L.: Laying out graphs using queues. SIAM J. Comput. 21(5), 927–958 (1992), https://doi.org/10.1137/0221055

14 Malitz, S.M.: Graphs with E edges have pagenumber $O(\sqrt{E})$. J. Algorithms 17(1), 71–84 (1994), https://doi.org/10.1006/jagm.1994.1027

15 Pupyrev, S.: Mixed linear layouts of planar graphs. In: Frati, F., Ma, K. (eds.) Graph Drawing and Network Visualization. LNCS, vol. 10692, pp. 197–209. Springer (2017), https://doi.org/10.1007/978-3-319-73915-1_17

16 Wagner, K.W.: Monotonic coverings of finite sets. J. Inf. Process. Cybern. 20(12), 633–639 (1984)

17 Wiechert, V.: On the queue-number of graphs with bounded tree-width. Electr. J. Comb. 24(1), P1.65 (2017), http://www.combinatorics.org/ojs/index.php/eljc/article/view/v24i1p65

18 Yannakakis, M.: Embedding planar graphs in four pages. J. Comput. Syst. Sci. 38(1), 36–67 (1989), https://doi.org/10.1016/0022-0000(89)90032-9

Gioan's Theorem for complete bipartite graphs*

Oswin Aichholzer[1]**, Man-Kwun Chiu**[2]**, Hung P. Hoang**[3]**, Michael Hoffmann**[3]**, Yannic Maus**[1]**, Birgit Vogtenhuber**[1]**, and Alexandra Weinberger**[1]

1 **Institute of Software Technology, Graz University of Technology, Austria**
[oaich,yannic.maus,bvogt,weinberger]@ist.tugraz.at

2 **Institut für Informatik, Freie Universität Berlin, Germany**
chiumk@zedat.fu-berlin.de

3 **Department of Computer Science, ETH Zürich, Switzerland**
[hung.hoang|hoffmann]@inf.ethz.ch

Abstract

For a drawing of a labeled graph, the rotation of a vertex or crossing is the cyclic order of its incident edges, presented by the labels of their other endpoints. The extended rotation system of the drawing is the collection of the rotations of all vertices and crossings. A drawing is simple if each pair of edges has at most one common point. Gioan's Theorem states that for any two simple drawings of the complete graph K_n with the same crossing edge pairs, one drawing can be transformed into the other by a sequence of *triangle flips* (a.k.a. Reidemeister moves of Type 3). Intuitively, this operation refers to the act of moving one edge of a triangular cell formed by three pairwise crossing edges over the opposite vertex of the cell.

We investigate to what extent Gioan's Theorem generalizes to other classes of graphs. On the one hand, we show that it holds for complete bipartite graphs $K_{m,n}$, provided that the two drawings share the same extended rotation system. Note that the assumption is also implicit in Gioan's Theorem, because for simple drawings of the complete graph the crossing edge pairs uniquely determine the extended rotation system; however, this is not the case for complete bipartite graphs. Our proof uses a Carathéodory-type theorem for simple drawings of complete bipartite graphs, which may be of independent interest. On the other hand, we show that the theorem does not hold if the graph is slightly sparser: When removing two edges from $K_{m,n}$, there exist two simple drawings with the same extended rotation system that cannot be transformed into each other using triangle flips.

1 Introduction

Given a simple drawing of a graph $G = (V, E)$ on the sphere $\mathcal{S}$, an *edge fragment* is a maximal connected part of an edge that does not contain any endpoint or crossing. The *rotation of a vertex* is the clockwise circular order of incident edges. The *rotation of a crossing* χ is the clockwise cyclic order of the four vertices of the crossing edge pair which is induced by the cyclic order of edge fragments around χ. (In other words, the rotation of a crossing χ is the rotation of an additional degree-4 vertex v_χ obtained by splitting the crossing edge pair at χ and replacing χ by v_χ.) The *extended rotation system* (ERS) of a drawing is the collection of rotations of all vertices and crossings.

* Research on this work was initiated during the 5th D-A-CH Workshop on Arrangements and Drawings in March 2021. We thank all participants of this workshop for the great atmosphere and fruitful discussions. O.A. and A.W. partially supported by the Austrian Science Fund (FWF), project W1230. M.C. partially supported by ERC StG 757609. M.H. supported by the Swiss National Science Foundation within the collaborative D-A-CH project *Arrangements and Drawings* as SNSF project 200021E-171681. B.V. partially supported by the Austrian Science Fund within the collaborative D-A-CH project *Arrangements and Drawings* as FWF project I 3340-N35.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

A *crossing triangle* is a cell in the subdrawing of three pairwise crossing edges that is bounded by three edge fragments. To define the orientation of a crossing triangle, we fix an arbitrary orientation for each edge of G. The *orientation* of a crossing triangle Δ is the parity (odd or even) of the number of edges that bound Δ and where Δ lies to the left of the edge. A crossing triangle Δ is *invertible* if there exists another simple drawing of the same graph and with the same extended rotation system (ERS) in which Δ appears in the opposite orientation. A *triangle flip* is the elementary operation of changing the orientation of an unintersected crossing triangle by a local transformation of the given drawing; see Figure 1.

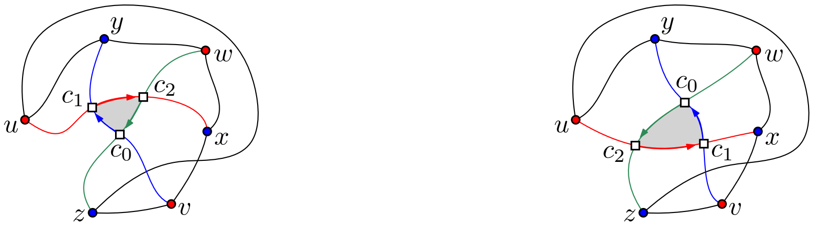


Figure 1 Two drawings of $K_{3,3}$ that can be transformed into each other via one triangle flip.

Two simple drawings γ and η of G are strongly isomorphic, denoted by $\gamma \cong \eta$, if there exists an orientation-preserving homeomorphism of $\mathcal{S}$ that maps γ to η, that is, $\gamma_v \mapsto \eta_v$, for all $v \in V$, and $\gamma_e \mapsto \eta_e$, for all $e \in E$. By Kynčl [5], the following combinatorial formulation is equivalent for connected drawings: (1) the same pairs of edges cross (this is called *weak isomorphism*); (2) the order of crossings along each edge is the same; and (3) the drawings have the same ERS. In this work, strongly isomorphic drawings are considered the same.

Gioan's Theorem [4] states that any two weakly isomorphic simple drawings of K_n can be transformed into each other via a sequence of triangle flips. Gioan announced his theorem in 2005 [4]. The original presentation contained a proof sketch, but a full proof was published only 10 years later by Arroyo, McQuillan, Richter, and Salazar [1], who also coined the name "Gioan's Theorem". In 2021, Schaefer generalized Gioan's Theorem to slightly sparser graphs by proving that any two weakly isomorphic simple drawings of $K_n \setminus M$, where M is a non-perfect matching, can be transformed into each other using triangle flips [7]. His work also includes an alternative proof for Gioan's Theorem.

Our main result is that an analogue of Gioan's Theorem also holds for simple drawings of much sparser graphs, namely, for complete bipartite graphs. To show this, we rephrase the statement to require both drawings to have the same ERS.

▶ **Theorem 1.1.** *Let D_1 and D_2 be two simple drawings of $K_{m,n}$, $m, n \geq 1$, on the sphere with the same ERS. Then there is a sequence of triangle flips that transforms D_1 into D_2.*

Note that triangle flips only change the order of crossings along edges. Hence, having the same ERS is a necessary requirement for any two drawings of any graph to be transformable into each other via triangle flips. For the complete graph, the requirement that the drawings have the same crossing edge pairs is equivalent to the requirement that they have the same ERS because the crossing edge pairs of a drawing uniquely determine its ERS [5, 6]. However, for complete bipartite graphs this is not the case, as two simple drawings of $K_{m,n}$ with the same crossing edge pairs might have different ERSs; see Figure 2 for an example.

We also show that both Gioan's Theorem and our Theorem 1.1 are almost tight.

▶ **Theorem 1.2.** *For any $m \geq 2$ and $n \geq 3$ and $K_{m,n}$ minus two edges, there exist two simple drawings with the same ERS that cannot be transformed into each other using triangle flips. The same holds for any $n \geq 5$ and K_n minus a four-cycle C_4.*

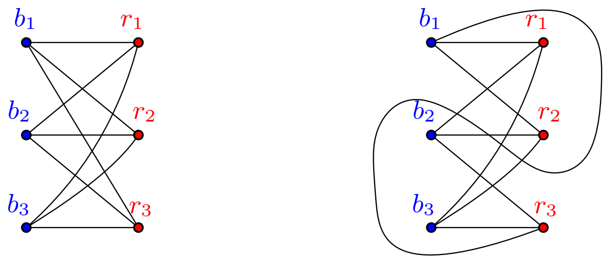


Figure 2 Two simple drawings of $K_{3,3}$ with the same crossing edge pairs but different ERSs.

In particular, the first part of Theorem 1.2 implies that an analogue to Schaefer's generalization of Gioan's Theorem for K_n minus a non-perfect matching cannot be achieved for complete bipartite graphs, as not even a generalization from $K_{m,n}$ to $K_{m,n}$ minus a matching of size two holds. Moreover, note that $K_{m,n}$ with $m \geq 4$ and $n \geq 1$ is a subgraph of $K_{n+m} \setminus C_4$. Hence the second part of Theorem 1.2 implies that—quite counterintuitively—the set of graphs for which a Gioan-type statement holds is not closed under adding edges.

To prove Theorem 1.1, we use a similar approach as Arroyo et al. [1]. In their proof, they iteratively transform one of the drawings so as to increase the parts of both drawings that are strongly isomorphic. However, several ingredients that are necessary for this transformation are known properties of drawings of complete graphs or follow directly, while it was unknown whether analogous statements hold for drawings of complete bipartite graphs. Hence, for our proof, we discover a number of useful, fundamental properties of simple drawings of complete bipartite graphs. For example, we establish an analogue to Carathéodory's Theorem for simple drawings of $K_{m,n}$.

The classic Carathéodory Theorem states that if a point $p \in \mathbb{R}^2$ lies in the convex hull of a set $A \subset \mathbb{R}^2$ of $n \geq 3$ points, then there exists a triangle spanned by points of A that contains p. In the terminology of drawings, this means that if a point p lies in a bounded cell of a straight-line drawing D of K_n in the plane, then there also exists a 3-cycle C of D so that p lies in the bounded cell of C. This statement has been generalized to simple (not necessarily straight-line) drawings of K_n [2, 3]. However, it clearly does not generalize to arbitrary (non-complete) graphs. A natural question is, for which classes of graphs this statement, or a variation of it, holds. We show that it holds for complete bipartite graphs if we replace the (non-existing) 3-cycle by a 4-cycle, which is the shortest available cycle.

▶ **Theorem 1.3** (Carathéodory's Theorem for simple drawings of $K_{m,n}$). *Let D be a simple drawing of $K_{m,n}$ in the plane, for $m, n \geq 2$, and let p be a point in some bounded cell of D. Then there exists a 4-cycle C of D such that p is contained in a bounded cell of C. This statement is tight in the sense that it does not hold for $K_{m,n}$ minus one edge.*

Outline. We prove Theorem 1.1 in Section 2. The proof relies on several lemmata, whose proofs are deferred to the upcoming full version of this paper. A sketch of the proof of Theorem 1.2 can be found in Section 3.

2 Proof of Gioan's Theorem for simple drawings of $K_{m,n}$

Denote the bipartition sets by $B = \{b_1, b_2, ..., b_m\}$ and $R = \{r_1, r_2, ..., r_n\}$. Let $D :\cong D_1$. We will do triangle flips in D, by this changing D, until we obtain $D \cong D_2$. We iteratively consider the vertices $r_1, \ldots, r_n$. For each vertex r_i, we iteratively consider the incident edges $r_ib_1 \dots, r_ib_m$. We denote by $K_{m,i}$ the subgraph of $K_{m,n}$ induced by B and the vertices $r_1, r_2, \dots, r_i$, and let $X_{i,j} = K_{m,i-1} \cup \{r_ib_k \colon 1 \leq k \leq j\}$ for $0 \leq k \leq m$, with $X_{i,0} = K_{m,i-1}$.

When considering an edge r_ib_j, the goal is to establish $D[X_{i,j}] \cong D_2[X_{i,j}]$, where $D[X_{i,j}]$ and $D_2[X_{i,j}]$ are the according subdrawings of D and D_2, respectively.

For the base case $i = 1$ observe that $D[K_{m,1}] \cong D_2[K_{m,1}]$ because there is only one simple drawing of $K_{m,1}$ (our graphs are labeled but the ERS is given).

For the general case $2 \leq i \leq n$ and $1 \leq j \leq m$, assume that $D[X_{i,j-1}] \cong D_2[X_{i,j-1}]$. To handle the case $j = 1$, we first argue that the position of vertex r_i is consistent between $D[K_{m,i-1}]$ and $D_2[K_{m,i-1}]$. To show this, we use the following lemma, whose proof relies on Theorem 1.3 (Carathéodory's Theorem for simple drawings of $K_{m,n}$).

▶ **Lemma 2.1.** *Let F be a simple drawing of $K_{m,n}$, $m, n \geq 1$, on the sphere. For any vertex v in F, the ERS of F uniquely determines in which cell of $F' := F \setminus \{v\}$ the vertex v lies.*

Since $D[K_{m,i-1}] \cong D_2[K_{m,i-1}]$, the two drawings topologically have the same cells. As D and D_2 have the same ERS, by Lemma 2.1 applied to $F = D[K_{m,i}]$ and to $D_2[K_{m,i}]$, both times with $v = r_i$, we conclude that r_i lies in the same cell in $D[K_{m,i-1}]$ and $D_2[K_{m,i-1}]$.

Now consider the edge r_ib_j. The aim is to use a sequence of triangle flips to transform D such that $D[X_{i,j}] \cong D_2[X_{i,j}]$. Let e_1 denote the curve that represents r_ib_j in D. We imagine to add another copy $\widetilde{e_2}$ of r_ib_j to D, which corresponds to the curve e_2 that represents the edge r_ib_j in D_2 and serves as a "target" curve which we aim to transform e_1 into.

▶ **Lemma 2.2.** *There exists a simple curve $\widetilde{e_2}$ such that $D[X_{i,j-1}] \cup \widetilde{e_2} \cong D_2[X_{i,j}]$ and e_1 and $\widetilde{e_2}$ have finitely many intersections in $D[X_{i,j}] \cup \widetilde{e_2}$.*

Now fix such a curve $\widetilde{e_2}$. Then $\Gamma = e_1 \cup \widetilde{e_2}$ forms a (not necessarily simple) closed curve. With the next lemma, we show that there is a *lens* in Γ which we can use as a starting point for transforming e_1 to $\widetilde{e_2}$. A *lens* in Γ is a cell whose boundary is formed by exactly two edge fragments of Γ, one from e_1 and one from $\widetilde{e_2}$.

▶ **Lemma 2.3.** *In Γ there is a lens that does not contain any vertex of $K_{m,i}$.*

Now consider a lens L as guaranteed by Lemma 2.3. While L does not contain any vertex of $D[X_{i,j-1}]$, it may contain crossings of $D[X_{i,j-1}]$. As a next step, we aim to transform D using triangle flips such that L does not contain any crossings of $D[X_{i,j-1}]$. Let $\chi \in L$ be a crossing of two edges a_1, a_2 in $D[X_{i,j-1}]$. As r_i and b_j are the only vertices on $e_1 \cup \widetilde{e_2}$, it follows that each of a_1, a_2 crosses ∂L twice; as both D and D_2 are simple drawings, one of these crossings is with e_1 and the other is with $\widetilde{e_2}$. Thus, a_1, a_2, and e_1 form a crossing triangle Δ_{e_1}. Moreover, the corresponding crossing triangle in D_2 has the opposite orientation, and hence Δ_{e_1} is invertible. By the following lemma, Δ_{e_1} is empty of *all* vertices of D (we already knew this for the vertices of $K_{m,i}$, but not yet for $r_{i+1}, \dots, r_n$).

▶ **Lemma 2.4** (Invertible triangles are empty). *Let D be a simple drawing of $K_{m,n}$ and Δ be an invertible crossing triangle in D. Then all vertices of D lie outside Δ.*

Since Δ_{e_1} is empty of all vertices, all edges crossing Δ_{e_1} can be "swept" outside Δ_{e_1} using a finite sequence of triangle flips (analogous to topological sweeps). None of those flips increases the number of crossings in L (while some of them might decrease this number) and after them, Δ_{e_1} is unintersected. Finally, we also flip Δ_{e_1} so that $\chi \notin L$.

Processing all remaining crossings inside L in the described fashion, we establish that in the resulting drawing, the lens L does not contain any vertex or crossing of $D[X_{i,j-1}]$. In other words, locally around L, the edge e_1 is topologically identical to $\widetilde{e_2}$ with respect to $D[X_{i,j-1}]$. Thus, we can adapt $\widetilde{e_2}$ by replacing its edge part on ∂L with a close copy of the edge part of e_1 on ∂L, effectively removing the lens L from Γ. As a result, the edges e_1 and $\widetilde{e_2}$ have fewer crossings than before in D, and the parameters D and $\Gamma = e_1 \cup \widetilde{e_2}$ again meet the conditions of Lemma 2.3. Repeatedly applying this procedure to the next cell (which exists by Lemma 2.3), we eventually obtain a drawing $D[X_{i,j}] \cup \widetilde{e_2}$ where e_1 and $\widetilde{e_2}$ do not cross, and hence Γ is a simple closed curve. By Lemma 2.3, one of the two cells bounded by Γ contains no vertices of $D[X_{i,j}]$. So after one last round of transformations as described above, we obtain a drawing $D[X_{i,j}] \cup \widetilde{e_2}$ in which all vertices and crossings lie on one side of Γ. Hence we have obtained $D[X_{i,j}] \cong D_2[X_{i,j}]$. Processing all vertices r_i, for $i = 2, \ldots, n$, and in turn handling all edges incident to r_i eventually yields a drawing $D \cong D_2$.

3 Sketch of the proof of Theorem 1.2

Figure 3 depicts the drawings we use in the proof of Theorem 1.2. The first row contains drawings of $K_{m,n}$ minus two adjacent edges, the second one drawings of $K_{m,n}$ minus two disjoint edges, and the third row is for K_{n+m} minus a C_4. In each row, the (green) edge r_1b_1 crosses b_2r_2 and b_2r_3 in a different order and these three edges do not form any crossing triangle. Thus, the drawings cannot be transformed into each other via triangle flips.

4 Conclusion & Open Questions

We have shown Gioan's Theorem for complete bipartite graphs (Theorem 1.1) and that an according statement does not hold for $K_{m,n}$ minus two edges or K_n minus a C_4 (Theorem 1.2). These results relevantly extend previous results [1, 4, 7] and show that the class of graphs for which an according statement holds is also not closed under adding edges. We believe that our result can be extended to complete k-partite graphs. But a complete characterization of graphs for which an according statement holds remains open.

▶ **Question 1.** *What is a complete characterization of all graphs for which Gioan's Theorem holds, that is, for which graphs is it true that any two drawings with the same ERS can be transformed into each other?*

Further, we have shown that an analogue of Caratheorody's Theorem holds for simple drawings of $K_{m,n}$ (Theorem 1.3). It would be interesting to know for which further graphs a similar statement is true.

Finally, in our proof of Theorem 1.1, we did not address algorithmical questions, and neither did the according proofs for Gioan's Theorem for K_n. Naturally, the minimum flip distance, that is, the minimum number of triangle flips that need to be done to transform the drawings, is of interest.

▶ **Question 2.** *What is the worst case minimum flip distance between two simple drawings of $K_{m,n}$ with the same ERS? And what is the worst case minimum flip distance between two simple drawings of K_n with the same rotation system?*

References

1 Alan Arroyo, Dan McQuillan, R. Bruce Richter, and Gelasio Salazar. Drawings of K_n with the same rotation scheme are the same up to triangle-flips Gioan's theorem). *Australasian J. Combinatorics*, 67(2):131–144, 2017. URL: https://ajc.maths.uq.edu.au/pdf/67/ajc_v67_p131.pdf.

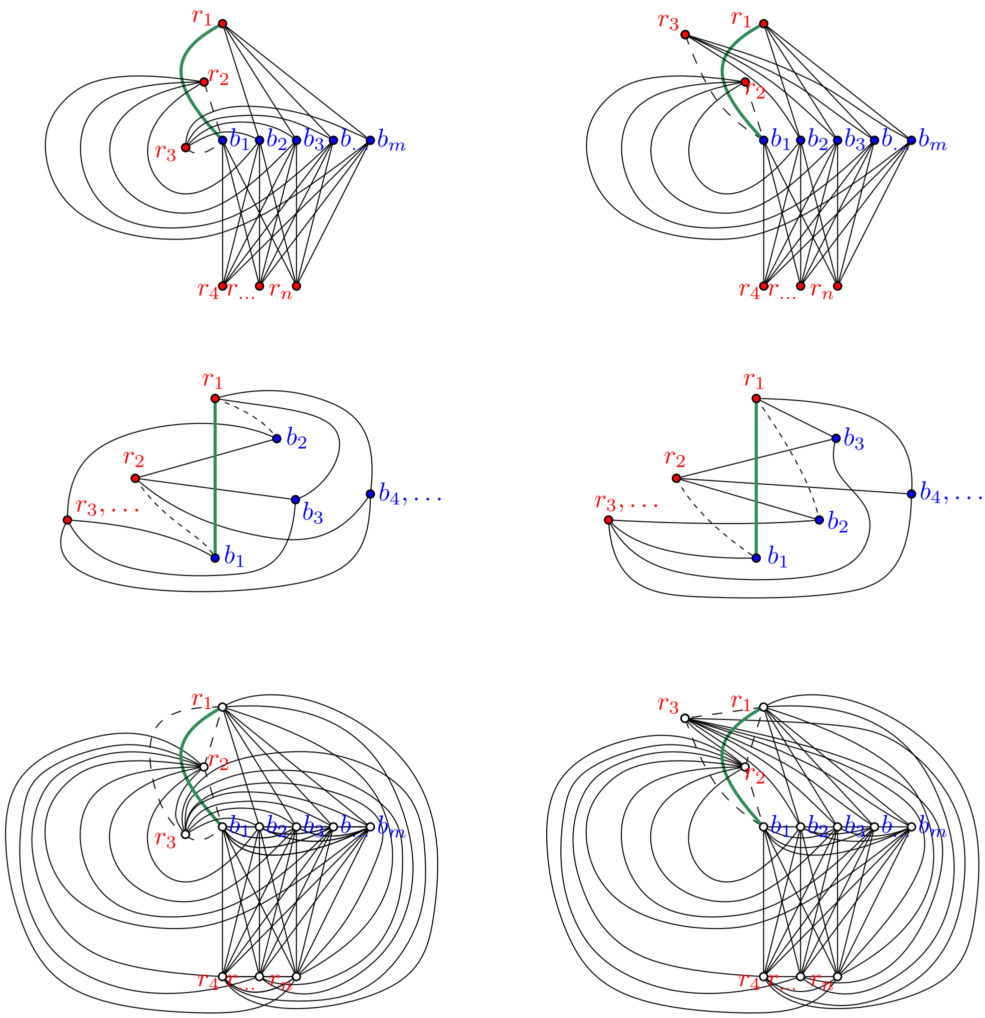


Figure 3 Constructions used in the proof of Theorem 1.2. Dashed arcs indicate omitted edges.

2 Martin Balko, Radoslav Fulek, and Jan Kynčl. Crossing numbers and combinatorial characterization of monotone drawings of K_n. *Discrete Comput. Geom.*, 53:107–143, 2015. doi:10.1007/s00454-014-9644-z.

3 Helena Bergold, Stefan Felsner, Manfred Scheucher, Felix Schröder, and Raphael Steiner. Topological drawings meet classical theorems from convex geometry. In *Proc. 28th Internat. Sympos. Graph Drawing*, volume 12590 of *Lecture Notes Comput. Sci.*, pages 281–294. Springer-Verlag, 2020. doi:10.1007/978-3-030-68766-3_22.

4 Emeric Gioan. Complete graph drawings up to triangle mutations. In *Proc. 31st Internat. Workshop Graph-Theoret. Concepts Comput. Sci.*, volume 3787 of *Lecture Notes Comput. Sci.*, pages 139–150. Springer, 2005. doi:10.1007/11604686_13.

5 Jan Kynčl. Enumeration of simple complete topological graphs. *European J. Combinatorics*, 30:1676–1685, 2009. doi:10.1016/j.ejc.2009.03.005.

6 Jan Kynčl. Simple realizability of complete abstract topological graphs in P. *Discrete Comput. Geom.*, 45:383–399, 2011. doi:10.1007/s00454-010-9320-x.

7 Marcus Schaefer. Taking a detour; or, Gioan's theorem, and pseudolinear drawings of complete graphs. *Discrete & Computational Geometry*, 66:12–31, 2021. doi:10.1007/s00454-021-00296-2.

A Note on Rectilinear Crossing number of Hypergraphs

Rahul Gangopadhyay[1] **and Gaiane Panina**[2]

1 **Department of Mathematics and Computer Science,Saint-Petersburg State University, Saint Petersburg, Russia**
`rahulg@iiitd.ac.in`

2 **Steklov Institute of Mathematics at Saint Petersburg, Saint Petersburg, Russia**
`gaiane-panina@rambler.ru`

Abstract

We improve the lower bound on the d-dimensional rectilinear crossing number of the complete d-uniform hypergraph having $2d$ vertices to $\Omega\left(\frac{(4\sqrt{2}/3^{3/4})^d}{d}\right)$ from $\Omega(2^d d)$. This result improves the lower bound on the d-dimensional rectilinear crossing number of the complete d-uniform hypergraph having n vertices to $\Omega\left(\frac{(4\sqrt{2}/3^{3/4})^d}{d}\right)\binom{n}{2d}$ which is approximately $\Omega\left(\frac{2.481^d}{d}\right)$.

Keywords: *Geometric Hypergraph*; *Crossing Number*; *Gale Transform*; *Upper Bound Theorem.*

1 Introduction

In a *rectilinear drawing of a graph*, its vertices are mapped to points in general position (i.e., no three points are colinear) in $\mathbb{R}^2$ and its edges are drawn as straight line segments connecting the corresponding vertices. In a rectilinear drawing of a graph, a pair of edges is said to be *crossing* if they are vertex disjoint and contain a common point in their relative interiors. The *rectilinear crossing number* of a graph G, denoted by $\overline{cr}(G)$, is the minimum number of crossing pairs of edges among all rectilinear drawings of the graph. The study of rectilinear crossing numbers of graphs is an active field of research, see [14].

A *hypergraph* is defined as an ordered pair (V, E) where V is the set of vertices and $E \subseteq 2^V \setminus \{\emptyset\}$ is the set of hyperedges. A hypergraph is said to be *d-uniform* if each hyperedge contains exactly d vertices. Let K_n^d denote the *complete d-uniform hypergraph* having n vertices and $\binom{n}{d}$ hyperedges. Dey and Edelsbrunner [6], and later Dey and Pach [7] extended the idea of a rectilinear drawing of a graph to a rectilinear drawing of a uniform hypergraph. Consider a set P of $n \geq d+1$ points in $\mathbb{R}^d$. The points are said to be in *general position* if no set of $d+1$ points of P lies on a $(d-1)$-dimensional hyperplane. In a *d-dimensional rectilinear drawing* of a d-uniform hypergraph H, the vertices of H are placed in general position in $\mathbb{R}^d$ and the hyperedges are drawn as the convex hulls of d corresponding vertices, i.e., as $(d-1)$-simplices. Let σ be a k-dimensional ($k \leq d-1$) simplex in $\mathbb{R}^d$. We denote the set of vertices of σ by $Vert(\sigma)$. Let τ be an l-dimensional ($l \leq d-1$) simplex in $\mathbb{R}^d$. We say that σ *crosses* τ if they contain a common point in their relative interiors and $Vert(\sigma) \cap Vert(\tau) = \emptyset$, see [7]. The *d-dimensional rectilinear crossing number* of a hypergraph H, denoted by $\overline{cr}_d(H)$, is the minimum number of crossing pairs of hyperedges among all d-dimensional rectilinear drawings of H, see [4]. Let us denote the d-dimensional rectilinear crossing number of K_{2d}^d by c_d, i.e., $c_d = \overline{cr}_d(K_{2d}^d)$. Note that we need at least $2d$ vertices to form a crossing pair of hyperedges since they need to be vertex disjoint, and each set of $2d$ vertices creates distinct crossing pairs of hyperedges. This implies that $\overline{cr}_d(K_n^d) \geq c_d\binom{n}{2d}$.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Dey and Edelsbrunner [6] showed that a 3-uniform hypergraph H having n vertices can have at most $\frac{3n^2}{2}$ hyperedges if $\overline{cr}_3(H) = 0$. This result can be seen as a generalization of Euler's formula for planar graphs. In the same paper, they also proved a generalization of crossing lemma [2]. Later, Dey and Pach [7] extended these results for d-uniform hypergraphs. Though there was a lot of research on determining the crossing number of special graphs, e.g. complete graph, complete bipartite graph etc., see [1, 5, 11], no significant progress was made in case of structured uniform hypergraphs. Through a series of papers [3, 10], lower bound on c_d was improved to $\Omega(d2^d)$ [9] from $\Omega(2^d \log d/\sqrt{d})$ [4]. The best-known upper bound on the d-dimensional rectilinear crossing number of K_{2d}^d is $O(4^d/\sqrt{d})$ [3]. Though the lower and upper bounds on the rectilinear crossing number of complete graph is fairly tight, there is a significant gap between the lower and upper bounds on c_d.
In this paper, we improve this lower bound on c_d:

▶ **Theorem 1.1.** *The d-dimensional rectilinear crossing number of K_{2d}^d is $\Omega\left(\frac{(4\sqrt{2}/3^{3/4})^d}{d}\right)$ which is approximately $\Omega\left(\frac{2.481^d}{d}\right)$.*

▶ **Corollary 1.2.** *The d-dimensional rectilinear crossing number of K_n^d is $\Omega\left(\frac{(4\sqrt{2}/3^{3/4})^d}{d}\right)\binom{n}{2d}$.*

2 Preliminaries

In order to prove Theorem 1.1, we use the following two lemmas and some properties of the Gale transform.

▶ **Lemma 2.1.** *[10, Proof of Theorem 1] Let C' be a set containing $d+4$ points in general position in $\mathbb{R}^d$. There exist at least $\lfloor (d+4)/2 \rfloor$ pairs of disjoint subsets $\{C'_{i1}, C'_{i2}\}$ of C' for each i satisfying $1 \le i \le \lfloor (d+4)/2 \rfloor$ such that the following properties hold.*

1. *$C'_{i1} \cup C'_{i2} = C'$ and $|C'_{i1}|, |C'_{i2}| \ge \lfloor (d+2)/2 \rfloor$*
2. *$(|C'_{i1}|-1)$-simplex $Conv(C'_{i1})$ crosses the $(|C'_{i2}|-1)$-simplex $Conv(C'_{i2})$ (i.e., $C'_{i1} \cap C'_{i2} = \emptyset$ and $Conv(C'_{i1}) \cap Conv(C'_{i2}) \neq \emptyset$).*
3. *There exist $C''_{i1} \subseteq C'_{i1}$ and $C''_{i2} \subseteq C'_{i2}$ such that $|C''_{i1}|, |C''_{i2}| \ge \lfloor (d+2)/2 \rfloor, |C''_{i1}| + |C''_{i2}| = d+2$ and $(|C''_{i1}|-1)$-simplex $Conv(C''_{i1})$ crosses the $(|C''_{i2}|-1)$-simplex $Conv(C''_{i2})$.*

▶ **Lemma 2.2.** *[10] Let a set C contain $2d$ points in general position in $\mathbb{R}^d$. Let $C' \subset C$ be a subset such that $|C'| = d+4$. Let C'_1 and C'_2 be two disjoint subsets of C' such that $|C'_1| = c'_1, |C'_2| = c'_2, C'_1 \cup C'_2 = C'$ and $c'_1, c'_2 \ge \lfloor (d+2)/2 \rfloor$. If the (c'_1-1)-simplex formed by C'_1 crosses the (c'_2-1)-simplex formed by C'_2, then the $(d-1)$-simplex formed by some point set $B'_1 \supset C'_1$ and the $(d-1)$-simplex formed by some point set $B'_2 \supset C'_2$ satisfying $B'_1 \cap B'_2 = \emptyset$, $|B'_1|, |B'_2| = d$ and $B'_1 \cup B'_2 = C$ also form a crossing pair.*

2.1 Gale Transform

The Gale transform [8] turns a sequence of points $P = \langle v_1, v_2, \ldots, v_n \rangle$ into a sequence of vectors $G(P) = \langle g_1, g_2, \ldots, g_n \rangle$. If the affine hull of the set P is $\mathbb{R}^d$, the Gale transform $G(P) = \langle g_1, g_2, \ldots, g_n \rangle$ is a sequence of n vectors in $\mathbb{R}^{n-d-1}$. The Gale transform $G(P)$ has the following properties:

▶ **Lemma 2.3.** *[12] A sequence of vectors $G = \langle g_1, g_2, \ldots, g_n \rangle$ in $\mathbb{R}^{n-d-1}$ is a Gale transform of some $P \subset \mathbb{R}^d$ if and only if G spans $\mathbb{R}^{n-d-1}$ and $\sum_{i=1}^n g_i = \vec{0}$.*

▶ **Definition 2.4** (Totally cyclic vector configuration:)**.** A vector configuration $A = \{a_1, a_2, \dots, a_n\} \subset \mathbb{R}^d$ is said to be *totally cyclic*, if there exists a vector $\beta = (\beta_1, \beta_2, \dots, \beta_n)$ in $\mathbb{R}^n$ such that each $\beta_i > 0$ and $\beta_1 a_1 + \beta_2 a_2 + \dots + \beta_n a_n = \vec{0}$.

▶ **Corollary 2.5.** *Each totally cyclic vector configuration having n vectors which span $\mathbb{R}^{n-d-1}$ becomes the Gale transform of some $P \subset \mathbb{R}^d$ after an appropriate scaling.*

▶ **Lemma 2.6.** *[12] Points in P are in general position in $\mathbb{R}^d$ if and only if every $n - d - 1$ vectors in $G(P)$ span $\mathbb{R}^{n-d-1}$.*

▶ **Lemma 2.7.** *[12] For $t \leq d$, consider a tuple $(i_1, i_2, \dots, i_t)$, where $1 \leq i_1 < i_2 < \dots < i_t \leq n$. A t-element subset $P' = \{v_{i_1}, v_{i_2}, \dots, v_{i_t}\} \subset P$ forms a $(t-1)$-dimensional face of $Conv(P)$ if and only if the relative interior of the convex hull of the points in $G(P) \setminus \{g_{i_1}, g_{i_2}, \dots, g_{i_t}\}$ contains the origin.*

▶ **Theorem 2.8** (Upper bound theorem)**.** *[13] Among all convex polytopes with a given dimension and number of vertices, cyclic polytopes have the largest possible number of faces of each dimension.*

The following corollary can be deducted from Upper bound theorem, see [15].

▶ **Corollary 2.9.** *[15] The convex hull of an n-point set in $\mathbb{R}^d$ has at most $\binom{n - \lceil d/2 \rceil}{n-d} + \binom{n - \lfloor d/2 \rfloor - 1}{n-d}$ facets.*

3 Proof of Theorem 1.1

We start with the following lemma which is interesting for its own sake.

▶ **Lemma 3.1.** *Let σ be a $\lfloor d/2 \rfloor$-simplex, and τ be a $(d-1)$-simplex such that all the $d + \lfloor d/2 \rfloor + 1$ points of $Vert(\sigma) \cup Vert(\tau)$ are in general position in $\mathbb{R}^d$. At most $O((3^{3/4}/\sqrt{2})^d/\sqrt{d})$ $\lceil d/2 \rceil$-faces of τ cross σ.*

Proof. Let us denote by $\tilde{\sigma}$ the affine hull of $Vert(\sigma)$. Note that any $\lceil d/2 \rceil$-face of τ that crosses σ, intersects $\tilde{\sigma}$. Also, note that no vertex of τ lies on $\tilde{\sigma}$ due to the general position of the points in $Vert(\sigma) \cup Vert(\tau)$. Let us consider the orthogonal complement space $\tilde{\sigma}^\perp$ of $\tilde{\sigma}$. Let $\{\tau'_1, \tau'_2, \dots, \tau'_l\}$ be the set of all $\lceil d/2 \rceil$-faces of τ that intersects $\tilde{\sigma}$. Denote by $\mathcal{V}$ the set of vertices $\cup_{j=1}^{l} Vert(\tau'_j)$. Let us assume that at least one $\lceil d/2 \rceil$-face of τ intersects $\tilde{\sigma}$. Then, we can assume that $|\mathcal{V}| = \lceil d/2 \rceil + 1 + k$ for some positive integer k. Let $\mathcal{V}$ be the set $\mathcal{V} = \{v_1, v_2, \dots, v_{\lceil d/2 \rceil + 1 + k}\}$. We project the points in $Vert(\sigma) \cup \mathcal{V}$ onto $\tilde{\sigma}^\perp$. The set $Vert(\sigma)$ maps to a single point in $\tilde{\sigma}^\perp$. Without loss of generality, let us assume that this point is the origin O. Denote by $\tilde{v}_i$ the projection of v_i for each $v_i \in \mathcal{V}$. Let $\tilde{\mathcal{V}}$ denote the set $\{\tilde{v}_2, \tilde{v}_2, \dots, \tilde{v}_{\lceil d/2 \rceil + 1 + k}\}$.

For a tuple $(i_1, i_2, \dots, i_{\lceil d/2 \rceil + 1})$, where $1 \leq i_1 < i_2 < \dots < i_{\lceil d/2 \rceil + 1} \leq \lceil d/2 \rceil + 1 + k$, the convex hull of $\{\tilde{v}_{i_1}, \tilde{v}_{i_2}, \dots, \tilde{v}_{i_{\lceil d/2 \rceil + 1}}\}$ contains the origin O if and only if $\lceil d/2 \rceil$-simplex spanned by $\{v_{i_1}, v_{i_2}, \dots, v_{i_{\lceil d/2 \rceil + 1}}\}$ intersects $\tilde{\sigma}$. Let us denote by $\hat{v}_i$ the vector $O\tilde{v}_i$. Consider the vector configuration $\hat{\mathcal{V}} = \{\hat{v}_1, \hat{v}_2, \dots, \hat{v}_{\lceil d/2 \rceil + 1 + k}\}$. The following observations hold.

▶ Observation 1. No two points of $\mathcal{V}$ map to the same point in $\tilde{\sigma}^\perp$, i.e., all the points in $\tilde{\mathcal{V}}$ are distinct.

For the sake of contradiction, let us assume that $\tilde{v}_{i_1}, \tilde{v}_{i_2}$ maps to a single point q. Then there exist $\lfloor d/2 \rfloor + 3$ points $\{v_{i_1}, v_{i_2}\} \cup Vert(\sigma)$ that lie on a $(\lfloor d/2 \rfloor + 1)$-dimensional hyperplane. This contradicts the general position assumption.

▶ Observation 2. Any subset of $\lceil d/2 \rceil$ vectors of $\hat{\mathcal{V}}$ spans $\mathbb{R}^{\lceil d/2 \rceil}$.

For the sake of contradiction, let us assume that $\{\hat{v}_{i_1}, \hat{v}_{i_2}, \dots, \hat{v}_{i_{\lceil d/2 \rceil}}\}$ spans some $\mathbb{R}^q$ where $q < \lceil d/2 \rceil$. Then, $d+1$ points $\{v_{i_1}, v_{i_2}, \dots, v_{i_{\lceil d/2 \rceil}}\} \cup Vert(\sigma)$ lie in a subspace whose dimension is less than $\lfloor d/2 \rfloor + q \leq d-1$. This contradicts the fact that $Vert(\sigma) \cup Vert(\tau)$ are in general position in $\mathbb{R}^d$.

▶ Observation 3. $\hat{\mathcal{V}}$ is a totally cyclic vector configuration.

For $1 \leq i \leq \lceil d/2 \rceil + k + 1$, each $\tilde{v}_i$ is a vertex of a convex hull of some $\lceil d/2 \rceil + 1$ points of $\tilde{\mathcal{V}}$ which contains the origin O.

These observations along with Lemma 2.3 imply that with proper scaling $\hat{\mathcal{V}}$ is a Gale transformation of some point set $Q = \{q_1, q_2, \dots, q_{\lceil d/2 \rceil + 1 + k}\}$ having $\lceil d/2 \rceil + 1 + k$ points in $\mathbb{R}^k$. Lemma 2.6 and Observation 2 imply that these $\lceil d/2 \rceil + 1 + k$ points of Q are in general position in $\mathbb{R}^k$. Since the points in Q are in general position in $\mathbb{R}^k$, Lemma 2.7 implies that the convex hull of $\{\tilde{v}_{i_1}, \tilde{v}_{i_2}, \dots, \tilde{v}_{i_{\lceil d/2 \rceil + 1}}\}$ contains the origin O if and only if the k points in $Q \setminus \{q_{i_1}, q_{i_2}, \dots, q_{i_{\lceil d/2 \rceil + 1}}\}$ span a $k-1$ dimensional face of the convex hull of Q. By Corollary 2.9, the maximum number of $k-1$ dimensional faces of the convex hull of Q is $\binom{\lceil d/2 \rceil + 1 + k - \lceil k/2 \rceil}{\lceil d/2 \rceil + 1 + k - k} + \binom{\lceil d/2 \rceil + 1 + k - \lfloor k/2 \rfloor - 1}{\lceil d/2 \rceil + 1 + k - k} = \binom{\lceil d/2 \rceil + 1 + \lfloor k/2 \rfloor}{\lceil d/2 \rceil + 1} + \binom{\lceil d/2 \rceil + \lceil k/2 \rceil}{\lceil d/2 \rceil + 1}$. This quantity increases as k increases. The maximum value k can take is $\lfloor d/2 \rfloor - 1$. By setting $k = \lfloor d/2 \rfloor - 1$ and using Stirling's approximation, we obtain the upper bound $O((3^{3/4}/\sqrt{2})^d/\sqrt{d})$ on the number of $\lceil d/2 \rceil$-faces of τ that intersect σ. ◀

Proof of Theorem 1.1: Let $V = \{v_1, v_2, \dots, v_{2d}\}$ be the vertices of K_{2d}^d in its d-dimensional rectilinear drawing. Note that the points in V are in general position in $\mathbb{R}^d$. Let E be the set of $(d-1)$-simplices that correspond to hyperedges of the drawing. Choose a subset of $V' \subset V$ having $d+4$ points. Lemma 2.1 implies that there exist $\lfloor (d+4)/2 \rfloor$ pairs of subsets $\{V'_{i1}, V'_{i2}\}$ for each i satisfying $1 \leq i \leq \lfloor (d+4)/2 \rfloor$ such that all the three conditions mentioned in Lemma 2.1 hold.

It follows from Lemma 2.2 that each such crossing pair of $(|V'_{i1}|-1)$-simplex and $(|V'_{i2}|-1)$-simplex can be extended to at least $\binom{d-4}{d - \lfloor (d+2)/2 \rfloor} = \Omega\left(2^d/\sqrt{d}\right)$ crossing pairs of $(d-1)$-simplices corresponding to the crossing pairs of hyperedges in E. Therefore, the total number of crossing pairs of hyperedges, originated from a particular choice of V', in a d-dimensional rectilinear drawing of K_{2d}^d is at least $\lfloor (d+4)/2 \rfloor \, \Omega\left(2^d/\sqrt{d}\right) = \Omega\left(2^d\sqrt{d}\right)$.

We can choose V' in $\binom{2d}{d+4} = \Theta\left(4^d/\sqrt{d}\right)$ ways. On the one hand, there exist $\Omega\left(2^d\sqrt{d}\right)$ crossing pairs of hyperedges in a d-dimensional rectilinear drawing of K_{2d}^d for each choice of V'. On the other hand, each crossing pair of hyperedges may originate from the different choices of subsets having $d+4$ points from V. Next, we estimate an upper bound on the number of such choices.

Let $e_k, e_l \in E$ be a crossing pair of hyperedges. Let $V(e_k)$, $V(e_l)$ respectively denote the set of vertices of e_k and e_l. Note that $|V(e_k)| = |V(e_l)| = d$, $V(e_k) \cup V(e_l) = V$ and $V(e_k) \cap V(e_l) = \emptyset$.

Lemma 3.1 implies that there can be at most $2\binom{d}{\lceil d+2/2\rceil}O((3^{3/4}/\sqrt{2})^d/\sqrt{d}) = O((3^{3/4}\sqrt{2})^d/d)$ pairs of $\{V''_{i1}, V''_{i2}\}$ such that following conditions hold.

1. $V''_{i1} \subset V(e_k)$, $V''_{i2} \subset V(e_l)$ or $V''_{i1} \subset V(e_l)$, $V''_{i2} \subset V(e_k)$
2. $|V''_{i1}|, |V''_{i2}| \geq \lfloor (d+2)/2 \rfloor, |V''_{i1}| + |V''_{i2}| = d+2$.
3. $Conv(V''_{i1})$ crosses $Conv(V''_{i2})$.

Let us assume that the crossing pair of hyperedges $e_k, e_l \in E$ originates from a $d+4$ sized subset V'_i of V. Note that V'_i contains two subsets V'_{i1} and V'_{i2} such that following conditions hold.

1. $|V'_{i1}|, |V'_{i1}| \geq \lfloor d+2/2 \rfloor$, $V'_{i1} \cup V'_{i2} = V'$ and $V'_{i1} \cap V'_{i2} = \emptyset$.
2. There exist either $V''_{i1} \subseteq V'_{i1} \subset V(e_k)$, $V''_{i2} \subseteq V'_{i2} \subset V(e_l)$ or $V''_{i1} \subseteq V'_{i1} \subset V(e_l)$, $V''_{i2} \subseteq V'_{i2} \subset V(e_k)$ such that $|V''_{i1}|, |V''_{i2}| \geq \lfloor (d+2)/2 \rfloor, |V''_{i1}|+|V''_{i2}| = d+2$ and $Conv(V''_{i1})$ crosses $Conv(V''_{i2})$.

Since each V'_i contains a pair $\{V''_{i1}, V''_{i2}\}$ and each pair $\{V''_{i1}, V''_{i2}\}$ can be extended in $O(d^2)$ ways to a set of size $d+4$, a crossing pair of hyperedges can originate from $O((3^{3/4}\sqrt{2})^d/d) \times O(d^2) = O((3^{3/4}\sqrt{2})^d d)$ distinct subsets $V'_i \subset V$ of size $d+4$. This implies that there exist at least $\dfrac{\Omega\left(2^d\sqrt{d}\right)\Theta\left(4^d/\sqrt{d}\right)}{O((3^{3/4}\sqrt{2})^d d)} = \Omega\left(\dfrac{(4\sqrt{2}/3^{3/4})^d}{d}\right)$ crossing pairs of hyperedges in any d-dimensional rectilinear drawing of K^d_{2d}.

4 Conclusion

In this paper we improved the lower bound on the d-dimensional rectilinear crossing number of K^d_{2d} which in turn improves the d-dimensional rectilinear crossing number of K^d_n. In order to prove this result, we also proved a non-trivial upper bound on the number of crossing pairs of $\lfloor d/2 \rfloor$-simplex and $\lceil d/2 \rceil$-simplex when the $(d-1)$-simplex containing one of them crosses the $(d-1)$-simplex containing the other one. There is still a significant gap between the best-known upper and lower bounds on c_d.

Acknowledgments. Rahul Gangopadhyay was supported by Ministry of Science and Higher Education of the Russian Federation, agreement no. 075–15–2019–1619.

References

1 O. Aichholzer, F. Duque, R. Fabila-Monroy, C. Hidalgo-Toscano and O. E. García-Quintero. An ongoing project to improve the rectilinear and the pseudolinear crossing constants. arXiv preprint arXiv:1907.07796 (2019).

2 M. Ajtai, V. Chvátal, M. M. Newborn and E. Szemerédi. Crossing-free subgraphs. North-Holland Mathematics Studies 60, 9-12 (1982).

3 A. Anshu, R. Gangopadhyay, S. Shannigrahi and S. Vusirikala. On the rectilinear crossing number of complete uniform hypergraphs. Computational Geometry: Theory and Applications 61, 38-47 (2017).

4 A. Anshu and S. Shannigrahi. A lower bound on the crossing number of uniform hypergraphs. Discrete Applied Mathematics 209, 11-15 (2016).

5 B. M. Ábrego, M. Cetina, S. Fernández-Merchant, J. Leaños and G. Salazar. On $(\leq k)$-edges, crossings, and halving lines of geometric drawings of K_n. Discrete and Computational Geometry 48, 192-215 (2012).

6 T. K. Dey and H. Edelsbrunner. Counting triangle crossings and halving planes. Discrete and Computational Geometry 12, 281-289 (1994).

7 T. K. Dey and J. Pach. Extremal problems for geometric hypergraphs. Algorithms and Computation (Proc. ISAAC '96, Osaka; T. Asano et al., eds.), Lecture Notes in Computer Science 1178, Springer-Verlag, 105-114 (1996). Also in: Discrete and Computational Geometry 19, 473-484 (1998).

8 D. Gale. Neighborly and cyclic polytopes. Proceedings of Symposia in Pure Mathematics, 225-232 (1963).

9 R. Gangopadhyay and S. A. Khan. Maximum Rectilinear Crossing Number of Uniform Hypergraphs. EuroCG 2020.

10 R. Gangopadhyay and S. Shannigrahi. k-Sets and Rectilinear Crossings in Complete Uniform Hypergraphs. Computational Geometry: Theory and Applications 86, 101578 (2020).

11 D. J. Kleitman. The crossing number of $K_{5,n}$. Journal of Combinatorial Theory 9, 315-323 (1970).

12 J. Matoušek. Lectures in Discrete Geometry. Springer, 2002.

13 P. McMullen. The maximum numbers of faces of a convex polytope. Mathematika 17, 179-184 (1970).

14 Marcus Schaefer. The graph crossing number and its variants: A survey. The Electronic Journal of Combinatorics, DS21, (2012).

15 G. M. Ziegler. Lectures on Polytopes. Springer, 1995.

Experimental analysis of Delaunay flip algorithms on genus two hyperbolic surfaces*

Vincent Despré[1], Loïc Dubois[2], Benedikt Kolbe[3], and Monique Teillaud[4]

1 **Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France**
vincent.despre@loria.fr
2 **Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France**
loic.dubois@ens-lyon.fr
3 **Hausdorff Center for Mathematics, University of Bonn, Germany** †
benedikt.kolbe@physik.hu-berlin.de
4 **Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France**
monique.teillaud@inria.fr

Abstract

We give experimental evidence that the only known upper bound on the diameter of the flip graph of a hyperbolic surface recently proven by Despré, Schlenker, and Teillaud (SoCG'20), is largely overestimated. To this aim, we develop an experimental framework for the storage of triangulations of hyperbolic surfaces and modifications through twists. We show that the computations with algebraic numbers can be overcome, and we propose ways to generate surfaces that are meaningful for the experiments.

Related Version https://hal.inria.fr/hal-03462834

The source code is available at

`https://members.loria.fr/Monique.Teillaud/Exp-hyperb-flips/`

1 Introduction

It was recently proven that the geometric flip graph of a closed oriented hyperbolic surface is connected [7]. A Delaunay flip algorithm can thus transform any input geometric triangulation T, i.e., a triangulation whose edges are embedded as geodesic segments only intersecting at common endpoints, into a Delaunay triangulation. This is particularly useful in practice as a crucial preprocessing step to computing Delaunay triangulations on a surface: it transforms a "bad" representation of a surface, e.g., by a very elongated fundamental domain, to a "nice" representation by a Delaunay triangulation with only one vertex. Inserting a lot of points would rather be done by Bowyer's incremental algorithm [8, 6], inspired from previous work in the flat case [11].

The authors prove an upper bound on the number of flips: $C_h \cdot \Delta(T)^{6g-4} \cdot n^2$, where C_h is a constant, $\Delta(T)$ is the diameter of T, g is the genus of the surface, and n is the number of vertices [7]. The diameter $\Delta(T)$ is the smallest diameter of a fundamental domain that is the union of lifts of the triangles of T in $\mathbb{H}$. If T is a triangulation of a genus two surface with

* The authors were partially supported by the grant(s) ANR-17-CE40-0033 of the French National Research Agency ANR (project SoS) and INTER/ANR/16/11554412/SoS of the Luxembourg National Research fund FNR (https://members.loria.fr/Monique.Teillaud/collab/SoS/)

† This work was done while Benedikt Kolbe was working at Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

only one vertex then $\Delta(T)$ and the diameter of any other such domain differ by a constant factor at most. In the experiments, we will thus use the domain that naturally appears.

In this paper, we experimentally study the dependence of the number of flips on $\Delta(T)$ (Section 5), for surfaces of genus two. We suspect that the factor $\Delta(T)^{6g-4}$ is largely overestimated. We focus on triangulations having only one vertex, both because the dependence on the number of vertices is clear, and because we are motivated by the abovementioned preprocessing aspect of the algorithm.

Our setup for experiments relies on the representation of genus two surfaces by octagons in $\mathbb{H}$ (Section 2.3). We obtain input triangulations with a large diameter by twisting the abovementioned octagons (Section 4). The data structure we use offers a representation of a triangulation that intrinsically lies on the surface (Section 3).

2 Background

2.1 Hyperbolic surfaces

Consider a closed oriented hyperbolic surface $\mathcal{S}$ (i.e., a connected compact oriented surface without boundary) of genus 2 and the underlying topological surface S_2. Given a hyperbolic structure h on $\mathcal{S}$, associated to a metric of constant curvature -1, the surface $\mathcal{S} = (S_2, h)$ is isometric to the quotient $\mathbb{H}/G$, where $\mathbb{H}$ is the hyperbolic plane and G is a (non-Abelian) discrete subgroup of the isometry group of $\mathbb{H}$ isomorphic to the fundamental group $\pi_1(S_2)$.

The universal cover of $\mathcal{S}$ is isometric to $\mathbb{H}$ equipped with a projection $\rho : \mathbb{H} \to \mathcal{S}$ that is a local isometry. The group G acts on $\mathbb{H}$, so that for any $p \in \mathcal{S}$, $\rho^{-1}(p)$ is an orbit under the action of G. A lift $\widetilde{p}$ of a point $p \in \mathcal{S}$ is one of the elements of the orbit $\rho^{-1}(p)$.

We use the Poincaré disk model, in which $\mathbb{H}$ is represented as the open unit disk $\mathbb{D} \subset \mathbb{C}$.

2.2 Triangulations and flips on hyperbolic surfaces

We call *triangulation* T of a hyperbolic surface $\mathcal{S}$ any geodesic embedding of an undirected graph with a finite number of vertices onto $\mathcal{S}$ such that each resulting face is homeomorphic to an open disk and is bounded by exactly three distinct edge-embeddings. The lift $\widetilde{T}$ of T is the (infinite) triangulation of $\mathbb{H}$ whose vertices and edges are the lifts of the vertices and the edges of T. A Delaunay triangulation T of $\mathcal{S}$ is a triangulation whose lift $\widetilde{T}$ is a Delaunay triangulation in $\mathbb{H}$; for each face t of T and any of its lifts $\widetilde{t}$, the open disk in $\mathbb{D}$ circumscribing $\widetilde{t}$ contains no vertex of $\widetilde{T}$.

Lifting an edge e of T to some $\widetilde{e}$, together with the two triangles incident to $\widetilde{e}$ in the lifted triangulation $\widetilde{T}$, we say that e is *Delaunay-flippable* if the open disks of these triangles contain the fourth vertex of the quadrilateral they form. In this case, the geodesic segment $\widetilde{e}'$ that is the other diagonal of the quadrilateral is contained in it. The *Delaunay flip* of e in T consists in replacing $\widetilde{e}$ by $\widetilde{e}'$ and projecting it back to $\mathcal{S}$ by ρ. A *Delaunay flip algorithm* takes as input a triangulation of $\mathcal{S}$ and flips Delaunay-flippable edges (in any order) until there is none left. Such an algorithm terminates and outputs a Delaunay triangulation [7].

2.3 Admissible loosely-symmetric octagons

We use a slight extension of a set of parameters introduced by Aigon-Dupuy, Buser *et al.* [1], who proved that any closed hyperbolic surface of genus 2 has a fundamental domain that is an octagon in $\mathbb{D}$. This versatile representation allows us to easily construct and manipulate surfaces in our experiments.

We say that a hyperbolic octagon P is *loosely-symmetric* if the opposite sides of P are isometric and the opposite interior angles of P are equal. If moreover the hyperbolic area of P is 4π then P is *admissible*. Clearly, the symmetric octagons introduced by Aigon-Dupuy, Buser *et al.* [1] are loosely-symmetric and the notions of admissibility coincide. Identifying the opposite sides of an admissible loosely-symmetric octagon gives a closed hyperbolic surface of genus 2 [3, Theorem 1.3.5]. Each such surface can be obtained this way [1]. We refer to the full paper [5, Section 3.2] for details and computations.

3 Data structure

Though an *ad hoc* data structure was previously proposed for flipping triangulations [7], we choose to use *combinatorial maps* [10, Section 3.3], which are commonly used to represent graphs embedded on a surface. The data structure we use offers a representation of the triangulation that intrinsically lies on the surface, while the earlier data structure [7, Section 4.1] stuck to specific representatives of all vertices and faces of the lifted triangulation.

For our experiments, we use the flexible implementation of combinatorial maps that is publicly available in CGAL [4]. The *dart* (or *flag*) is the central object in a combinatorial map: it gives access to all incidence relations of an edge of the graph (see Figure 1).

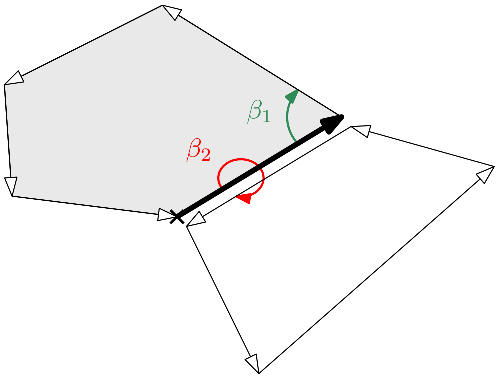


Figure 1 A dart in a combinatorial map (bold).

The geometric information for the triangulation is stored as a cross-ratio for each edge. Recall that the cross-ratio of four pairwise-distinct points in $\mathbb{H}$ represented by $z_1, z_2, z_3, z_4 \in \mathbb{D}$ is the complex $[z_1, z_2, z_3, z_4] = \dfrac{(z_4 - z_2)(z_3 - z_1)}{(z_4 - z_1)(z_3 - z_2)}$ [2]. Let $\mathrm{Im}\,[\cdot]$ denote the imaginary part of a complex. Cross-ratios are suitable for a flip algorithm, due to their well-known property: assuming that z_1, z_2, z_3, z_4 are counterclockwise, $\mathrm{Im}\,[z_1, z_2, z_3, z_4] > 0$ if and only if z_4 lies in the open disk circumscribing (z_1, z_2, z_3).

Given an edge e of a triangulation T of $\mathcal{S}$ we consider a lift $\widetilde{e} = (\widetilde{u_1}, \widetilde{u_3})$ of e in $\mathbb{D}$ and the other vertices $\widetilde{u_2}$ and $\widetilde{u_4}$ of the two faces incident to $\widetilde{e}$ in $\widetilde{T}$, numbering vertices counterclockwise. The cross-ratio $\mathcal{R}_T(e)$ is defined as $[\widetilde{u_1}, \widetilde{u_2}, \widetilde{u_3}, \widetilde{u_4}]$; it is independent of the choice of the lift of e, as the cross-ratio is invariant under orientation preserving isometries of $\mathbb{D}$. An edge e of T is Delaunay-flippable if and only if $\mathrm{Im}\,[\mathcal{R}_T(e)] > 0$.

Note that in our experiments, the lifts in $\mathbb{D}$ are only used to initialize the cross-ratios of a given input triangulation T; they are ignored during the flips, thus preserving the property that the data structure only considers the embedding of the triangulation on the surface. However, in order to be able to recover a lift in $\mathbb{D}$ in the end, e.g., for drawing a representation in $\mathbb{D}$ of the final Delaunay triangulation, we need to maintain an *anchor* during flips. The anchor $A = (\delta, a_1, a_2, a_3)$ consists in some dart δ, chosen arbitrarily, together with a triple (a_1, a_2, a_3) of points in $\mathbb{D}$ that are the vertices of a lift of the face containing δ.

A triangulation T is thus represented by (M, F, A), where M is the combinatorial map, F maps edges of M to their cross-ratios, and A is the anchor. We refer to the full paper [5, Section 3.4] for details on how to update the data structure during a flip.

4 Generating input for the experiments

We generate input for the Delaunay flips algorithms by triangulating admissible loosely-symmetric octagons. An algorithm [5, Appendix C.2] produces such octagons whose vertices are represented in $\mathbb{D}$ by complex numbers with rational real and imaginary parts. We proved a density result [5, Theorem 2] on such rational coordinates allowing us to run experiments using rational numbers only. This is crucial as computing with algebraic numbers is problematic in practice [5, Section 4.1]. For the experiments in Section 5, we need to generate surfaces with large diameter. Our attempts to directly compute such surfaces, taking the diameter as a parameter, were not conclusive. An effective approach consists in starting by generating octagons with a small diameter, then we twist them many times to obtain octagons with a very large diameter. This way we will also study the dependency of the number of flips on those twists.

4.1 Twisting admissible loosely-symmetric octagons

Given $j \geq 3$ and $z_0, \ldots, z_j \in \mathbb{D}$ in geodesically convex position, $G[z_0, \ldots, z_j]$ denotes the hyperbolic polygon whose vertices are $z_1, \ldots, z_j$. Let $G[z_0, \ldots, z_7]$ be an admissible loosely-symmetric octagon. We will consider the Dehn twists [9] along the axes of its side-pairings, as follows (see Figure 2). For every $k \in \{0, \ldots, 7\}$ let τ_k be the orientation preserving isometry of $\mathbb{D}$ satisfying $\tau_k(z_{k+5}) = z_k$ and $\tau_k(z_{k+4}) = z_{k+1}$. Let $t \in \{0, \ldots, 7\}$. For $k \in \{0, \ldots, 7\}$ we set

$$z'_k = \begin{cases} \tau_t(z_k) & \text{if } k - t \in \{1,2,3,4\} \mod 8, \\ z_k & \text{otherwise.} \end{cases}$$

The polygon $G[z'_0, \ldots, z'_7]$ is an admissible loosely-symmetric octagon defining a surface isometric to the one defined by $G[z_0, \ldots, z_7]$ [5, Section 3.2]. We say that $(z'_0, \ldots, z'_7)$ is obtained by *t-twisting* $(z_0, \ldots, z_7)$.

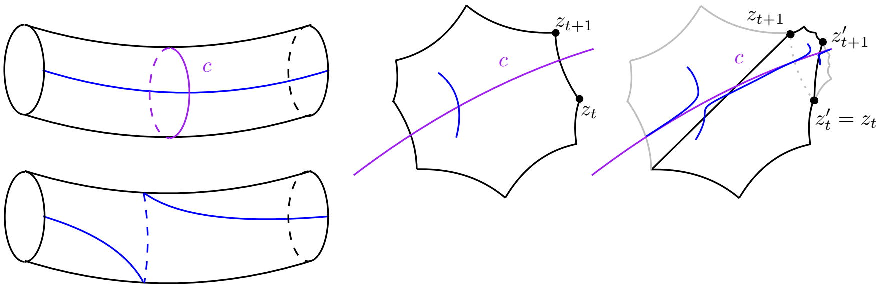


Figure 2 (Left) A Dehn twist along the curve c modifies the blue curve as shown. (Right) A t-twist on an admissible loosely-symmetric octagon.

For a word $t = t_1 \ldots t_m$, we define the t-twist as the composition of the t_k-twists, $k = 1, \ldots, m$, in this order. We pick $t_1, \ldots, t_m$ in $\{0, \ldots, 3\}^m$ instead of $\{0, \ldots, 7\}^m$ to consider the generators without their inverses and quickly obtain large diameters. Indeed for $k \in \mathbb{Z}$ we have $\tau_{k+4} = \tau_k^{-1}$ so a $(t+4)$-twist is the inverse of a t-twist (the indices are modulo 8).

4.2 Generating triangulations

We generate a large number of triangulations having a large diameter following three steps. In the full paper [5, Section 5] a fourth step (**[step 2]**) enables to generate input triangulations with more than one vertex. This step is omitted here as not used in this version.

[step 1] We construct an initial admissible symmetric octagon O.

[step 3] We choose $m \geq 0$ and a sequence $t = t_1 \dots t_m$ of twists.

[step 4] We construct an admissible loosely-symmetric octagon O' by t-twisting O and build the input triangulation T by first cutting O' into 5 triangles and then identifying the edges of the resulting triangulation that correspond to opposite sides of O'.

We refer to the full paper [5, Section 5] for details on these steps. The triangulation T of the hyperbolic surface defined by O has 1 vertex, 9 edges and 5 faces.

We will study two kinds of twists sequences (step 3) in Section 5:

- A *power sequence* is represented by a word u^m for some $u \in \{0, \dots, 3\}$.
- In a *random sequence*, $t_1, \dots, t_m$ are chosen uniformly and independently in $\{0, \dots, 3\}$.

Section 5 will refer to the above three steps. Before doing any experiment Step 1 was applied a thousand times to construct octagons $Q_1, \dots, Q_{1{,}000}$; the experiments consider the first n_q octagons. We also constructed (for step 3) some 10,000 random sequences of twists noted $S_1, \dots, S_{10{,}000}$, each of length 10, of which some of the experiments will use the first n_s sequences. The values of n_q, n_s will be specified in the description of each experiment.

5 Exploring the relationship between number of flips and diameter

As recalled in Section 2.2, a Delaunay flip algorithm can flip Delaunay-flippable edges in any order. In the full paper [5, Section 6] we studied various orders, and observed that the number of flips obtained by the *naive* strategy is close to the minimum: we choose the first Delaunay-flippable edge given by the iterator `DartRange::iterator` of the CGAL combinatorial map. As it runs much faster than all other strategies, we stick to it.

Two sets of experiments will be carried out: experiments I and J use power sequences while experiments K, L, and M use random sequences. We use the notations of Section 4.2.

Experiments I and J are parameterized by the number n_q of octagons: $n_q = 1$ in I and $n_q = 1,000$ in J. We perform step 4 with $O = Q_k$ and $t_1 \dots t_m = u^{3l}$ for $k \in \{1, \dots, n_q\}$, $u \in \{0, 1, 2, 3\}$, $l \in \{0, \dots, 50\}$ and we compute the approximate hyperbolic diameter $\varnothing_{k,l,u}$ of O'. We run the Delaunay flip algorithm, counting the number $\alpha_{k,l,u}$ of flips that were needed by the algorithm to terminate. Figure 3 shows the result.

For experiments K, L, and M the values of (n_q, n_s) are respectively $(1, 10.000)$, $(10, 1.000)$ and $(1.000, 100)$. We first construct the set X containing the 11 prefixes of S_k (including the empty sequence) for every $k \in \{1, \dots, n_s\}$. Then for every $k \in \{1, \dots, n_q\}$ and every $s \in X$, we perform step 4 with $O = Q_k$, and $t_1 \dots t_m = s$. We compute the approximate hyperbolic diameter $\varnothing_{k,s}$ of O'. We run the Delaunay flip algorithm and count the number $\alpha_{k,s}$ of flips that were needed by the algorithm to terminate. Figure 4 shows $\alpha_{k,s}$ as a function of $10 \ln(\varnothing_{k,s})$ for $k \in \{1, \dots, n_q\}, s \in X$.

Our experiments show that controlling the sequence of twists actually allows us to control the number of flips needed by the flip algorithm. Indeed, in the case of power sequences, we observe that the number of flips is linear in the diameter of the input triangulation: Delaunay flips untwist the triangulation by performing a constant number of flips per iteration of the twist. However, for random sequences, we observe that the number of flips is logarithmic in

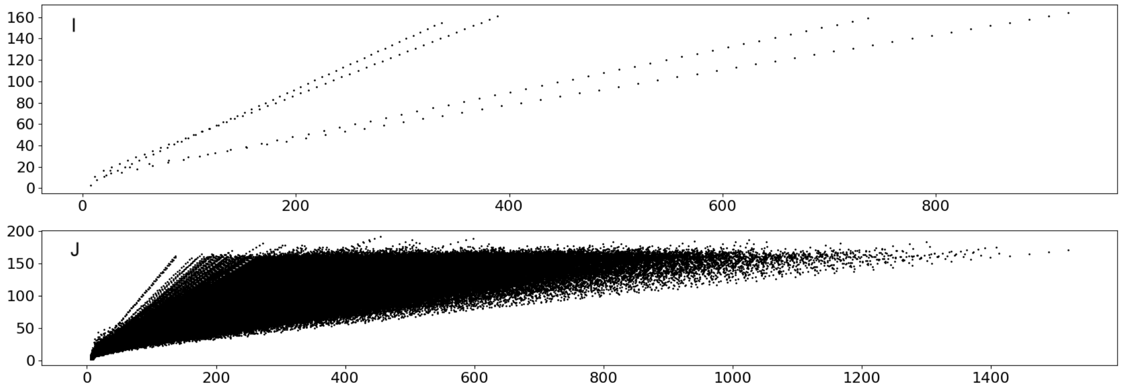


Figure 3 Experiments I and J: number of flips $\alpha_{k,l,u}$ with respect to the (approximate) diameter $\varnothing_{k,l,u}$, $k \in \{1, \ldots, n_q\}, l \in \{0, \ldots, 50\}, u \in \{0, 1, 2, 3\}$

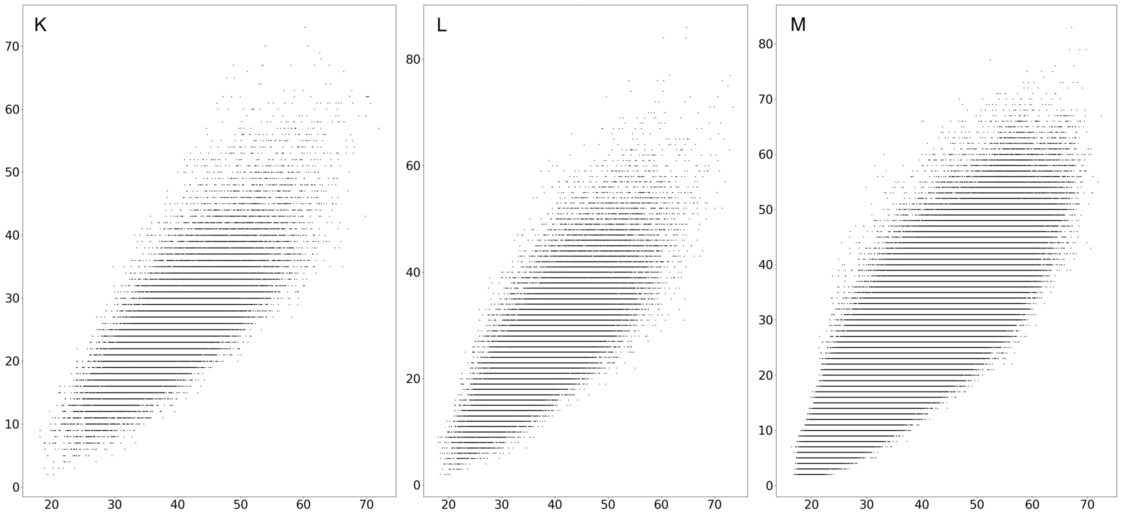


Figure 4 Experiments K, L, and M: number of flips $\alpha_{k,s}$ with respect to $10 \ln(\varnothing_{k,s})$, $k \in \{1, \ldots, n_q\}, s \in X$; the maximum diameter is about 1500

the diameter of the input triangulation. These results can be interpreted using insights on the mapping class group [5, Section 7.4].

In light of our results, we conjecture that the complexity of the Delaunay flip algorithm is worst-case linear in the diameter of the triangulation, and logarithmic on average.

Acknowledgments. The authors want to thank Vincent Delecroix, Matthijs Ebbens, Hugo Parlier, Jean-Marc Schlenker, and Gert Vegter for helpful discussions over many years.

References

1 Aline Aigon-Dupuy, Peter Buser, Michel Cibils, Alfred F. Künzle, and Frank Steiner. Hyperbolic octagons and Teichmüller space in genus 2. *Journal of mathematical physics*, 46(3):033513, 2005. doi:10.1063/1.1850177.

2 M. Berger. *Geometry (vols. 1-2)*. Springer-Verlag, 1987.

3 Peter Buser. *Geometry and Spectra of Compact Riemann Surfaces*. Birkhäuser, Boston, 1992. doi:10.1007/978-0-8176-4992-0.

4 Guillaume Damiand. Combinatorial maps. In *CGAL User and Reference Manual*. CGAL Editorial Board, 5.2.1 edition, 2021. URL: https://doc.cgal.org/5.2.1/Manual/packages.html#PkgCombinatorialMaps.

5 Vincent Despré, Loïc Dubois, Benedikt Kolbe, and Monique Teillaud. Experimental analysis of Delaunay flip algorithms on genus two hyperbolic surfaces. Research report, INRIA, December 2021. URL: https://hal.inria.fr/hal-03462834.

6 Vincent Despré, Benedikt Kolbe, and Monique Teillaud. Representing infinite hyperbolic periodic Delaunay triangulations using finitely many Dirichlet domains. Research report, INRIA, July 2021. URL: https://hal.inria.fr/hal-03045921.

7 Vincent Despré, Jean-Marc Schlenker, and Monique Teillaud. Flipping geometric triangulations on hyperbolic surfaces. In *Proceedings of the 36th International Symposium on Computational Geometry (SoCG'20)*, pages 35:1–35:16, 2020. doi:10.4230/LIPIcs.SoCG.2020.35.

8 Iordan Iordanov and Monique Teillaud. Implementing Delaunay triangulations of the Bolza surface. In *Proceedings of the 33rd International Symposium on Computational Geometry (SoCG'17)*, pages 44:1–44:15, 2017. doi:10.4230/LIPIcs.SoCG.2017.44.

9 Joseph Maher. Random walks on the mapping class group. *Duke Mathematical Journal*, 156(3):429–468, 2011. doi:10.1215/00127094-2010-216.

10 Bojan Mohar and Carsten Thomassen. *Graphs on Surfaces*. Johns Hopkins University Press, Baltimore, 2001.

11 Georg Osang, Mael Rouxel-Labbé, and Monique Teillaud. Generalizing CGAL periodic Delaunay triangulations. In *Proceedings 28th European Symposium on Algorithms*, pages 75:1–75:17, 2020. Best Paper Award (Track B: Engineering and Applications). doi:10.4230/LIPIcs.ESA.2020.75.

Unique Sink Orientations of Grids is in Unique End of Potential Line

Michaela Borzechowski[*1] and Wolfgang Mulzer[†2]

1 Institut für Informatik, Freie Universität Berlin
michaela.borzechowski@fu-berlin.de

2 Institut für Informatik, Freie Universität Berlin
mulzer@inf.fu-berlin.de

Abstract

The complexity classes *Unique End of Potential Line* (UEOPL) and its promise version PromiseUEOPL were introduced in 2018 by Fearnly et al. [4]. PromiseUEOPL captures search problems where the instances are promised to have a unique solution. UEOPL captures total search versions of these promise problems. The promise problems can be made total by defining *violations* that are returned as a short certificate of an unfulfilled promise.

Grid-USO is the problem of finding the sink in a grid with a unique sink orientation. It was introduced by Gärtner et al. [7]. We describe a promise preserving reduction from Grid-USO to Unique Forward EOPL, a UEOPL-complete problem. Thus, we show that Grid-USO is in UEOPL and its promise version is in PromiseUEOPL.

Related Version www.mi.fu-berlin.de/inf/groups/ag-ti/theses/download/Borzechowski21.pdf

1 Introduction

Many tasks in computer science are naturally formulated as *search problems*, where the goal is to *find* a "solution" for a given instance. *Promise problems*, where it is guaranteed that we see only instances that have a certain property, are also an intuitive approach to formulate certain tasks. Nonetheless, standard complexity theory works with decision problems, and it may happen that the computational complexity of the decision problem and the search problem are not equivalent. In particular, this is the case for problems for which it is guaranteed that a solution always exists. The complexity of such *total* search problems has been studied since at least 1991, when Megiddo and Papadimitriou defined the class *Total Function NP* (TFNP) [9]. Here, we consider a subclass of TFNP, namely the complexity class *Unique End of Potential Line* (UEOPL). It contains some interesting problems from computational geometry for which no polynomial time algorithm is known but which are unlikely to be NP-hard, for example α-Ham-Sandwich [3] and Arrival [6]. Currently UEOPL contains one complete problem: One-Permutation-Discrete-Contraction. The problem Grid-USO is an abstraction of the simplex algorithm executed on a *general P-Matrix linear complementarity problem.* Since UEOPL was introduced only recently, in 2018, by Fearnly et al. [4], the knowledge about this class is still very limited. We show that the problem Grid-USO lies in UEOPL, making progress towards elucidating the nature of this class and the problems in it. In particular, with more problems that are known to lie in UEOPL, it becomes more likely that additional complete problems are found.

[*] Supported by DFG within the Research Training Group GRK 2434 *Facets of Complexity.*

[†] Supported in part by ERC StG 757609.

2 Unique End of Potential Line

A UNIQUE FORWARD EOPL instance is defined by two circuits $S : \{0,1\}^d \rightarrow \{0,1\}^d$ and $c : \{0,1\}^d \rightarrow \{0,1\}^m$. A circuit is a compact polynomial sized representation of information which in a map would be exponentially large (for a more formal definition, see [1, Definition 6.1]). The circuits represent a directed graph G. The vertices of G are bit strings $v \in \{0,1\}^d$ with $S(v) \neq v$, and there is a directed edge from a node v to a node w if and only if $S(v) = w$ and $c(w) > c(v)$ (where the result of c is interpreted as an m-bit positive integer). Thus, each node in G has out-degree of at most 1, and the circuit S computes the candidate *successor* of a node. The circuit c assigns a positive *cost* (also called *potential*) from $\{0, \dots, 2^m - 1\}$ to every node, and all edges go in the direction of strictly increasing cost. Furthermore, we define that the bit string 0^d is a node of G, and that $c(0^d) = 0$. These properties ensure that G is a collection of directed paths, which we call *lines*, and that 0^d is the start vertex of a line. Our computational task is as follows: if the nodes of G form a single line (that necessarily starts in 0^d), then we should find the unique end node of this line — the *sink*. Otherwise, we should find a *violation certificate* that shows that G does not consist of a single line. UNIQUE FORWARD EOPL is a total search problem. There always exists a valid sink or a violation. Note that there might exist a valid sink and a violation simultaneously. The promise version of UNIQUE FORWARD EOPL is: under the promise that no violations exist for the given instance, find the unique end of the line. The formal definition of the total search problem UNIQUE FORWARD EOPL is as follows:

▶ **Definition 2.1.** ([5, Definition 10]) Let $d, m \in \mathbb{N}^+$ with $m \geq d$. Given Boolean circuits $S : \{0,1\}^d \rightarrow \{0,1\}^d$ and $c : \{0,1\}^d \rightarrow \{0,1,\dots,2^m-1\}$ which must have the property that $S(0^d) \neq 0^d$ and $c(0^d) = 0$, find one of the following:

(UF1) A bit string $v \in \{0,1\}^d$ with $S(v) \neq v$ and either $S(S(v)) = S(v)$ or $c(S(v)) \leq c(v)$. Then, $S(v)$ is not a valid node and v is a sink node in G and thus a valid solution.

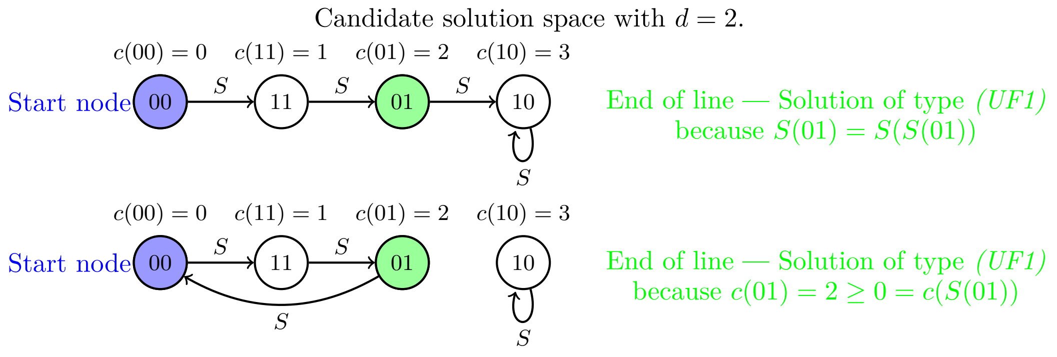
(UFV1) Two bit strings $v, w \in \{0,1\}^d$ with $v \neq w$, $S(v) \neq v$, $S(w) \neq w$ and either *(a)* $c(v) = c(w)$ or *(b)* $c(v) < c(w) < c(S(v))$. Then, v and w are two different nodes that violate the promise of the strictly increasing potential.

(UFV2) Two nodes $v, w \in \{0,1\}^d$ such that v is a solution of type *(UF1)*, $v \neq w$, $S(w) \neq w$ and $c(v) < c(w)$. This encodes a break in the line. The node v is the end of one line, but there exists a different line with a node w that has higher cost than v.

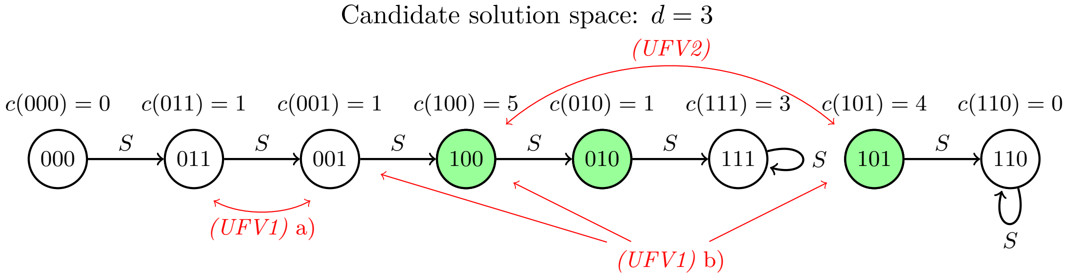
Two examples of an instances with no violations are shown in Figure 1 and an instance with all types of violations can be seen in Figure 2.

▶ **Definition 2.2.** ([5, Definition 7]) Let R be a search problem, and $\mathcal{I}^R \subseteq \{0,1\}^*$ be the set of all instances for R. For an instance $I \in \mathcal{I}^R$, let $\mathcal{S}^R(I)$ be the set of candidate solutions of I. A search problem R can be reduced by a *promise preserving Karp reduction in polynomial time* to a search problem R' if there exist two polynomial-time functions $f : \mathcal{I}^R \rightarrow \mathcal{I}^{R'}$ and $g : \mathcal{I}^R \times \mathcal{S}^{R'}(f(I)) \rightarrow \mathcal{S}^R(I)$ such that if s' is a violation of $f(I)$, then $g(I, s')$ is a violation of I and if s' is a valid solution of $f(I)$, then $g(I, s')$ is a valid solution or a violation of I. We are given an instance of R from which we construct with f an instance of problem R'. If we then solve R', we can re-translate the solution from R' to a solution of R with g. Promise preserving reductions are transitive.

▶ **Definition 2.3.** ([4]) The search problem complexity class UEOPL contains all problems that can be reduced in polynomial time to UNIQUE FORWARD EOPL. Thus, the complexity


Candidate solution space with $d = 2$.
$c(00) = 0$
$c(11) = 1$
$c(01) = 2$
$c(10) = 3$
Start node
00
11
01
10
S
End of line — Solution of type (UF1) because $S(01) = S(S(01))$
End of line — Solution of type (UF1) because $c(01) = 2 \geq 0 = c(S(01))$

■ **Figure 1** Unique Forward EOPL instances that form a valid line without violations.



■ **Figure 2** A Unique Forward EOPL instance line with all types of violations.

class UEOPL captures all total search problems where the space of candidate solutions has the structure of a unique line with increasing cost. The relationship of UEOPL to other classes is shown in Figure 3.

PromiseUEOPL is the promise version of the search problem class UEOPL. When containment of a search problem R in UEOPL is shown via a promise preserving reduction, the promise version of R is contained in PromiseUEOPL.

3 Unique Sink Orientations of Grids

▶ **Definition 3.1.** ([7, p. 206]) Let $n, d \in \mathbb{N}^+$. Let $\mathcal{M} = \{1, \ldots, n\}$ be an ordered set of integers called *directions* and $K = (\kappa_1, \ldots, \kappa_d)$ be a partition of $\mathcal{M}$ with κ_i being ordered and $|\kappa_i| \geq 2$ for all *dimensions* $i = 1, \ldots, d$. The *d-dimensional grid* Γ is the undirected graph derived from $\Gamma = (\mathcal{M}, K)$ with a set of vertices $V := \{v \subseteq \mathcal{M} \mid i = 1, \ldots, d, |v \cap \kappa_i| = 1\}$ and a set of edges $E := \{\{p, q\} \mid p, q \in V, |p \oplus q| = 2\}$, where $\oplus$ denotes the *symmetric difference*.

▶ **Definition 3.2.** ([7, p. 211]) The *outmap function* $\sigma : V \to Powerset(\mathcal{M})$ defines an *orientation* of the edges of a grid Γ. For each point $p \in V$, the set $\sigma(p)$ contains all directions to which p has outgoing edges. The edges of p for all other directions are incoming. In particular, we have $\sigma(p) \cap p = \emptyset$. An outmap σ is called *unique sink orientation* of Γ if all nonempty induced subgrids of Γ have a unique sink.

▶ **Definition 3.3.** ([7, Definition 2.13]) The *refined index* r_σ of an outmap σ with $r_\sigma : V \to \{0, \ldots, |\kappa_1| - 1\} \times \cdots \times \{0, \ldots, |\kappa_d| - 1\}$ is defined as: $r_\sigma(p) := (|\sigma(p) \cap \kappa_1|, \ldots, |\sigma(p) \cap \kappa_d|)$.

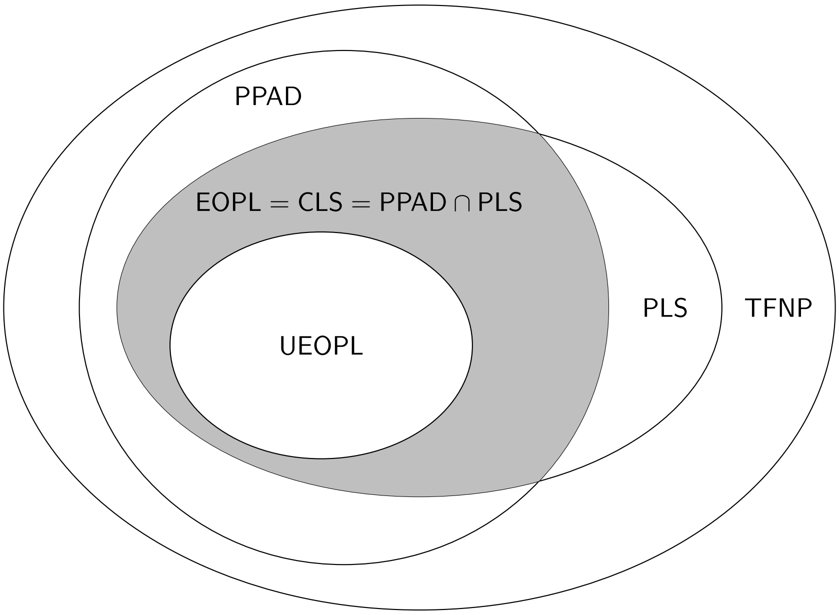


Figure 3 Relation of UEOPL to other search problem complexity classes according to [8].

The refined index assigns to each point a d-tuple containing at index i the number of outgoing edges in dimension i.

▶ **Theorem 3.4.** *([7, Theorem 2.14]) If σ is a unique sink orientation, then r_σ is a bijection.*

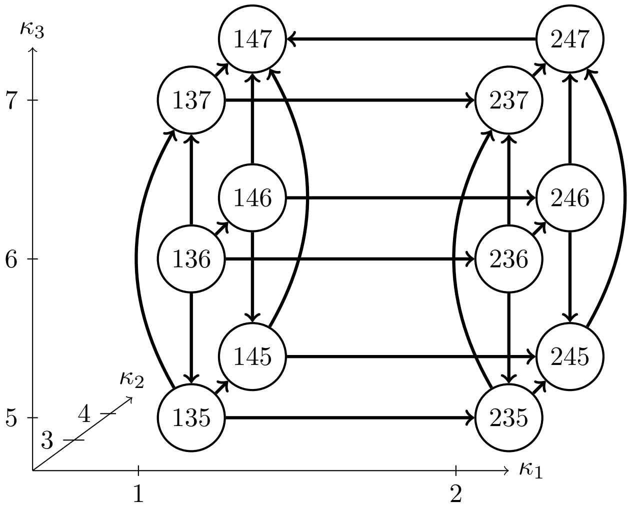


Figure 4 Example grid Γ with $\mathcal{M} = \{1, \dots, 7\}$, $\kappa_1 = \{1, 2\}$, $\kappa_2 = \{3, 4\}$ and $\kappa_3 = \{5, 6, 7\}$. The orientation of the edges is a unique sink orientation. The unique sink is the point (147). The outmap of point (246) is $\sigma(246) = \{5, 7\}$ and its refined index is $r_\sigma(246) = (0, 0, 2)$.

▶ **Definition 3.5.** ([2, Definition 6.1.23]) The search problem GRID-USO is defined as follows: Given a d-dimensional grid, represented implicitly by $\Gamma = (\mathcal{M}, K)$, and a circuit computing an outmap function $\sigma\colon V \to \mathit{Powerset}(\mathcal{M})$, find one of the following:

(GU1) A point $p \in V$ with $\sigma(p) = \emptyset$. The point p is a sink.

(GUV1) A point $p \in V$ with $p \cap \sigma(p) \neq \emptyset$. The point p has a directed edge to itself, thus it is a certificate of σ not being a valid unique sink orientation.

(GUV2) An induced subgrid $\Gamma' = (\mathcal{M}', K')$ with $\sigma'(p) := \sigma(p) \cap \mathcal{M}'$ and two points $p, q \in V'$ with $p \neq q$ and $r_{\sigma'}(p) = r_{\sigma'}(q)$. The points p, q and the subgrid Γ' are a polynomial time verifiable certificate that $r_{\sigma'}$ is not a bijection and thus, σ not a unique sink orientation.

Grid-USO is a total search problem. Under the promise that the outmap σ is a unique sink orientation, the unique sink will be found and returned as solution *(GU1)*. If σ is *not* a unique sink orientation, there exists at least one of the violations *(GUV1)* or *(GUV2)*. An example instance can be seen in Figure 4. Unique sink orientations on grids were introduced by Gärtner et al. [7] as a combinatorial abstraction of linear programming over products of simplices and the generalized linear complementarity problems over P-matrices. There is no polynomial time algorithm known to solve Grid-USO.

4 Grid-USO is in UEOPL

▶ **Theorem 4.1.** *Grid-USO can be reduced via a promise preserving reduction to Unique Forward EOPL.*

Proof. Given an instance $I = (\Gamma, \sigma)$ of Grid-USO, we construct one instance of Unique Forward EOPL $I' = (S, c)$ such that solutions and violations can be mapped accordingly.

Idea of the reduction. We construct a line following algorithm that finds for any given grid its unique sink or a violation, such that each step can be calculated in polynomial time. The vertices of I' are encodings of the states of this line following algorithm. The successor function S calculates the next state. To do so, it is allowed to call the outmap function σ from the Grid-USO instance.

▶ **Lemma 4.2.** *([2, Lemma 6.3.5]) Given an outmap σ on a grid Γ, two disjoint induced subgrids Γ' and Γ'' of Γ whose union is again a valid subgrid, and their respective unique sinks x and y, then* (a) *the unique sink of the grid* $(\Gamma' \cup \Gamma'')$ *is either x or y or* (b) *σ is not a unique sink orientation and we found a violation of Grid-USO.*

The line following algorithm starts at the bottom left point of the grid. It iterates over all directions, in each step looking at the subgrid that is formed by the union of the previous directions (e.g., the subgrid colored in blue in Figure 5) and its sink x. By adding the next lexicographic direction, we add another subgrid (e.g., the yellow subgrid in Figure 5) for which we can find the sink y recursively. The sink of the grid which is the union of the blue and the yellow subgrids is either x or y. None of the other nodes in the subgrids is a sink of the respective subgrid, and thus not a sink of the combined grid. If neither x nor y is a sink, then it can be proven that there is a refined index violation of type *(GUV2)*.

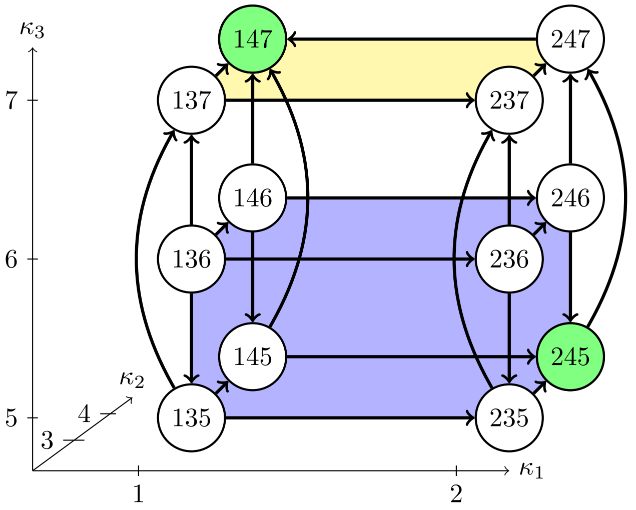


Figure 5 Step of the line-following algorithm: either (245) or (147) is the sink of the whole grid.

Algorithm 1: `find_sink`$(\sigma, \mathcal{M}, \kappa_1, \ldots, \kappa_d)$

```
$x$:= $((\kappa_1)_1, \ldots, (\kappa_d)_1)$;
for $i \in \mathcal{M}$ do
    if $i \notin \sigma(x)$                    // $x$ is also sink in $i$'th subgrid then
        continue with next $i$;
    Let $j$ be the index such that $i \in \kappa_j$ ;
    $\mathcal{M}' := (\mathcal{M} \setminus \kappa_j) \cup \{i\}$ ;
    $y$:= find_sink$(\sigma, \mathcal{M}', (\kappa_1 \cap \mathcal{M}'), \ldots, (\kappa_d \cap \mathcal{M}'))$ // Search recursively for
    sink in yellow subgrid ;
    for $k \in \kappa_j \wedge k \leq i$ do
        if $k \in \sigma(y)$                   // If $y$ is not sink in $i$'th subgrid then
            return Violation;
    $x$:= $y$;
return $x$;
```

Construction of the vertices. Each node of I' is the bit-encoding of an $(n+1)$-tuple of points of the grid. Let $V' := (V \cup \{\perp\})^{n+1}$. Its contents encode the state of Algorithm 1. The points stored in the vertices are the unique sinks of the subgrids in which Algorithm 1 searches recursively. The position corresponds to the directions through which the algorithm iterates. Thus, we can identify for each tuple which step of the algorithm it represents. Let the start node be the $(n+1)$-tuple consisting of the bottom left point of the grid and n many $\perp$'s: $(((\kappa_1)_1, \ldots, (\kappa_d)_1), \perp, \ldots, \perp)$. We define the function `isVertex` which checks in polynomial time, whether any given $(n+1)$-tuple is a valid step of Algorithm 1, the representation of a Grid-USO violation or not a proper encoding at all.

Construction of the successor function S. The successor function S checks for the given node whether it encodes a valid step of Algorithm 1 and if so, calculates the next step in polynomial time. Each step, including the steps of the recursive calls, is a separate node on the resulting unique line. The line corresponds to traversing the tree of the call hierarchy.

Let $S\colon V' \to V'$. Given a vertex $v = (p_1, \dots, p_n, p_{n+1})$, let $S(v)$ be:

1. If `isVertex`(v) says v is *not* a valid encoding, then set $S(v) := v$.
2. If `isVertex`(v) says v is a valid encoding and not a violation:
   - **a.** If the node has the form $v = (\perp, \dots, \perp, p_{n+1})$ it encodes the end of the for-loop in line 2 of Algorithm 1. Thus, the algorithm returns the sink p_{n+1}. Thus, set $S(v) := v$ to indicate the end of the line.
   - **b.** If the node has the form $v = (\perp, \dots, \perp, p_i, p_{i+1}, p_{i+2}, \dots, p_{n+1})$, then p_i is the sink x of the i'th iteration of the for-loop in line 2.
     - **i.** If the check in line 3 is true, we know that x is also the sink of the next bigger subgrid. Thus, set $S(v) := (\perp, \dots, \perp, \perp, p_i, p_{i+2}, \dots, p_{n+1})$.
     - **ii.** If the check in line 3 is false, then we know that x is *not* the sink of the next bigger subgrid. By Lemma 4.2, we must search recursively for the sink of the yellow subgrid. But we want to remember x, so that we can identify a violation if one exists. Thus, set $S(v) := (s, \perp, \dots, \perp, p_i, p_{i+1}, \dots, p_{n+1})$, where s is the start point of the subgrid in the recursive call.

Construction of the cost function c. Let $\omega = n + 2$ and $h\colon V' \times \{1, ..., n\} \times \{1, \dots, d\} \to \{0, \dots, \omega - 1\}$ a help function with

$$h(v,i,j) := \begin{cases} 0 & \text{if } p_i = \perp, \\ \omega - 1 & \text{if } \sigma(p_i) \cap \{1, \dots, i\} = \emptyset, \text{i.e., step } \textit{(2.b.i)} \text{ holds,} \\ (p_i)_j & \text{otherwise.} \end{cases} \tag{1}$$

Because the grid works with ordered sets, lexicographically bigger points have a higher value in h. Also, if two points are sink of their corresponding subgrid, they have the same value in h. The help values of each point and every dimension are scaled and summed up. The cost of a node then is the sum of these scaled values, where each summand is scaled again to enforce that the value for a point at position i which is not $\perp$ is always higher than the sum of all values of points with indices smaller than i. Thus, we always give steps later on in the algorithm higher cost.

$$c(v) := \left(\sum_{i=1}^{n} \left(\omega^{i-1} \cdot \sum_{j=1}^{d} \omega^j h(v,i,j) \right) \right) + \begin{cases} 0 & \text{if } p_{n+1} = \perp, \\ \omega^{nd+1} & \text{if } p_{n+1} \neq \perp. \end{cases} \tag{2}$$

Correctness The successor function and the cost function can be constructed in polynomial time. It can be proven that every solution of type *(GU1)* is only mapped to solutions of type *(UF1)*. Every violation of the created Unique Forward EOPL instance can be mapped back to a violation of the Grid-USO instance. Therefore this reduction is promise preserving. It follows that Grid-USO is in UEOPL and the promise version of Grid-USO is in PromiseUEOPL. The full proof can be found in [2, Proof of Theorem 6.3.1]. ◀

References

1 Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, USA, 1st edition, 2009.

2 Michaela Borzechowski. The complexity class unique end of potential line. Master's thesis, 2021. URL: https://www.mi.fu-berlin.de/inf/groups/ag-ti/theses/download/Borzechowski21.pdf.

3 Man-Kwun Chiu, Aruni Choudhary, and Wolfgang Mulzer. Computational complexity of the α-ham-sandwich problem. *arXiv preprint arXiv:2003.09266*, 2020.

4 John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *CoRR*, abs/1811.03841, 2018. URL: `http://arxiv.org/abs/1811.03841`.

5 John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *Journal of Computer and System Sciences*, 114:1 – 35, 2020. `doi:https://doi.org/10.1016/j.jcss.2020.05.007`.

6 Bernd Gärtner, Thomas Dueholm Hansen, Pavel Hubáček, Karel Král, Hagar Mosaad, and Veronika Slívová. ARRIVAL: next stop in CLS. *CoRR*, abs/1802.07702, 2018. URL: `http://arxiv.org/abs/1802.07702`.

7 Bernd Gärtner, D Walter Jr, Leo Rüst, et al. Unique sink orientations of grids. *Algorithmica*, 51(2):200–235, 2008. `doi:https://doi.org/10.1007/s00453-007-9090-x`.

8 Mika Göös, Alexandros Hollender, Siddhartha Jain, Gilbert Maystre, William Pires, Robert Robere, and Ran Tao. Further collapses in tfnp, 2022. `arXiv:2202.07761`.

9 Nimrod Megiddo and Christos H Papadimitriou. On total functions, existence theorems and computational complexity. *Theoretical Computer Science*, 81(2):317–324, 1991. `doi:https://doi.org/10.1016/0304-3975(91)90200-L`.

On the Number of Optimal Paths in Multicriteria Route Planning

Florian Barth[1], **Stefan Funke**[2], **and Claudius Proissl**[3]

1 **Universität Stuttgart**
`barth@fmi.uni-stuttgart.de`

2 **Universität Stuttgart**
`funke@fmi.uni-stuttgart.de`

3 **Universität Stuttgart**
`proissl@fmi.uni-stuttgart.de`

Abstract

Graphs with multiple edge costs arise naturally in the route planning domain when apart from travel time other criteria like fuel consumption or positive height difference are also objectives to be minimized. In such a scenario, this paper investigates the number of 'optimal' paths between a given source-target pair s, t. We prove a substantial gap between the number of Pareto-optimal and the number of unique shortest paths in a natural model of linear aggregation of the cost metrics. While there are simple graph instances exhibiting an exponential number of Pareto-optimal paths even for only 2 cost metrics, we show that the number of unique shortest st-paths is subexponential (for a fixed number of cost metrics). We can create graphs, however, where the number of unique shortest paths is exponential in the number of metrics. The underlying arguments are highly geometric in that they rely, e.g., on the consideration of hyperplane arrangements in the parameter space.

1 Introduction

In this paper we consider the problem of counting 'optimal' paths in multicriteria networks, i.e., graphs with several scalar values as edge costs. More precisely, we are given a graph $G(V, E)$ and a function $c : E \to \mathbb{R}^d_{\geq 0}$ which assigns each edge d cost values which are to be minimized. We refer to the value d as the dimension of G. A path $\pi(s, t) = e_1 e_2 \ldots e_k$ from node s to node t in V (also called st-path) is a connected sequence of edges of E with $e_1 = (s, \cdot)$ and $e_k = (\cdot, t)$. We define its cost vector naturally as the sum $\sum_{i=1}^{k} c(e_i)$.

For the remainder of this paper, we fix two nodes s and t, and are interested in the number of 'optimal' paths between them. We use the notion of optimality that is common in the context of personalized route planning [5, 4]. Here the idea is that every driver has a weighting $\alpha \in [0, 1]^d$ with $\sum \alpha_i = 1$ which quantifies the importance of each one of the d metrics. For a path π and preference vector α, the aggregated cost of π is then defined as $c(\pi, \alpha) := c(\pi)^T \alpha$. An st-path π is called optimal for preference α if π has smallest aggregated cost from s to t with respect to α. Furthermore, we call a path π a shortest path if it is optimal for at least one preference. Path π from s to t is called *unique shortest path* if there exists an α such that $c(\pi, \alpha)$ has strictly smaller cost than any other st-path with respect to α. [2] showed that in the cost space, the unique shortest paths correspond to the extreme points of (the lower left part of) the convex hull of all Pareto-optimal paths (see Figure 1). An st-path is Pareto-optimal if one cannot find any other st-path that is better in at least one metric and not worse in all other metrics.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

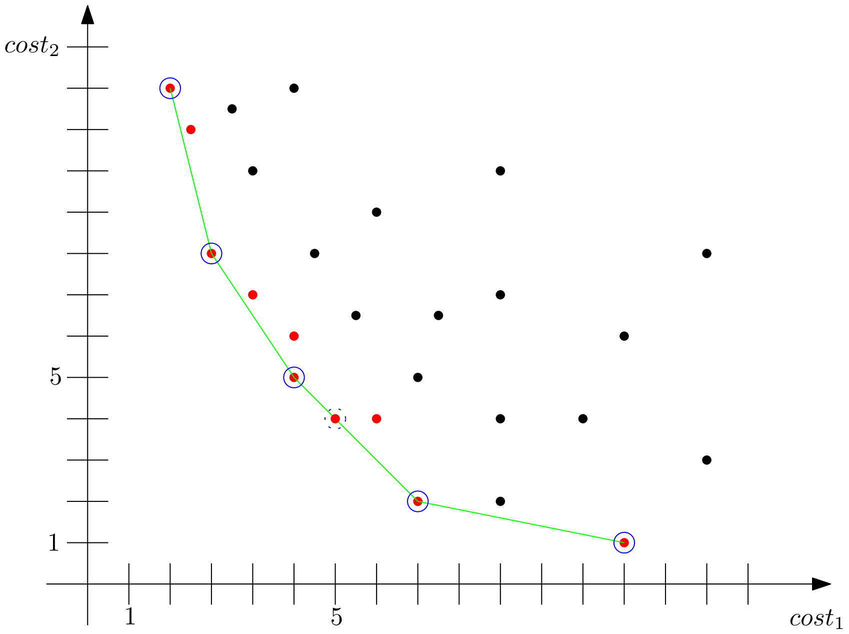


Figure 1 Paths in cost space (black and red dots); Pareto-optimal paths (red); (lower left part of) the boundary of the convex hull of all Pareto-optimal paths in green; unique shortest paths/extreme points of the CH circled in blue.

Related Work

The fact that the number of Pareto-optimal paths can be exponential in the graph size can be considered folklore. Both, Pareto-optimal paths as well as unique shortest paths have been instrumented to create alternative route recommendations. The former approach, pursued e.g. in [7, 3, 6], unfortunately only seems to be viable on rather small graphs due to the too rapidly growing number of Pareto-optimal paths. Restricting to unique shortest paths, though, as in [4], has been shown to be feasible in different practical applications [2, 1].

Our Contribution

In this paper we show that while there are graphs with an exponential number of Pareto-optimal (and shortest) paths (even for dimension $d = 2$), the number of *unique shortest paths* is in $O(n^{2d\sqrt{n}+d+1})$ (n is number of nodes), which is subexponential for fixed d. On the other hand we construct d-metric graphs with $\Omega(n^{d-1})$ unique shortest paths.

2 Preliminaries

In this section we introduce the notions used in Section 3 and show some basic properties.

▶ **Definition 2.1.** The set of all possible preferences

$$\mathcal{P}_d := \{(\alpha_1, \alpha_2, \ldots, \alpha_d) \in \mathbb{R}^d_+ \mid \sum_{i=1}^{d} \alpha_i = 1\}$$

is called d-metric preference space. Note that $\mathcal{P}_d$ is a $(d-1)$-dimensional simplex.

We simply write $\mathcal{P}$ instead of $\mathcal{P}_d$ if the dimension is unimportant.

▶ **Definition 2.2.** Given a graph $G(V, E)$ and a path π in G. The set $A(\pi) := \{\alpha \in \mathcal{P} \mid \pi \text{ is optimal for } \alpha\}$ is called the preference polyhedron of π.

The following lemma justifies the chosen term preference polyhedron.

▶ **Lemma 2.3.** *Given a graph $G(V, E)$. For any path π in G the preference polyhedron $A(\pi)$ is convex and closed.*

Proof. Closure follows from the continuity of the aggregated cost. Regarding convexity, consider a path π that is optimal for $k \in \mathbb{N}$ preferences $\alpha^{(1)}, \alpha^{(2)}, \dots, \alpha^{(k)}$ and any convex combination $\beta := \sum_{i=1}^{k} \gamma_i \alpha^{(i)}$ of the preferences induced by a vector $\gamma \in [0,1]^k$ with $\sum \gamma_i = 1$. We claim that π is also optimal for preference choice β. The cost of π with preference β can be written as

$$c(\pi, \beta) = \gamma_1 c(\pi, \alpha^{(1)}) + \cdots + \gamma_k c(\pi, \alpha^{(k)})$$

Assume there is a path π' with $cost(\pi', \beta) < cost(\pi, \beta)$. But then, since π is optimal for each preference $\alpha^{(i)}$, we know that for every summand we have $\gamma_i cost(\pi, \alpha^{(i)}) \leq \gamma_i cost(\pi', \alpha^{(i)})$, which is a contradiction to our assumption $cost(\pi', \beta) < cost(\pi, \beta)$. Thus, π is also optimal for any such β and $A(\pi)$ is convex. ◀

▶ **Definition 2.4.** Given a graph $G(V, E)$ and two nodes s and t in V. The set

$$\mathcal{A}(s, t) := \{A(\pi(s, t)) \mid \pi \text{ is unique shortest path}\}$$

is called the preference space subdivision from s to t (see Figure 2).

The following lemma says that counting the unique shortest paths between two nodes s and t is equal to counting the cells in the preference space subdivision $\mathcal{A}(s, t)$ (see Figure 2 for an example).

▶ **Lemma 2.5.** *Given a graph $G(V, E)$. A path π in G is a unique shortest path if and only if $A(\pi)$ has non-zero volume.*

Proof. Given a shortest path $\pi(s, t)$. If π is a unique shortest path we find a preference α such that $c(\pi, \alpha)$ is strictly less than the aggregated cost of all other st-paths. Since the aggregated cost is continuous regarding the preference α we can find an $\epsilon > 0$ such that the sphere with radius ϵ and midpoint α is contained in $A(\pi)$.

On the other hand, if $A(\pi)$ has non zero volume, one can find a sphere within $A(\pi)$ that does not touch the boundary of $A(\pi)$. Obviously, for any preference in this sphere π is the unique shortest st-path. ◀

3 Bounds on the Number of Shortest Paths

In this section we investigate upper and lower bounds on the number of shortest paths and unique shortest paths.

3.1 Multi-edge Path Graphs

▶ **Definition 3.1.** A multi-edge path graph is a directed graph $G(V, E)$ with n vertices $V := \{v_1, v_2, \dots, v_n\}$ and the property $e := (v_i, v_j) \in E \Rightarrow j = i + 1$. Multiple edges between the same pair of nodes are allowed.

Figure 3 shows an example path graph. There are 2-metric path graphs with n nodes and $2n - 2$ edges that have $\Theta(2^n)$ shortest paths between v_1 and v_n. The same statement holds for Pareto-optimal paths. See Figure 4 for a sketch of the proof.

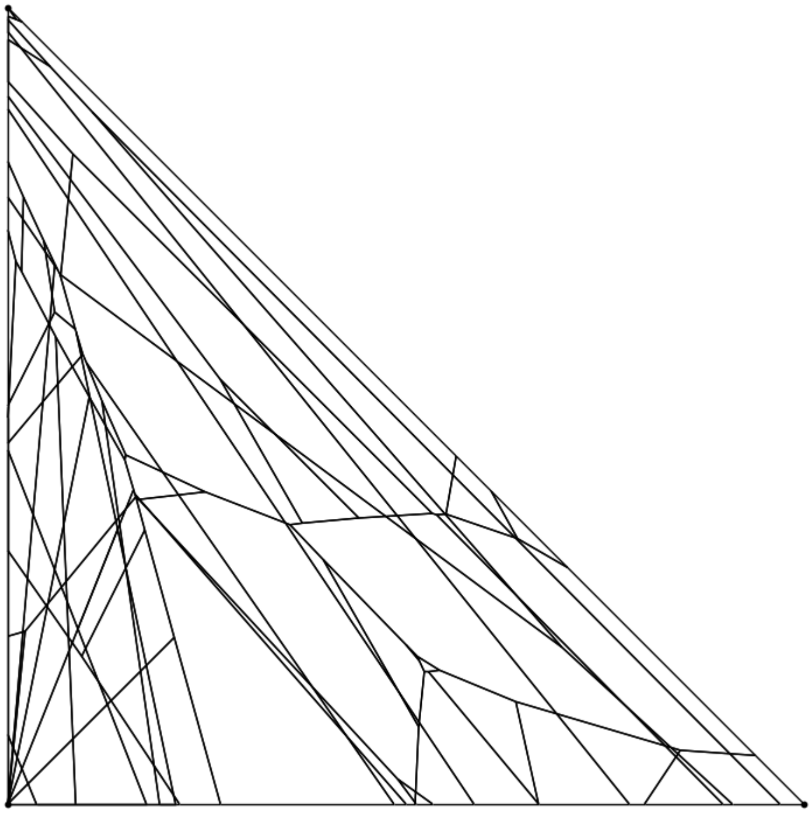


Figure 2 Example preference space subdivision (with $d = 3$) from a real world street network for bicyclists in Germany. Each cell is a preference polyhedron $A(\pi)$ of a unique shortest path π.

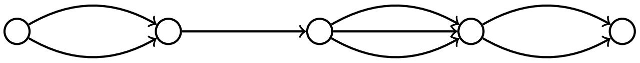


Figure 3 Example path graph

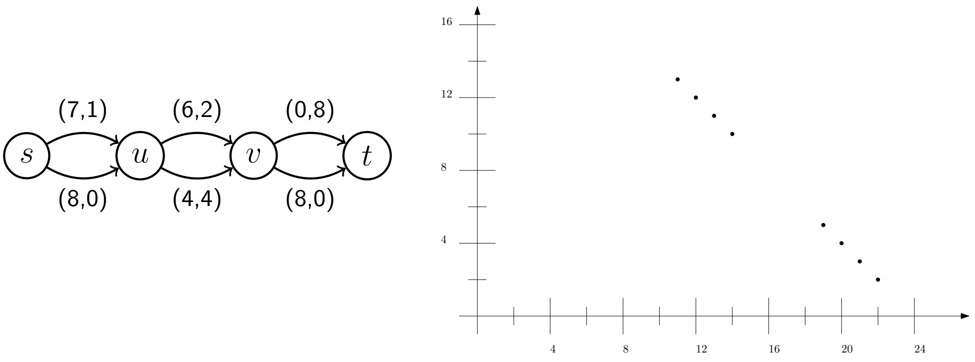


Figure 4 Each path from s to t in the path graph corresponds to a distinct point on the line $y = 24 - x$ in the cost space on the right. They are all optimal for $\alpha^T = (0.5, 0.5)$. Only $(11, 13)$ (e.g., for $\alpha^T = (1, 0)$) and $(22, 2)$ are unique shortest.

3.2 Preference Spaces of multi-edge Path Graphs

In the following we will show that unique shortest paths behave quite differently.

▶ **Lemma 3.2.** *Given a path graph $G(V, E)$ and a path π in G. Then for the preference polyhedron $A(\pi)$ we have*

$$A(\pi) = \bigcap_{e \in \pi} A(e).$$

Lemma 3.2 implies that in a path graph G we obtain the preference space subdivision $\mathcal{A}(v_1, v_n)$ by computing the overlay of $\mathcal{A}(v_1, v_2)$, $\mathcal{A}(v_2, v_3)$ and so forth until $\mathcal{A}(v_{n-1}, v_n)$ as illustrated in Figure 5. This is the main ingredient of the proof of Lemma 3.3.

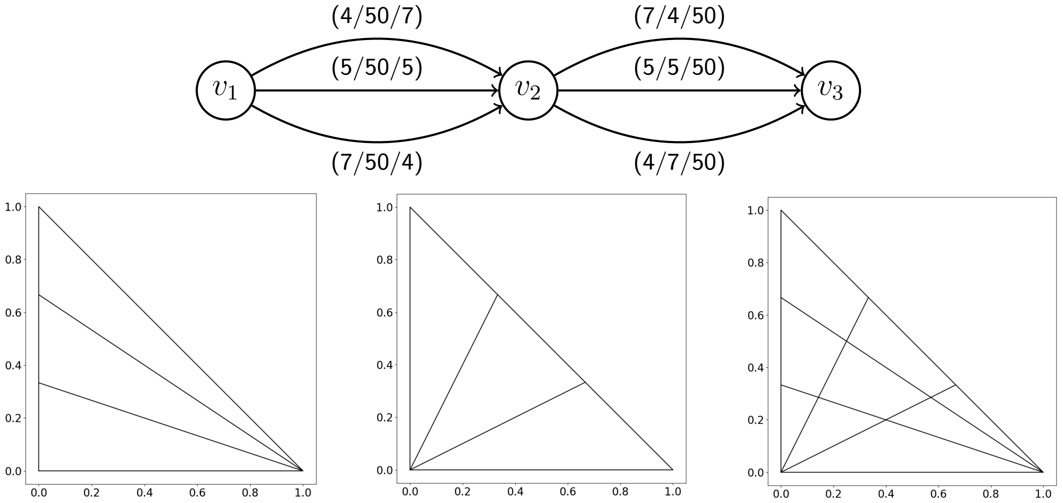


Figure 5 Top: Example graph G with nine unique shortest paths. **Bottom:** $(d-1)$-dimensional Preference space subdivisions $\mathcal{A}(v_1, v_2)$, $\mathcal{A}(v_2, v_3)$ and $\mathcal{A}(v_1, v_3)$ of G (from left to right). The preference space subdivision $\mathcal{A}(v_1, v_3)$ is the intersection of $\mathcal{A}(v_1, v_2)$ and $\mathcal{A}(v_2, v_3)$.

▶ **Lemma 3.3.** *Given a d-metric path graph $G(V, E)$. There are $O\left(\left(n\Delta^2\right)^{d-1}\right)$ unique shortest paths between any two nodes in G, where Δ is the maximum node (out)degree in G.*

Proof. We first consider two consecutive nodes v_i and v_{i+1} and their preference space subdivision $\mathcal{A}(v_i, v_{i+1})$. For each pair of edges e_1, e_2 from v_i to v_{i+1} consider the hyperplane

$$H(e_1, e_2) := \{\alpha \in \mathcal{P}_d \mid (c(e_1) - c(e_2))^T \alpha = 0\}.$$

All boundaries of preference polyhedra in $\mathcal{A}(v_i, v_{i+1})$ are supported by such hyperplanes. Since there are $O(\Delta^2)$ edge pairs from v_i to v_{i+1}, the preference polyhedra in $\mathcal{A}(v_i, v_{i+1})$ are separated by $O(\Delta^2)$ hyperplanes. From Lemma 3.2 it follows that the preference space subdivision $\mathcal{A}(v_1, v_n)$ is the overlay of $\mathcal{A}(v_1, v_2), \mathcal{A}(v_2, v_3), \ldots, \mathcal{A}(v_{n-1}, v_n)$. Therefore, the preference polyhedra in $\mathcal{A}(v_1, v_n)$ are separated by $O(n\Delta^2)$ hyperplanes. Considering $\mathcal{A}(v_1, v_n)$ as an arrangement with $O(n\Delta^2)$ hyperplanes it follows that there are $O(\left(n\Delta^2\right)^{d-1})$ cells or preference polyhedra in $\mathcal{A}(v_1, v_n)$ (the preference space $\mathcal{P}_d$ is $(d-1)$-dimensional). ◀

We show that the upper bound in Lemma 3.3 is tight up to the factor Δ^{2d-2} by constructing an arbitrary arrangement of n hyperplanes within the preference space. The following lemma says that we can in fact construct such arbitrary hyperplanes.

▶ **Lemma 3.4.** *For any halfspace $h \subset \mathbb{R}^{d-1}$ with $h \cap \mathcal{P}_d \neq \emptyset$, there exist a path graph with two nodes, two edges and respective edge cost vectors such that $A(e_1) \subset h$ and $A(e_2) \cap h$ has 0 volume.*

This can be shown by choosing $c(e_1), c(e_2)$ appropriately. All that remains is to show that having n hyperplanes there is an arrangement with $\Omega(n^{d-1})$ cells within the preference space.

▶ **Theorem 3.5.** *There are d-metric path graphs with n nodes, $2n-2$ edges and $\Theta\left(n^{d-1}\right)$ unique shortest paths between two nodes.*

Proof. We know that in general position $(d-1)$-dimensional arrangements with n hyperplanes have $\Theta\left(n^{d-1}\right)$ cells. From Lemma 3.4 we know that we can translate any arrangement within the preference space into a path graph with two outgoing edges per node. We need one node and two edges per hyperplane (plus one end node). All that remains is to show that there are arrangements with $\Theta\left(n^{d-1}\right)$ cells *within* the preference space. Since the preference space $\mathcal{P}_d$ is a $(d-1)$-dimensional simplex we can place a $(d-1)$-dimensional sphere in it with positive volume. Having any arrangement with n hyperplanes and $\Theta\left(n^{d-1}\right)$ vertices we can scale it down such that all vertices of the arrangement fit into this sphere. Clearly, with this new arrangement we still have n hyperplanes and $\Theta\left(n^{d-1}\right)$ cells in the preference space. ◀

3.3 Bounds for General Graphs

Let us now consider general graphs. Since most graphs do not contain multi-edges, we restrict our considerations to this case. First, we introduce the family of layered graphs.

3.3.1 Layered Graphs

▶ **Definition 3.6.** A layered graph $L(V, E, r, c)$ is a graph without multi-edges that is partitioned into r disjoint node sets $V_1, V_2, \ldots, V_r$ with c nodes each such that any edge $(v, u) \in E$ connects nodes of consecutive node sets (also called *layers* of L), i.e., there is an index i with $v \in V_i$ and $u \in V_{i+1}$.

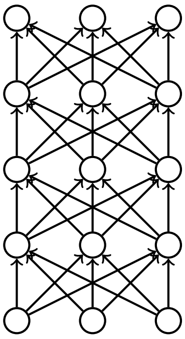


Figure 6 Example layered graph with $c = 3$ and $r = 5$.

▶ **Lemma 3.7.** *Given a d-metric layered graph $L(V, E, r, c)$. There are $O(r^d c^{2d\sqrt{r}})$ unique shortest paths between a node in the first and a node in the last layer of L.*

Proof. Let $T(r, c)$ be the maximum possible number of unique shortest paths between a node v_1 of the first layer and a node v_r of the last layer of any layered graph $L(V, E, r, c)$. Our approach to find an upper bound for T is to decompose L into path graphs and to use Lemma 3.3 for each of them. It is clear that for two numbers $r_1 < r_2$ and a fixed number c it holds $T(r_1, c) \leq T(r_2, c)$. Let $r_0 = 1 < r_1 < \cdots < r_k = r$ be $k+1$ layers of graph L with $r_i - r_{i-1} \leq \lceil \frac{r}{k} \rceil$ for each $1 \leq i \leq k$. Thus, from a node $v_{r_{i-1}}$ in layer r_{i-1} to a node v_{r_i} in layer r_i there are at most $T\left(\lceil \frac{r}{k} \rceil + 1, c\right)$ unique shortest paths. We remove all layers that do not belong to the $k+1$ chosen layers and connect the $k+1$ layers with the unique shortest paths between consecutive layers. In that way, all unique shortest paths between the first and the last layer are preserved. In the new graph there are c^{k-1} node sequences to reach v_r from v_1. Each node sequence can be considered as a path graph with $k+1$ nodes and maximum node degree $\Delta \leq T\left(\lceil \frac{r}{k} \rceil + 1, c\right)$. Hence, with Lemma 3.3 we get

$$T(r, c) \in O\left(c^{k-1}\left(k \cdot T\left(\left\lceil \frac{r}{k} \right\rceil + 1, c\right)^2\right)^{d-1}\right).$$

Setting $k = \lceil \sqrt{r} \rceil$ we get

$$T(r, c) \in O\left(c^{\lceil \sqrt{r} \rceil - 1}\left(\lceil \sqrt{r} \rceil \cdot T\left(\lceil \sqrt{r} \rceil + 1, c\right)^2\right)^{d-1}\right)$$
$$\Rightarrow T(r, c) \in O\left(c^{\sqrt{r}} \cdot r^{\frac{d-1}{2}} \cdot T\left(\lceil \sqrt{r} \rceil + 1, c\right)^{2(d-1)}\right).$$

A trivial upper bound for $T(r, c)$ is c^{r-2} as this is the number of all possible paths. Thus, with $T(\lceil \sqrt{r} \rceil + 1, c) \leq c^{\sqrt{r}}$ it follows

$$T(r, c) \in O\left(c^{\sqrt{r}} \cdot r^{\frac{d-1}{2}} \cdot c^{2(d-1)\sqrt{r}}\right) \Rightarrow T(r, c) \in O\left(r^d c^{2d\sqrt{r}}\right).$$

◀

3.3.2 Extending Upper Bounds to general Graphs

To be able to generalize our results from the previous section, we first show how to represent any graph without multi-edges as layered graph without losing unique shortest paths in the process. Then we use this representation to derive general bounds.

▶ **Definition 3.8.** Given any graph $G(V, E)$ with n nodes, m edges and without multi-edges. The layered graph L of G has n rows and n columns. Each row of L consists of one copy of V and there is one copy of E between each two consecutive rows. Each edge points from one row to the next while the end nodes are the same as in G.

See Figure 7 for an example. The following lemma allows us to focus on layered graphs regarding general upper bounds.

▶ **Lemma 3.9.** *Given a graph $G(V, E)$ without multi-edges and its layered graph L. Then for any two nodes $v_i, v_j \in V$ it holds*

$$|\mathcal{A}(v_i, v_j)| \leq \sum_{r=1}^{n} |\mathcal{A}(v_{i,1}, v_{j,r})|.$$

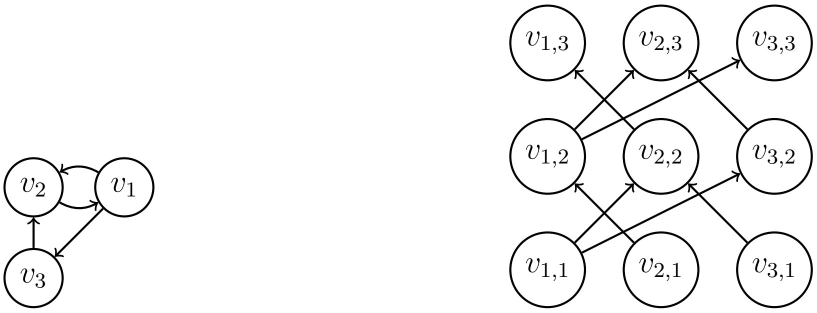


Figure 7 Graph G (left) is translated into a layered graph L (right).

Lemma 3.9 can be proved by showing that any unique shortest path in G can be mapped injectively to an unique shortest path in L.

▶ **Theorem 3.10.** *In any d-metric graph $G(V, E)$ with n nodes there are $O(n^{2d\sqrt{n}+d+1})$ unique shortest paths between any two nodes.*

Proof. Follows from Lemma 3.9 and Lemma 3.7 when choosing $r = n$ and $c = n$. ◀

4 Conclusion

In this paper we gave upper and lower bounds on the maximum number of 'optimal' paths in multicriteria networks. In case of Pareto-optimality, it is well-known that an exponential (in the number of nodes) many optimal paths exist. We show that for a commonly used model of linear aggregation (very popular, e.g., for personalized route planning) the number of optimal paths is subexponential in the graph size, yet possibly exponential in the number of criteria.

References

1 Florian Barth, Stefan Funke, and Claudius Proissl. Preference-based trajectory clustering: An application of geometric hitting sets. In *32nd International Symposium on Algorithms and Computation (ISAAC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

2 Florian Barth, Stefan Funke, and Sabine Storandt. Alternative multicriteria routes. In *2019 Proceedings of the Meeting on Algorithm Engineering and Experiments (ALENEX)*, pages 66–80, 2019.

3 Daniel Delling and Dorothea Wagner. Pareto paths with SHARC. In *Proc. 8th International Symposium on Experimental Algorithms (SEA)*, volume 5526 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2009.

4 Stefan Funke and Sabine Storandt. Personalized route planning in road networks. In Jie Bao, Christian Sengstock, Mohammed Eunus Ali, Yan Huang, Michael Gertz, Matthias Renz, and Jagan Sankaranarayanan, editors, *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems, Bellevue, WA, USA*, pages 45:1–45:10. ACM, 2015.

5 Robert Geisberger, Moritz Kobitzsch, and Peter Sanders. Route planning with flexible objective functions. In *Workshop on Algorithms and Experimentation*, ALENEX '10, page 124–137, USA, 2010. Society for Industrial and Applied Mathematics.

6 Hans-Peter Kriegel, Matthias Renz, and Matthias Schubert. Route skyline queries: A multi-preference path planning approach. In *26th International Conference on Data Engineering (ICDE)*, pages 261–272. IEEE Computer Society, 2010.

7 Matthias Müller-Hannemann and Karsten Weihe. On the cardinality of the pareto set in bicriteria shortest path problems. *Annals of Operations Research*, 147(1):269–286, 2006.

Ray Shooting amid Tetrahedra in Four Dimensions: A Range Search Approach*

Esther Ezra[1], **Micha Sharir**[2], **and Tslil Tsabari**[3]

1 **School of Computer Science, Bar Ilan University, Ramat Gan, Israel**
ezraest@cs.biu.ac.il

2 **School of Computer Science, Tel Aviv University, Tel Aviv Israel**
michas@tauex.tau.ac.il

3 **School of Computer Science, Bar Ilan University, Ramat Gan, Israel**
tslil.tsabari@live.biu.ac.il

Abstract

We present an algorithm that preprocesses a set of n tetrahedra (3-simplices) in $\mathbb{R}^4$ into a data structure for answering ray shooting queries amid the given tetrahedra. Specifically, we show that the classical approach to ray shooting amid triangles in $\mathbb{R}^3$, as described in Pellegrini [6], can be extended in a nontrivial manner to solve the four-dimensional problem with $O^*(s)$ storage and query time $O^*(n/s^{1/6})$, for any storage parameter s between n and n^6 (where the $O^*(\cdot)$ notation hides polylogarithmic factors or factors of the form $O(n^\varepsilon)$, for any $\varepsilon > 0$). This problem arises as a basic ingredient in collision detection in a collection of moving objects in three dimensions. As far as we can tell, the problem has not been previously studied.

1 Introduction

In this paper we consider an extension to four dimensions of the classical *ray shooting* problem, which has mostly been studied in two and three dimensions. In a general setting, we are given a collection S of n simply-shaped objects, and the goal is to preprocess S into a data structure that supports efficient ray shooting queries, where each query specifies a ray ρ and asks for the first object of S hit by ρ, if such an object exists.

The main motivation for ray shooting comes from visibility and rendering problems in computer graphics, modeling, and related topics. The input objects can have various shapes, e.g., triangles in $\mathbb{R}^3$, disks or spheres, and so on. A recent summary of the state of the art in ray shooting is given by Pellegrini [6].

In this work we extend the setup of 2D triangles in $\mathbb{R}^3$ to 3D tetrahedra in four dimensions. Besides being an interesting problem in its own right, it arises, e.g., when we want to perform ray shooting (that is, visibility) queries in a time-varying scene. Concretely, we can think of a ray as the space-time trajectory of some particle that moves from some starting position at constant velocity, and the query asks for the first (time-varying) object that the particle will hit. In this example the ray direction is arbitrary, and the input objects are n (not necessarily pairwise openly disjoint) tetrahedra in $\mathbb{R}^4$. As another application, one may think of a video game, or a simulator, in which objects pop up, move on the screen, and then disappear. In this application, though, the query rays are all 'horizontal', i.e., orthogonal to the fourth, time axis.

[*] Work by Esther Ezra was partially supported by NSF CAREER under grant CCF:AF-1553354 and by Grant 824/17 from the Israel Science Foundation. Work by Micha Sharir was partially supported by ISF Grant 260/18, by grant 1367/2016 from the German-Israeli Science Foundation (GIF), and by Blavatnik Research Fund in Computer Science at Tel Aviv University.

As far as we can tell, this problem has not been explicitly studied so far. We present a solution in which the problem is reduced to a range searching problem in a suitable parametric space, which, in the case of (lines supporting) rays in $\mathbb{R}^4$, is six-dimensional. By carefully adapting and combining off-the-shelf techniques, we are able to solve the problem so that, allowing s storage for the structure, a ray shooting query can be answered in $O^*(n/s^{1/6})$ time, for any $n \le s \le n^6$.

2 A Detailed Description of the Algorithm

Let $\mathcal{T}$ be a set of n input tetrahedra in $\mathbb{R}^4$. Before presenting the algorithm, we note that we use the parametric search technique of Agarwal and Matoušek [3], which reduces ray shooting queries to segment intersection emptiness queries. That is, the query specifies a segment e, and needs to determine whether e intersects any of the input tetrahedra.

To obtain a tradeoff between the storage of the structure and the query time, our algorithm uses a primal-dual approach. However, both the primal and dual setups suffer from the fact that segments and tetrahedra require too many parameters to specify. Specifically, a segment requires eight parameters (e.g., by specifying its two endpoints), while a tetrahedron requires 16 parameters (e.g., by specifying the coordinates of its four vertices). But if we use a multi-level data structure, where each level caters to one aspect of the condition that a segment crosses a tetrahedron, at each of these levels, the number of parameters that a segment or a tetrahedron requires is at most six.

Specifically, the condition that a segment e, that lies on a line ℓ, intersects a tetrahedron Δ, supported by a hyperplane h_Δ, is the conjunction of the following conditions:

(i) The two endpoints of e lie on different sides of h_Δ.

(ii) With a suitable choice of a direction of ℓ and an orientation of Δ, ℓ has a positive orientation with respect to each of the 2-planes that support the four 2-faces of Δ.

Concrete details concerning condition (ii) are given below. Conditions (i) and (ii) are the conjunction of six sub-conditions, where each of the first two tests the position of some endpoint of e with respect to the hyperplanes h_Δ, and each of the other four tests the orientation of ℓ with respect to the planes supporting specific 2-faces of the tetrahedra. Consequently, the dual structure has six levels, two for testing for the sub-conditions comprising condition (i) and four for the sub-conditions comprising condition (ii).

More precisely, each but the last level collects all the tetrahedra Δ in a canonical set produced by the preceding levels that satisfy the corresponding sub-condition for the query segment (that a specific endpoint of e lies in a specific side of h_Δ for the first two levels, and that the oriented 2-plane supporting a specific 2-face of Δ is positively oriented with respect to ℓ for the last four levels), as the disjoint union of precomputed canonical sets of tetrahedra. The last level just tests whether the last sub-condition is satisfied for any tetrahedron in the current canonical set.

We use the fact that lines in $\mathbb{R}^4$ require six real parameters to specify. The space of lines in $\mathbb{R}^4$ is actually projective, but for simplicity of presentation we regard it as a real space, and ignore the special cases in which the real representation fails. Handling these cases follows the same approach, and is in fact simpler.

One simple way to represent a line ℓ in $\mathbb{R}^4$ is by the points $u_\ell^0 = (x_0, y_0, z_0, 0)$ and $u_\ell^1 = (x_1, y_1, z_1, 1)$ at which ℓ crosses the hyperplanes $w = 0$ and $w = 1$, respectively (ignoring lines that are orthogonal to the w-axis), so the line ℓ can be represented as the point $p_\ell = (x_0, y_0, z_0, x_1, y_1, z_1)$ in $\mathbb{R}^6$, as desired.

Similarly, 2-planes in $\mathbb{R}^4$ also require six parameters to specify. This is simply because the duality in $\mathbb{R}^4$ maps lines to 2-planes and vice versa, but a concrete way to represent 2-planes by six parameters is to specify three points on a 2-plane π that are intersections of π with three fixed 2-planes, such as, say, $x = y = 0$, $x = 0$ and $y = 1$, and $x = y = 1$ (again ignoring special directions of π). Each of the intersection points has two degrees of freedom (as two of its coordinates are fixed), for a total of six. Denote these points as $v_\pi^{(00)}$, $v_\pi^{(01)}$, and $v_\pi^{(11)}$, and put $q_\pi = \left(v_\pi^{(00)}, v_\pi^{(01)}, v_\pi^{(11)}\right)$, listing only the w- and z-coordinates of each point, so q_π is a point in $\mathbb{R}^6$. (This is of course only one way to represent a 2-plane in $\mathbb{R}^4$, and there are many other ways where the representation requires only six parameters. For example, one can take the two projections of the 2-plane onto the xyz-space and the yzw-space, say. Nevertheless, the above representation is useful for our analysis so we adopt it.)

These observations are meaningful only for the last four levels of the structure. The first two levels are simpler, as they deal with points (the endpoints of e) and hyperplanes (those supporting the tetrahedra of $\mathcal{T}$) in $\mathbb{R}^4$. Thus each of the first two levels is a halfspace range searching structure for points and halfspaces in $\mathbb{R}^4$. (Actually, this is the case when we pass to the dual 4-space; in the primal we have a point-enclosure problem, where the query is a point and the input consists of halfspaces (or hyperplanes).) Using standard techniques (see, e.g., [1]), this can be done, for N halfspaces in the current canonical subset, and using $O^*(N)$ storage, so that a query costs $O^*(N^{3/4})$ time.[1] As we will shortly see, this cost will be subsumed by the query time bounds for the last four levels. The cost of a query includes the cost of reporting its output, as a list of canonical sets (but not of enumerating the elements of these sets, which is not needed anyway).

We next consider the (more involved) situation in the last four levels of the structure. Here the query segment is replaced by its supporting line ℓ, and each tetrahedron Δ is replaced by the 2-plane supporting a specific 2-face of Δ. In the primal setup, the line ℓ is represented as a point in (projective) 6-space, in the manner just described, and a tetrahedron Δ, represented by a suitable 2-plane π, is represented as a semi-algebraic region K_π, consisting of all points that represent (directed) lines that are positively oriented with respect to π. In the dual setup, the 2-planes π are represented as points in $\mathbb{R}^6$, and the query line ℓ is represented as a semi-algebraic region Q_ℓ that consists of all (oriented) 2-planes that are positively oriented with respect to ℓ.

The orientation test of ℓ with respect to π amounts to computing the sign of the 5×5 determinant

$$\begin{vmatrix} u_\ell^0 & 1 \\ u_\ell^1 & 1 \\ v_\pi^{(00)} & 1 \\ v_\pi^{(01)} & 1 \\ v_\pi^{(11)} & 1 \end{vmatrix}, \tag{1}$$

with a suitable orientation of the pair of points u_ℓ^0, u_ℓ^1 on ℓ (dictating the direction of ℓ), and of the triple of points $v_\pi^{(00)}$, $v_\pi^{(01)}$, $v_\pi^{(11)}$ on π (dictating the orientation of π).

To compute these signs, at each of the four latter levels of the structure, we use a primal-dual approach, where the top part of the structure is in the primal, and at each of its leaf nodes we pass to the dual.

[1] A tradeoff between storage and query time is also available, but we will not need it here.

The dual setup. The dual setup is simpler, so we begin with its description. In the dual setup, each tetrahedron Δ of the current canonical subset of $\mathcal{T}$ is mapped to the point $q_\pi = \left(v_\pi^{(00)}, v_\pi^{(01)}, v_\pi^{(11)}\right)$ in $\mathbb{R}^6$, where π is the 2-plane supporting the 2-face of Δ that corresponds to the present level. The query line ℓ is mapped to a semi-algebraic region Q_ℓ in $\mathbb{R}^6$, consisting of all points that represent (oriented) 2-planes that have positive orientation with respect to ℓ; Q_ℓ is a semi-algebraic region of constant complexity, consisting of those points q_π for which the determinant in (1) is positive (the corresponding polynomial is cubic in q_π). As already mentioned, the task at hand, at each but the last level, is to collect the points q_π that lie in Q_ℓ, as the disjoint union of a small number of precomputed canonical sets of tetrahedra, and the task at the last level is to determine whether Q_ℓ contains any point q_π, for π corresponding to tetrahedra Δ in the present canonical subset of $\mathcal{T}$. In other words, we have, at each of these levels, a problem involving range searching with semi-algebraic sets in $\mathbb{R}^6$. Using the algorithm of Matoušek and Patáková [5], which is a simplified version of the algorithm of Agarwal et al. [4], this can be done, for N tetrahedra with $O^*(N)$ storage, so that a query takes $O^*(N^{5/6})$ time (including the cost of reporting the output canonical sets). See [1, Theorem 6.1] for more details.

The primal problem. With this tool as a black box, we go back to the primal structure, at each of the last four levels. As noted, the problem that we face there is a point enclosure problem, where the input consists of some N constant-complexity semi-algebraic regions in $\mathbb{R}^6$ of the form K_π, as defined earlier, the query is the point p_ℓ that represents ℓ, as defined earlier, and the task is to collect all the regions K_π that contain p_ℓ, as the disjoint union of a small number of precomputed canonical sets, or, at the last level, to determine whether p_ℓ is contained in any such region.

This problem has recently been studied in Agarwal et al. [2], using a multilevel polynomial partitioning technique, but only for the case where we allow maximum storage for the structure (that is, $O^*(N^6)$ in our case) and want the query time to be logarithmic. We next show that the structure can be modified so that its preprocessing stops "prematurely" when its overall storage attains some prescribed value, and each of the subproblems at the new leaves can be handled via the dual algorithm presented above.

The crucial tool in [2], on which their approach is based, is the following result. We give here a restricted specialized version that suffices for our purposes (where the set Ψ below consists of the boundaries of the regions K_π):

[A specialized version of Agarwal et al. [2, Corollary 4.8]] Given a set Ψ of N constant-degree algebraic surfaces in $\mathbb{R}^6$, and a parameter $0 < \delta < 1/6$, there are finite collections $\Omega_0, \ldots, \Omega_6$ of semi-algebraic sets in $\mathbb{R}^6$ with the following properties.

- For each index i, each cell $\omega \in \Omega_i$ is a connected semi-algebraic set of constant complexity.
- For each index i and each $\omega \in \Omega_i$, at most $\frac{N}{4|\Omega_i|^{1/6-\delta}}$ surfaces from Ψ cross ω (intersect ω without fully containing it).
- The cells partition $\mathbb{R}^6$, in the sense that

$$\mathbb{R}^6 = \bigsqcup_{i=0}^{6} \bigsqcup_{\omega \in \Omega_i} \omega,$$

where $\bigsqcup$ denotes disjoint union. The sets in $\Omega_0, \ldots, \Omega_6$ can be computed in $O(N)$ expected time, where the constant of proportionality depends on δ, by a randomized algorithm. For each i and for every set $\omega \in \Omega_i$, the algorithm returns a semi-algebraic representation of ω, a reference point inside ω, and the subset of surfaces of Ψ that cross ω.

Due to lack of space, we defer the rest of the details of the analysis to the full version of this paper. This eventually leads to the following main result:

▶ **Theorem 2.1.** *Given a collection $\mathcal{T}$ of n tetrahedra in $\mathbb{R}^4$, and any storage parameter s between n and n^6, we can preprocess $\mathcal{T}$ into a data structure of size $O^*(s)$, in $O^*(s)$ time, so that we can answer any ray shooting query in $\mathcal{T}$ in $O^*(n/s^{1/6})$ time.*

▶ Remark. It seems that this technique can be extended to any dimension d. In that case the structure has $d+2$ levels. The first two levels ensure that the endpoints of the query segment e lie on different sides of the hyperplane containing the input simplex Δ, and are implemented by halfspace range searching structures in $\mathbb{R}^d$. The last d levels ensure that the line containing e has positive orientation with respect to each of the $(d-2)$-flats containing the facets of Δ, with suitable orientations of the line and the flats. Since lines and $(d-2)$-flats in $\mathbb{R}^d$ have $2d-2$ degrees of freedom, these levels are implemented using semi-algebraic range searching structures in $\mathbb{R}^{2d-2}$. Hence the cost of the query at each of the last d levels dominates the overall cost, which is thus $O^*(n/s^{1/(2d-2)})$.

References

1 P. K. Agarwal. Simplex range searching and its variants: A review. In *Journey through Discrete Mathematics: A Tribute to Jiří Matoušek*, pages 1–30. Springer Verlag, Berlin-Heidelberg, 2017.

2 P. K. Agarwal, B. Aronov, E. Ezra, and J. Zahl. An efficient algorithm for generalized polynomial partitioning and its applications. *SIAM J. Comput.*, 50:760–787, 2021. Also in *Proc. Sympos. on Computational Geometry (SoCG)*, 2019, 5:1–5:14. Also in arXiv:1812.10269.

3 P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM J. Comput.*, 22:794–806, 1993.

4 P. K. Agarwal, J. Matoušek, and M. Sharir. On range searching with semialgebraic sets ii. *SIAM J. Comput.*, 42:2039–2062, 2013. Also in arXiv:1208.3384.

5 J. Matoušek and Z. Patáková. Multilevel polynomial partitions and simplified range searching. *Discrete Comput. Geom.*, 54:22–41, 2015.

6 M. Pellegrini. Ray shooting and lines in space. In *Handbook on Discrete and Computational Geometry*, chapter 41, pages 1093–1112. CRC Press, Boca Raton, Florida, 3rd edition, 2017.

On Stable Range Assignments in $\mathbb{S}^1$

Mark de Berg[1], Arpan Sadhukhan[1], and Frits Spieksma[1]

1 **Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands.** M.T.d.Berg@tue.nl, A.Sadhukhan@tue.nl, f.c.r.spieksma@tue.nl

Abstract

Let P be a set of points in a metric space, where each point $p \in P$ has an associated transmission range, denoted $\rho(p)$. The range assignment ρ induces a directed communication graph $\mathcal{G}_\rho(P)$ on P, which contains an edge (p,q) iff $|pq| \leqslant \rho(p)$. In the broadcast range-assignment problem, the goal is to assign the ranges such that $\mathcal{G}_\rho(P)$ contains an arborescence rooted at a designated root node and the cost $\sum_{p\in P} \rho(p)^\alpha$ of the assignment is minimized, where $\alpha > 1$ is some constant. We study the dynamic version of the problem, where points can be inserted into or deleted from P. In particular, we study trade-offs between the stability of the solution—the number of ranges that are modified when a point is inserted or deleted—and its approximation ratio. In the full version of the paper [11], we study such trade-offs in $\mathbb{R}^1$, in $\mathbb{S}^1$, and in $\mathbb{R}^2$. In this short note we focus on the problem in $\mathbb{S}^1$, where we show that a so-called *stable approximation scheme* does not exist.

Related Version arXiv:2112.05426

1 Introduction

The broadcast range-assignment problem. Let P be a set of points in $\mathbb{R}^d$, representing transmission devices in a wireless network. By assigning each point $p \in P$ a transmission range $\rho(p)$, we obtain a *communication graph* $\mathcal{G}_\rho(P)$. The nodes in $\mathcal{G}_\rho(P)$ are the points from P and there is a directed edge (p,q) iff $|pq| \leqslant \rho(p)$, where $|pq|$ denotes the Euclidean distance between p and q. The energy consumption of a device depends on its transmission range: the larger the range, the more energy it needs. More precisely, the energy needed to obtain a transmission range $\rho(p)$ is given by $\rho(p)^\alpha$, for some real constant $\alpha > 1$ called the *distance-power gradient*. In practice, α depends on the environment and ranges from 1 to 6 [13]. Thus the overall cost of a range assignment is $\text{cost}_\alpha(\rho(P)) := \sum_{p\in P} \rho(p)^\alpha$, where we use $\rho(P)$ to denote the set of ranges given to the points in P by the assignment ρ. The goal of the range-assignment problem is to assign the ranges such that $\mathcal{G}_\rho(P)$ has certain connectivity properties while minimizing the total cost [3]. In the *broadcast range-assignment problem* this property is that $\mathcal{G}_\rho(P)$ contains a *broadcast tree*, i.e., an arborescence rooted at a given source $s \in P$.

The static version of broadcast range-assignment problem has been studied extensively, both in $\mathbb{R}^1$ and in $\mathbb{R}^2$ [1, 2, 4, 5, 6, 7, 8, 9, 12]. Our interest lies in the dynamic version, where points can be inserted into and deleted from P (except the source, which should always remain present). This corresponds to new sensors being deployed and existing sensors being removed. The question we want to answer is: is it possible to maintain a close-to-optimal range assignment that is relatively stable, that is, an assignment for which only few ranges are modified when a point is inserted into or deleted from P? And which trade-offs can be achieved between the quality of the solution and its stability?

To the best of our knowledge, the dynamic problem has not been studied so far. The online problem, where the points from P arrive one by one (there are no deletions) and it is not allowed to decrease ranges, is studied by De Berg *et al.* [10]. When ranges cannot be decreased, a bounded approximation ratio cannot be achieved [11]. By allowing to also

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

decrease a few ranges, it turns out to be possible to maintain solutions whose cost is very close to the static optimum.

Our contribution. Before we state our results, we first define the framework we use to analyze our algorithms. Let P be a dynamic set of points in $\mathbb{R}^d$, which includes a fixed source point s that cannot be deleted.

An update algorithm ALG for the dynamic broadcast range-assignment problem is an algorithm that, given the current solution (the current ranges of the points in the current set P) and the location of the new point to be inserted into or deleted from P, modifies the range assignment so that the updated solution is a valid broadcast range assignment for the updated set P. We call such an update algorithm *k-stable* if it modifies at most k ranges when a point is inserted into or deleted from P. Here we define the range of a point currently not in P to be zero. Thus, if a newly inserted point receives a positive range it will be counted as receiving a modified range; similarly, if a point with positive range is deleted then it will be counted as receiving a modified range.

We are not only interested in the stability of our update algorithms, but also in the quality of the solutions they provide. We measure this in the usual way, by considering the approximation ratio of the solution. Of particular interest are so-called stable approximation schemes, defined as follows. (In the context of dynamic scheduling problems, a related concept has been introduced under the name *robust PTAS* [14, 15].)

▶ **Definition 1.** A *stable approximation scheme*, or *SAS* for short, is an update algorithm ALG that, for any given yet fixed parameter $\varepsilon > 0$, is $k(\varepsilon)$-stable and that maintains a solution with approximation ratio $1+\varepsilon$, where the stability parameter $k(\varepsilon)$ only depends on ε and not on the size of P.

Recall that $\mathrm{cost}_\alpha(\rho(P)) := \sum_{p\in P} \rho(p)^\alpha$, is the cost of a range assignment ρ, where $\alpha > 1$ is a constant. To make our results easier to interpret, we state them here only for $\alpha = 2$.

- We present a SAS for the broadcast range-assignment problem in $\mathbb{R}^1$, with $k(\varepsilon) = O(1/\varepsilon)$, and we prove that this is tight in the worst case.
- Our SAS needs to know an optimal solution after each update. The fastest existing algorithms to compute an optimal solution in $\mathbb{R}^1$ run in $O(n^2)$ time. We show how to recompute an optimal solution in $O(n \log n)$ time after each update.
- In $\mathbb{R}^1$ we also show that a 1-stable algorithm with bounded approximation ratio does not exist when both insertions and deletions must be handled. For the insertion-only case, however, we give a 1-stable $(6+2\sqrt{5})$-approximation algorithm. We also give a very simple 2-stable 2-approximation algorithm, and a 3-stable 1.97-approximation algorithm.
- Next we study the problem in $\mathbb{S}^1$, that is, when the underlying 1-dimensional space is circular. This version has, as far as we know, not been studied so far. We first prove that in $\mathbb{S}^1$ an optimal solution for the static problem can always be obtained by cutting the circle at an appropriate point and solving the resulting problem in $\mathbb{R}^1$. This leads to an algorithm to solve the static problem optimally in $O(n^2 \log n)$ time. We also prove that, in spite of this, a SAS does not exist in $\mathbb{S}^1$.
- Finally, we consider the problem in $\mathbb{R}^2$. Based on the no-SAS proof in $\mathbb{S}^1$, we show that the 2-dimensional problem does not admit a SAS either. In addition, we present a 17-stable 12-approximation algorithm for the 2-dimensional version of the problem.

In this short note we only discuss the problem in $\mathbb{S}^1$, where we show that a SAS does not exist. All other results can be found in the full version of the paper [11].

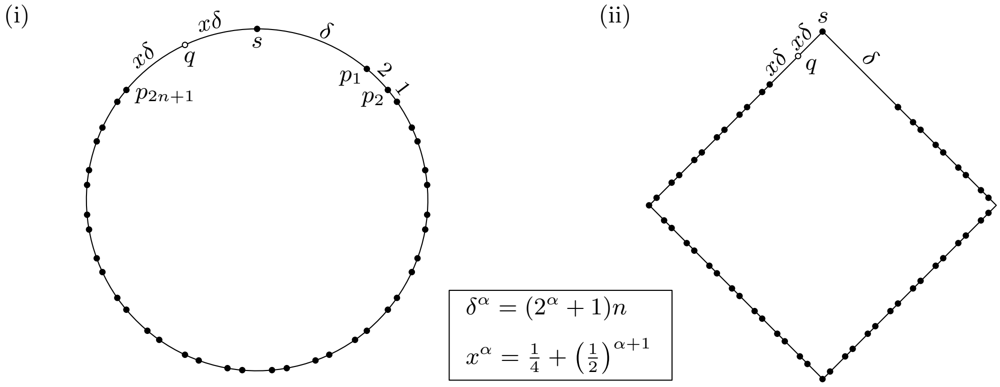


Figure 1 (i) The instance showing that there is no SAS in $\mathbb{S}^1$. (ii) The instance in $\mathbb{R}^2$.

2 Non-existence of a SAS in $\mathbb{S}^1$

Let P be a set of points in $\mathbb{S}^1$, that is, the points lie on a circle and distances are measured along the circle. Let $s \in P$ denote the (fixed) source point for which we want to maintain a broadcast tree. The clockwise distance from a point $p \in \mathbb{S}^1$ to a point $q \in \mathbb{S}^1$ is denoted by $d_{\text{cw}}(p, q)$, and the counterclockwise distance by $d_{\text{ccw}}(p, q)$. The actual distance is then $d(p, q) := \min(d_{\text{cw}}(p, q), d_{\text{ccw}}(p, q))$. In the full version we show that an optimal range assignment for the set P can be obtained from an optimal solution in $\mathbb{R}^1$, if we cut $\mathbb{S}^1$ at an appropriate point. We also present a SAS for the problem in $\mathbb{R}^1$. Here we show that, despite of this, a SAS does not exist for the problem in $\mathbb{S}^1$.

▶ **Theorem 2.** *The dynamic broadcast range-assignment problem in $\mathbb{S}^1$ with distance power gradient $\alpha > 1$ does not admit a SAS. In particular, there is a constant $c_\alpha > 1$ such that the following holds: for any n large enough, there is a set $P := \{s, p_1, \dots, p_{2n+1}\}$ and a point q in $\mathbb{S}^1$ such that any update algorithm* ALG *that maintains a c_α-approximation must modify more than $2n/3 - 1$ ranges upon the insertion of q into P.*

The rest of this section is dedicated to proving Theorem 2. We will prove the theorem for

$$c_\alpha := \min\left(1 + 2^{\alpha-4} - \frac{1}{8},\ 1 + \frac{2^{\alpha-1} - 1}{3 \cdot 2^\alpha + 2},\ 1 + \frac{\min\left(2^\alpha - 1, \frac{3^\alpha - 2^\alpha - 1}{2}, \frac{4^\alpha - 2^\alpha - 2}{3}\right)}{4(2^\alpha + 1)}\right).$$

Note that each term is a constant strictly greater than 1 for any fixed constant $\alpha > 1$. In particular, for $\alpha = 2$ we have $c_\alpha = 1 + \frac{1}{14}$.

Let $P := \{s, p_1, \dots, p_{2n+1}\}$, where $d_{\text{cw}}(p_i, p_{i+1}) = 2$ for odd i and $d_{\text{cw}}(p_i, p_{i+1}) = 1$ for even i; see Fig. 1(i). Let $d_{\text{cw}}(s, p_1) = \delta$, where $\delta^\alpha = (2^\alpha + 1)n$. Finally, let $d_{\text{cw}}(p_{2n+1}, q) = d_{\text{cw}}(q, s) = x\delta$, where $x^\alpha = \frac{1}{4} + \left(\frac{1}{2}\right)^{\alpha+1}$. Note that $(1/2)^\alpha < x^\alpha < 1/2$ for any $\alpha > 1$.

Let $\rho(p)$ denote the range given to a point p by ALG. A directed edge (p, p') in the communication graph induced by ρ is called a *clockwise edge* if $\rho(p) \geqslant d_{\text{cw}}(p, p')$, and it is called a *counterclockwise edge* if $\rho(p) \geqslant d_{\text{ccw}}(p, p')$. Observe that we may assume that no edge (p, p') is both clockwise and counterclockwise, because otherwise $\rho(p) \geqslant (\delta + 3n + 2x\delta)/2$, which is much too expensive for an approximation ratio of at most c_α. Define the range $\rho(p)$ of a point in P to be CW-*minimal* if $\rho(p)$ equals the distance from p to its clockwise neighbor in P. Similarly, $\rho(p)$ is CCW-*minimal* if $\rho(p)$ equals the distance from p to its counterclockwise neighbor. The idea of the proof is to show that before the insertion of q, most of the points $s, p_1, \ldots, p_{2n+1}$ must have a CW-minimal range, while after the insertion most points must have a CCW-minimal range. This will imply that many ranges must be modified from being CW-minimal to being CCW-minimal.

Before the insertion of q, giving every point a CW-minimal range leads to a feasible assignment of total cost $\delta^\alpha + (2^\alpha + 1)n = 2\delta^\alpha$. After the insertion of q, giving every point a CCW-minimal range leads to a feasible assignment of total cost $2(x\delta)^\alpha + (2^\alpha + 1)n = (2x^\alpha + 1)\delta^\alpha$. So if $\text{OPT}(\cdot)$ denotes the cost of an optimal range assignment, then we have:

▶ **Observation 3.** $\text{OPT}(P) \leqslant 2\delta^\alpha$ *and* $\text{OPT}(P \cup \{q\}) \leqslant (2x^\alpha + 1)\delta^\alpha < 2\delta^\alpha$.

We first prove a lower bound on the total cost of the points $p_1, \ldots, p_{2n+1}$. Intuitively, only $o(n)$ of those points can be reached from s or q (otherwise the range of s or q would be too expensive) and the cheapest way to reach the remaining points will be to use only CW-minimal or CCW-minimal ranges. A formal proof of the lemma is given in the full version of the paper [11].

▶ **Lemma 4.** $\sum_{i=1}^{2n+1} \rho(p_i)^\alpha \geqslant (2^\alpha + 1)n - o(n)$, *both before and after the insertion of q.*

The following lemma gives a key property of the construction.

▶ **Lemma 5.** *The point p_{2n+1} cannot have an incoming counterclockwise edge before q is inserted, and the point p_1 cannot have an incoming clockwise edge after q has been inserted.*

Proof. The cheapest incoming counterclockwise edge for p_{2n+1} before the insertion of q is from s, but this is too expensive for ALG to achieve approximation ratio c_α. Similarly, the cheapest incoming clockwise edge for p_1 is from s, but this is too expensive after the insertion of q. The computations for both cases can be found in the full version of the paper [11]. ◀

We are now ready to prove that many edges must change from being CW-minimal to being CCW-minimal when q is inserted.

▶ **Lemma 6.** *Before the insertion of q, at least $4n/3 + 1$ of the points from $\{s, p_1, \ldots, p_{2n}\}$ have a* CW*-minimal range and after the insertion of q at least $4n/3 + 1$ of the points from $\{q, p_1, \ldots, p_{2n}\}$ have a* CCW*-minimal range.*

Proof. We prove the lemma for the situation before q is inserted; the proof for the situation after the insertion of q is similar. Observe that before and after the insertion of q, the distance between any two points is either 1, 2 or at least 3. Hence, in what follows we may assume that $\rho(p) \in \{0, 1, 2\} \cup [3, \infty)$ for any point $p \in P \cup \{q\}$.

It will be convenient to define $p_0 := s$ (although we may still use s if we want to stress that we are talking about the source). Recall that p_{2n+1} does not have an incoming counterclockwise edge in the communication graph $\mathcal{G}_\rho(P)$ before the insertion of q. Let π^* be a minimum-hop path from s to p_{2n+1} in $\mathcal{G}_\rho(P)$. Since p_{2n+1} does not have an incoming counterclockwise edge and π^* is a minimum-hop path, all edges in π are clockwise. We assign each point p_j with $1 \leqslant j \leqslant 2n + 1$ to the edge (p_i, p_t) in π^* such that $i + 1 \leqslant j \leqslant t$, and we define $A(p_i, p_t) := \{p_{i+1}, \ldots, p_t\}$ to be the set of all points assigned to (p_i, p_t). We define the *excess* of a point $p_j \in A(p_i, p_t)$ to be

$$\text{excess}(p_j) := \frac{1}{|A(p_i, p_t)|} \cdot \left(\rho(p_i)^\alpha - \sum_{p_\ell \in A(p_i, p_t)} d(p_{\ell-1}, p_\ell)^\alpha \right).$$

We say that an edge (p_i, p_t) in π^* is CW-minimal if p_i has a CW-minimal range. Note that if a point p_j is assigned to a CW-minimal edge, then this is the edge (p_{j-1}, p_j) and

$\text{excess}(p_j) = 0$. Intuitively, $\text{excess}(p_j)$ denotes the additional cost we pay for reaching p_j compared to reaching it by a CW-minimal edge, if we distribute the additional cost of a non-CW-minimal edge over the points assigned to it. Because each of the points $p_1, \ldots, p_{2n+1}$ is assigned to exactly one edge on the path π^*, we have

$$\sum_{p_i \in \pi^*} \rho(p)^\alpha \geqslant \sum_{j=1}^{2n+1} d(p_{j-1}, p_j)^\alpha + \sum_{j=1}^{2n+1} \text{excess}(p_j) \geqslant \text{OPT}(P) + \sum_{j=1}^{2n+1} \text{excess}(p_j) \tag{1}$$

where the second inequality follows from Observation 3 and because $p_0 = s$. The following claim is proved in the full version of the paper [11]. (Essentially, the smallest possible excess is obtained when $|A(p_i, p_t)| \in \{1, 2, 3\}$; the three terms in the claim correspond to these cases.)

Claim. If p_j is not assigned to a CW-minimal edge then $\text{excess}(p_j) \geqslant c'_\alpha$, where $c'_\alpha = \min\left(2^\alpha - 1, \frac{3^\alpha - 2^\alpha - 1}{2}, \frac{4^\alpha - 2^\alpha - 2}{3}\right)$.

Now suppose for a contradiction that less than $4n/3 + 1$ points from $\{s, p_1, \ldots, p_{2n+1}\}$ have a CW-minimal range. Then at least $2n/3 + 1$ points p_j have $\text{excess}(p_j) \geqslant c'_\alpha$ by the claim above. By Inequality (1) the total cost incurred by ALG is therefore more than

$$\begin{aligned} \text{OPT}(P) + c'_\alpha \cdot (2n/3) &= \text{OPT}(P) + \frac{c'_\alpha}{3(2^\alpha + 1)} \cdot 2(2^\alpha + 1)n && (2) \\ &> \left(1 + \frac{\min\left(2^\alpha - 1, \frac{3^\alpha - 2^\alpha - 1}{2}, \frac{4^\alpha - 2^\alpha - 2}{3}\right)}{4(2^\alpha + 1)}\right) \cdot \text{OPT}(P) && (3) \\ &\geqslant c_\alpha \cdot \text{OPT}(P) && (4) \end{aligned}$$

which contradicts the approximation ratio achieved by ALG. ◀

Lemma 6 implies that at least $4n/3$ of the points $p_1, \ldots, p_{2n+1}$ have a CW-minimal range before q is inserted, and at least $4n/3$ of those points have a CCW-minimal range after the insertion. Hence, at least $2n + 1 - 2 \cdot (2n/3 + 1) = 2n/3 - 1$ points must change from being CW-minimal to being CCW-minimal, thus finishing the proof of Theorem 2.

References

1 Christoph Ambühl. An optimal bound for the MST algorithm to compute energy efficient broadcast trees in wireless networks. In *Proc. 32nd International Colloquium on Automata, Languages and Programming (ICALP 2005)*, volume 3580 of *Lecture Notes in Computer Science*, pages 1139–1150, 2005.

2 Ioannis Caragiannis, Christos Kaklamanis, and Panagiotis Kanellopoulos. New results for energy-efficient broadcasting in wireless networks. In *Proc. 13th International Symposium on Algorithms and Computation (ISAAC 2002)*, volume 2518 of *Lecture Notes in Computer Science*, pages 332–343, 2002.

3 Andrea E. F. Clementi, Gurvan Huiban, Paolo Penna, Gianluca Rossi, and Yann C. Verhoeven. Some recent theoretical advances and open questions on energy consumption in ad-hoc wireless networks. In *Proc. 3rd Workshop on Approximation and Randomization Algorithms in Communication Networks (ARACNE 2002)*, 2002.

4 Andrea E. F. Clementi, Miriam Di Ianni, and Riccardo Silvestri. The minimum broadcast range assignment problem on linear multi-hop wireless networks. *Theor. Comput. Sci.*, 299(1-3):751–761, 2003.

5 Andrea E. F. Clementi, Paolo Penna, Afonso Ferreira, Stephane Perennes, and Riccardo Silvestri. The minimum range assignment problem on linear radio networks. *Algorithmica*, 35(2):95–110, 2003.

6 Andrea E. F. Clementi, Paolo Penna, and Riccardo Silvestri. Hardness results for the power range assignment problem in packet radio networks. In *Proc. 3rd International Workshop on Randomization and Approximation Techniques in Computer Science, and 2nd International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (RANDOM-APPROX'99)*, volume 1671 of *Lecture Notes in Computer Science*, pages 197–208, 1999.

7 Andrea E. F. Clementi, Paolo Penna, and Riccardo Silvestri. The power range assignment problem in radio networks on the plane. In *Proc. 17th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 1770 of *Lecture Notes in Computer Science*, pages 651–660, 2000.

8 Gautam K. Das, Sandip Das, and Subhas C. Nandy. Range assignment for energy efficient broadcasting in linear radio networks. *Theor. Comput. Sci.*, 352(1-3):332–341, 2006.

9 Gautam K. Das and Subhas C. Nandy. Weighted broadcast in linear radio networks. *Inf. Process. Lett.*, 106(4):136–143, 2008.

10 Mark de Berg, Aleksandar Markovic, and Seeun William Umboh. The online broadcast range-assignment problem. In *Proc. 31st International Symposium on Algorithms and Computation (ISAAC)*, volume 181 of *LIPIcs*, pages 60:1–60:15, 2020.

11 Mark de Berg, Arpan Sadhukhan, and Frits C. R. Spieksma. Stable approximation algorithms for the dynamic broadcast range-assignment problem. *CoRR*, abs/2112.05426, 2021.

12 Lefteris M. Kirousis, Evangelos Kranakis, Danny Krizanc, and Andrzej Pelc. Power consumption in packet radio networks. *Theor. Comput. Sci.*, 243(1-2):289–305, 2000.

13 Kaveh Pahlavan and Allen H. Levesque. *Wireless information networks, Second Edition.* Wiley series in telecommunications and signal processing. Wiley-VCH, 2005.

14 Peter Sanders, Naveen Sivadasan, and Martin Skutella. Online scheduling with bounded migration. *Math. Oper. Res.*, 34(2):481–498, 2009.

15 Martin Skutella and José Verschae. A robust PTAS for machine covering and packing. In *Proc. 18th Annual European Symposium (ESA 201)*, volume 6346 of *Lecture Notes in Computer Science*, pages 36–47, 2010.

Removing Popular Faces in Curve Arrangements by Inserting one more Curve[*]

Phoebe de Nooijer[1], Soeren Nickel[2], Alexandra Weinberger[3], Zuzana Masárová[4], Tamara Mchedlidze[1], Maarten Löffler[1], and Günter Rote[5]

1 **Utrecht University, the Netherlands**
p.denooijer@students.uu.nl | t.mtsentlintze@uu.nl | m.loffler@uu.nl
2 **TU Wien, Austria** soeren.nickel@ac.tuwien.ac.at
3 **TU Graz, Austria** weinberger@ist.tugraz.at
4 **IST Austria, Austria** zuzana.masarova@ist.ac.at
5 **Freie Universität Berlin,** rote@inf.fu-berlin.de

Abstract

A face in a curve arrangement is called *popular* if it is bounded by the same curve multiple times. Motivated by the automatic generation of curved nonogram puzzles, we investigate possibilities to eliminate popular faces in an arrangement by inserting a single additional curve. This turns out to be already NP-hard; however, we present a probabilistic FPT-approach in the number of such faces.

Related Version arXiv:2202.12175

1 Introduction

Let $\mathcal{A}$ be a set of curves which lie inside the area bounded by a closed curve F, called the *frame*. All curves in $\mathcal{A}$ are either *closed* or they are *open* with a start and end point on F. We refer to $\mathcal{A}$ as a *curve arrangement*, see Figure 1a. We consider only *simple* arrangements, where no three curves meet in a point, and all intersections are crossings (no tangencies).

The arrangement $\mathcal{A}$ can be seen as an embedded multigraph whose vertices are crossings between curves and whose edges are *curve segments*. $\mathcal{A}$ subdivides the region bounded by F into *faces*. We call a face *popular* when it is incident to multiple curve segments belonging to the same curve in $\mathcal{A}$ (see Figures 1b-c). We study the following problem: is it possible to insert an additional curve ℓ into $\mathcal{A}$, such that no faces of $\mathcal{A} \cup \{\ell\}$ are popular (see Figure 1d)?

Nonograms. Our question is motivated by the problem of generating *curved nonograms*. Nonograms, also known as *Japanese puzzles*, *paint-by-numbers*, or *griddlers*, are a popular

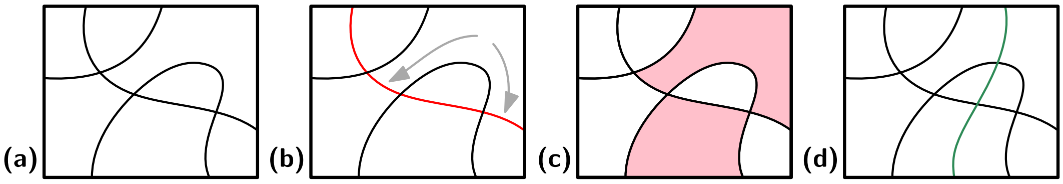


Figure 1 **(a)** An arrangement of curves inside a frame. **(b)** The red curve is incident to the top right face in two disconnected segments, making the face *popular*. **(c)** All popular faces are highlighted. **(d)** After inserting an additional curve, no more popular faces remain.

[*] A.W. is supported by the Austrian Science Fund (FWF): W1230. The authors are ordered by seniority.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

puzzle type where one is given an empty grid and a set of *clues* on which grid cells need to be colored. A clue consists of a sequence of numbers specifying the numbers of consecutive filled cells in a row or column. A solved nonogram typically results in a picture (see Figure 2 (a)). There is quite some work in the literature on the difficulty of solving nonograms [1, 3, 5].

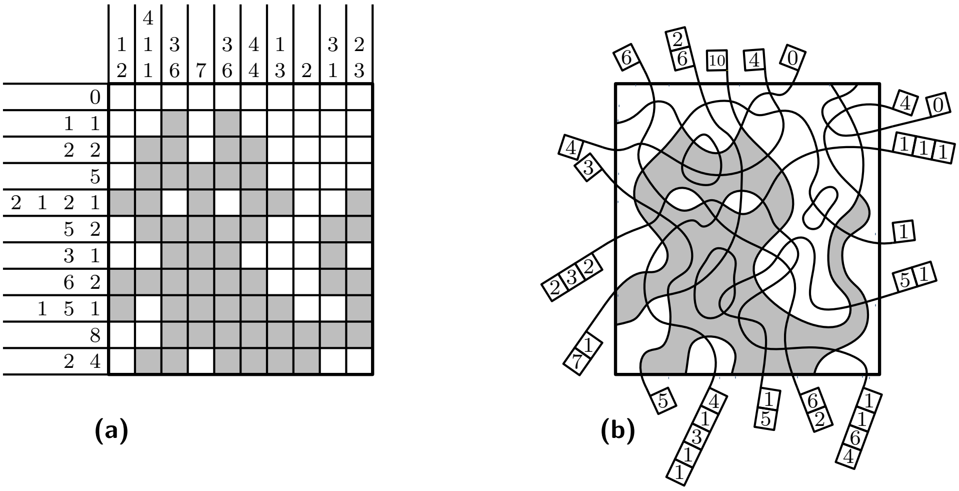


Figure 2 Two nonogram puzzles in solved state. **(a)** A classic nonogram. **(b)** A curved nonogram.

Van de Kerkhof et al. introduced *curved* nonograms, a variant in which the puzzle is no longer played on a grid but on any arrangement of curves [7] (see Figure 2b). In curved nonograms, a clue specifies the numbers of filled faces of the arrangement in the sequence of faces that are incident to a common curve on one side. Van de Kerkhof et al. focus on heuristics to automatically generate such puzzles from a desired solution picture by extending curve segments to a complete curve arrangement.

Nonogram complexity. Van de Kerkhof et al. observe that curved nonograms come in different flavors of increasing complexity — not in terms of how hard it is to *solve* a puzzle, but how hard it is to understand the rules (see Figure 3). They state that it would be of interest to generate puzzles of a specific complexity level; their generators are currently not able to do so other than by trial and error.

- *Basic* nonograms are puzzles in which each clue corresponds to a sequence of unique faces. The analogy with clues in classical nonograms is straightforward.
- *Advanced* nonograms may have clues that correspond to a sequence of faces in which some faces may appear multiple times because the face is incident to the same curve (on the same side) multiple times. When such a face is filled, it is also counted multiple times; in particular, it is no longer true that the sum of the numbers in a clue is equal to the total number of filled faces incident to the curve. This makes the rules harder to understand, and thus advanced nonograms are only suitable for more experienced puzzle freaks.
- *Expert* nonograms may have clues in which a single face is incident to the same curve on *both* sides. They are even more confusing than advanced nonograms.

It is not hard to see that expert puzzles correspond exactly to arrangements with self-intersecting curves. The difference between basic and advanced puzzles is more subtle; it corresponds exactly to the presence of *popular faces* in the arrangement.

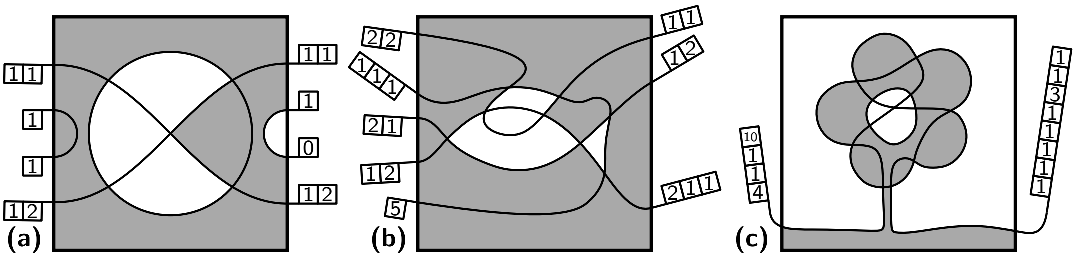


Figure 3 Three types of curved nonograms of increasing complexity [7], shown with solutions. **(a)** *Basic* puzzles have no popular faces. **(b)** *Advanced* puzzles may have popular faces, but no self-intersections. **(c)** *Expert* puzzles have self-intersecting curves. We can observe closed curves (without clues) in **(a)** and **(c)**.

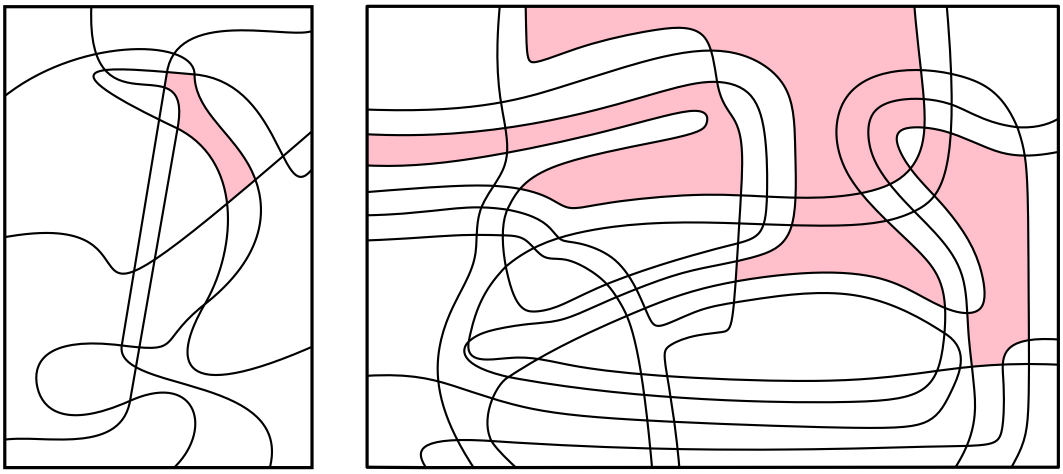


Figure 4 Some examples of real puzzles (without the clues) with all popular faces highlighted.

One possibility to generate nonograms of a specific complexity would be to take an existing generator and modify the output. In this paper, we explore what we can do by inserting a single new curve into the output arrangement. Clearly, inserting more curves will not get rid of self-intersections, so we focus on changing advanced puzzles into basic puzzles; i.e., removing all popular faces.

Results. We show in Section 3 that deciding whether we can remove all popular faces from a given curve arrangement by inserting a single curve – which we call the N1R problem – is NP-complete. However, often the number of popular faces is small, see Figure 4. Hence, we are also interested in the problem parametrized by the number of popular faces k; we show in Section 4 that the problem can be solved by a randomized algorithm in FPT time.

2 Resolving one Popular Face by Adding a Single Curve

As a preparation, we analyze how a single bad face F can be resolved. If F is visited more than twice by some curve, it is easy to see that it cannot be resolved with a single additional curve ℓ, and we can immediately abort. Otherwise, there are *duplicate* edges among the edges of F, which belong to a curve that visits F twice. As a visual aid, we indicate each such pair of edges by connecting them with a red *curtain*, see Figure 5a or 6d.

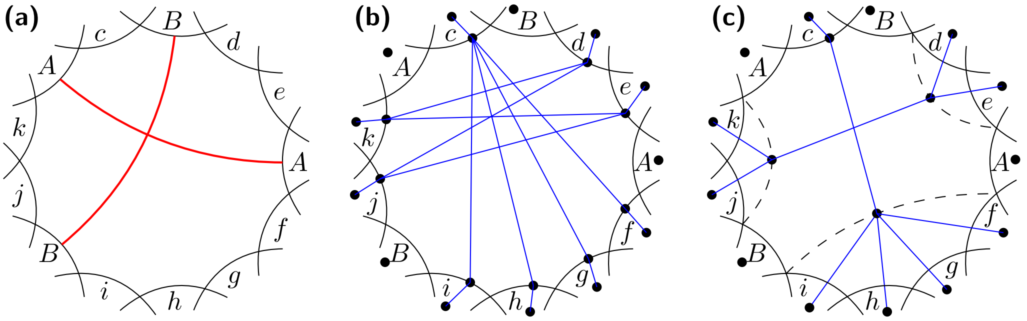


Figure 5 Resolving a popular face F

▶ **Lemma 1.** *To ensure that a popular face F becomes unpopular after insertion of a single curve ℓ into the arrangement, it is necessary and sufficient that the curve ℓ*

1. *visits the face F exactly once;*
2. *does not enter or exit through a duplicate edge;*
3. *separates each pair of duplicate edges. In other words, ℓ must cut all curtains.* ◀

The blue segments in Figure 5b show the ways how ℓ may pass through a popular face.

3 Removing Popular Faces with a Single Curve is NP-Complete

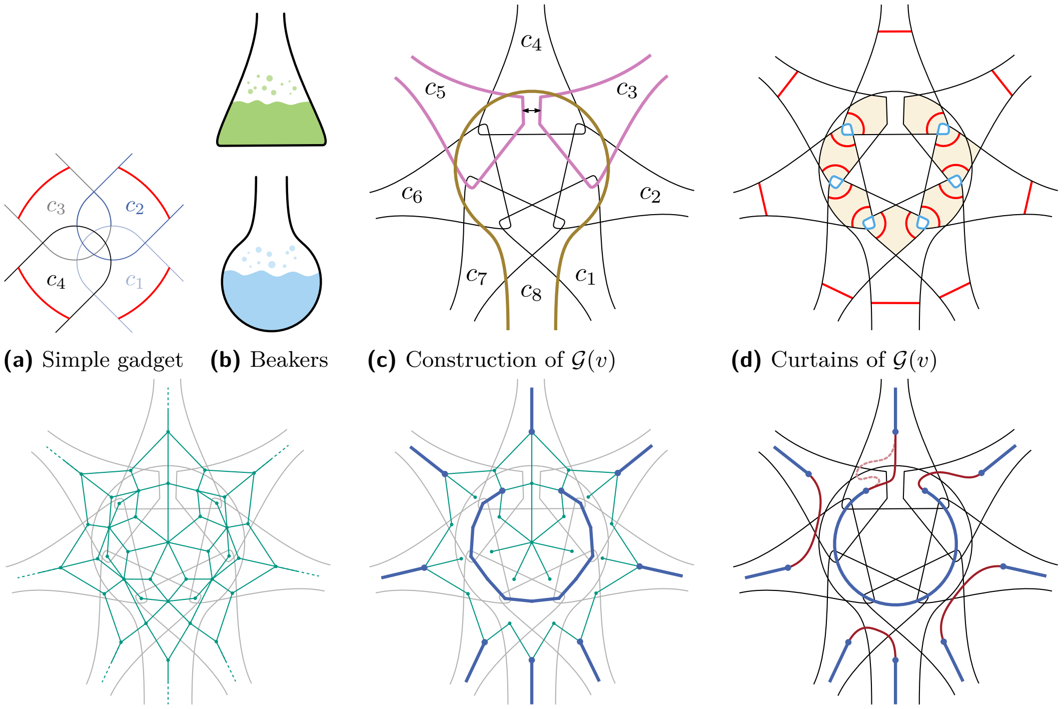
We reduce the NP-hard problem of finding a *non-intersecting Eulerian cycle in a plane graph G*, i.e., a cycle that visits every edge exactly once, such that consecutive edges belong to a common face [2]. We represent every vertex v and edge (u,v) in G by a vertex gadget $\mathcal{G}(v)$ and an edge gadget $\mathcal{G}(u,v)$, consisting of open curves starting and ending at the frame, resulting in a curve arrangement $\mathcal{A}$. For detailed constructions and analysis see Appendix A in the full version [6].

Every vertex has even degree. Degree-2 vertices can be eliminated. For degree-4 vertices, we use the simple construction of Figure 6a. For a vertex v of degree $d \geq 6$, $\mathcal{G}(v)$ consists of $d-1$ curves (*beakers*, see Figure 6b) $c_1, \ldots, c_{d-1}$ placed symmetrically around the location of v, which extend outwards to form the incident edges. Each beaker intersects four adjacent beakers (two on each side), except for $c_{d/2-1}$ and $c_{d/2+1}$ (lilac curves). We place an additional circular beaker c_d (the brown curve) to form the intersection pattern of Figure 6c.

All popular faces and the curtains in $\mathcal{G}(v)$ are shown in Figure 6d. To cut all curtains (item 3 in Lemma 1), ℓ has to enter and exit the orange faces through the blue segments. This forces ℓ to traverse these faces (and the degree 2 faces in between) in sequence according to the central blue chain in Figure 6f. To cut the curtains in the open ends of the beakers ℓ also has to traverse them outward as shown by the outer blue parts.

Every curve traversing faces of a curve arrangement $\mathcal{A}$ corresponds to a path in the dual graph of $\mathcal{A}$. Since ℓ is a single curve, all endpoints in $\mathcal{G}(v)$ have to be matched up without creating closed loops or having ℓ cross itself. Therefore, every endpoint in a beaker c_i can only be connected to the one in either c_{i+1} or c_{i-1}, corresponding to a non-intersecting Eulerian cycle connecting the i-th edge to its right or left neighboring edge. The inner endpoints also have to be connected, and can only connect to the endpoint in $c_{d/2}$ and one of $c_{d/2\pm1}$. This results in two possible connection pairs, one of which is shown in Figure 6g.

The edge gadget $\mathcal{G}(u,v)$ connects two beakers (one from $\mathcal{G}(u)$ and $\mathcal{G}(v)$) by routing their open ends close together, placing three curves over them and bending all curves to either



(a) Simple gadget **(b)** Beakers **(c)** Construction of $\mathcal{G}(v)$ **(d)** Curtains of $\mathcal{G}(v)$

(e) Dual graph of $\mathcal{G}(v)$ **(f)** Forced paths in $\mathcal{G}(v)$ **(g)** Possible routings of ℓ

Figure 6 **(a)** Vertex gadget for degree-4 vertices. **(b)** Basic beaker shapes. **(c)** Vertex gadget $\mathcal{G}(v)$ for a degree-8 vertex, **(d)** its curtains and **(e)** the dual graph of $\mathcal{G}(v)$. **(f)** The highlighted orange faces in **(d)** force the blue connections in the dual graph and restrict the dual graph. **(g)** One of two symmetric possibilities for the splitting curve ℓ. Light red shows an alternative routing of ℓ.

side as shown in Figure 7 to form two bundles A and B of parallel curves. A and B have different curves on the outside and therefore cannot create popular faces between them. As a direct consequence, all popular faces are contained in either a vertex or an edge gadget.

This connects a popular face in $\mathcal{G}(u)$ to one in $\mathcal{G}(v)$ via a chain of consecutive popular faces in $\mathcal{G}(u,v)$ and ℓ has to traverse $\mathcal{G}(u,v)$ along the thin blue axis. All bundles are routed through the faces and beakers of other edges until they reach the frame (see Figure 8).

It is now obvious that a curve ℓ visiting all gadgets and splitting all popular faces of an arrangement $\mathcal{A}$ gives rise to a non-intersecting Eulerian cycle in G, and vice versa. Since $\mathcal{A} \cup \ell$ can be represented as a plane graph, we can represent it in polynomial space and verify in polynomial time if it contains any popular faces and we conclude that N1R is NP-complete.

4 Resolving an Arrangement with Few Popular Faces

In this section, we will show that N1R with k popular faces can be solved by a randomized algorithm in $O\left(2^k \text{poly}(n)\right)$ time, thus placing N1R in the class randomized FPT when parameterized by the number k of popular faces.

We model the problem as a problem of finding a simple (i.e. vertex-disjoint) cycle in a modified dual graph G, subject to a constraint that certain edges must be visited:

▶ Problem (Simple Cycle with Edge Set Constraints). Given an undirected graph $G = (V, E)$ and k subsets $S_1, S_2, \ldots, S_k \subseteq E$ of edges, find a simple cycle, if it exists, that contains exactly one edge from each set S_i.

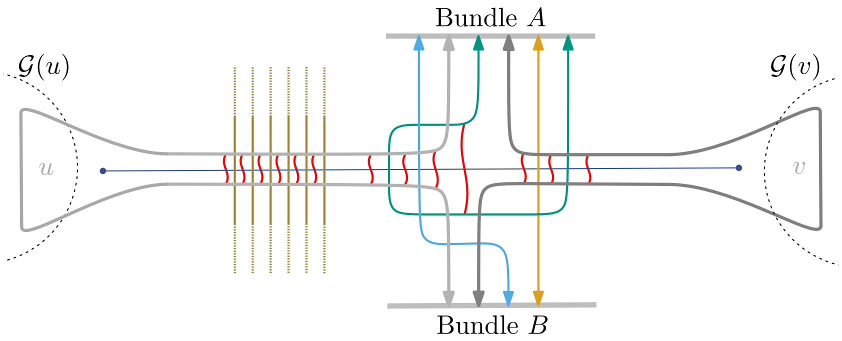


Figure 7 $\mathcal{G}(u, v)$ connects two beakers with three additional curves creating two bundles, which lead into incident faces. Any number of other bundles (brown curves) can cross either beaker.

We have one edge set S_i for each popular face F_i. To take care of the conditions of Lemma 1, we use a modified dual graph in which we model the passage through F_i directly by an edge instead of having a dual node for F_i, see Figure 5b. On each edge of F_i that is not a double edge, we place a *terminal node* that represents the entrance or exit through that edge. Inside F_i, we add an edge between all pairs of terminal nodes that satisfy the conditions of Lemma 1. These edges form the special set of edges S_i corresponding to the face F_i.

Outside F_i, we connect each terminal to the dual node corresponding to the incident face (unless that face is also popular and does not have a dual node; in that case, the node plays the role of terminal node in both faces).

▶ **Lemma 2.** *The simple cycles in the modified dual graph defined above that use exactly one edge from each set S_i are in one-to-one correspondence with the curves that can be added to the arrangement so that no popular faces remain.*

If we want, we can force the curve ℓ to start and edge at the frame by defining an additional set S_i containing all edges incident to the dual vertex of the outer face.

Now we apply a randomized algorithm for the edge-set constrained simple cycle problem, adapting an algorithm of Björklund, Husfeld, and Taslaman [4], which computes simple paths through k specified vertices or (single) edges. Our focus on edges instead of vertices actually makes the algorithm simpler: it avoids certain technicalities that are associated with visiting the same edge twice in succession. The extension to *sets* of edges is straightforward.

▶ **Theorem 3.** *The problem* Simple Cycle with Edge Set Constraints *with k edge sets in a graph with n vertices can be solved with a randomized algorithm in $O\left(2^k \text{poly}(n)\right)$ time.*

The algorithm assigns a random weight $w(e)$ from a sufficiently large finite field F of characteristic 2 to every edge e. The weight of a (possibly nonsimple) *walk* $c = (e_1, \ldots, e_m)$ of length m is defined as the product $\gamma(c) = \prod_{e \in c} w(e) \in F$ of the edge weights. The algorithm then computes in a dynamic-programming fashion sums of weights of certain sets C of walks. Each set C is characterized by the length of the walks, by their starting point and endpoint, and by the subset of edge sets $\mathcal{S}$ that are visited. Eventually, the algorithms computes the sequence T_m, $m = 1, 2, \ldots, n$ of sums of closed walks of length m which use exactly one edge from each S_i. (This condition is explicitly enforced by the dynamic-programming recursion, leading to the factor 2^k.) The condition that the walk should be simple is guaranteed by the trickery of the field F of characteristic 2: If a walk c contains a cycle, it can be matched with another walk c' that uses the same set of edges, but visits the cycle in reverse order. Therefore $\gamma(c') = \gamma(c)$, and $\gamma(c) + \gamma(c') = 0$ in F. It is ensured that every nonsimple walk is

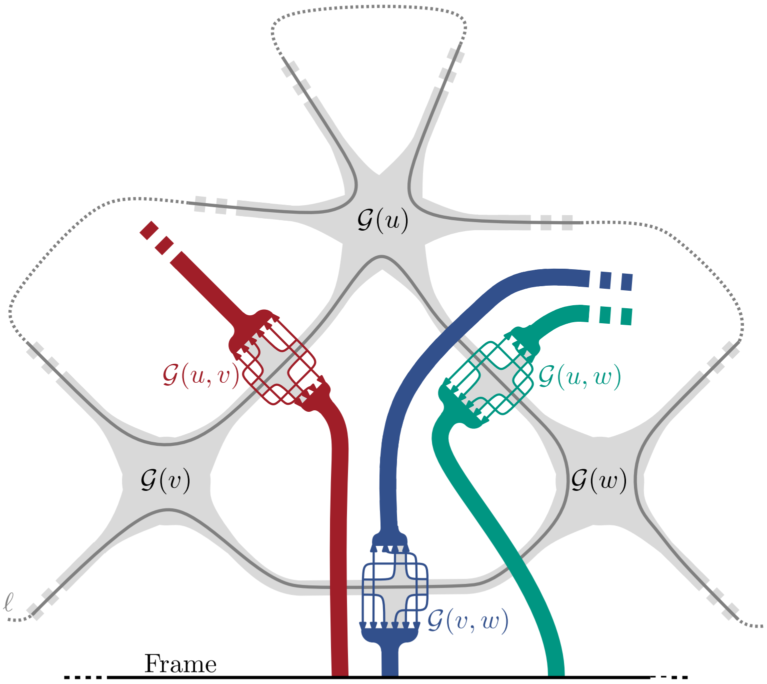


Figure 8 Schematic representation of three vertex and edge gadgets. The bundles are routed through beakers of other edges ending at the frame (partially shown at the bottom of the figure). A possible routing of ℓ is shown in dark gray.

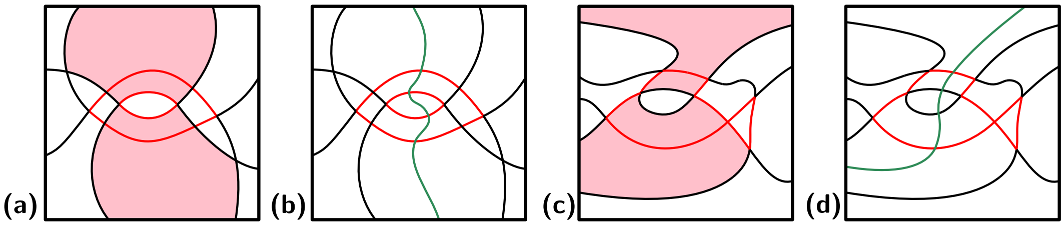


Figure 9 Input **(a,c)** and resulting output **(b,d)** generated by the implementation; the green curve is the curve with the smallest number of crossings that resolves the popular faces.

matched exactly once, and hence the nonsimple walks cancel in the sums T_m. If the shortest simple walk has m edges, it follows that $T_1 = T_2 = \cdots = T_{m-1} = 0$, and it can be shown that T_m is nonzero with high probability. In this case, such a simple walk can be constructed in $O\left(2^k \text{poly}(n)\right)$ time. See Appendix B in the full version [6] for details.

Connecting all terminals between two runs of consecutive edges in the dual graph incurs a quadratic blowup in the number of edges. This blowup can be avoided: We cut off each run of consecutive edges, like *fghi*, by an additional edge (shown dotted in Figure 5c) and place a single terminal node there. Now, one has to take care that the cycle does not use two such terminal edges in succession, like the edges crossing f and h, because such a cycle would not correspond to a valid curve. This constraint must be added to the problem definition, and the algorithm modified accordingly, see Appendix D in the full version [6].

Figure 9 shows initial results of an implementation of our algorithm on two small test instances. More details are given in the full version [6].

References

1 Kees Joost Batenburg and Walter A. Kosters. On the difficulty of nonograms. *ICGA Journal*, 35(4):195–205, 2012. doi:10.3233/ICG-2012-35402.

2 Samuel W. Bent and Udi Manber. On non-intersecting Eulerian circuits. *Discrete Applied Mathematics*, 18(1):87–94, 1987. doi:10.1016/0166-218X(87)90045-X.

3 Daniel Berend, Dolev Pomeranz, Ronen Rabani, and Ben Raziel. Nonograms: Combinatorial questions and algorithms. *Discrete Applied Mathematics*, 169:30–42, 2014. doi:10.1016/j.dam.2014.01.004.

4 Andreas Björklund, Thore Husfeld, and Nina Taslaman. Shortest cycle through specified elements. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 1747–1753, USA, 2012. Society for Industrial and Applied Mathematics. doi:2095116.2095255.

5 Yen-Chi Chen and Shun-Shii Lin. A fast nonogram solver that won the TAAI 2017 and ICGA 2018 tournaments. *ICGA Journal*, 41(1):2–14, 2019. doi:10.3233/ICG-190097.

6 Phoebe de Nooijer, Soeren Nickel, Alexandra Weinberger, Zuzana Masárová, Tamara Mchedlidze, Maarten Löffler, and Günter Rote. Removing popular faces in curve arrangements, 2022. arXiv:2202.12175.

7 Mees van de Kerkhof, Tim de Jong, Raphael Parment, Maarten Löffler, Amir Vaxman, and Marc J. van Kreveld. Design and automated generation of Japanese picture puzzles. *Comput. Graph. Forum*, 38(2):343–353, 2019. doi:10.1111/cgf.13642.

Transitions in Dynamic Map Labeling*

Thomas Depian, Guangping Li, Martin Nöllenburg, and Jules Wulms

Algorithms and Complexity Group, TU Wien, Vienna, Austria
thomas.depian@tuwien.ac.at,{guangping, noellenburg, jwulms}@ac.tuwien.ac.at

Abstract

The labeling of point features on a map is a well-studied topic. In a static setting, the goal is to find a non-overlapping label placement for (a subset of) point features. In a dynamic setting, the set of point features and their corresponding labels changes, and the labeling has to adapt to such changes. To aid the user in tracking these changes, we can use morphs, here called *transitions*, to indicate how a labeling changes. Such transitions have not gained much attention yet, and we investigate different types of transitions for labelings of points, most notably *consecutive* transitions and *simultaneous* transitions. We give (tight) bounds on the number of overlaps that can occur during these transitions. When each label has a (non-negative) weight associated to it, and each overlap imposes a penalty proportional to the weight of the overlapping labels, we show that it is NP-complete to decide whether the penalty during a simultaneous transition has weight at most k.

Related Version arXiv:2202.11562

1 Introduction

Maps are ubiquitous in the modern world: from geographic to political maps, and from detailed road networks to schematized metro maps, maps are used on a daily basis. Advances in technology allow us to use digital maps on-the-fly and in a highly interactive fashion, by means of panning, zooming, and searching for map features. Besides changes induced by the user, maps can also change passively, for example automated panning during gps routing, or changing points of interest when visualizing time-varying geospatial (point) data.

Important features on a map are often labeled. Examples of such features are areas (such as countries and mountain ranges), curves (for example roads and rivers), and most importantly points (of interest). The aforementioned interactions force map features and their corresponding labels to change, by appearing, disappearing, or changing position. Instead of swapping between the map before and after such changes, we can use morphs, here called *transitions*, to allow the user to more easily follow changes in map features and labelings. Figure 1 shows why such transitions are important: even for two very similar map labelings, a lot of mental effort can be required to identify the differences.

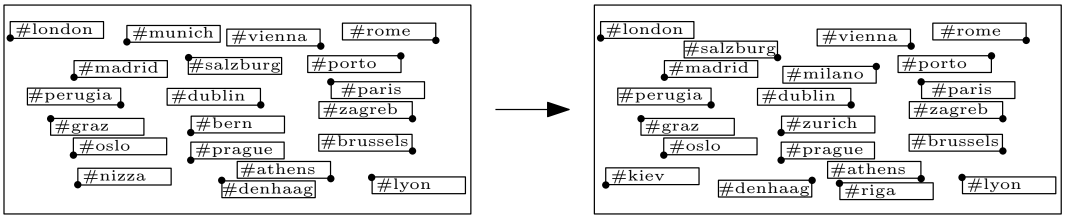


Figure 1 A visual scan of the individual labels is necessary to identify all changes [11].

* GL and JW are (partially) funded by the Austrian Science Fund (FWF) under grant P31119 and JW is partially funded by the Vienna Science and Technology Fund (WWTF) under grant ICT19-035.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

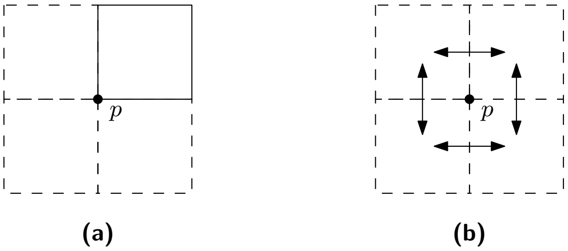


Figure 2 **(a)** The four candidate positions for label l of point p, with l placed in the top-right position. **(b)** Labels continuously move between candidate positions using the sliding-position model.

While previous research focused mainly on (the complexity of) computing labelings, in various static [1, 8, 12], interactive [3, 4, 9, 10], and dynamic [2, 5] settings, in this abstract we study transitions on maps that show point features P and their labels L. Let P be a finite point set in $\mathbb{R}^2$, where each point $p_i \in P$ has a label $l_i \in L$ associated to it. Labels are axis-aligned unit-sized squares in the frequently used four-position model, that is, each point p_i has four possible candidate positions to place label l_i [8] (see Figure 2a). While labels are often modeled as arbitrary (axis-aligned) rectangles, we use squares with side length $\sigma = 1$ for simplicity, and show in [7] how our results extend to arbitrary rectangles. A labeling $\mathcal{L} \subseteq L$ of P consists of a set of pairwise non-overlapping labels, and can be drawn on a map conflict-free, by drawing only the labels that are in $\mathcal{L}$ with their associated points. If the label $l \in L$ for a point $p \in P$ is not contained in $\mathcal{L}$, we do not draw p either.

Furthermore, we work in a dynamic setting, where points appear and disappear at different moments in time, and hence the set P changes through additions and deletions. Every time additions and deletions are made to P, a new overlap-free labeling must be computed, thus resulting in a change from labeling $\mathcal{L}_1$, before the changes, to labeling $\mathcal{L}_2$, afterwards. In this abstract we study different types of transitions from $\mathcal{L}_1$ to $\mathcal{L}_2$. During such a transition, the individual labels are allowed to move in the sliding-position model [12] (see Figure 2b). Our aim is to find transitions that achieve optimization criteria, such as minimizing the number of overlaps during a transition, or minimizing the time required to perform a transition. To our knowledge, this is the first time transitions have been studied in this way.

Problem description. Given two (overlap-free) labelings $\mathcal{L}_1$ and $\mathcal{L}_2$, we denote a transition between them with $\mathcal{L}_1 \rightarrow \mathcal{L}_2$. Such a transition consists of changes of the following types.

Additions If only label l_i of a feature point p_i must be added, we denote this by $\mathcal{L}_1 \xrightarrow{A_i} \mathcal{L}_2$.

Removals If only label l_i of a feature point p_i must be removed, we denote this by $\mathcal{L}_1 \xrightarrow{R_i} \mathcal{L}_2$.

Movements If only label l_i of a feature point p_i must change from its position in $\mathcal{L}_1$ to a new position in $\mathcal{L}_2$, we denote this by $\mathcal{L}_1 \xrightarrow{M_i} \mathcal{L}_2$. Movements are unit speed and axis-aligned, in the sliding-position model. Note that a diagonal movement, as in Figure 3a (left), takes twice as long as a movement to an adjacent position.

A label is *stationary* if it remains unchanged during a transition. Applying multiple transitions consecutively is indicated by chaining the corresponding transition symbols: $\mathcal{L}_1 \xrightarrow{M_i M_j} \mathcal{L}_2$ denotes that label l_i moves before label l_j. Furthermore, $\mathcal{L}_1 \xrightarrow{M} \mathcal{L}_2$ is a shorthand for applying all movement-transitions simultaneously. All these notions extend to additions and removals, using A and R, respectively, instead of M. A transition has no effect if no point must be transformed with the respective transition, e.g., even if there are no additions, the transition $\mathcal{L}_1 \xrightarrow{A} \mathcal{L}_2$ is still applicable; it simply does not modify the labeling.

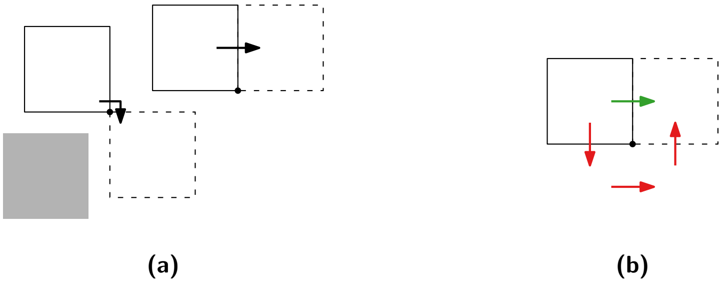


Figure 3 **(a)** Minimizing overlaps by moving around the gray stationary label. **(b)** Minimizing duration by using a single movement along the green arrow, instead of moving along the red arrows.

We aim to identify types of transitions that try to achieve the following goals.

$\mathcal{G}_1$– **Minimize overlaps** While the two labelings are overlap-free, overlaps can occur during the transition from $\mathcal{L}_1$ to $\mathcal{L}_2$. Those overlaps should be avoided as much as possible, by, for instance, adjusting the movement direction of labels, as shown in Figure 3a.

$\mathcal{G}_2$– **Minimize transition duration** The main goal is still to show a map in a (mostly) static state. Hence, we want to perform the transitions as fast as possible. This can be achieved by disallowing detours, as in Figure 3b, or by performing the changes simultaneously.

Optimizing both goals simultaneously is often impossible as there can be a trade-off: performing the transition as fast as possible to achieve $\mathcal{G}_2$ often leads to unnecessary overlaps, while preventing as many overlaps as possible to achieve $\mathcal{G}_1$ may require more time. However, to work towards both $\mathcal{G}_1$ and $\mathcal{G}_2$, we can perform all additions simultaneously, as well as all removals. Furthermore, if we perform removals before movements, and movements before the additions, we create free space for the movements, to reduce the number of overlaps without wasting time. Let X be an arbitrary way of performing all movements required to change from $\mathcal{L}_1$ to $\mathcal{L}_2$ (consecutively and/or simultaneously), then we can observe the following.

▶ **Observation 1.** *A transition of the form $\mathcal{L}_1 \xrightarrow{RXA} \mathcal{L}_2$ aids in achieving both $\mathcal{G}_1$ and $\mathcal{G}_2$.*

In the following sections we introduce and analyze different *transition styles*, each a variant of the style RXA, as prescribed by Observation 1, while filling in X in a unique way.

All omitted proofs and details can be found in the complete version [7].

2 Consecutive Transitions

Naive transitions. Before we can propose more elaborate transition styles, we first evaluate the potential overlaps for a single label performing its movement. Figure 4a shows how only a single stationary square label can interfere with the moving label.

▶ **Lemma 2.1.** *In $\mathcal{L}_1 \xrightarrow{RM_iA} \mathcal{L}_2$, where only label l_i moves, at most one overlap can occur.*

Next we consider an arbitrary order of all n moving labels in a transition. We define a conflict graph, which has a vertex for each moving label, and an edge between overlapping labels. With a packing argument we locally bound the degree of each of the n moving labels to 14 by considering the start, intermediate, and end position of such a label (these overlaps are achieved in Figure 4b). By the handshaking lemma this results in at most $7n$ overlaps.

▶ **Lemma 2.2.** *In $\mathcal{L}_1 \xrightarrow{RM_1 \ldots M_nA} \mathcal{L}_2$ at most $7n$ overlaps can occur.*

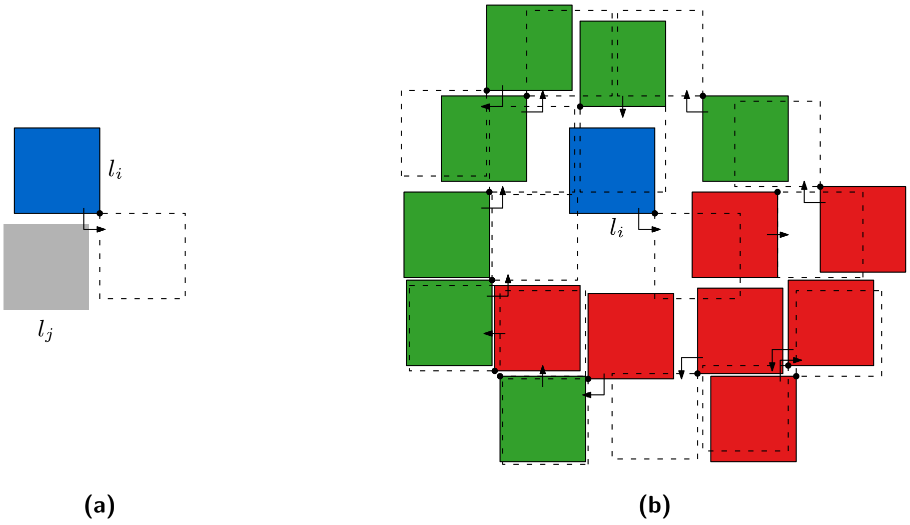


Figure 4 (a) Since all labels are squares with side length σ, the moving blue label l_i can overlap only a single gray stationary label l_j. **(b)** The blue label l_i overlaps 14 other labels during the movement transitions. The green labels move before l_i, red labels move after l_i.

DAG-based transitions. To refine the naive approach, we model dependencies between movements in a *movement graph*, and use it to order movements and avoid certain overlaps.

▶ **Definition 2.3** (Movement graph). Let $\mathcal{M} = \{M_1, \dots, M_n\}$ be a set of movements. Create for each movement $M_i \in \mathcal{M}$ a vertex v_i, and create a directed edge from v_i to v_j, $v_i \to v_j$, if some intermediate or end position of M_j overlaps with the start position of M_i, or the end position of M_j overlaps with some intermediate position of M_i. If intermediate positions of M_i and M_j overlap, create the edge $v_i \to v_j$, $i < j$. This results in the movement graph $G_{\mathcal{M}}$.

An example for a movement graph is shown in Figure 5.

▶ **Theorem 2.4.** *Movements in* $\mathcal{L}_1 \xrightarrow{RM_1 \dots M_n A} \mathcal{L}_2$ *can be rearranged such that at most* $n + m$ *overlaps occur, if removing* m *edges transforms* $G_{\mathcal{M}}$*, with* $\mathcal{M} = \{M_1, \dots, M_n\}$*, into a DAG.*

Proof. By Lemma 2.1, we know that at most one overlap occurs when moving a single label to a free end position. This leads to at most n overlaps for n consecutively moving labels, if no label moves to (or through) a position occupied by a label, which starts moving later.

Let $G_{\mathcal{M}}$ be a movement graph with $\mathcal{M} = \{M_1, \dots, M_n\}$. There are two cases:

Case (1) If $G_{\mathcal{M}}$ is acyclic, then handling all movements according to any topological ordering of the vertices of $G_{\mathcal{M}}$ produces no additional overlaps.

Case (2) If $G_{\mathcal{M}}$ contains cycles, then overlaps may be inevitable because each label in such a cycle wants to move to or through a position that is occupied by another moving label. Moreover, as the movements happen sequentially, one label in this cycle must move first and therefore may cause an overlap. Let m be the smallest number of edges that must be removed to break each cycle in $G_{\mathcal{M}}$, i.e., the size of a minimum feedback arc set S. As $G_{\mathcal{M}}$ is cycle-free after removing S, case (1) applies and m additional overlaps suffice. ◀

We can see in Figure 5 that this bound is tight. Furthermore, it is not always necessary to perform all movements consecutively. We can observe that movements which are unrelated in $G_{\mathcal{M}}$ can be performed simultaneously: when no overlap is possible, there is no edge in $G_{\mathcal{M}}$.

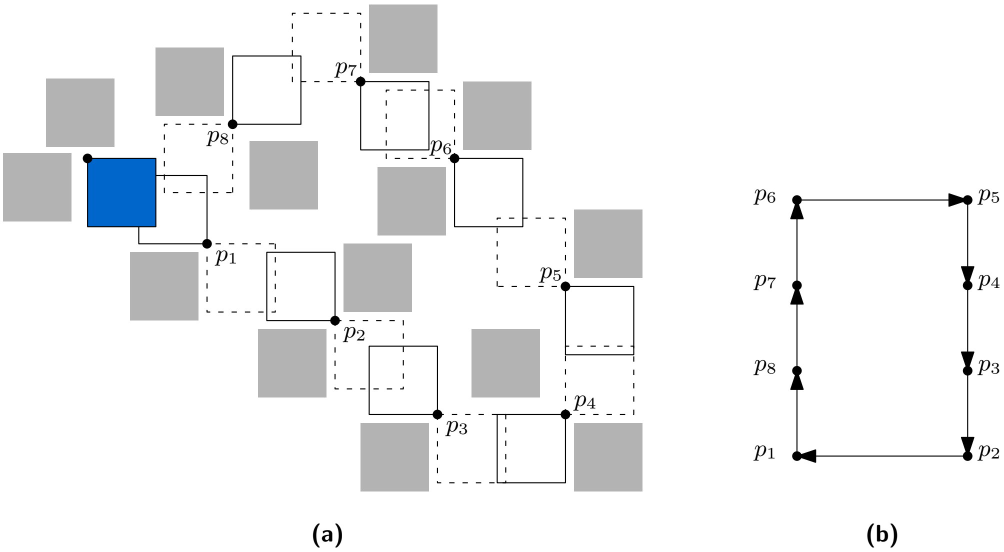


Figure 5 **(a)** The blue label is added in this transition and forces $n + m$ inevitable overlaps during movement ($n = 8$ and $m = 1$). Gray labels are stationary. **(b)** The corresponding movement graph.

3 Simultaneous Transitions

Figure 6 shows three timelines of different transition styles, (1) a naive consecutive transition, (2) a DAG-based transition, and (3) simultaneous movement. While (1) produces four overlaps and takes four units of time, (2) and (3) produce no overlaps, and (3) only takes a single unit of time. This shows that it is sometimes unnecessary to perform the movements consecutively to minimize overlaps. In this section, we investigate how simultaneous movements influence the number of overlaps, and the complexity of minimizing overlaps.

▶ **Theorem 3.1.** *In $\mathcal{L}_1 \xrightarrow{RMA} \mathcal{L}_2$ at most $6n$ overlaps can occur, where n is the number of labels that must be moved, and all movements are performed at unit speed.*

Proof sketch. We again use a conflict graph, as for Lemma 2.2, with a more intricate packing argument than before (see Figure 7). We consider a σ-wide area around the movement of each label l, and argue where the start positions of labels overlapping l can be located inside this area. We then bound the degree of each of the n moving labels to 12 (and this degree is achieved in Figure 7d), which by the handshaking lemma results in at most $6n$ overlaps. ◀

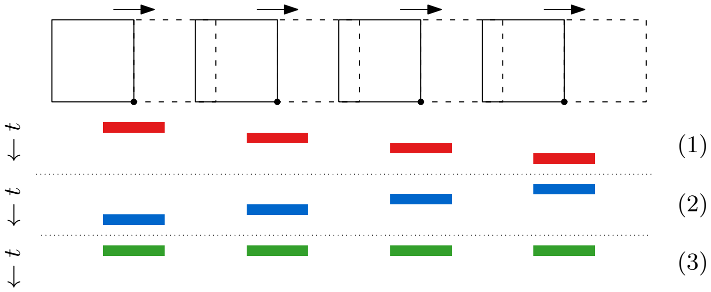


Figure 6 Comparison of possible movement orderings with respect to $\mathcal{G}_1$ and $\mathcal{G}_2$.

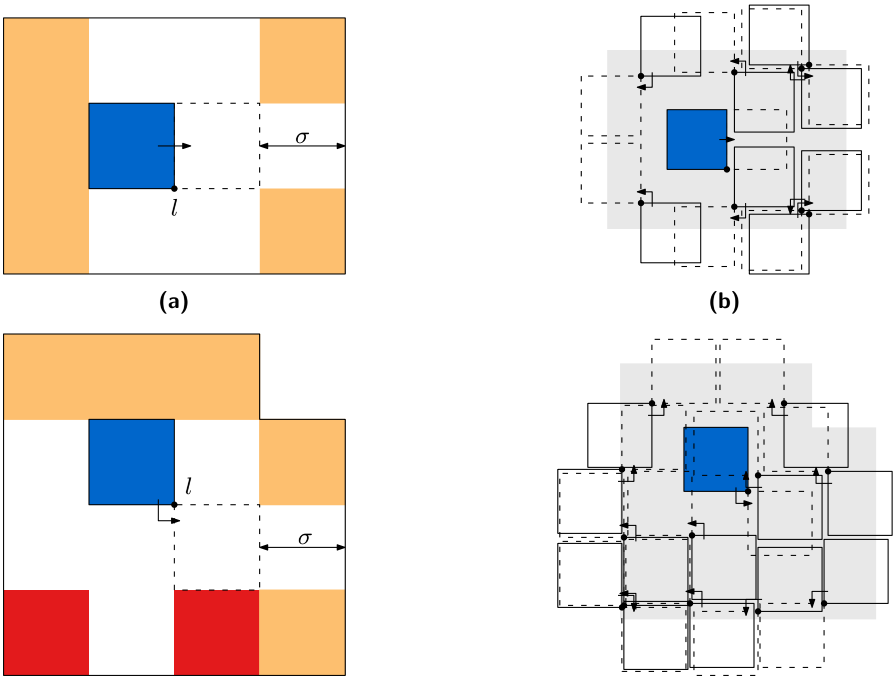


Figure 7 Overlapping regions for **(a)** non-diagonal and **(c)** diagonal movement of the blue label l. Label l has at most **(b)** eight overlaps, **(d)** twelve overlaps with moving (white) labels. Labels starting in orange/red areas cannot overlap l, as l moves away, or they overlap the end position of l.

3.1 Complexity of Computing Simultaneous Transitions

In this section, we show that it is NP-complete to minimize the number of overlaps in a *weighted* $\mathcal{L}_1 \xrightarrow{RMA} \mathcal{L}_2$-transition by choosing the direction of diagonal movements.

▶ **Definition 3.2** (Weighted Transition). Let $\mathcal{L}_1 \xrightarrow{\Sigma} \mathcal{L}_2$ be a transition, where Σ denotes an arbitrary transition style of additions, movements, and removals, and let w be a weight function that assigns to each label $l \in L$ a non-negative weight $w(l) \in \mathbb{R}_0^+$. A weighted transition $\mathcal{L}_1 \xrightarrow[w]{\Sigma} \mathcal{L}_2$ performs $\mathcal{L}_1 \xrightarrow{\Sigma} \mathcal{L}_2$, but when two labels l_i and l_j overlap, a penalty of weight $w(l_i) \cdot w(l_j)$ is introduced. The *total penalty* W is equal to the sum of penalty weights.

▶ Problem 1. Given a weighted transition $\mathcal{L}_1 \xrightarrow[w]{RMA} \mathcal{L}_2$ and $k \in \mathbb{R}_0^+$, can we assign a movement direction to each diagonal movement such that the total penalty W is at most k?

▶ **Theorem 3.3.** *It is NP-complete to decide whether W is at most k for $\mathcal{L}_1 \xrightarrow[w]{RMA} \mathcal{L}_2$.*

Proof sketch. Given a movement direction for each label, it is easy to check whether W is at most k by considering each pair of labels and checking for overlaps. Hence Problem 1 is contained in NP. For NP-hardness, we reduce from an instance F of Planar Monotone Max 2-Sat [6]. Figure 8 gives an overview of the required gadgets. Clause and variable gadgets consist of two opposing labels at their core, corresponding, respectively, to the assignments of the two literals in a clause, or the binary choice for a variable. For an unsatisfied clause, an overlap occurs inside the clause gadget, whenever both labels move towards each other (inwards). The corresponding labels have weight one, and hence such

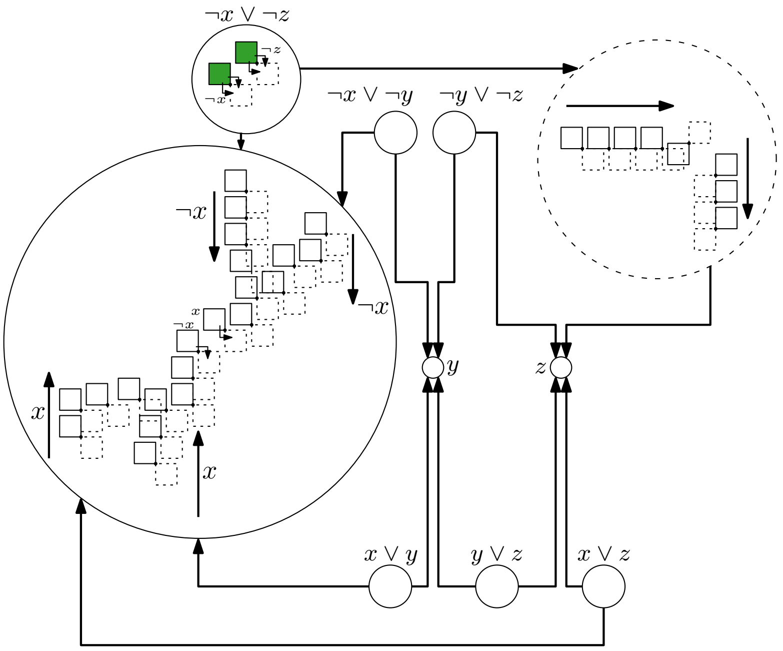


Figure 8 Reduced instance for the formula $F = (\neg x \lor \neg z) \land (\neg x \lor \neg y) \land (\neg y \lor \neg z) \land (x \lor y) \land (y \lor z) \land (x \lor z)$. The weight of white and green labels is $n + 1$ and 1, respectively.

an overlap would incur a penalty of weight one. A variable gadget has two opposing labels for setting the variable to *true* or *false*. Choosing a movement direction outward from the variable gadget, for example on the "true"-side, will cause a domino effect, propagating towards the gadgets of clauses with negative occurrences of this variable. There it results in inward movement, and hence this corresponds to setting the variable to not be false (and thus be true). Choosing the outward movement for both variable states is never beneficial: that variable is neither true nor false. The movement directions chosen in the variable gadgets are propagated to the appropriate clauses using the (planar) embedding of the incidence graph of F. All labels outside of clause gadgets have weight $n + 1$ and hence producing an overlap outside of a clause gadget will result in a large penalty of weight greater than n. As such, we either have movement directions that produce a total penalty of at most k for some positive $k < n$, and overlaps correspond to unsatisfied clauses, or we have a total penalty of at least n, and no clauses can be satisfied (or the variable assignment is inconsistent). Thus, $n - k$ clauses are satisfiable in F, if and only if we have k overlaps in our reduced instance. ◀

4 Conclusion

In this abstract we performed a first investigation into the number of overlaps produced by transitions on labelings of points, and started by proving tight upper bounds for various transition styles. Finally, we showed that it is NP-complete to decide whether a weighted simultaneous transition has a penalty of at most k. We see this abstract as a first step towards understanding such transitions in map labeling. Therefore we have many open questions for future work, such as:

- Do transitions work well in practice? Can we verify our results with a prototype?

- Should we develop new transition styles or improve the existing ones? Can we utilize more structured movement, like performing all movements in the same direction simultaneously?
- Is choosing the direction of labels in simultaneous transitions still NP-hard in the unit weight case?
- Can we analyze transitions from the point of view of (algorithmic) stability?

References

1 Pankaj K. Agarwal, Marc J. van Kreveld, and Subhash Suri. Label placement by maximum independent set in rectangles. *Computational Geometry*, 11(3-4):209–218, 1998. doi:10.1016/S0925-7721(98)00028-5.

2 Lukas Barth, Benjamin Niedermann, Martin Nöllenburg, and Darren Strash. Temporal map labeling: a new unified framework with experiments. In *Proc. 24th ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 1–10, 2016. doi:10.1145/2996913.2996957.

3 Ken Been, Eli Daiches, and Chee-Keng Yap. Dynamic map labeling. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):773–780, 2006. doi:10.1109/TVCG.2006.136.

4 Ken Been, Martin Nöllenburg, Sheung-Hung Poon, and Alexander Wolff. Optimizing active ranges for consistent dynamic map labeling. *Computational Geometry*, 43(3):312–328, 2010. doi:10.1016/j.comgeo.2009.03.006.

5 Sujoy Bhore, Guangping Li, and Martin Nöllenburg. An Algorithmic Study of Fully Dynamic Independent Sets for Map Labeling. In *Proc. 28th European Symposium on Algorithms (ESA)*, volume 173 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 19:1–19:24, 2020. doi:10.4230/LIPIcs.ESA.2020.19.

6 Kevin Buchin, Valentin Polishchuk, Leonid Sedov, and Roman Voronov. Geometric Secluded Paths and Planar Satisfiability. In *Proc. 36th International Symposium on Computational Geometry (SoCG)*, volume 164 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 24:1–24:15, 2020.

7 Thomas Depian, Guangping Li, Martin Nöllenburg, and Jules Wulms. Transitions in Dynamic Map Labeling, 2022. doi:10.48550/arXiv.2202.11562.

8 Michael Formann and Frank Wagner. A packing problem with applications to lettering of maps. In *Proc. 7th International Symposium on Computational Geometry (SoCG)*, pages 281–288, 1991.

9 Andreas Gemsa, Martin Nöllenburg, and Ignaz Rutter. Consistent Labeling of Rotating Maps. *Journal of Computational Geometry*, 7(1):308–331, 2016. doi:10.20382/jocg.v7i1a15.

10 Chung-Shou Liao, Chih-Wei Liang, and Sheung H. Poon. Approximation algorithms on consistent dynamic map labeling. *Theoretical Computer Science*, 640:84–93, 2016. doi:10.1016/j.tcs.2016.06.006.

11 Ronald A. Rensink, John K. O'Regan, and James J. Clark. To See or not to See: The Need for Attention to Perceive Changes in Scenes. *Psychological Science*, 8(5):368–373, 1997. doi:10.1111/j.1467-9280.1997.tb00427.x.

12 Marc J. van Kreveld, Tycho Strijk, and Alexander Wolff. Point labeling with sliding labels. *Computational Geometry*, 13(1):21–47, 1999. doi:10.1016/S0925-7721(99)00005-X.

Preprocessing Imprecise Points for Furthest Distance Queries

Vahideh Keikha[1], Sepehr Moradi[2], and Ali Mohades[2]

1 **The Czech Academy of Sciences, Institute of Computer Science, Pod Vodárenskou věží 2, 182 07 Prague, Czech Republic**
keikha@cs.cas.cz

2 **Department of Computer Science, Amirkabir University of Technology**
(moradi,mohades)@aut.ac.ir

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 is not precise due to several resources generated by bounded precision of measuring devices, rounding errors, 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 may assume such a region as an *imprecise point*, that could be a disk, rectangle, line segment, etc. This uncertainty model is called *region-based* by Löffler and van Kreveld [11].

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- or the best-case behavior of the input, however, all cases are likely to happen. Another standpoint is preprocessing uncertain data for speeding-up the further computations on precise instances received later. We address the diameter problem in this context.

▶ 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 [19], computing an arbitrary triangulation [9, 20], the Delaunay triangulation [2, 12], 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 [20]. The same problem was also studied in [12] at which the same results

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

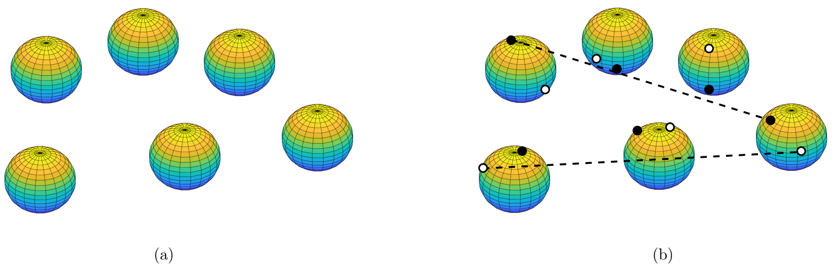


Figure 1 Problem definition: (a) A set D of 6 imprecise points modeled as unit balls, (b) and the diameter of two different realizations of D.

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, 15]. 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, 13] 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).

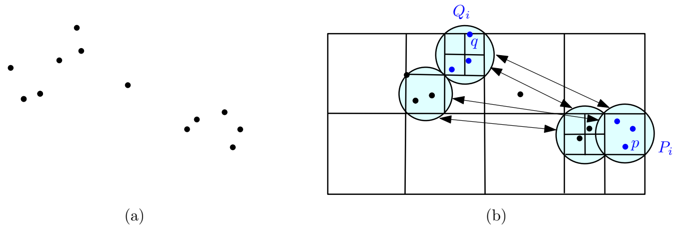
2 Preliminaries

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



(a) (b)

Figure 2 (a) Illustration of a point set and (b) a well-separated pair decomposition of it with 4 pairs (computed from the quadtree; see [16] for the definition and the algorithm).

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.

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 2.1** (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 $O(s^d n)$.

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

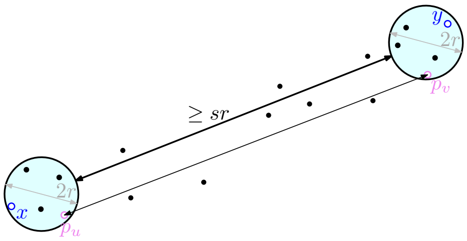


Figure 3 Diameter approximation using WSPD. The points x, y determine the diameter, and p_u, p_v approximate the diameter within a factor $1 + \epsilon$.

diameter approximation algorithm [7] in the query phase, using the computed WSPD.

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 [13]. 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 [17]. 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 [18].

In $\mathbb{R}^d$, we focus on approximation algorithms. An $f(d)$-approximation algorithm for this problem is the minimum enclosing ball (MEB) of a set of points that approximates the diameter of the points. For clarity, in $\mathbb{R}^3$ consider the configuration at which four points on the boundary of the MEB form a regular tetrahedron, and the side length of each triangular face determines the diameter. If one translates any pair of these points on the boundary of the MEB, to get closer, the diameter enlarges between at least one pair. The side length of the tetrahedron inside a sphere of radius r equals $\sqrt{\frac{8}{3}}r$. Hence, an $(\sqrt{\frac{3}{8}})$-approximation of the diameter of any set of points in $\mathbb{R}^3$ is achievable in $O(n)$ time. In $\mathbb{R}^d$, such an approximation factor grows exponentially to d, however, if d is constant, MEB can still be computed in $O(n)$ time using Megiddo's algorithm [14]. For general values of d, computation of MEB is more complicated, if an $f(d)$-approximation factor suffices. An approximation of the MEB can be computed in $O(dnz/\epsilon^{O(1)})$ time by using the randomized $(1 + \epsilon)$-approximation algorithm in [10] for computing the MEB 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 for general values of d.

[1] To the best of our knowledge, this is the best-known algorithm for computing the MEB, that has a

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 3.1.** *Let $\{(A_i, B_i) | 1 \leq i \leq m\}$ be a WSPD on the set $\{c_1, \dots, c_n\}$ of given disjoint unit disks with respect to $s' = \frac{8}{\epsilon} + 2$. Let $\mathcal{Q} = \{p_1, \dots, p_n\}$ 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 2.1 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 3.1 that we can 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.

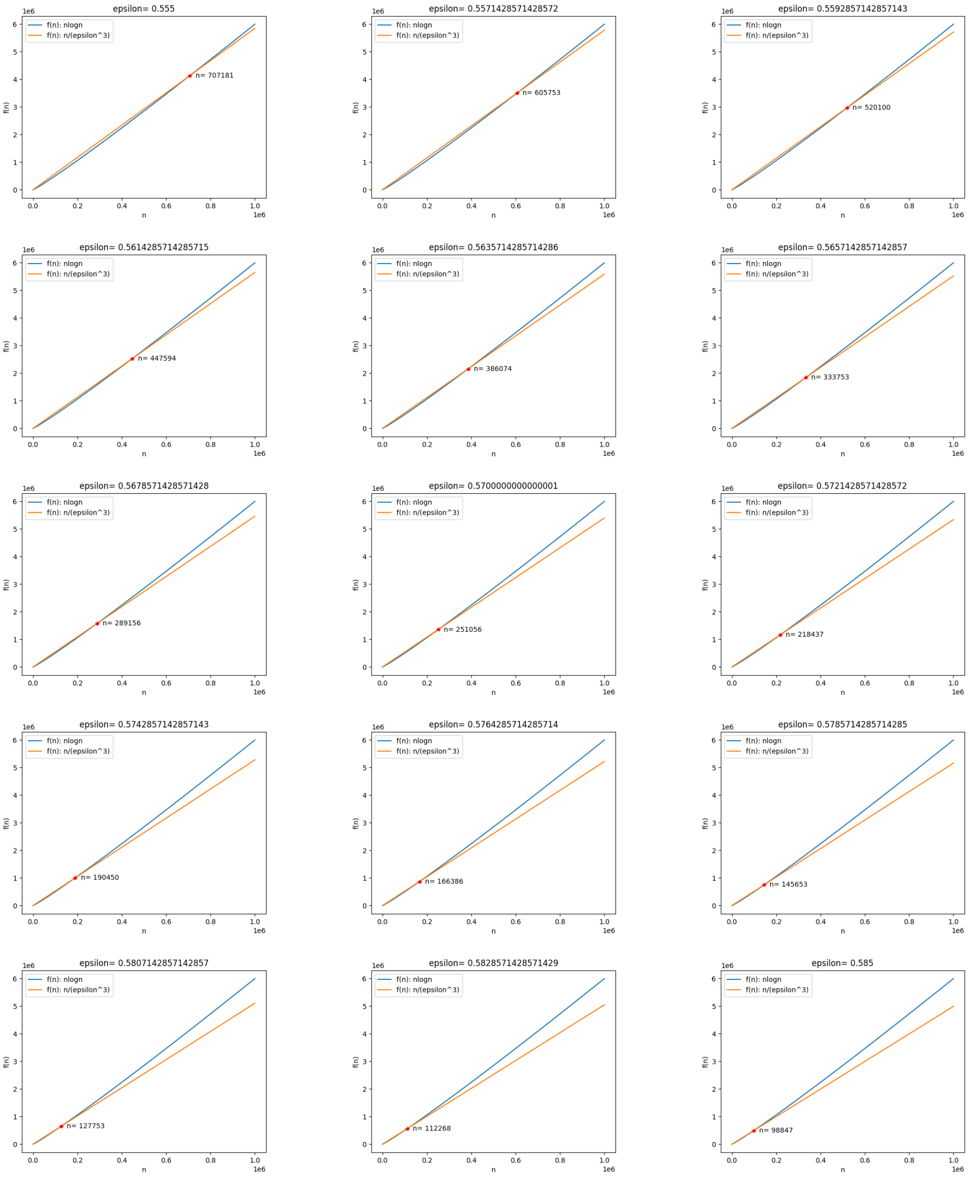
▶ **Theorem 3.2.** *For any given set $D = \{D_1, \dots, D_n\}$ of n imprecise points modeled as the same size balls which are pairwise disjoint, a $(1+\epsilon)$-approximation of the diameter of a realization $\mathcal{Q}$ of D can be computed in $O(\frac{n}{\epsilon^d})$ time, after $O(n \log n + \frac{n}{\epsilon^d})$ preprocessing time.*

Proof. Let $s = \frac{4}{\epsilon}$ and $s' = 2s + 2 = \frac{8}{\epsilon} + 2$ and $\mathcal{Q} = \{p_1, \dots, p_n\}$ be the set of precise points. Let $\{(A_i, B_i)\}_{i=1,\dots,m}$ 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 2.1 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. ◀

Moreover, for the particular case $d = 3$, we consider the efficiency of our method by considering the intersection point of the running time function of the query and the best known existing algorithm for computing the diameter. See Figure 4. This clarifies that for which values of n and ϵ the preprocessing is meaningful. Observe that for large values of n it is always efficient to perform the preprocessing.

linear dependency on d for general values of d.


epsilon= 0.555

epsilon= 0.5571428571428572

epsilon= 0.5592857142857143

epsilon= 0.5614285714285715

epsilon= 0.5635714285714286

epsilon= 0.5657142857142857

epsilon= 0.5678571428571428

epsilon= 0.5700000000000001

epsilon= 0.5721428571428572

epsilon= 0.5742857142857143

epsilon= 0.5764285714285714

epsilon= 0.5785714285714285

epsilon= 0.5807142857142857

epsilon= 0.5828571428571429

epsilon= 0.585

Figure 4 For $d = 3$ and for different values of ϵ with worst behavior, we consider the optimal values of n for which our algorithm is efficient. Whenever the red function is below the blue function the preprocessing is meaningful.

4 Discussion

The main open question is finding an algorithm for the general version, as our approach cannot be extended to overlapping balls or balls of arbitrary size. We note that a $(1+\epsilon)$-approximation for the nearest neighbour query or the shortest path tree query is also solvable by a similar idea of using the WSPD in the preprocessing, in $\mathbb{R}^2$, as discussed in [2]. But, the input disks must be disjoint and unit. These restrictions are in principal because of the WSPD properties. Finding an approach that breaks this barrier generally, or for using the WSPD for the overlapping balls in the preprocessing is an interesting open problem.

Acknowledgement The authors would like to thank an anonymous reviewer for pointing out the exact algorithm for the problem in $\mathbb{R}^2$. V.K is Supported by the Czech Science Foundation, grant number GJ19-06792Y, and by institutional support RVO: 67985807.

References

1 Mohammad Ali Abam, Paz Carmi, Mohammad Farshi, and Michiel Smid. On the power of the semi-separated pair decomposition. *Comput. Geom.*, 46(6):631–639, 2013.

2 Kevin Buchin, Maarten Löffler, Pat Morin, and Wolfgang Mulzer. Delaunay triangulation of imprecise points simplified and extended. In *WADS*, pages 131–143. Springer, 2009.

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

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

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

6 Xi Guo, Xiaochun Yang, Danni Chen, and Changyu Chen. Diameter-aware extreme group queries. *IEEE Access*, 6:58687–58701, 2018.

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

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

9 Vahideh Keikha, Ali Mohades, and Mansoor Davoodi Monfared. On the triangulation of non-fat imprecise points. In *CCCG*, pages 114–121, 2016.

10 Amer 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 Maarten Löffler. *Data imprecision in computational geometry.* PhD thesis, University Utrecht, 2009.

12 Maarten Löffler and Jack Snoeyink. Delaunay triangulation of imprecise points in linear time after preprocessing. *Comput. Geom.*, 43(3):234–242, 2010.

13 Grégoire Malandain and Jean-Daniel Boissonnat. Computing the diameter of a point set. *Internat. J. Comput. Geom. Appl.*, 12(06):489–509, 2002.

14 Nimrod Megiddo. Linear-time algorithms for linear programming in Rˆ3 and related problems. *SIAM J. on computing*, 12(4):759–776, 1983.

15 Franco P Preparata and Michael Ian Shamos. Computational Geometry: an Introduction. pages 95–149. Springer, 1985.

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

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

18 M. I. Shamos. Computational geometry. Ph.D. Thesis. 1978.

19 Ivor van der Hoog, Irina Kostitsyna, Maarten Löffler, and Bettina Speckmann. Preprocessing ambiguous imprecise points. *arXiv preprint arXiv:1903.08280*, 2019.

20 Marc Van Kreveld, Maarten Löffler, and Joseph SB Mitchell. Preprocessing imprecise points and splitting triangulations. *SIAM J. Comput.*, 39(7):2990–3000, 2010.

Augmenting Graphs with Maximal Matchings

Maike Buchin[1], **Antonia Kalb**[2], **and Bernd Zey**[2]

1 **Faculty of Computer Science, Ruhr University Bochum, Germany**
`maike.buchin@rub.de`

2 **Faculty of Computer Science, TU Dortmund University, Germany**
`antonia.kalb@tu-dortmund.de, bernd.zey@tu-dortmund.de`

Abstract

We study the augmentation of planar straight-line geometric graphs with maximal compatible matchings. We show a lower bound of $\frac{n-4}{18}$ for the size of a maximal compatible matching of 3-regular geometric graphs with n vertices and give an example that closely approaches this bound.

1 Introduction

Problem Setup A graph is a pair $G = (V, E)$ of *vertices* V and *edges* $E \subseteq \{V \times V\} \setminus \{(v, v) \mid v \in V\}$ (*loops* are not allowed). We assume that every graph G is assigned a fixed drawing which gives the position of the vertices and edges in $\mathbb{R}^2$. A graph is drawn *straight-line* and *planar* if the edges are noncrossing and drawn as straight lines. If no three points of such a graph G are co-linear, we call G a *geometric* graph; throughout this work we solely consider geometric graphs.

A *matching* in a graph is an edge set that is disjoint to the edges of the graph and it is *compatible* if it preserves the planarity of its fixed straight-line drawing [1, 2, 8, 11]. The constraint on fixed drawings is common, as constraints on fixed positions are often considered in practice, for example in maps or networks depicting the real world. A matching is *maximal* if it cannot be expanded to a larger compatible matching and the size of the *smallest maximal compatible matching* of a graph G is denoted by $\mathrm{mm}(G) = \min\{|M| \mid M \text{ is a maximal } G\text{-compatible matching}\}$. For a graph class $\mathcal{G}$ we have $\mathrm{mm}(\mathcal{G}) = \min\{\mathrm{mm}(G) \mid G \in \mathcal{G}\}$. We are interested in a *tight lower bound* for $\mathrm{mm}(\mathcal{G})$, i.e., we search for one graph $G \in \mathcal{G}$ with $\mathrm{mm}(G) = \mathrm{mm}(\mathcal{G})$. Thereby, the matching size is always given in relation to the number of vertices.

Our work continues the work of [11], who determine lower bounds on the size of maximal compatible matchings for different graph classes, mainly focusing on 1- and 2-regular graphs; a graph $G = (V, E)$ is *k-regular* if every vertex $v \in V$ has degree $\deg(v) = k$. In this work, we deduce a lower bound for 3-regular graphs via graph properties and investigate its tightness. For this, Lemma 1 of [11] is very important to us. However, simply applying this lemma to 3-regular graphs does not give good bounds; instead, we evaluate the terms more carefully and show how they influence each other to obtain a strong lower bound for these graphs.

Basic Definitions We call a drawing a *deformation* of another drawing if the positions of the vertices differ, but not the faces with which they are incident (see Figure 1). A vertex v is a *reflex vertex* if there is an angle $> 180°$ between two incident edges.

The *augmentation* of a graph is the addition of vertices or edges; we consider only *edge augmentations*. Here, the graph $G + E' = (V, E \cup E')$ denotes the augmentation of the graph $G = (V, E)$ with the edge set E'. The notation $\deg_G(v)$ or $\deg_{G+E'}(v)$ is used to distinguish which degree the same vertex $v \in V$ has in the graph G or $G + E'$, respectively. Here, we augment fixed planar drawings of graphs and all inserted edges are straight-line as well.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

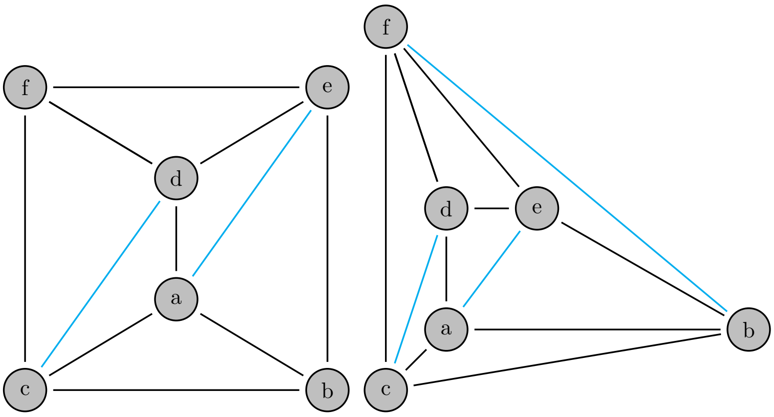


Figure 1 Different drawings can have different sized maximal compatible matchings (cyan)

Hence, the vertices in $G + E'$ have the same position in $\mathbb{R}^2$ as in G. An edge e is called *compatible with a graph G (is G-compatible)*, if $e \notin E(G)$ and e does not cross any edge in $G + \{e\}$. A matching M with respect to G is *maximal and compatible* if there is no pair of vertices u and v such that both vertices are unmatched and the edge (u, v) is compatible to $G + M$. A face of $G + M$ is called *fully (un-)matched* if every vertex incident to that face is (un-)matched.

The size $|M|$ of a G-compatible matching M always refers to its unique fixed drawing of the graph G. Figure 1 shows that the size depends on the exact position of the vertices in $\mathbb{R}^2$. Hence, even isomorphic graphs can have different sized maximal compatible matchings. For a graph class $\mathcal{G}$, we search for a lower bound over all graphs and all possible drawings, in $\mathcal{G}$.

Related Results There are various problems in which graphs are augmented with a set of edges. Some problems deal with augmenting edges such that certain graph properties are achieved, e.g. regularity or connectivity. This may additionally require that existing properties are preserved, e.g. planarity [3, 5, 6, 7, 10, 14]. Other augmentation problems impose requirements on the set of edges, e.g. it is a matching [1, 2, 8, 11]. A famous example is Christofides' algorithm where a perfect matching between the vertices of odd degree is determined [4]. As early as 1986, Rapport et al. [12, 13] studied the complexity of augmenting a disjoint edge set to a simple cycle, i.e., augmenting an 1-regular graph with a perfect compatible matching.

Lemma 1.1 [11] as well as general graph properties and properties of specific graph classes are used in [9, 11] to give guarantees for minimal maximal compatible matchings for different graph classes. These and other bounds are summarized in Table 1. Note that this paper is based on the master thesis by Kalb [9]; hence, results marked with [9] are new results.

Graph class $\mathcal{G}$	**Size of a maximal compatible matching M**
0-regular geometric graphs	$\mathrm{mm}(\mathcal{G}) = \frac{\lvert V\rvert-1}{3}$, [11, Theorem 2]
1-regular geometric graphs	$\mathrm{mm}(\mathcal{G}) = \frac{\lvert V\rvert-2}{6}$, [11, Theorem 2]
1-regular geometric graphs with $\lvert E\rvert \mod 2 = 0$	$\mathrm{mm}(\mathcal{G}) \geq \frac{2\lvert V\rvert-1}{5}$ [1, Theorem 14]
2-regular geometric graphs	$\mathrm{mm}(\mathcal{G} = \frac{\lvert V\rvert-3}{11}$, [11, Theorem 2]
connected 2-regular geometric graphs with $\lvert V\rvert \geq 4$	$\mathrm{mm}(\mathcal{G}) = \frac{\lvert V\rvert}{7}$ [11, Theorem 3]
3-regular geometric graphs	$\mathrm{mm}(\mathcal{G}) \geq \frac{\lvert V\rvert-4}{18}$ [9, Theorem 31]
4-regular geometric graphs	$\mathrm{mm}(\mathcal{G}) \geq \frac{\lvert V\rvert-6}{32}$ [9, Theorem 33]
outerplanar graphs	$\mathrm{mm}(\mathcal{G}) = 0$ [9, Theorem 26]
geometric graphs with $2\lvert V\rvert - 3 \leq \lvert E\rvert \leq 3\lvert V\rvert - 6$	$\mathrm{mm}(\mathcal{G}) = 0$ [9, 11]
geometric graphs with $\lvert E\rvert \leq \lvert V\rvert \cdot d$, $\frac{7}{10} < d < 2$	$\mathrm{mm}(\mathcal{G}) = \lvert V\rvert \cdot \frac{2-d}{13}$ [11, Lemma 2]
geometric graphs with bounded degree $d < 4$	same bound as d-regular graphs with $d < 4$ [9, Theorem 36]
geometric graphs with bounded degree $d \geq 4$	$\mathrm{mm}(\mathcal{G}) = 0$ [9, Theorem 36]

Table 1 Overview of lower bounds on the size of maximal compatible matchings

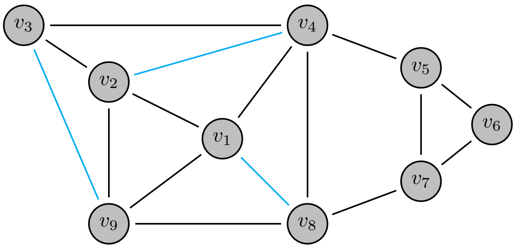


Figure 2 Geometric graph G and maximal G-compatible matching (cyan) with $\nu_{GM} = 2$, $\sigma_{GM} = 6$, $r^u_{GM} = 3$, $r^m_{GM} = 4$, $\sum_{u \text{ matched}} \deg_G(u) = 18$ and $\Delta_{GM} = 1$

▶ Lemma 1.1. [11, Lemma 1] Let $G = (V, E)$ be a geometric graph and M be a maximal G-compatible matching. The matching size can bounded using the following parameters:

- ν_{GM}: $\#(u, v) \in E$ where u unmatched, v matched, and u or v is reflex in $G + M$
- σ_{GM}: # fully matched faces of $G + M$
- r^u_{GM}: # unmatched reflex vertices in $G + M$
- r^m_{GM}: # matched reflex vertices in $G + M$
- Δ_{GM}: # fully unmatched triangular faces of G

It holds that

$$2|V| + \nu_{GM} + 2 \cdot \sigma_{GM} - r^u_{GM} - 2 \cdot r^m_{GM} - \sum_{u \text{ matched}} \deg_G(u) - \Delta_{GM} - 2 \;\leq\; 2|M|.$$

Figure 2 gives an example of the values used in Lemma 1.1.

2 Lower Bound for 3-regular Graphs

▶ **Lemma 2.1.** *The K_4, the complete graph on four vertices, is the only 3-regular geometric graph that does not allow for a compatible matching.*

Proof. (*Sketch*) First, the K_4 cannot be augmented since it is already complete. Second, a graph has no compatible matching, if and only if it has a convex outer face and only triangular inner faces. For a 3-regular graph it holds $h = \frac{3}{2}n - 3$ for the number of outer edges (for details see [9, Lemma 32]). A 3-regular graph cannot be outerplanar, thus we have the restriction $h < n$; combined with $h = \frac{3}{2}n - 3$ this leads to $n < 6$. Because $n \mod 2 = 0$ and $n \geq 4$ for 3-regular graphs, the statement follows. ◀

▶ **Theorem 2.2.** *Let G be a 3-regular geometric graph. It holds* $\mathrm{mm}(G) \geq \frac{n-4}{18}$ *for the minimal size of a maximal G-compatible matching.*

Proof. Let G be a 3-regular geometric graph, then the following properties hold for the values of Lemma 1.1. First, some values can be trivially bounded from below by 0, e.g., $\nu_{GM} \geq 0$ and $\sigma_{GM} \geq 0$ always hold. For the number of (un-)matched reflex vertices in $G + M$, we have $r^u_{GM} \leq n - 2|M|$ and $r^m_{GM} \leq 2\,|M|$, because it is possible that each of the $n - 2\,|M|$ unmatched and $2\,|M|$ matched vertices is a reflex vertex (see Figure 3). Moreover, we have $\sum_{u \text{ matched}} \deg_G(u) = 3 \cdot 2|M|$, because $\deg_G(v) = 3$ for all $v \in V$ and there are $2|M|$ matched vertices. Finally, for the number of fully unmatched triangular faces of $G + M$, $\Delta_{GM} \leq \frac{n-2|M|}{4} \cdot 3$ holds, because every four of the $n - 2|M|$ unmatched vertices can be a

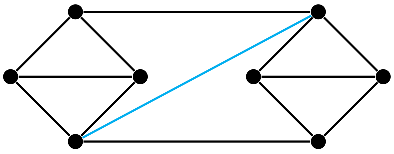


Figure 3 3-regular graph with maximal compatible Matching (cyan) where each vertex is reflex

K_4, and a K_4 is the only 3-regular graph with only triangular inner faces, cf. proof sketch of Lemma 2.1. By inserting these values into Lemma 1.1 we obtain the bound

$$\underbrace{2n}_{=2|V|} + \underbrace{0}_{\leq \nu_{GM}} + \underbrace{2 \cdot 0}_{\leq 2\sigma_{GM}} - \underbrace{(n-2|M|)}_{\geq r^u_{GM}} - \underbrace{2 \cdot 2|M|}_{\geq 2r^m_{GM}} - \underbrace{6|M|}_{=\sum_{\text{matched}} \deg_G} - \underbrace{\frac{n-2|M|}{4} \cdot 3}_{\geq \Delta_{GM}} - 2 \leq 2|M|$$

$$\iff \frac{n-8}{34} \leq \mathrm{mm}(G).$$

This bound can be improved further, since for 3-regular graphs the following correlation of the individual values can be observed: The number Δ_{GM} of fully unmatched triangular faces is maximized by K_4, but the vertex within the K_4 is always not a reflex vertex. Moreover, the number r^u_{GM} of unmatched reflex vertices is thus reduced by one per each K_4. Since we are upper bounding subtrahenders for a lower bound, the following holds

$$r^u_{GM} + \Delta_{GM} \leq \underbrace{n - 2|M| - \frac{n-2|M|}{4}}_{\geq r^u_{GM}} + \underbrace{\frac{n-2|M|}{4} \cdot 3}_{\geq \Delta_{GM}} = n - 2|M| + \frac{n-2|M|}{4} \cdot 2.$$

With this addition the bound for 3-regular geometric graphs increases to

$$\underbrace{2n}_{=2|V|} + \underbrace{0}_{\leq \nu_{GM}} + 2 \cdot \underbrace{0}_{\leq \sigma_{GM}} - 2 \cdot \underbrace{2|M|}_{\geq r^m_{GM}} - \underbrace{6|M|}_{=\sum_{\text{matched}} \deg_G} - \underbrace{\left(n - 2|M| + \frac{n-2|M|}{2}\right)}_{\geq r^u_{GM} + \Delta_{GM}} - 2 \leq 2|M|$$

$$\iff \frac{n-4}{18} \leq \mathrm{mm}(G)$$ ◀

3 On the Tightness of the Bound of $\frac{n-4}{18}$

We use the following ideas to construct a graph with minimal maximal compatible matching. Thereby, we use C to denote a connected component with $\mathrm{mm}(C) = 0$, e.g. the K_4.

i) Extending a fully matched face with C increases the number of vertices of the graph $G + M + C$ without increasing the matching.

ii) A fully matched face of $G + M$ with n' incident vertices can be augmented by $\lfloor n'/2 \rfloor$ C-components without any new compatible edge. This is achieved by a "zigzag"-shaped face, i.e. a face bounded by two "interleaving" chains of alternating acute and reflex angles (see Figure 4). If done properly, the C-components can be placed such that they do not "see" each other (connecting the C-components by compatible edges is not possible).

iii) If the outer face of $G + M$ is convex, we again are able to insert C-components without allowing new compatible edges between these. In case the outer face is fully matched, no vertex of the C-components can be matched with any vertex of G.

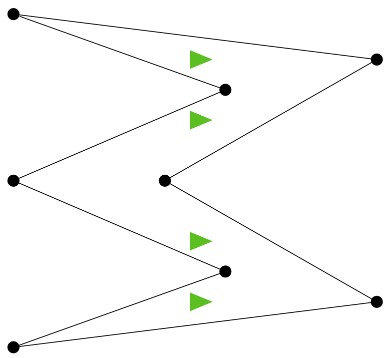


Figure 4 An inner “zigzag”-shaped face incident to 8 vertices that can be extended by $\lfloor 8/2 \rfloor$ components (green) with no compatible edges between them

iv) The number of fully matched inner faces and the number of matched vertices incident to the outer face can be maximized by augmenting a graph with a perfect compatible matching. For this, the number of vertices must be even.

Hence, the goal is iv) to augment a graph G with a perfect matching M such that iii) $G + M$ has the largest possible convex outer face and ii) such that $G + M$ consists of the largest possible number of inner faces with an even number of vertices.

Then, i) each inner face of $G + M$ as well as each outer edge is extended with copies of C. Notice that, for 3-regular graphs, K_4 is the only graph with $\mathrm{mm}(G) = 0$ (Lemma 2.1). Moreover, the bounds for 0-, 1- and 2-regular graphs proved by [11] are tight due to graphs that correspond to these ideas.

For a graph constructed using this approach, we can specify the relative matching size using the following formula:

▶ Formula 3.1. Let G be a graph with n vertices, M a perfect G-compatible matching, and C be a graph with $\mathrm{mm}(C) = 0$. If the augmented graph $G + M$ is deformed such that each inner face $f \in F_{\text{in}}$ with $|f|$ incident vertices is expanded by $\lfloor |f|/2 \rfloor$ copies of C, each of the h outer edges is expanded with one copy of C, and there exist no compatible edges between the C-copies, then the matching size is

$$\frac{n \cdot \overbrace{|V(G)|/2}^{\text{perfect matching}}}{\left(\underbrace{\sum_{f \in F_{\text{in}}} \left(\lfloor |f|/2 \rfloor\right)}_{\text{inner face expansions}} + \underbrace{h}_{\text{outer face expansions}}\right) \cdot \underbrace{|V(C)|}_{\text{vertices per expansion}} + \underbrace{|V(G)|}_{\text{vertices}}}$$

Now, we construct a graph that nearly achieves the bound of Theorem 2.2. The graph is constructed in several steps, see Figure 5: First, we construct a graph of 20 vertices, which is then augmented with a perfect matching. In the faces of the augmented graph, we place 30 K_4, and then double this construction. Finally, we “deform” each face of the graph into a “zigzag”-shape, such that the K_4’s placed inside it cannot be connected to each other.

Overall, this leads to a matching size

$$\frac{n \cdot \overbrace{10}^{\text{perfect matching per graph}} \cdot 2}{\Big(\underbrace{30}_{K_4 \text{ per graph}} \cdot 2 + \underbrace{14}_{K_4 \text{ between graphs}} + \underbrace{4}_{K_4 \text{ outer edges}}\Big) \cdot \underbrace{4}_{|V(K_4)|} + \underbrace{20}_{\text{vertices per graph}} \cdot 2} = \frac{n}{17.6}.$$

Next, we aim to expand the constructed graph such that the matching size comes arbitrarily close to the bound of Theorem 2.2.

Notice that, if the number of triangular faces is increased relative to the vertex count, the matching size decreases. However, the graph in Figure 5 can be expanded arbitrarily without increasing the number of triangular faces as shown by Figure 6. This leads to a larger number of quadrilaterals that can be deformed into the described “zigzag”-shapes. We can do so combinatorially, but we also need to show that the deformation of the graph in Figure 6b is always possible. We assume it works, because every face is triangular or quadrangular. Hence we just need to relocate one vertex per quadrangular inner face to make it concave (a concave quadrilateral is “zigzag”-shaped). Until now, we miss an algorithm for every number of vertices to deform our graph face by face without making already deformed faces convex again. If the deformation of a graph with γ extensions of twelve vertices is possible, the matching size decreases to

$$\frac{n \cdot \overbrace{(20 + 12 \cdot \gamma)/2}^{\text{perfect matching per graph}} \cdot 2}{\Big(\underbrace{(30 + 20 \cdot \gamma)}_{K_4 \text{ per graph}} \cdot 2 + \underbrace{14 + 8 \cdot \gamma}_{K_4 \text{ between graphs}} + \underbrace{4}_{K_4 \text{ outer edges}}\Big) \cdot 4 + \underbrace{(20 + 12 \cdot \gamma)}_{\text{vertices per graph}} \cdot 2} = \frac{n}{18 - \frac{2}{3\gamma+5}}.$$

Hence, for arbitrarily large γ this bound approaches $^{n}/_{18}$.

4 Conclusion and Future Research

In Theorem 2.2 we gave a lower bound of $\frac{n-4}{18}$ for the size of a maximal compatible matching in a 3-regular graph. The graph in Figure 5 with maximal compatible matching of size $\frac{n}{17.6}$ is the smallest we know so far. Assuming all inner faces can be “zigzag”-shaped, our construction of Figure 6 for large n approaches the bound arbitrarily close.

Note that, when considering graphs with a bounded degree $d \leq 3$, the obtained results for d-regular graphs can be transferred directly (see Table 1). For example, for graphs with maximum degree 3 we have the same $\mathrm{mm}(\mathcal{G})$-bound as for 3-regular graphs.

For 4-regular graphs, the proof of Theorem 2.2 and the construction ideas from Section 3 can be adapted. In [9] we prove the bound $\mathrm{mm}(G) \geq \frac{n-6}{32}$ for any 4-regular geometric graph G. Until now, smallest matching size we achieve is $\frac{n}{25^2/_3}$. For 5-regular graphs, however, our approach of augmenting with a perfect matching cannot be applied, since no planar k-regular graph with $k > 5$ exists.

In addition to regular graphs, we also consider other graph classes like outerplanar graphs. This class of graphs also includes the maximal outerplanar graphs, which always have a drawing to which no compatible matching exists. By restricting the maximum degree or the size of the edge set, this class would become more interesting. For future research, trees are another interesting graph class that has not yet been investigated.

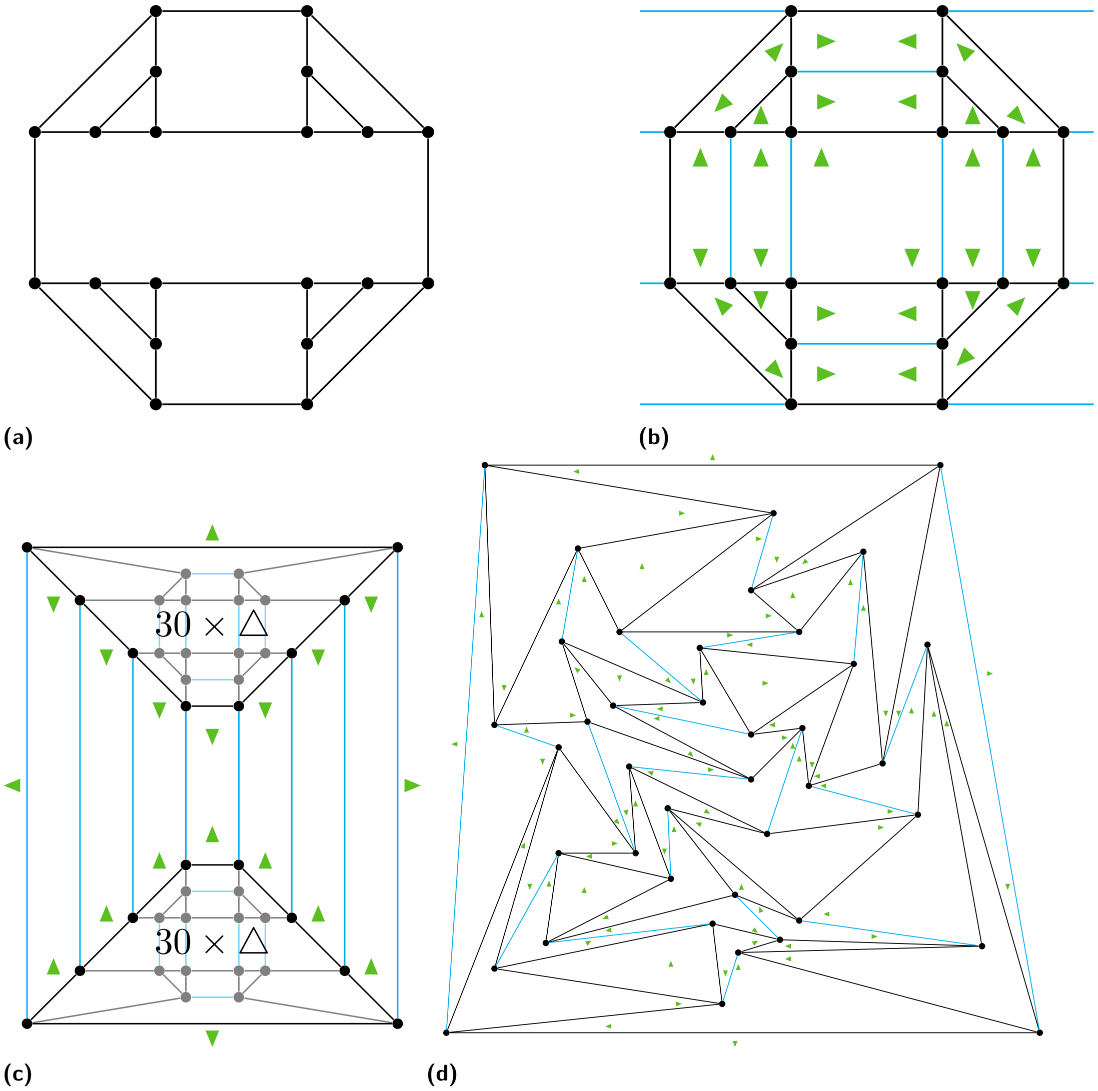


Figure 5 a) 3-regular graph with 20 vertices. b) The augmented graph (perfect compatible matching (cyan)), the inner faces are expanded by 30 K_4 (green). c) Augmented unified graph with two copies of the graph of b), 14 additional K_4 are added and 4 K_4 for the outer edges. d) Deformed graph with maximal compatible matching of size $\frac{n}{17.6}$.

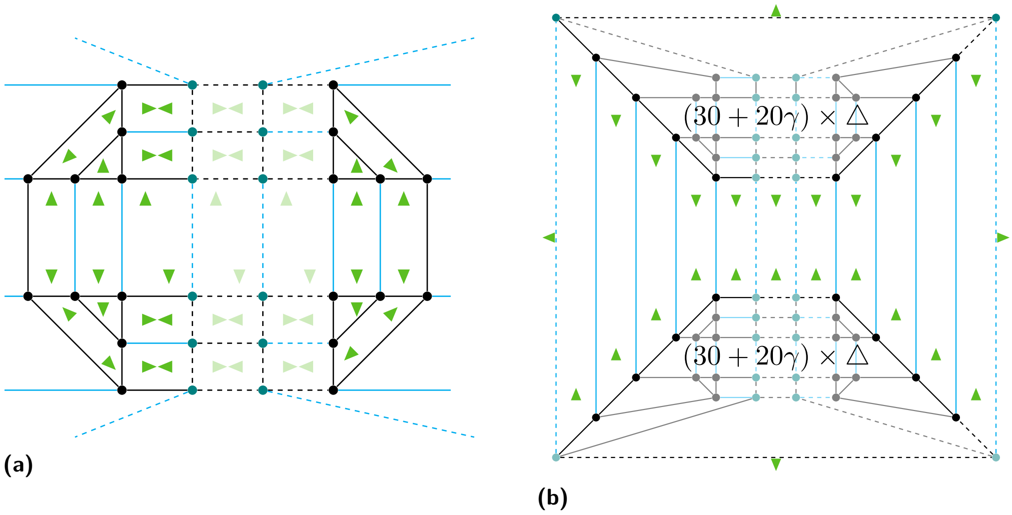


Figure 6 Construction of 3-regular graph with maximal compatible matching of size $\frac{n}{18-\frac{2}{3\gamma+5}}$. a) Graph of Figure 5b) is extended arbitrarily often by 12 vertices (teal, dashed edges), 20 K_4 are inserted per extension. b) Augmented unified graph, between two graph copies of Figure 6a). $14+8\gamma$ additional K_4 are added and 4 K_4 on the outer edges.

Acknowledgment We thank the reviewers for their helpful comments.

References

1 O. Aichholzer, S. Bereg, A. Dumitrescu, A. Olaverri, C. Huemer, F. Hurtado, M. Kano, A. Márquez, D. Rappaport, S. Smorodinsky, D. Souvaine, J. Urrutia, and D. Wood. Compatible geometric matchings. *Computational Geometry*, 42(6–7):617–626, 2009.

2 O. Aichholzer, A. Olaverri, F. Hurtado, and J. Tejel. Compatible matchings in geometric graphs. *XIV Spanish Meeting on Computational Geometry*, pages 27–30, 2011.

3 M. Al-Jubeh, G. Barequet, M. Ishaque, D. Souvaine, C. Tóth, and A. Winslow. Constrained tri-connected planar straight line graphs. In *Thirty Essays on Geometric Graph Theory*, pages 49–70. Springer, 2013.

4 N. Christofides. Worst-case analysis of a new heuristic for the travelling salesman problem. 1976.

5 T. Hartmann, J. Rollin, and I. Rutter. Regular augmentation of planar graphs. *Algorithmica*, 73(2):306–370, 2015.

6 M. Hoffmann and C. Tóth. Segment endpoint visibility graphs are hamiltonian. *Computational Geometry*, 26(1):47–68, 2003.

7 F. Hurtado and C. Tóth. *Plane Geometric Graph Augmentation: A Generic Perspective*, pages 327–354. Springer, 2013.

8 M. Ishaque, D. Souvaine, and C. Tóth. Disjoint compatible geometric matchings. *Discrete and Computational Geometry*, 49(1):89–131, 2013.

9 A. Kalb. Graph-Augmentierung mit maximalen Matchings, December 2021. Master thesis (in German), TU Dortmund University.

10 A. Mirzaian. Hamiltonian triangulations and circumscribing polygons of disjoint line segments. *Computational Geometry*, 2:15–30, 1992.

11 A. Pilz, J. Rollin, L. Schlipf, and A. Schulz. Augmenting geometric graphs with matchings. In *Graph Drawing and Network Visualization*, pages 490–504. Springer, 2020.

12 D. Rappaport. Computing simple circuits from a set of line segments is NP-complete. *SIAM Journal on Computing*, 18(6):1128–1139, 1989.

13 D. Rappaport, H. Imai, and G. Toussaint. On computing simple circuits on a set of line segments. In *Proceedings of the Second Annual Symposium on Computational Geometry*, SCG '86, pages 52—60. ACM, 1986.

14 D. Souvaine and C. Tóth. A vertex-face assignment for plane graphs. *Computational Geometry*, 42(5):388–394, 2009.

Small Area Drawings of Cactus-Graphs

Leonhard Löffler-Dauth[1]

1 **Department of Mathematics and Computer Science, Freie Universität Berlin**
l.loeffler@fu-berlin.de

Abstract

We introduce an algorithm for the straight-line drawing of any cactus-graph of size n in $O(n \log n)$ area. This is an improvement on the area bound described by Frati, Patrignani and Roselli [1] in 2020 for a subclass of outerplanar graphs that has no restriction on the degrees of its nodes.

1 Introduction

An area bound for the straight-line drawing of trees in $o(n \log n)$ has been found by Chan [2] in 2020. Di Battista and Frati [3] described a straight-line drawing for any outerplanar graph in an area containing $O(n^{1.48})$ grid points in 2009. In 2020 Frati, Patrignani and Roselli [1] improved this result by introducing an algorithm for the straight-line outerplanar drawing of any outerplanar graph in $O\left(n \cdot 2^{\sqrt{2 \log_2 n}} \sqrt{\log n}\right)$ area. An overview on the topic of graph drawings and a description of related open problems can be found in [4]. A *cactus-graph* is a connected graph in which two cycles share at most one common vertex. Every tree is a cactus-graph and every cactus-graph is an outerplanar graph. It is obvious, that the complexity of the area bound for cactus-graphs is in between the complexity of the area bounds for trees and outerplanar graphs. Frati [5] has shown in 2012 that restricting the degrees of the nodes in an outerplanar graph to be bounded by a parameter d yields a straight-line drawing in $O(dn \log n)$ area. We present an algorithm for the straight-line drawing of any cactus-graph of size n in $O(n \log n)$ area and thereby improve on previously found area bounds for a subclass of outerplanar graphs that has no restriction on the degrees of its nodes. In general, the produced drawings are not outerplanar and adjusting the algorithm to produce outerplanar drawings in $O(n \log n)$ area does not seem to be feasible.

2 Definitions and Notations

A *straight-line drawing* of a planar graph is an assignment of every vertex to a point in the plane and of every edge to the line segment connecting its endpoints. The *bounding box* of a drawing is the smallest rectangle with axis-parallel sides, such that the drawing is completely contained in that rectangle. A drawing, where every vertex is placed at a point on the integer grid is called a *grid-drawing*. We will only consider straight-line grid-drawings and thus, use *straight-line grid-drawing* and *drawing* interchangeably. Define the *area* of a grid-drawing to be the number of points on the integer grid contained in the bounding box of the drawing. The *width* and *height* of a grid-drawing are then defined to be the number of integer grid points contained in the horizontal and vertical side of its bounding box.

▶ **Definition 2.1.** Define a *rooted graph* to be a graph $G = (V, E)$ with a distinguished vertex called the *root*. The root provides a hierarchy similar to the parent-child hierarchy in a tree. For a rooted graph G with root r and a vertex $v \in G$ consider the subgraph induced by $V \setminus \{v\}$. Let $\mathcal{C}_r$ be the unique connected component of that induced subgraph that contains the root r. If $r \neq v$ we write $G(v)$ to denote the subgraph that is induced by the vertices

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

in $V \setminus V(\mathcal{C}_r)$. If $r = v$ we set $G(v) = G$. As a reference consider Figure 1. With $|G(v)|$ we denote the number of vertices in $G(v)$. Consider all neighbors of v in $G(v)$ and all vertices that lie on a common simple cycle with v in $G(v)$ and call the union of them the *extended neighborhood* of v. Denote the extended neighborhood of v with $S(v)$. Note, that $S(v)$ does not contain any vertices in $V \setminus V(G(v))$ by construction. Define $h(v)$ to be the "heaviest" element in $S(v)$, i.e. the vertex that maximizes the function $|G(\,\cdot\,)|$. Furthermore:

- With $v_1, \dots, v_s$ we denote the elements in the extended neighborhood of v that do not lie on a common cycle with v. Assume without loss of generality that v_s maximizes the function $|G(\,\cdot\,)|$ for all elements in $\{v_1, \dots, v_s\}$.
- We use the notation $C_1, \dots, C_t$ for the cycles in $G(v)$ that contain v. With $v_1^k, \dots, v_{j_k}^k$ we denote the vertices in $V(C_k) \setminus \{v\}$. Hereby, we assume without loss of generality that $v_m^t \in C_t$ maximizes the function $|G(\,\cdot\,)|$ for all vertices in $\bigcup_{k=1}^t V(C_k) \setminus \{v\}$.

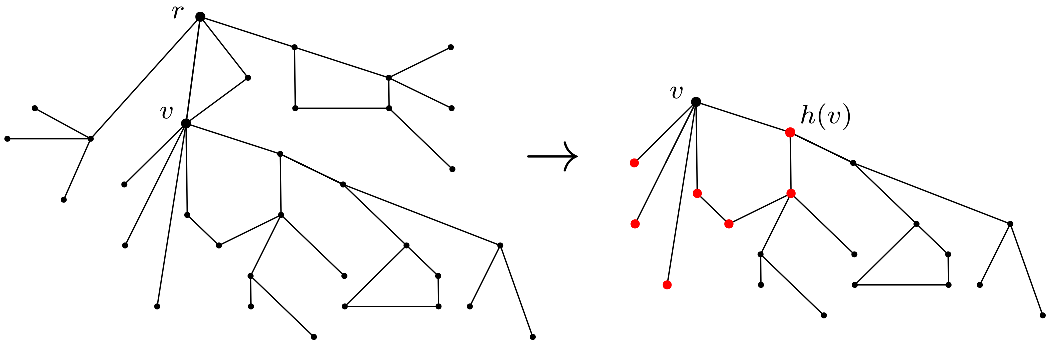


Figure 1 $G(v)$ and the extended neighborhood $S(v)$, that is represented by the red dots.

3 The Algorithm for the Drawing of Cactus-Graphs

▶ **Theorem 3.1.** *Let G be a cactus-graph of size $n \geq 2$ and root r. Then G has a straight-line drawing with area $3n\lceil \log_2 n \rceil$, where r is placed at the top left corner of the bounding box.*

Place r at the point $(1, -1)$. The drawing is obtained by recursion, where we denote a recursive step with $\operatorname{ext}(v)$ for a given vertex $v \in G$ that has previously been placed. Such a recursive step extends the drawing from the placement of v to the drawing of the vertices in $S(v)$ and in some cases to the drawing of additional vertices. The following invariants are maintained, when $\operatorname{ext}(v)$ is called. As a reference consider Figure 2 and Figure 3 and the explanation of the coloring of the Figures in the figure legend of Figure 2.

- v is the only vertex in $G(v)$ that has previously been placed. Furthermore, we have that $\{w \in V \mid v \in G(w)\}$ is a subset of the vertices that have been drawn in previous steps.
- v has been placed at the top left or the top right corner of a rectangle B with axis-parallel sides that has width $3\lceil \log_2 n_v \rceil$ and height n_v for $|G(v)| = n_v \geq 2$. Let D be the set of points that are contained in the drawing of the edges that have been placed in previous steps. The rectangle B has the property that $D \cap \operatorname{int}(\operatorname{conv}(B)) = \emptyset$, where conv denotes the convex hull. If $D \cap B$ contains only points with the same y-coordinate as v, i.e. points on the top side of B we say that v is in *good position.* If $D \cap B$ contains points on the left or right side of B other than the ones in the corners we say that v is in *bad-position.*

- Depending on the position of v ext(v) is performed by the *good-position algorithm* or the *bad-position algorithm* and places the vertex $h(v)$ at a point in good position.

3.1 The Good-Position Algorithm

Both algorithms that we use as subroutines distinguish between two cases, where in the first case $h(v)$ is a neighbor of v and in the second case $h(v)$ is not a neighbor of v. Call the following algorithm the *good-position algorithm.* As a reference consider Figure 2. Let v be a vertex in good position with the notation from Definition 2.1 for the elements in $S(v)$.

Case 1.1 v and $h(v)$ are neighbors and do not lie on a common cycle.

Stack the vertices in $S(v) \setminus \{h(v)\}$ at points on the vertical line through $v + (1, \cdot)$. The distance between the drawing of a vertex $w \in S(v)$ and the vertex that is placed below it has to be large enough to fit the vertices in $G(w)$ on grid points between them. This is necessary to avoid edge-crossings in the later described recursive steps. Place $h(v) = v_s$ at a point below v with a sufficiently large gap between them, i.e. such that the vertices in $G(v) \setminus G(h(v))$ fit on grid points between them.

Case 1.2 v and $h(v)$ are neighbors and lie on the common cycle C_t.

Stack the vertices in $S(v) \setminus V(C_t)$ vertically at points on the vertical line through $v + (1, \cdot)$. Stack the vertices in $V(C_t) \setminus \{v_m^t\}$ underneath the drawing of these vertices, such that v_1^t is drawn on the uppermost point and $v_{m-1}^1 = v_{j_t-1}^t$ is drawn on the lowermost point. Place $h(v) = v_m^t$ at a point below v with a sufficiently large gap between them.

Case 2. v and $h(v)$ are not neighbors. Note, that they lie on the common cycle C_t.

Since v and $h(v) = v_m^t$ are not neighbors we can deduce that there are at least two more vertices $v_{m-1}^t, v_{j_t}^t$ that are contained in C_t. We construct a drawing of C_t, such that its vertices lie on a quadrangle with height $n - n_m^t$ and width $3\lceil \log_2 n \rceil$, where $n_m^t = |G(v_m^t)|$.

1. Place $h(v) = v_m^t$ at the point $v + (3\lceil \log_2 n \rceil - 1, -(n - n_m^t - 1))$. Place v_{m-1}^t at the point $v + (0, -(n - n_m^t - 1))$. Place $v_{j_t}^t$ at the point $v + (3\lceil \log_2 n \rceil - 1, -1)$.
2. Stack the vertices $v_1^t, \dots, v_{m-2}^t$ on the vertical line segment between v and v_{m-1}^t. Stack the vertices $v_{m+1}^t, \dots, v_{j_t-1}^t$ on the vertical line segment between $v_{j_t}^t$ and $h(v)$.
3. Stack the vertices in $S(v) \setminus V(C_t)$ at points on the vertical line through $v + (1, -1)$. Stack the vertices in $S(v_{j_t}^t), \dots, S(v_{m+1}^t)$ at points on the vertical line through $h(v) + (-1, \cdot)$ below the vertices in $S(v) \setminus V(C_t)$. Stack the vertices in $S(v_1^t), \dots, S(v_{m-1}^t)$ at points on the vertical line through $v + (1, \cdot)$ below the vertices in $S(v_{m+1}^t)$.

In Case 1.1 and Case 1.2 take the vertices in $S(v)$ as the roots of the subgraphs at them for the next recursive steps. In Case 2 take the vertices that were drawn inside the quadrangle and $h(v)$ as the roots for the next recursive steps. Mirror the drawing in the next step horizontally, if a vertex that was placed on the right side of the quadrangle is taken as a root.

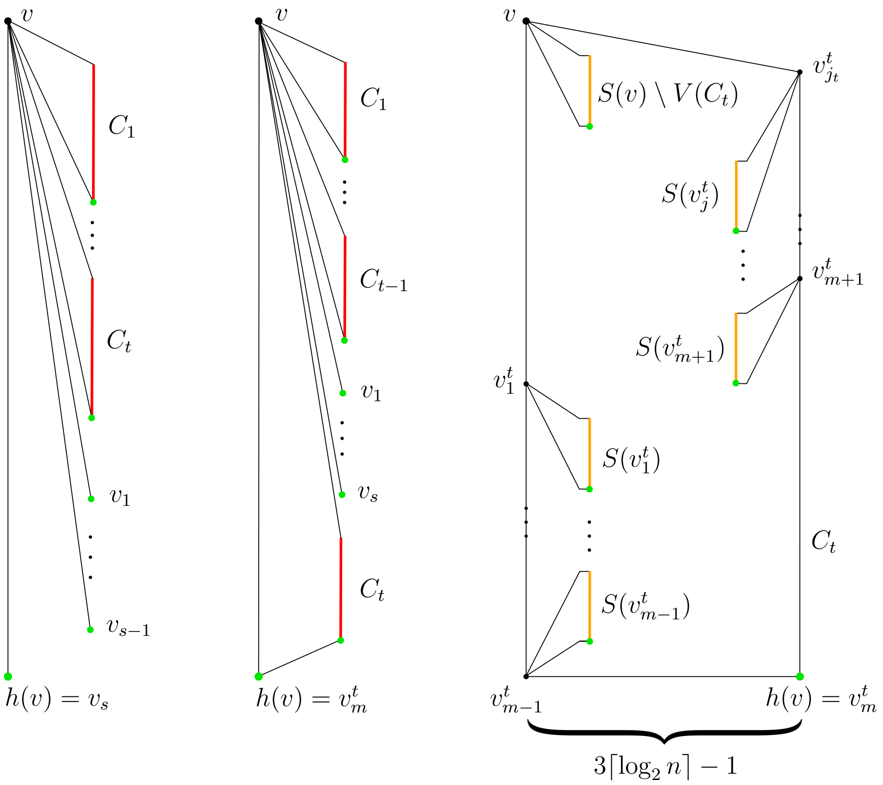


Figure 2 The good-position algorithm (from left to right Case 1.1, Case 1.2 and Case 2). The green dots represent the vertices that are taken as the roots for the following recursive steps that are in good position. The red line segments represent the areas on which the vertices in bad position were placed. The orange line segments can represent the drawing of multiple cycles at a vertex v_k^t and its children. Thus, vertices that were drawn on the orange line segment are not necessarily in good or necessarily in bad position.

3.2 The Bad-Position Algorithm

Call the following algorithm the *bad-position algorithm.* As a reference consider Figure 3. Let v be a vertex in bad position with the notation from Definition 2.1 for the elements in $S(v)$. This algorithm does not draw a vertex in $S(v)$ at a point with the same x-coordinate as its "parent" v. In particular, it places the element $h(v)$ at a point in good position for the next recursive step.

Case 1. v is a neighbor of $h(v)$.

Stack the vertices in $S(v)$ at points on the vertical line through $v + (1, \cdot)$.

Case 2. v is not a neighbor of $h(v)$.

Since v and v_m^t are not neighbors there are at least two other vertices $v_{m-1}^t, v_{j_t}^t \in C_t$.

1. Place v_m^t at the point $v + (3\lceil \log_2 n_v \rceil - 1, -(n_v - n_m^t - 1))$, where $n_v = |G(v)|$ and $n_m^t = |G(v_m^t)|$. Place $v_{j_t}^t$ at the point $v + (3\lceil \log_2 n_v \rceil - 1, -1)$. Place v_{m-1}^t at the point $v + (1, -(n_v - n_m^t - 1))$.
2. Stack the vertices $v_1^t, \ldots, v_{m-2}^t$ on the vertical line through $v + (1, 0)$ and v_{m-1}^t. Stack the vertices $v_{m+1}^t, \ldots, v_{j_t-1}^t$ on the vertical line segment between $v_{j_t}^t$ and v_m^t.

3. Stack the vertices in $S(v)\setminus V(C_t)$ at points on the vertical line through $v+(1,-1)$. Stack the vertices in $S(v^t_{j_t}),\dots,S(v^t_{m+1})$ at points on the vertical line through $v^t_m+(-1,\cdot)$ below the vertices in $S(v)\setminus V(C_t)$. Stack the vertices in $S(v^t_1),\dots,S(v^t_{m-1})$ at points on the vertical line through $v+(2,\cdot)$ below the vertices in $S(v^t_{m+1})$.

In Case 1 take the vertices in $S(v)$ as the roots of the subgraphs at them for the next recursive steps. In Case 2 take the vertices that were drawn inside C_t and v^t_m as the roots for the next recursive steps. Mirror the drawing in the next step horizontally, if a vertex that was placed on the right side of the drawing of C_t is taken as a root.

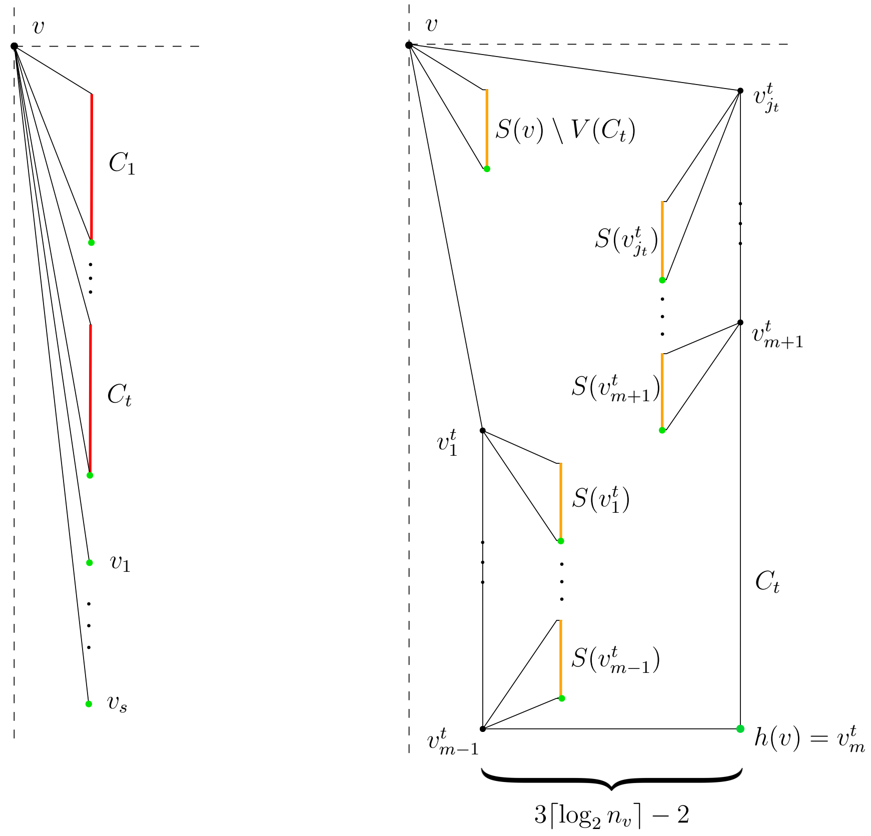


Figure 3 The bad-position algorithm (from left to right Case 1 and Case 2).

The drawing of the entire graph G can be obtained by applying either the good-position algorithm or the bad-position algorithm in each recursive step.

4 Absence of Egde-Crossings and Size of the Drawing

We constructed the gaps on the y-axis between the drawing of the vertices of G large enough to not cause an edge-crossing at the top- or bottommost side of the drawing of distinct subgraphs. The application of Case 1.1 or Case 1.2 of the good-position algorithm or Case 1 of the bad-position algorithm can not cause edge-crossings with the previously drawn subgraph. Consider a subgraph $G(v)$ of size n_v at a vertex $v\in G$, where $S(v)$ was drawn with Case 2 of the good-position algorithm, or Case 2 of the bad-position algorithm. Define the polygon at v to be the polygon consisting of the drawing of the edges of C_t. Let $w\neq v^t_m$

be a vertex that was drawn inside C_t in the recursive step in which v was the root. Assume without loss of generality, that w was placed at a point on the left side of the polygon at v and that v was placed at its right side. As a reference consider Figure 4, where an example with such a configuration of vertices is depicted. Let x_v denote the x-coordinate of v. The polygon at v has its right vertical side at the x-coordinate $x_v - 1$ and its left vertical side at the x-coordinate $x_v - (3\lceil \log_2 n_v \rceil - 1)$, if the polygon was drawn with the bad-position algorithm. If it was drawn with the good-position algorithm it has its right vertical side at the x-coordinate x_v. In either case we have that the x-coordinate of w is given by $x_w = x_v - (3\lceil \log_2 n_v \rceil - 1) + 1$. We want to show that the polygon at w, i.e. the smaller polygon that was drawn in the recursive step in which w was the root does not intersect at its vertical sides with the larger polygon at v. The previous observation implies that the distance between w and the right vertical side of the polygon at v is at least

$$(x_v - 1) - x_w = (x_v - 1) - (x_v - (3\lceil \log_2 n_v \rceil - 1) + 1) = 3\lceil \log_2 n_v \rceil - 3.$$

An application of the good-position algorithm or an application of the bad-position algorithm places any vertex in $G(w)$ at a point with x-coordinate of at most $x_w + 3\lceil \log_2 n_w \rceil - 1$, where $n_w = |G(w)|$. By construction we have $n_w < n_v/2$, since $w \neq v_m^t$. Thus, in this step any point in $G(w)$ is placed at a point with x-coordinate at most

$$x_w + 3\lceil \log_2 n_v/2 \rceil - 1 = x_w + 3\lceil \log_2 n_v \rceil - 4.$$

The difference of x_w and the drawing of any point in $G(w)$ in this step is less than the difference of x_w and the x-coordinate of the right side of the larger polygon at v. Thus, we do not obtain an edge-crossing with the edges that were drawn at the polygon at v and the edges that were drawn at the polygon at w.

We obtain the total height of at most n for the drawing of G, since the vertices are stacked tightly. What is left to be shown, is that we obtain the total width of $3\lceil \log_2 n \rceil$. Drawing the subgraph $G(h(r))$ at $h(r)$ does not exceed the width bound of $3\lceil \log_2 n \rceil$ as can be seen by applying a simple induction argument. Let $v \in G$, such that $|G(v)| > 1$. Consider any vertex $w \neq h(v)$ that is drawn in the recursive step in which v is the root and that is taken as the root for a next step. In each recursive step we may augment the number of vertical lines in the plane that contain drawings of vertices in G. If the vertices in such a step are drawn inside a polygon that has been drawn in a previous step we do not increase the total width of the drawing of G but we may augment the number of vertical lines that contain drawings of vertices. In Case 1.1 and Case 1.2 of the good-position algorithm and in Case 1 of the bad-position algorithm w is placed at a point with x-coordinate $x_v \pm 1$. In this step we therefore obtain an increase of at most 1 to the number of vertical lines that contain vertices. In Case 2 of the bad-position algorithm w is placed at a point with x-coordinate in $\{x_v \pm 1, x_v \pm 2, x_v \pm (3\lceil \log_2 n_v \rceil - 2)\}$. An example is depicted in Figure 4. Previously, we have shown, that the drawing of the edges in $G(w)$ can not intersect with the vertical sides of the polygon at v. In this step, we therefore obtain an increase of at most 3 to the number of vertical lines that contain vertices. In Case 2 of the good-position algorithm w is placed at a point with x-coordinate in $\{x_v \pm 1, x_v \pm (3\lceil \log_2 n_v \rceil - 2)\}$. With the same argument that was used for Case 2 of the bad-position algorithm, we obtain an increase of at most 2 to the number of vertical lines that contain vertices.

In summary, we have that an application of each case yields an increase of at most 3 to the number of vertical lines that contain vertices. Since $w \neq h(v)$, in each such recursive step we halve the number of vertices that are left to be drawn. Thus, we obtain the width bound of $3\lceil \log_2 n \rceil$ for the recursive drawing of G and thereby its total area bound of $3n\lceil \log_2 n \rceil$.

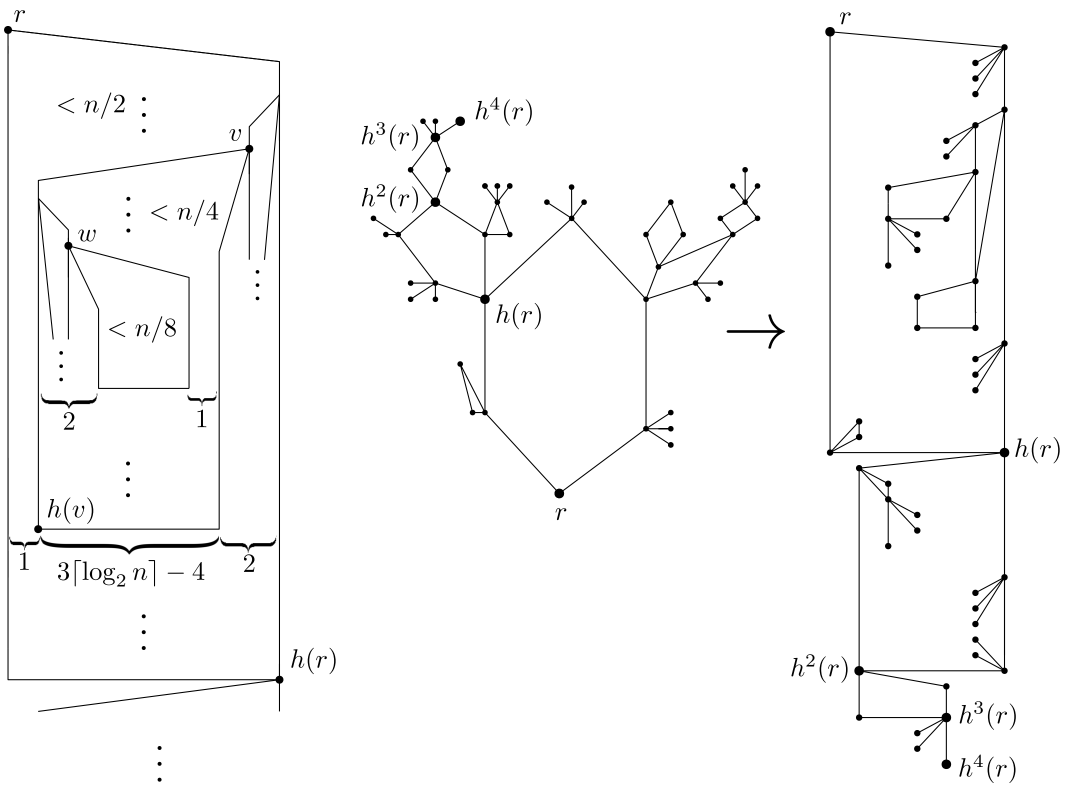


Figure 4 On the left side: an example for the recursive drawing and absence of edge-crossings. In the middle and on the right side: an example of the drawing of a cactus-graph.

5 Open Problems

It seems to be feasible to find a drawing for any cactus-graph in $o(n \log n)$. This drawing could possibly be constructed by employing a similar strategy as described in the algorithm by Chan [2] for the drawing of trees in $o(n \log n)$. The algorithm for the drawing of cactus-graphs in $O(n \log n)$ that is described in this paper could be used as a subroutine similar to the use of the standard algorithm [2],[6] for trees in Chans algorithm. Considering the intricate algorithm for the drawing of trees by Chan that takes many steps however, the attempt to find such a drawing with that strategy would require a very elaborate case analysis.

Acknowledgments. I want to thank both Klaus Kriegel for his support and for introducing this topic to me and the reviewers for their valuable hints concerning my first submission.

References

1 Frati, Fabrizio, Patrignani, Maurizio, Roselli, Vincenzo, *LR-drawings of ordered rooted binary trees and near-linear area drawings of outerplanar graphs*, Journal of Computer and System Sciences 107, 28-53, 2020.

2 Chan, Timothy M., *Tree Drawings Revisited*, Discrete & Computational Geometry 63, 799–820, 2020.

3 Di Battista, Giuseppe and Frati, Fabrizio, *Small Area Drawings of Outerplanar Graphs*, Algorithmica 54, 25–53, 2009.

4 Di Battista, Giuseppe and Frati, Fabrizio, *A Survey on Small-Area Planar Graph Drawing*, University of Rome/ University of Sydney, 2014.

5 Frati, Fabrizio, *Straight-line drawings of outerplanar graphs in $O(dn \log n)$ area*, Computational Geometry: Theory and Applications 45, 524–533, 2012.

6 Shiloach, Yossi, *Linear and Planar Arrangement of Graphs.* PhD thesis, page 94, Weizmann Institute of Science, 1976.

Towards the Minimization of Global Measures of Congestion Potential for Moving Points*

Will Evans[1], Ivor v.d. Hoog[2], David Kirkpatrick[1], Maarten Löffler[3]

1 **University of British Columbia**
`will@cs.ubc.ca` | `kirk@cs.ubc.ca`
2 **Technical University of Denmark**
`vanderhoog@gmail.com`
3 **Utrecht University**
`m.loffler@uu.nl`

Abstract

Imagine a collection of entities moving unpredictably at some bounded speed. Information about entity locations can be maintained by queries, at most one per unit of time, between which location uncertainty grows. Our goal is to minimize global measures of potential congestion defined in terms of the intersection graph of the entities' associated uncertainty regions. As a step towards this larger goal, we study the problem of minimizing the total degree (i.e., twice the number of edges) in the graph when entities are *static* points in $\mathbb{R}^1$.

1 Introduction

Data in motion is increasingly becoming a topic of interest to researchers in fields such as GIS, sensor networks, social networks, etc. [2, 9, 20, 23]. Although these applications are very different, they share the property that the motion of entities is largely *unpredictable*. In computational geometry there is an extensive history on data in *motion*: kinetic data models [1, 4, 16, 15] can be used to study the complexity and structural changes of geometric objects in motion. However, they were developed for moving data in a controlled environment and rely on the possibility to predict, at least locally, the trajectory of a moving point. We follow recent efforts that try to deal with less restricted models of motion as we assume the motion of each entity to be unpredictable but to have an upper bound on speed [7, 8, 10, 14, 19, 22]. Following [6, 13] we assume that in this setting we can query one entity per unit time for their exact location. A sequence of queries results in a set of possible locations for each entity, which is a ball (called its *uncertainty region*) centred at the entity's last observed location, with diameter proportional to the time since the corresponding query.

Our goal is to devise query strategies to maintain *potential congestion* of the entities as low as possible. We measure this potential congestion by how much the entities' uncertainty regions intersect. For example, this could be the maximum degree of their intersection graph, or their *ply* [18] (the maximum number of regions that contain any given point in $\mathbb{R}^d$). In many applications involving imprecise numbers or points, low congestion implies more efficient algorithms [5, 17, 21].

* This work has been partially supported by the Dutch Science Foundation (NWO) under grant number 614.001.504, and by NSERC of Canada.

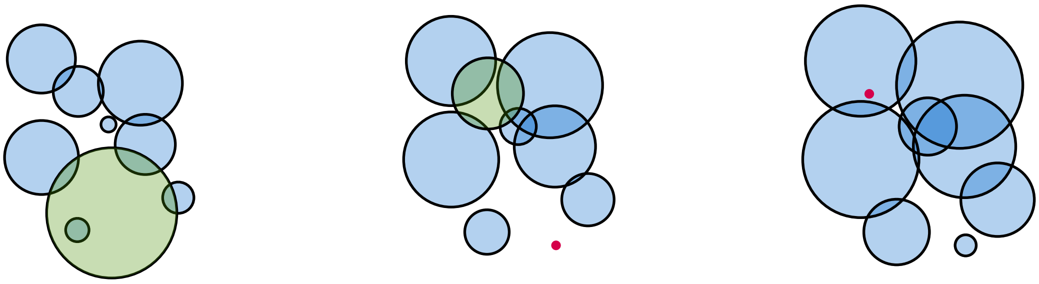


Figure 1 The model at three successive time steps. In each step, one disk (green) is queried, result in a point (red) at the next time step. All other disks grow in diameter by 1.

For any configuration of entities and any congestion measure (e.g. the max degree of their associated intersection graph), the *intrinsic congestion* at any fixed time t^* is the minimum over all query strategies of the congestion at time t^*. Evans *et al.* [13] show a query strategy that (up to a constant factor) realises the intrinsic ply (or max degree) at any specified target time t^*. In this sense it is *competitively optimal* at time t^*. Later, Busto *et al.* [6] showed that no single strategy can be competitively optimal in this strong sense on a continuous basis. Instead they formulate a query strategy Γ that maintains low ply (or max degree) and is competitive in every modest-sized time window T in the following weaker sense: if the maximum congestion realized by Γ over T is τ^* then, for any other (even clairvoyant) query strategy Γ', there is at least one time $t^* \in T$ for which the congestion realized by Γ' is $\Omega(\tau^*)$. The result of Busto *et al.* [6] is based on a strategy that first assumes a target congestion τ. This basic strategy then either realises a congestion of τ, or adapts the target value as entities move. Surprisingly, much of the intuition behind these competitive strategies was derived from an initial study of the case where entities are static (because even this restricted case captures an important component of the problem of maintaining low potential congestion).

The ply and the max degree of the intersection graph are in a sense local measures: they are values that are realised by a local configuration of the uncertainty regions. Global congestion measures have been recently considered as more robust and fine-grained alternatives to local congestion measures [21]. Optimizing for global measures can require different, non-local, query strategies. Indeed, even when entities are static, minimizing max degree can fail to provide a reasonable solution to the problem of minimizing the sum of all degrees (Theorem 1.1), which we call the *total degree.* We study how to construct a query strategy that maintains a target *global* measure (the total degree of the intersection graph). We show, in the restricted setting involving static entities, a query strategy that is competitively optimal in the weaker sense (Theorem 2.2).

Model of computation and problem statement. We study a model for analysis of unpredictable data that was previously explored by Evans *et al.* [11, 12, 13] and Busto *et al.* [6]: let $\mathcal{E}$ be a set $\{e_1, e_2, \ldots, e_n\}$ of point entities in $\mathbb{R}^d$. Every entity e_i has an associated (unknown) *trajectory* $f_i : \mathbb{R} \to \mathbb{R}^d$, which is a continuous function from time to position. Each entity e_i moves along f_i with maximum speed of 1/2 per unit of time (i.e. $\forall t$, $\|f_i(t) - f_i(t+1)\| \leq 1/2$). We are allowed to *query* an entity for its current location, which takes unit time[1]. This query returns, for a time t and entity e_i, the entity's true location $f_i(t)$.

[1] Setting the speed and sampling rate does not restrict the problem's generality. We chose 1/2 for ease.

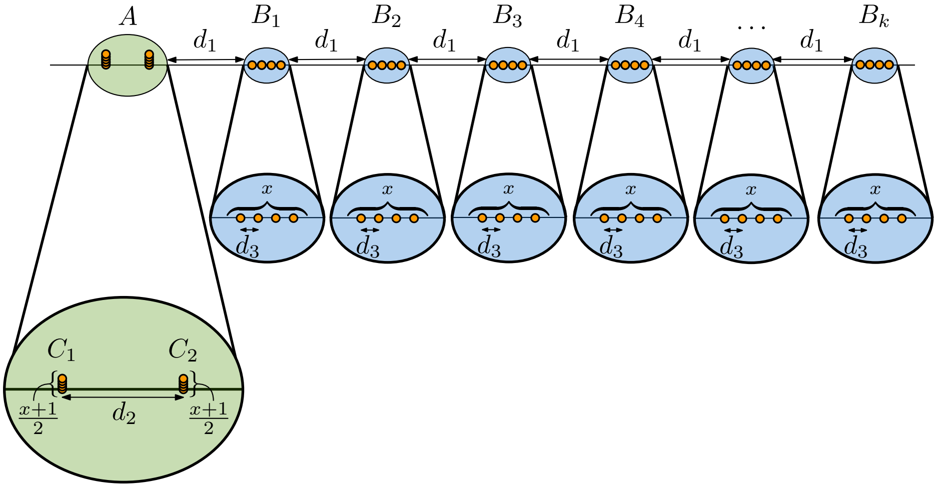


Figure 2 An example of a point set for which no good query strategy can maintain the maximum degree, but for the total degree this is possible. Essentially, the construction consists of two parts: part A with a single cluster of $x+1$ points, and part B with many clusters of x points.

A *query strategy* is a sequence $\Gamma = (\gamma(1), \gamma(2), \ldots)$ for some function $\gamma : \mathbb{N} \to \{1, \ldots, n\}$. That is, Γ is a sequence of indices where at time t we query entity $e_{\gamma(t)}$ for its location. Given a query strategy Γ, the possible locations of e_i at time t form a ball $B_i^\Gamma(t)$ with center $f_i(t_i)$ (the location of e_i at its *most recent query time*) and diameter $B_i^\Gamma(t) = t - t_i$. We refer to $B_i^\Gamma(t)$ as an *uncertainty region* and denote by $\mathcal{B}^\Gamma(t)$ the set $\{B_i^\Gamma(t) \mid e_i \in \mathcal{E}\}$. (When Γ is clear from context, we drop the superscript.) Figure 1 illustrates the model in $\mathbb{R}^2$.

Henceforth, we focus on the following problem: given $\mathcal{E}$, devise a query strategy Γ that continuously minimizes the total degree of the intersection graph of $\mathcal{B}^\Gamma(t)$. To make progress towards this goal, we first study this algorithmic problem in a more restricted setting:

- We assume the entities reside in $\mathbb{R}^1$.
- We assume that the entities are *static* (i.e, $f_i(\cdot)$ is constant, for all $i \in [n]$).

We are now ready to state our first result, which motivates our focus on global congestion measures (specifically the total degree of the intersection graph of the uncertainty regions):

▶ **Theorem 1.1.** *There exists a static point set in $\mathbb{R}^1$ with the property that*

(i) any strategy with query frequency 1 that maintains max degree less than x must experience total degree more than $(n-x-1)(x-1)$; and
(ii) there is a strategy with query frequency 1 that maintains total degree at most $(x+1)x$.

Proof. We construct a set of $n = (k+1)x+1$ points, for an odd positive integer x and a positive integer k. The construction is illustrated in Figure 2. Specifically, it consists of $k+1$ primary clusters $A, B_1, \ldots, B_k$ separated by distance d_1. The first of these clusters (A) is made up of two sub-clusters (C_1 and C_2), separated by distance d_2, each of which consists of $\frac{x+1}{2}$ co-located points (that is, for each sub-cluster, the points have the same x-coordinate). The remaining primary clusters B_i are all identical copies of a collection of x points separated by distance d_3. We set the values of d_1, d_2, d_3 during the remainder of the proof.

For (i), observe that the points in cluster A must each be queried with frequency at least $1/(2d_2)$ (recall that the speed of a point is $1/2$) to avoid degree x. If there is less than

$\frac{1}{2d_3(x-1)}$ additional frequency available then, at some time, all other points, i.e., those in the B_i clusters, must have uncertainty regions of size at least $d_3(x-1)$, and hence uncertainty degree at least $x-1$, making the total degree, not even counting points in the A cluster, at least $(n-x-1)(x-1)$.

For (ii), observe that if all of the B_i cluster points are queried with frequency at least $\frac{1}{d_3}$ then their uncertainty regions are disjoint, so they contribute nothing to the total degree. This requires total frequency at least $\frac{n-x-1}{d_3}$, so we need $d_3 \geq n-x-1$. If we have even a little excess frequency, it can be used to keep the uncertainty regions of points in the A cluster from intersecting those of the points in the B clusters, provided we choose d_1 large enough. Hence total degree at most $(x+1)x$ (associated with the points in the A cluster) can be maintained. This can be realized by choosing d_1, d_2, and d_3 as powers of 2 with $1 - \frac{x+1}{2d_2} < \frac{1}{2d_3(x-1)}$ and $\frac{n-x-1}{d_3} + \frac{x+1}{d_1} \leq 1$. ◀

The next section describes our second and main result (Theorem 2.2): in the restricted setting described above, we obtain a query strategy Γ that maintains a total degree of at most some τ^*, where any other strategy must realize total degree $\Omega(\tau^*)$ at some point in every time interval of length at least τ^*.

2 Minimizing the Total Degree for Static Entities in $\mathbb{R}^1$

We describe a query strategy that takes as input a collection of entities (points) $\mathcal{E}$, located in $\mathbb{R}^1$, and a target value τ for our total degree congestion measure. To formulate our strategy, we find it helpful to locate entities of $\mathcal{E}$ within a hierarchical, quadtree-like decomposition of $\mathbb{R}^1$ into cells (half-open intervals), where each cell C has two equal-sized children that partition C. Formally we create *levels* l for all $l \in \mathbb{Z}$. For each level we partition $\mathbb{R}^1$ into cells (i.e. half-open intervals in $\mathbb{R}^1$):

- a cell C_i (for $i \in \mathbb{Z}$) at level $l \in \mathbb{Z}$ has the form $C_i = [i \cdot 2^l, (i+1) \cdot 2^l)$.
- a cell C_i at level l has neighbors $[(i-1) \cdot 2^l, i \cdot 2^l)$ and $[(i+1) \cdot 2^l, (i+2) \cdot 2^l)$.
- a cell C_i at level l has two children at level $l-1$ that partition C_i.

We denote by $\mathcal{C}_l$ the set of all cells at level l. For any cell C we denote by $\pi(C)$ the *population* of C; that is, the number of points in $\mathcal{E}$ that are contained in C.

Active, critical and fringe cells. We say that a cell $C \in \mathcal{C}_l$ is τ*-active* if $\pi(C) \geq \frac{\tau}{|C|} = \frac{\tau}{2^l}$ (thus the smaller the cell, the larger its population must be before it is τ-active).

A cell is τ*-critical* if it is τ-active but neither of its children is τ-active. A cell is τ*-fringe* if it is not τ-active itself, but its parent is τ-active and not critical (refer to Figure 3). We denote by γ_l (respectively, ϕ_l) the number of critical (respectively, fringe) cells in $\mathcal{C}_l$. It is easy to confirm that critical and fringe cells provide a disjoint cover of $\mathcal{E}$. We begin by observing some other simple properties of active, critical and fringe cells:

▶ **Observation 1.** *If a cell C is τ-active then all its ancestors are τ-active.*

▶ **Observation 2.** *If a cell $C \in \mathcal{C}_l$ is τ-critical then $\tau/2^l \leq \pi(C) < 4\tau/2^l$.*

▶ **Observation 3.** *If a cell $C \in \mathcal{C}_l$ is τ-fringe then $\pi(C) < \tau/2^l$.*

▶ **Lemma 2.1.** *Let γ_l (ϕ_l) be the number of critical (fringe) cells at level l. It must be that:*

$$\sum_l \phi_l \frac{1}{2^l} < 2 \sum_l \gamma_l \frac{1}{2^l} \text{ and } \sum_l \phi_l (\frac{1}{2^l})^2 < \frac{4}{3} \sum_l \gamma_l (\frac{1}{2^l})^2. \qquad (1)$$

Proof. Consider for each fringe cell C' the subtree T' rooted at its parent and the *nearest* critical cell C in T'. We associate C' to C. Each critical cell $C \in \mathcal{C}_l$ may have several associated fringe cells, but at most one associated fringe cell C' in each level $l' \geq l$ (and no associated fringe cells at level $l' < l$). Indeed, since two fringe cells at level l' cannot share the same parent, their associated critical cells lie in different parent-rooted subtrees. By charging to each critical cell l at most one fringe cell in each level $l' \geq l$ we obtain the lemma. ◀

Satisfied entities and cells. Consider a sequence T of τ consecutive time steps. If an entity e_i is queried k times in T then its uncertainty interval B_i must have length at least $\tau/2k$ for at least half of the time steps in T. We will say that an entity e_i in a critical cell $C \in \mathcal{C}_l$ is *satisfied* if it is queried more than $40\tau/2^l$ times within T. A critical cell $C \in \mathcal{C}_l$ is *unsatisfied* if at least $\frac{1}{2}\tau/2^l$ entities in its population are not satisfied. To satisfy a critical cell, a query strategy Γ needs to allocate at least $(\frac{1}{2}\tau/2^l)\frac{40\tau}{2^l} = 20(\frac{\tau}{2^l})^2$ queries to entities in $\pi(C)$.

Given these definitions, we are ready to state our main result:

▶ **Theorem 2.2.** *For any target value τ, either (i) any query strategy must give rise to uncertainty intervals whose intersection graph has total degree $\Omega(\tau)$ at some point in any time interval of length τ, or (ii) there is a simple query strategy, in which points in τ-critical or τ-fringe nodes at level l are assigned query frequency at most 2^{-l}, that guarantees that the total degree in the intersection graph of the uncertainty intervals is $O(\tau)$ at all times.*

Proof strategy and supporting lemmas. Let T be any time interval of length τ. Given a value τ we consider the sum $\sum_t \gamma_t \cdot (\frac{\tau}{2^l})^2$. We show that either:

- Case (i): this sum is too large and there must at some time be many unsatisfied critical cells, contributing to a large total degree, or
- Case (ii): there exists a weighted round-robin query strategy Γ such that entities in all τ-critical and τ-fringe cell at level l have uncertainty region size 2^l at all times.

As a consequence of these respective cases:

▶ **Lemma 2.3.** *Let $C \in C_l$ be an unsatisfied cell. For* more *than a quarter of the time steps in T, at least $\frac{1}{6}\tau/2^l$ entities in C have an uncertainty region of size at least $K = \frac{1}{80}2^l$.*

▶ **Lemma 2.4.** *Suppose that $\sum_t \gamma_t \cdot (\frac{\tau}{2^l})^2 \leq \tau/10$, and let Γ be a query strategy where for all times $t \in T$, entities in a (critical or fringe) cell C at level l have an uncertainty region of size at most 2^l. Then the total degree is $O(\tau)$.*

Proving Theorem 2.2. With the above lemmas, we prove Theorem 2.2 in three steps:

▶ **Lemma 2.5.** *Let Γ be a query strategy and suppose that Γ satisfies $\textsc{Sat}(l)$ cells at level l. It must be that: $\sum_l \textsc{Sat}(l) \cdot (\frac{\tau}{2^l})^2 \leq \tau/20$.*

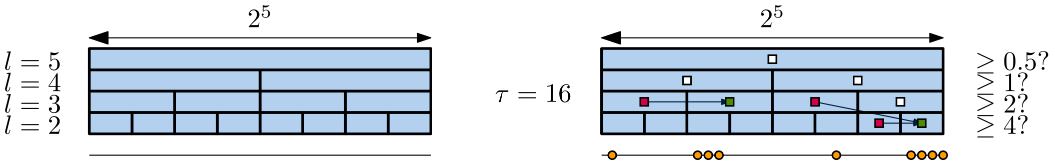


Figure 3 Our hierarchical decomposition of $\mathbb{R}^1$ into levels l and cells. Each cell C has two neighbors, two children and one parent. If the target value τ is 16, cells at level 3 are 16-active if they contain two or more entities. We mark active cells white, critical cells green and fringe cells red.

Proof. To satisfy a cell $C \in \mathcal{C}_l$ at least $20(\frac{\tau}{2^l})^2$ queries must be allocated to entities in $\pi(C)$. The sum of all queries over the time interval T is at most τ and so the inequality follows. ◀

▶ **Lemma 2.6.** *Let* Γ *be a query strategy that fails to satisfy* $\textsc{Unsat}(l)$ *cells at level* l*. Assume* $\sum_l \textsc{Unsat}(l) \cdot (\frac{\tau}{2^l})^2 > \tau/20$*. There exists a time* $t \in T$ *when the total degree of* $\mathcal{B}_i(t)$ *is* $\Omega(\tau)$*.*

Proof. We prove the above lemma through a counting argument, where we count the total degree over each time step. Let $C \in C_l$ be an unsatisfied cell. By Lemma 2.3, for at least $\frac{|T|}{4}$ time steps there is a set $M' \subset \pi(C)$ of $\frac{1}{6}\tau/2^l$ entities that each have uncertainty intervals of size at least $\frac{1}{80}2^l$. For each of these $\frac{1}{4}|T|$ time steps, the entities in M' contribute at least $\frac{1}{81}(\frac{1}{6}\tau/2^l)^2 = \frac{1}{81 \cdot 36}(\tau/2^l)^2$ to the total degree in the intersection graph. (This follows immediately from the observation that every interval of M', shrunk to size exactly $2^l/80$, with center in C must intersect exactly one of 81 equally-spaced spikes placed at separation $2^l/80$ within C. Thus the cliques in the intersection graph associated with the intervals intersecting each of these spikes contribute a total degree of at least $81(\frac{1}{6} \cdot \frac{1}{81}\tau/2^l)^2$.) If we sum for each critical cell, for each of the $\frac{|T|}{4}$ time steps, their contribution to the total degree we obtain the following lower bound:

$$\sum_{t \in T} \textsc{Totaldegree}(\mathcal{B}_i(t)) \geq \frac{|T|}{|4|} \cdot \frac{1}{81 \cdot 36} \cdot \sum_l \textsc{Unsat}(l) \cdot (\tau/2^l)^2 > \frac{|T| \cdot \tau}{4 \cdot 81 \cdot 36 \cdot 20}.$$

Where the last inequality follows from the lemma assumption. It immediately follows that there is a time $t \in |T|$ for which $\textsc{Totaldegree}(\mathcal{B}_i(t)) = \Omega(\tau)$. ◀

Proof of the two cases. Recall that for a given value of τ, we denote by γ_l the number of critical cells at level l. We make a case distinction (case (i) or case (ii)) based on whether:

$$\sum_l \gamma_l \cdot (\frac{\tau}{2^l})^2 = \sum_l \textsc{Sat}(l) \cdot (\frac{\tau}{2^l})^2 + \sum_l \textsc{Unsat}(l) \cdot (\frac{\tau}{2^l})^2 \leq \tau/10. \tag{2}$$

Case (i): Suppose first that Equation 2 is not true. By Lemma 2.5, it must be that for any query strategy Γ, $\sum_l \textsc{Sat}(l) \cdot (\frac{\tau}{2^l})^2 \leq \tau/20$ and thus $\sum_l \textsc{Unsat}(l) \cdot (\frac{\tau}{2^l})^2 > \tau/20$. Thus, by Lemma 2.6 there is at least one time $t \in T$ for which the total degree is $\Omega(\tau)$.

Case (ii): Suppose otherwise that Equation 2 is true. We claim that we can construct a query strategy for which at all times $t \in T$, the total degree is $O(\tau)$. By Lemma 2.1, $\sum_l \phi_l \frac{\tau}{4^l} < 4/3 \sum_l \gamma_l \frac{\tau}{4^l}$. Since we assumed that Equation 2 is true, it follows that:

$$\sum_l (\gamma_l + \phi_i) \frac{\tau}{4^{l-1}} < 28/3 \sum_l \gamma_l \frac{\tau}{4^l} < 28/30. \tag{3}$$

By Anily *et al.* [3] (Lemma 6.2) a query frequency of $\tau/4^{l-1}$ can be allocated to each critical (and fringe) cell on level l. Since critical and fringe cells contain at most $4\tau/2^l$ entities, for all times $t \in T$, entities in a (critical or fringe) cell C at level l have an uncertainty region of at most 2^l. By Lemma 2.4, this obtains a total degree of at most $O(\tau)$. ◀

References

1 Pankaj K. Agarwal, Jeff Erickson, and Leonidas J. Guibas. Kinetic BSPs for intersecting segments and disjoint triangles. In *Proc. 9th ACM-SIAM Symp. Discrete Algorithms*, pages 107–116, 1998.

2 J. Almeida and R. Araujo. Tracking multiple moving objects in a dynamic environment for autonomous navigation. In *Advanced Motion Control, 2008. AMC'08. 10th IEEE International Workshop on*, pages 21–26, 2008.

3 Shoshana Anily, Celia A. Glass, and Refael Hassin. The scheduling of maintenance service. *Discrete Applied Mathematics*, 82:27–42, 1998.

4 J. Basch, L. J. Guibas, C. Silverstein, and L. Zhang. A practical evaluation of kinetic data structures. In *Proc. 13th ACM Symp. Comput. Geom.*, pages 388–390, 1997.

5 Kevin Buchin, Maarten Löffler, Pat Morin, and Wolfgang Mulzer. Delaunay triangulation of imprecise points simplified and extended. *Algorithmica*, 61(3):674–693, 2011.

6 Daniel Busto, William S. Evans, and David G. Kirkpatrick. Minimizing interference potential among moving entities. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 2400–2418. SIAM, 2019.

7 Minkyoung Cho, David M. Mount, and Eunhui Park. Maintaining nets and net trees under incremental motion. In *Proc. 20th International Symposium on Algorithms and Computation*, ISAAC '09, pages 1134–1143. Springer, 2009.

8 Mark de Berg, Marcel Roeloffzen, and Bettina Speckmann. Kinetic convex hulls and Delaunay triangulations in the black-box model. In *Proc. 27th ACM Symp. on Comput. Geom.*, pages 244–253, 2011.

9 Shane Brophy Eisenman. *People-centric mobile sensing networks.* PhD thesis, Columbia University, New York, NY, USA, 2008.

10 David Eppstein, Michael T. Goodrich, and Maarten Löffler. Tracking moving objects with few handovers. In *Proc. 12th Algorithms and Data Structures Symposium*, pages 362–373, 2011.

11 William Evans, David Kirkpatrick, Maarten Löffler, and Frank Staals. Competitive query strategies for minimising the ply of the potential locations of moving points. In *Proceedings of the Twenty-ninth Annual Symposium on Computational Geometry*, SoCG '13, pages 155–164, 2013.

12 William Evans, David Kirkpatrick, Maarten Löffler", and Frank Staals. Query strategies for minimizing the ply of the potential locations of entities moving with different speeds. In *Abstr. 30th European Workshop on Computational Geometry (EuroCG)*, 2014.

13 William Evans, David Kirkpatrick, Maarten Löffler, and Frank Staals. Minimizing co-location potential of moving entities. *SIAM J. Comput.*, 45(5):1870–1893, 2016.

14 Jie Gao, Leonidas Guibas, and An Nguyen. Deformable spanners and their applications. *Computational Geometry: Theory and Applications*, 35:2–19, 2006.

15 L. J. Guibas. Kinetic data structures — a state of the art report. In P. K. Agarwal, L. E. Kavraki, and M. Mason, editors, *Proc. Workshop Algorithmic Found. Robot.*, pages 191–209. A. K. Peters, Wellesley, MA, 1998.

16 Leonidas Guibas, John Hershberger, Subash Suri, and Li Zhang. Kinetic connectivity for unit disks. In *Proc. 16th ACM Symp. Comput. Geom.*, pages 331–340, 2000.

17 Maarten Löffler and Marc van Kreveld. Largest and smallest convex hulls for imprecise points. *Algorithmica*, 56(2):235–269, 2010.

18 G.L. Miller, S.H. Teng, W.P. Thurston, and S.A. Vavasis. Separators for sphere-packings and nearest neighbor graphs. *Journal of the ACM*, 44(1):1–29, 1997.

19 David M. Mount, Nathan S. Netanyahu, Christine D. Piatko, Ruth Silverman, and Angela Y. Wu. A computational framework for incremental motion. In *Proc. 20th ACM Symp. on Comput. Geom.*, pages 200–209, 2004.

20 M. Schneider. Moving Objects in Databases and GIS: State-of-the-Art and Open Problems. *Research Trends in Geographic Information Science*, pages 169–187, 2009.

21 Ivor van der Hoog, Irina Kostitsyna, Maarten Löffler, and Bettina Speckmann. Preprocessing ambiguous imprecise points. In *35th International Symposium on Computational Geometry (SoCG 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

22 Ke Yi and Qin Zhang. Multi-dimensional online tracking. In *Proc. 20th ACM-SIAM Symposium on Discrete Algorithms*, pages 1098–1107. SIAM, 2009.

23 C. Zhu, L. Shu, T. Hara, L. Wang, and S. Nishio. Research issues on mobile sensor networks. In *Communications and Networking in China (CHINACOM), 2010 5th International ICST Conference on*, pages 1–6. IEEE, 2010.

Lions and Contamination: Monotone Clearings

Daniel Bertschinger, Meghana M. Reddy*†, and Enrico Mann

ETH Zürich, Department of Computer Science
{daniel.bertschinger, meghana.mreddy}@inf.ethz.ch, emann@student.ethz.ch

Abstract

We consider a special variant of a pursuit-evasion game called lions and contamination. In a graph whose vertices are originally contaminated, a set of lions walk around the graph and clear the contamination from every vertex they visit. The contamination, however, simultaneously spreads to any adjacent vertex not occupied by a lion. We study the relationship between different types of clearings of graphs, such as clearings which do not allow recontamination, clearings where at most one lion moves at each time step and clearings where lions are forbidden to be stacked on the same vertex. We answer several questions raised by Adams et al. [1] in last year's edition of EuroCG.

1 Introduction

Pursuit-evasion problems have a long and rich history going back more than 50 years [8, 9]. Countless similar problems have been studied under very different names in the past. What they all have in common is that there is a group of pursuers that try to catch an evader. The typical question asked in a pursuit-evasion problem is whether the evader can escape the pursuers, and if so, for how long. Naturally, the more pursuers there are, the harder it is for the evader to escape. Some other objectives of the pursuers can be to catch the evader fast (minimize the time taken) or with minimal effort (minimize the distance traveled).

There are different variations of the problem depending on the exact rules, among which we study one. For detailed definitions of the various problems, see the surveys [4, 3, 6] and the references therein. We study the problem of lions and contamination, which was first mentioned in [7]. A group of lions tries to eradicate contamination from a graph while the contamination spreads to all adjacent vertices that are not occupied by a lion.

More formally, suppose there is a graph $G = (V, E)$. At the very beginning, every vertex occupied by a lion is considered cleared of contamination, whereas the remaining vertices are considered to be contaminated. In this particular problem, time is viewed as discrete; and in every step, the lions and the contamination both move simultaneously. Every lion is allowed to move along an edge that is incident to its current position. Contamination, on the other hand, spreads along every incident edge to every adjacent vertex, unless the edge is used by a lion or a lion occupies the adjacent vertex. Figure 1 illustrates an example. The sequential variant of this problem, where the lions and contamination move one after the other in alternating time steps, results in a setting where the lions are more powerful and hence is a very different problem compared to the one we study.

Note that for some graphs, it is easy to see whether k lions can clear the graph of contamination. However, finding the minimum number of lions required to clear a graph seems to be a hard question. For example, the minimum number of lions required to clear the $n \times n$-grid is not known. Nevertheless, it is known that at least $\lfloor \frac{n}{2} \rfloor + 1$ lions are needed [5].

[*] Supported by the Swiss National Science Foundation within the collaborative DACH project *Arrangements and Drawings* as SNSF Project 200021E-171681.

[†] The second author's full last name consists of two words and is *Mallik Reddy*. However, she consistently refers to herself with the first word of her last name being abbreviated.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

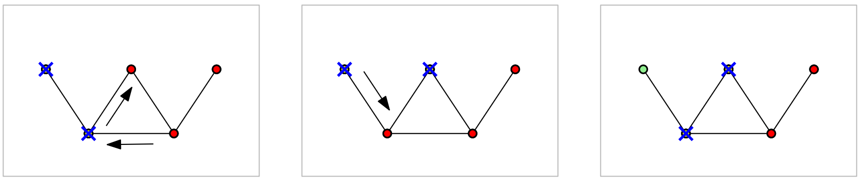


Figure 1 A graph and two lions (indicated by blue crosses). One lion moves in the first time step, however, contamination moves simultaneously and the vertex gets recontaminated. After the second step, a vertex that is not occupied by a lion is cleared of contamination (indicated in green).

Since n lions can simply sweep the graph from left to right and clear the grid of contamination, n is an upper bound on the number of lions needed for clearing the $n \times n$-grid, and this is the best upper bound currently known. Further, it is believed that $n-1$ lions are not sufficient. For higher-dimensional grids, it is known that $\Theta(n^{d-1}/\sqrt{d})$ lions are necessary and sufficient [2]. In this paper, we study different types of clearings that were defined by Adams et al. [1] and answer several questions raised in their paper.

We denote a clearing of a graph using k lions as a *k-clearing* and the graph itself is referred to as *k-clearable*. We say a clearing is *monotone* if no vertex ever gets recontaminated. The lions and the clearing are said to be *polite* if at most one lion moves in each time step and *non-stacked* if no two lions occupy the same vertex at any point in time.

The paper is organized in the following way. In Section 2, we show that there exist k-clearable graphs which require more than k lions for any monotone clearing. This implies that monotone clearings are harder to achieve than non-monotone clearings. In Section 3, we show that any monotone clearing can be paused at any time and no recontamination occurs. This allows us to show that any monotone clearing can be converted into a monotone and polite clearing. We also show that polite clearings can be transformed into non-stacked clearings (see Theorem 3.4). Finally, in Section 4, we tackle the subgraph question raised in [1]. Given a k-clearable graph G and some subgraph $H \subseteq G$, they ask whether H is k-clearable. We answer this question in some settings.

2 Monotone Clearings

2.1 The $n \times n$ Grid

Let us consider the $n \times n$ grid. As already mentioned, at least $\lfloor \frac{n}{2} \rfloor + 1$ lions are needed, while n lions are sufficient to clear the grid. If restricted to monotone clearings we can improve the lower bound and close the gap between the bounds.

Let $V(t)$ be the set of cleared vertices at time t. We define a *boundary vertex* as a vertex of $V(t)$ that has a neighbor in $V \setminus V(t)$. Before we state and prove our result, we first make two simple observations and recall a lemma proved by Berger et al. [2].

▶ Observation 2.1. The number of cleared vertices in a k-clearing cannot increase by more than k in one time step.

▶ Observation 2.2. If there are more than k boundary vertices at time t, then at least one vertex gets recontaminated in the next time step.

▶ **Lemma 2.3** (Lemma 5 of [2]). *Any vertex set S that is a subset of the $n \times n$ grid and satisfies $\frac{n^2}{2} - \frac{n}{2} \leq |S| \leq \frac{n^2}{2} + \frac{n}{2}$ has at least n boundary vertices.*

▶ **Theorem 2.4.** *A monotone clearing of the $n \times n$ grid needs at least n lions.*

Proof. Assume that the $n \times n$ grid has a monotone clearing with $n-1$ lions. Initially, the lions start with at most $n - 1 < \frac{n^2}{2} + \frac{n}{2}$ cleared vertices and eventually have to clear all n^2 vertices. Further, Observation 2.1 implies that at most $n-1$ vertices are cleared in each time step. Thus, at some time t, the set of cleared vertices $V(t)$ must satisfy the condition $\frac{n^2}{2} - \frac{n}{2} \leq |V(t)| \leq \frac{n^2}{2} + \frac{n}{2}$. The number of boundary vertices at such a time t will be at least n due to Lemma 2.3, and Observation 2.2 then implies that at least one vertex will get recontaminated at $t+1$. Hence, no monotone $(n-1)$-clearing of the $n \times n$ grid exists. ◀

In hindsight, this might not be a very surprising result. However, this does not necessarily improve the lower bound for non-monotone clearings as we will see in the next subsection.

2.2 Graphs with No Monotone Clearing

Unfortunately, not every graph with a k-clearing admits a monotone k-clearing. Indeed, we can show that the set of monotone k-clearable graphs is a proper subset of the set of all k-clearable graphs, which implies that monotonicity is a strong assumption on clearings. Theorem 2.4 does not improve the general lower bound for grids due to this reason.

▶ **Theorem 2.5.** *For any $k \geq 2$ there exist k-clearable graphs with no monotone k-clearing.*

Here, we construct a 2-clearable graph G that has no monotone 2-clearing. The arguments can be extended for $k > 2$ lions. A formal proof can be found in the full version of the paper.

Proof Sketch. Consider the graph G in Figure 2. It can be cleared using two lions: they first clear the left square, then walk over to the middle, where one lion guards the vertex u to block contamination from entering the left square, and one lion goes to vertex v. Next, they sweep the middle part from left to right and finally, they clear the right square.

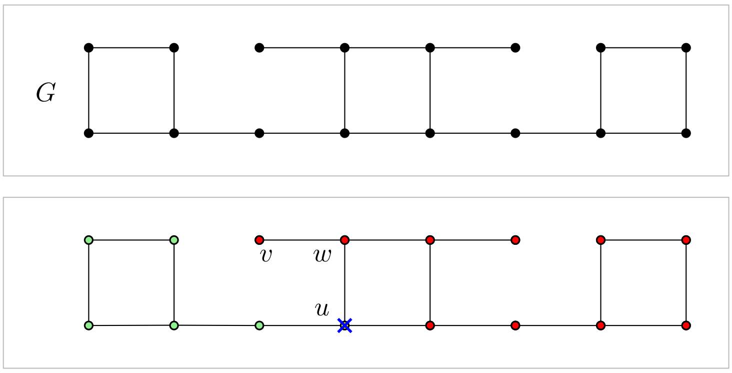


Figure 2 A graph G (i.e., a subgraph of the grid), and one particular situation in the clearing, where both lions occupy vertex u, the left part already cleared, the right part still contaminated.

On the other hand, G admits no monotone 2-clearing. The main idea is that independent of how the lions start, they always end up in a situation similar to the one illustrated in Figure 2, where they cannot clear vertex v without allowing recontamination. ◀

This result however raises the following question.

▶ **Open Question 1.** *Given a k-clearable graph $G = (V, E)$, is it always monotonically clearable with $k+1$ lions? More formally, is there a non-trivial upper bound on the number of lions required for a monotone clearing?*

3 Transforming between Different Types of Clearings

3.1 Pausing Monotone Clearings

We now analyse some properties of monotone clearings. These will then allow us to show that monotone clearings can always be adapted to monotone clearings with polite lions. We start with an important observation.

▶ **Proposition 3.1.** *Let $\mathcal{C}$ be any clearing of a graph G. Assume that all the lions are paused indefinitely at time t, i.e., no lion moves from $t+1$ onwards. If no vertex gets recontaminated at time $t+1$, then no vertex gets recontaminated at a later time either.*

Proof. Recall that contamination spreads along every incident edge at each time step. If no vertex gets recontaminated at $t+1$, it implies that every cleared vertex neighboring a contaminated vertex is occupied by a lion that blocks the contamination from spreading. Then, no vertex can get recontaminated after $t+1$ either, since the lions continue to block the contamination from spreading. ◀

▶ **Lemma 3.2.** *A monotone clearing can be paused at any time and no vertex gets recontaminated.*

The proof uses Proposition 3.1 along with a simple contradiction argument, and can be found in the full version of the paper. With a more careful analysis of pausing monotone clearings, we can prove a much stronger result.

▶ **Theorem 3.3.** *Let G be a graph and $\mathcal{C}$ be a monotone clearing of G with k lions. Then, there exists another monotone clearing which uses k polite lions.*

The idea of the proof is that whenever we pause the lions, there is an ordering of the lions that move in the next time step such that the lions can be moved in this order to their next vertices one after the other, without allowing any recontamination. A formal but rather technical proof can be found in the full version of the paper. We would like to remark at this point that this proof is algorithmic, i.e., given a monotone clearing, we can compute another monotone clearing that uses polite lions without increasing the number of lions.

3.2 Polite and Non-Stacked Clearings

In this subsection, we study clearings which need not be monotone and consider other restrictions on clearings. In particular, we study the relationship between clearings that use polite lions and clearings that do not stack lions. We can show the following relation.

▶ **Theorem 3.4.** *Let G be a graph and $\mathcal{C}$ be a polite clearing of G with $k \leq n$ lions. Then, there exists another k-clearing that does not stack lions.*

Instead of stacking lions, chains of consecutive lions are formed such that whenever a stacked lion needs to move, lions move along this chain instead. The formal proof is deferred to the full version of the paper. Note that this result implies that the set of graphs that are clearable with polite lions is a subset of graphs that admit non-stacked clearings. Though we were unable to prove it, we believe that the converse of Theorem 3.4 is false, because the time taken by polite lions might be much higher when compared to clearings where multiple lions can be moved simultaneously, even when stacking is not allowed.

▶ **Open Question 2.** *Is the converse of Theorem 3.4 also true or are there graphs with a non-stacked clearing but no polite clearing?*

When the clearing $\mathcal{C}$ is monotone, the converse is indeed true (follows from Theorem 3.3).

4 Clearable Subgraphs

In the final section of this paper, we study clearings of subgraphs of a k-clearable graph. More formally, let $G = (V, E)$ be a k-clearable graph, and let H be a subgraph of G. It is natural to ask if H also admits a k-clearing (see Question 6.1 in [1]). On one hand, the contamination is restricted due to some missing edges; on the other hand, some edges or paths in $G \setminus H$ might be crucial for the lions to clear the graph.

We show that this question can be answered in the affirmative if the clearing on G is monotone, and that surprisingly, this need not be possible in some other restricted settings.

▶ **Theorem 4.1.** *Let G be a graph with a monotone k-clearing. Then any connected subgraph $H \subset G$ also admits a k-clearing.*

Note that the clearing of the subgraph need not be monotone. Figure 3 illustrates a graph G^* that admits a monotone 2-clearing (sweeping from left to right). However, recall Figure 2, which illustrates a subgraph of G^* that has no monotone 2-clearing.

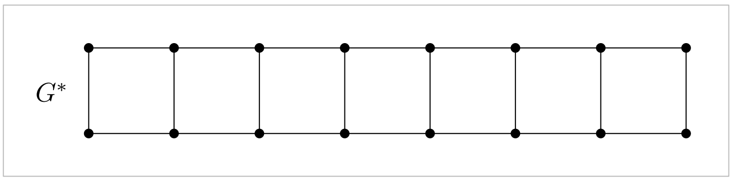


Figure 3 A supergraph of the graph illustrated in Figure 2. G^* is monotone 2-clearable even in the restricted setting of polite and non-stacked lions.

The main idea of the proof is to use a monotone polite clearing (that exists by Theorem 3.3), and whenever a lion would need to use a non-existent edge or move to a non-existent vertex, the lion is instead rerouted to its destination. A formal proof can be found in the full version of the paper.

Finally, we illustrate a graph where one of its subgraphs does not admit a 2-clearing with polite and non-stacked lions.

▶ Observation 4.2. The graph G illustrated in Figure 4 is 2-clearable with polite non-stacked lions. However, its subgraph H is not 2-clearable with polite non-stacked lions.

Note that H is not only a subgraph but also an induced subgraph of G.

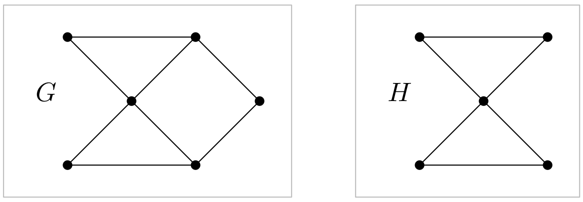


Figure 4 Graph G is 2-clearable with polite and non-stacked lions, whereas subgraph H is not.

5 Conclusion

In the first part of the paper, we studied different types of clearings, namely monotone, polite and non-stacked clearings and we showed some of the relations between them. This gives a good overview over the different restrictions, though a few questions remain open.

In the second part, we focused on the subgraph question raised by Adams et al. [1]. We were able to answer the question in some restricted settings. In the general setting, we believe that there exist graphs with subgraphs that admit no k-clearing. Such a graph might be rather large. A next step in this direction would be to design an algorithm that checks whether a given graph is k-clearable. Such an algorithm, however, may not be easy to find.

References

1 H. Adams, L. Gibson, and J. Pfaffinger. Lions and contamination, triangular grids, and cheeger constants. *arXiv*, 2020. URL: `https://arxiv.org/abs/2012.06702`.

2 F. Berger, A. Gilbers, A. Grüne, and R. Klein. How many lions are needed to clear a grid? *Algorithms*, 2(3):1069–1086, 2009. URL: `https://www.mdpi.com/1999-4893/2/3/1069`, `doi:10.3390/a2031069`.

3 A. Bonato and B. Yang. Graph searching and related problems. In P. M. Pardalos, D.-Z. Du, and R. L. Graham, editors, *Handbook of Combinatorial Optimization*, pages 1511–1558, 2013. `doi:10.1007/978-1-4419-7997-1_76`.

4 R. Borie, S. Koenig, and C. Tovey. Section 9.5: Pursuit-evasion problems. In J. Yellen J. Gross and P. Zhang, editors, *Handbook of Graph Theory*, pages 1145–1165. Chapman and Hall/CRC, 2013.

5 P. Brass, K. D. Kim, H.-S. Na, and C.-S. Shin. Escaping off-line searchers and a discrete isoperimetric theorem. In *Algorithms and Computation*, pages 65–74, 2007.

6 T. H. Chung, G. A. Hollinger, and V. Isler. Search and pursuit-evasion in mobile robotics, a survey, 2011. URL: `https://calhoun.nps.edu/handle/10945/45474`.

7 A. Dumitrescu, I. Suzuki, and P. Zylinski. Offline variants of the "lion and man" problem. In *Proceedings of the Twenty-Third Annual Symposium on Computational Geometry*, page 102–111. Association for Computing Machinery, 2007. URL: `10.1145/1247069.1247085`.

8 R. Isaacs. Differential games: A mathematical theory with applications to warfare and pursuit, control and optimization. 1965.

9 T. D. Parsons. Pursuit-evasion in a graph. In Y. Alavi and D. R. Lick, editors, *Theory and Applications of Graphs*, pages 426–441, 1978.

A Conditional Lower Bound for the Discrete Fréchet Distance in a Graph*

Anne Driemel[1], **David Göckede**[1], **Ivor van der Hoog**[2], **and Eva Rotenberg**[2]

1 **Institut für Informatik, Universität Bonn, Germany**
`driemel@cs.uni-bonn.de`

2 **Department of Applied Mathematics and Computer Science, Technical University of Denmark, Denmark**
`vanderhoog@gmail.com | erot@dtu.dk`

Abstract

The Fréchet distance is a well-studied similarity measure between curves that is widely used throughout computer science. Motivated by applications where curves stem from walks on an underlying graph (such as a road network), we define and study the Fréchet distance for walks in graphs. When provided with a distance oracle of G with $O(1)$ query time, the classical quadratic-time dynamic program can compute the Fréchet distance between two walks P and Q in a graph G in (quadratic) $O(|P| \cdot |Q|)$ time. We show that, without additional assumptions on G, P, or Q, quadratic running time is (likely) necessary. Specifically, we provide a conditional lower bound showing that the Fréchet distance between arbitrary *walks* in a weighted planar graph cannot be computed in $O((|P| \cdot |Q|)^{1-\delta})$ time for any $\delta > 0$ unless the Orthogonal Vector Hypothesis fails. Our result holds even for walks in a constant-complexity graph.

Related Version ArXiv:2201.02121

1 Introduction

The Fréchet distance is a popular metric for measuring the similarity between (polygonal) curves P and Q. The Fréchet distance is often intuitively defined through the following metaphor: suppose that we have two curves that are traversed by a person and their dog. Over all possible (monotone) traversals by both the person and the dog, what is the minimum length of their connecting leash? This distance measure is similar to the Hausdorff distance, which is defined for sets, except that it takes the ordering of points along the curve into account. The Fréchet distance has many applications; in particular in the analysis and visualization of movement data [7, 11, 17, 25]. It is a versatile distance measure that can be used for a variety of objects, such as handwriting [22], coastlines [19], outlines of geometric shapes in geographic information systems [13], trajectories of moving objects, such as vehicles, animals or sports players [21, 23, 4, 11], air traffic [3] and also protein structures [16]. The two most-studied variants of the Fréchet distance are the *continuous* and *discrete* Fréchet distance (based on whether the entities traverse a polygonal curve in a continuous manner or vertex-by-vertex).

* Anne Driemel is supported by the Hausdorff Center for Mathematics (DFG grant number EXC 2047). Ivor van der Hoog and Eva Rotenberg are partially supported by Independent Research Fund Denmark grant 2020-2023 (9131-00044B) “Dynamic Network Analysis”.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Alt and Godau [2] were the first to study the Fréchet distance from a computational perspective. They studied how to compute the continuous Fréchet distance of n and m vertices each in $O(mn \log(n+m))$ time. Recently, this running time was improved by Buchin *et al.* [8] to $O(n^2\sqrt{\log n}(\log\log n)^{3/2})$ on a real-valued pointer machine and $O(n^2 \log\log n)$ on a word RAM with word size $\Omega(\log n)$. Eiter and Manila [15] showed how to compute the discrete Fréchet distance $D_{\mathcal{F}}(P,Q)$ between two polygonal curves in $O(nm)$ time, which was later improved to $O(nm(\log\log nm)/\log nm)$ by Buchin *et al.* [8].

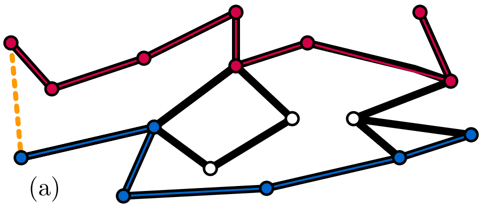


Figure 1 The Fréchet distance may be derived from the Euclidean or the shortest path metric.

Conditional lower bounds for the Fréchet distance. The above (near-) quadratic algorithms are accompanied by a series of conditional lower bounds for computing the Fréchet distance. All these results assume the Orthogonal Vector Hypothesis (OVH) or, by extension, the strong exponential time hypothesis (SETH) [24]. Bringmann [5] shows that there is no $O(n^{2-\delta})$ algorithm, for any $\delta > 0$, for computing the (discrete or continuous) Fréchet distance between two polygonal curves of n vertices each. Bringmann's original proof uses self-intersecting curves in the plane. Later, Bringmann and Mulzer [6] showed the same conditional lower bound for intersecting curves in $\mathbb{R}^1$. Bringmann [5] also showed the following conditional lower bound tailored to the unbalanced setting where the two input curves have different complexities: given two polygonal curves of n and m vertices each, there is no $O((nm)^{1-\delta})$ time algorithm for computing the Fréchet distance. Recently, Buchin, Ophelders and Speckmann [10] showed that (assuming OVH) there can be no $O((nm)^{1-\delta})$ time algorithm that computes anything better than a 3-approximation of the Fréchet distance for pairwise disjoint planar curves in $\mathbb{R}^2$ and intersecting curves in $\mathbb{R}^1$.

Fréchet distance variants. Variants of the Fréchet distance include those that model partial similarity by allowing straight-line shortcuts along a curve [14], or by maximizing the portions of the curves that are matched to each other within a fixed distance [9]. Other variants constrain the class of mappings by applying speed constraints [18] or topological constraints [12]. Even other variants extend the class of mappings, such as the weak Fréchet distance, which was already studied by Alt and Godau [2]. Strikingly, the Fréchet distance has not been studied for weighted graph metrics. Edge-weighted graphs with their shortest-path metric are commonly used to model discrete metric spaces [20], and the Fréchet distance can be derived from the underlying distance metric (Figure 1).

In this paper, we initiate a study of the computational complexity of the discrete Fréchet distance between walks in a planar graph, where distances between nodes are measured by their shortest path metric in this graph. This is a natural model when, for example, measuring the similarity of two trajectories in the same street network (Figure 2). In this setting, we show a conditional lower bound for computing the discrete Fréchet distance $D_{\mathcal{F}}(P,Q)$ between two walks P and Q in a graph. Specifically, assuming the Orthogonal Vector Hypothesis (OVH), we show that if G is an integer-weighted planar graph, P and Q

are walks in G and $m = n^\gamma$ for some constant $\gamma > 0$, then for every $\delta > 0$ there can be no algorithm that computes $D_\mathcal{F}(P, Q)$ (or a 1.01-approximation) in $O((nm)^{1-\delta})$ time unless OVH fails. In the full version, we extend this conditional lower bound to paths in a planar weighted graph and we show algorithmic results for when P is a shortest path.

2 Preliminaries

Let $G = (V, E)$ be an undirected weighted graph with N vertices, where every edge e_i has some corresponding integer weight ω_i and all weights can be expressed in a word of $\Theta(\log N)$ bits. For any two vertices $v_1, v_2 \in V$ their distance, denoted by $d(v_1, v_2)$, is the length of the shortest path from v_1 to v_2 in G. A walk in G is any sequence of vertices where every subsequent pair of vertices is connected by an edge in E. A path in G is a walk where no vertex appears twice in the sequence. Let P be any walk in G, represented by an ordered set of vertices $P = (p_1, p_2, \ldots p_n)$. We denote by $|P| = n$ the number of vertices in P and by $[n]$ the set $(1, 2, \ldots, n)$. We denote the walk $Q = (q_1, q_2, \ldots q_m)$, $|Q|$ and $[m]$ analogously.

Discrete Fréchet distance. Given two walks P and Q in G, we denote by $[n] \times [m] \subset \mathbb{N} \times \mathbb{N}$ the integer lattice of n by m integers. We say that an ordered sequence F of points in $[n] \times [m]$ is a *discrete walk* if for every consecutive pair $(i, j), (k, l) \in F$, we have $k \in \{i - 1, i, i + 1\}$ and $l \in \{j - 1, j, j + 1\}$. It is furthermore *xy-monotone* when we restrict to $k \in \{i, i+1\}$ and $l \in \{j, j + 1\}$. Let F be a discrete walk from $(1, 1)$ to (n, m). The *cost* of F is the maximum over $(i, j) \in F$ of $d(p_i, q_j)$. The (strong) discrete Fréchet distance is the minimum over all (xy-monotone) walks F from $(1, 1)$ to (n, m) of its associated cost:

$$D_\mathcal{F}(P, Q) := \min_F \operatorname{cost}(F) = \min_F \max_{(i,j) \in F} d(p_i, q_j).$$

Orthogonal Vectors Hypothesis. The Orthogonal Vectors problem can be stated as follows. Given are sets A and B of d-dimensional Boolean vectors with $|A| = n$ and $|B| = m$. The goal is to identify whether there exist two vectors $a = (a_1, a_2, \ldots a_d)$ and $b = (b_1, b_2, \ldots b_d)$ with $a \in A$ and $b \in B$, such that a and b are orthogonal (i.e. $\sum_{i=1}^d a_i \cdot b_i = 0$). In this paper, we use the following variant of the Orthogonal Vectors hypothesis. It is implied by SETH, see Abboud and Williams [1, Section 3], and it is equivalent to the standard variant of OHV defined by Williams [24], see Bringmann [5].

▶ **Definition 2.1.** The Orthogonal Vectors Hypothesis states that for every $\delta > 0$, there exists constants $\omega > 0$ and $1 > \gamma > 0$ such that the Orthogonal Vectors problem for d-dimensional vectors with $d = \omega \log n$ and $m = n^\gamma$, cannot be solved in $O((nm)^{1-\delta})$ time.

3 Hardness of walks in constant size graphs

We show a conditional lower bound for computing the Fréchet distance between two walks in a constant size graph G. We show that there cannot be an algorithm that always correctly computes $D_\mathcal{F}(P, Q)$ for two walks P and Q in a graph G, in truly subquadratic time, unless OVH fails, even if the graph G has unit weight. For any instance of Orthogonal Vectors A and B with n' and m' vectors, we create a constant complexity graph G and two walks P and Q with $n = O(n')$ and $m = O(m')$ vertices. We show that there exists a pair of vectors $(a, b) \in A \times B$ that are orthogonal if and only if $D_\mathcal{F}(P, Q) \leq 1.9$. Our construction closely matches the original conditional lower bound for the Fréchet distance by Bringmann [5].

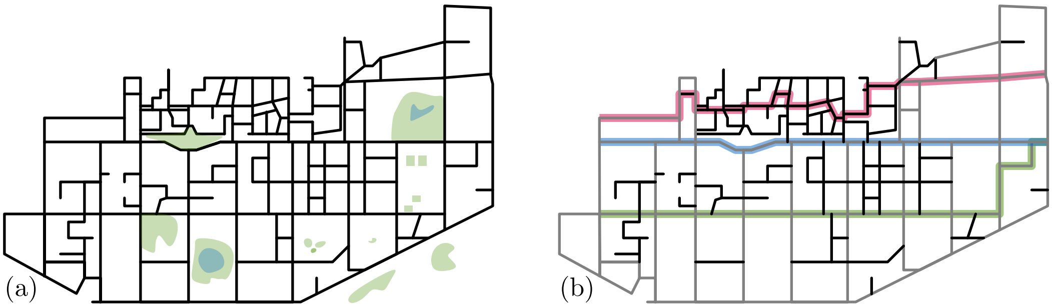


Figure 2 (a) A road network can be represented as a graph G. (b) Edges in G can be weighted, e.g. depending on whether traffic flows fast (grey) or slow (black). Under the graph distance metric, the Fréchet distance between blue and green may be smaller than the distance between red and blue; even though under the Euclidean metric, the red-blue Fréchet distance is smaller.

Vector gadgets and walks. For all instances $A = (a_1, \dots a_{n'})$ and $B = (b_1, \dots b_{m'})$, we assume that the vectors are of even dimension (otherwise, we add one dummy coordinate of 0 to each vector). For any instance A and B of Orthogonal Vectors, we start by constructing two constant complexity gadgets which we call the vector gadgets (Figure 3(a)).

We refer to the gadget that models vectors in A as the red gadget. The red gadget contains vertices $\{\alpha, \alpha^*, \gamma, A^{\{0\}}, A^{\{1\}}, R, \beta^*, \beta\}$ plus one additional 'sink' shown in white. For all vectors $a \in A$, we create what we call a *subwalk* $f(a)$ through this gadget as follows: the subwalk starts at γ and continues to either $A^{\{0\}}$ if the first value in a is a 0, or to $A^{\{1\}}$ otherwise. Then, the subwalk continues to A, and from there to either $A^{\{0\}}$ or $A^{\{1\}}$ depending on the second value in a. We continue alternating between A and a vertex in $(A^{\{0\}}, A^{\{1\}})$ until we reach the end of vector a. The walk $f(A)$ is now constructed as follows: it starts at α and moves to α^*. From there, it traverses the first subwalk $f(a_1)$. This subwalk ends at A, at which point the walk traverses through $A^{\{0\}}$ to γ to start $f(a_2)$. The walk ends at β^* into β. Formally, we denote by $\circ$ the concatenation of two walks and say:

$$f(A) = \{\alpha\} \circ \{\alpha^*\} \circ f(a_1) \circ \left\{A^{\{0\}}\right\} \circ f(a_2) \circ \left\{A^{\{0\}}\right\} \circ f(a_3) \circ \left\{A^{\{0\}}\right\} \circ \dots f(a_{n'}) \circ \{\beta^*\} \circ \{\beta\}.$$

Similarly, the blue gadget for vectors in B contains vertices $\{x, y, B^{\{0\}}, B^{\{1\}}, z\}$ plus one additional sink denoted by s. For all vectors $b \in B$ we create a subwalk $g(b)$ in a similar fashion. The subwalk $g(b)$ starts at y and then continues to either $B^{\{0\}}$ or $B^{\{1\}}$ depending on the first value in b. Then, the subwalk alternates between B and a vertex in $(B^{\{0\}}, B^{\{1\}})$ based on the successive value in b. The subwalk $g(b)$ ends at z. The walk $f(B)$ is now constructed as follows: the walk starts at x and proceeds with the subwalk $g(b_1)$. Then we walk goes from B to z, and from z *through* s to x, and we repeat this process for every vector in B. Formally we write:

$$g(B) = \{x\} \circ g(b_1) \circ \{z\} \circ \{s\} \circ \{x\} \circ g(b_2) \circ \{z\} \circ \{s\} \circ \dots \{x\} \circ g(b_{m'}) \circ \{z\}$$

From gadgets to a graph. We connect the vector gadgets as shown in Figure 3(b). We say that a pair of red and blue vertices is *close* whenever their distance is at most 1.9. Thus, the Fréchet distance is at most 1.9 if and only if there exists a traversal, where at all times the paired vertices are *close*. We observe the following distances between red and blue vertices:

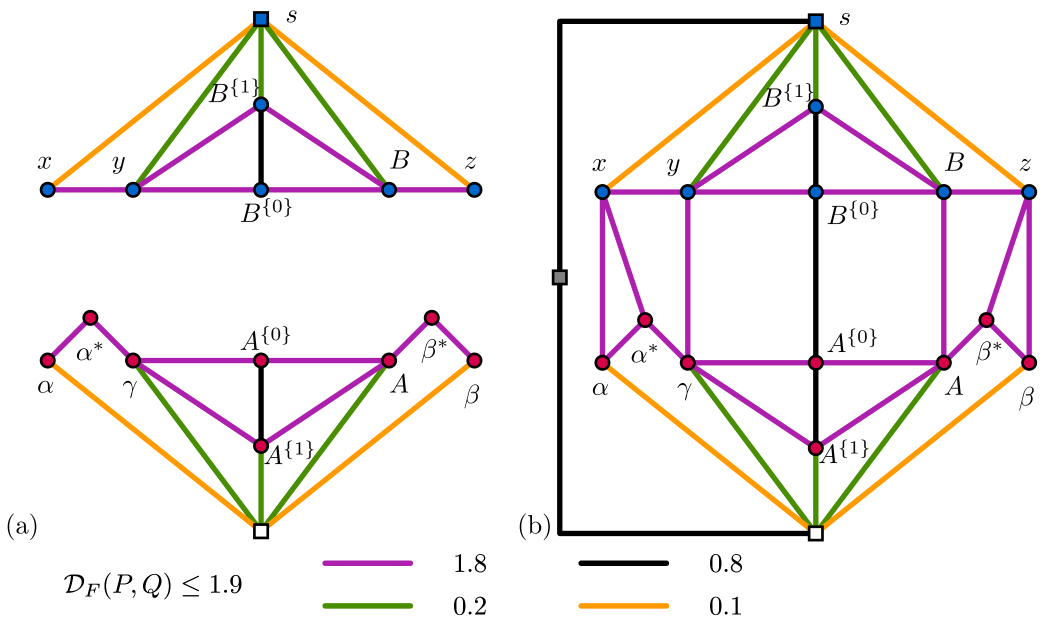


Figure 3 (a) The gadget corresponding to vectors in A (red) and vectors in B (blue). (b) We connect the two gadgets with weighted edges.

▶ **Theorem 3.1.** *Let G be a planar, integer-weighted graph, P and Q be two walks in G with n and m vertices and $n = m^\gamma$ for some constant $0 < \gamma \leq 1$. For all $\delta > 0$, there can be no algorithm that computes the Fréchet distance between P and Q in $O((nm)^{1-\delta})$ time.*

Proof. We observe that for any given A and B of n' and m' vectors, we can construct these two walks $P = f(A)$ and $Q = g(B)$ with $n = O(n')$ and $m = O(m')$ vertices respectively. We prove this theorem through showing that there are two orthogonal vectors if and only if $D_{\mathcal{F}}(P,Q) \leq 1.9$.

Two orthogonal vectors imply $D_{\mathcal{F}}(P,Q) \leq 1.9$. First, we show that if there exist two vectors $(a,b) \in A \times B$ such that a and b are orthogonal, then $D_{\mathcal{F}}(P,Q) \leq 1.9$. We construct a traversal where at all times the red entity (Red) traversing P is close to the blue entity (Blue) traversing Q. First, Red remains stationary at the vertex α, whilst the blue entity traverses B until it reaches $g(b)$. Since α is close to every blue vertex, they remain close.

Then, whilst Blue remains stationary at the vertex x, Red traverses P up to the start of $f(a)$ into the vertex γ. Every red vertex (except β^*) is close to x and thus they remain close.

At this point, Blue moves to y as Red remains stationary at γ. Then, both entities simultaneously traverse their respective vector gadgets. Observe that during this traversal, since a and b are orthogonal, the entities remain close. After this traversal, the Blue remains stationary at z, whilst Red traverses the remainder of P. Every red vertex (except α^*) is close to z and thus the entities remain close during this traversal. Finally, Red remains stationary at β whilst Blue traverses the remainder of Q.

$D_{\mathcal{F}}(P,Q) \leq 1.9$ implies two orthogonal vectors. We show that if the Fréchet distance between P and Q is at most 1.9, then there exists a pair of vectors $(a,b) \in A \times B$ such that a and b are orthogonal. Indeed, fix any traversal of P and Q that realises the Fréchet distance. When Red is at α^*, Blue must be at x on some subwalk $f(a)$.

dist.	α	α^*	β	β^*	γ	$A^{\{0\}}$	$A^{\{1\}}$	A
x	1.8	1.8	1.8	3.6	1.9	1.9	1.9	1.9
y	1.9	3.6	1.9	3.7	1.8	2	2	2
z	1.8	3.6	1.8	1.8	1.9	1.9	1.9	1.9
$B^{\{0\}}$	1.9	3.7	1.9	3.7	2	0.8	1.6	2
$B^{\{1\}}$	1.9	3.7	1.9	3.7	2	1.6	2	2
B	1.9	3.7	1.9	3.6	2	2	2	1.8

Table 1 Pairwise distances, close pairs are marked orange.

Consider now the time when Blue oves from x to y (where y lies in a gadget corresponding to some vector $b \in B$). At this time, Red cannot be at the vertex α because α precedes α^*. Similarly, Red cannot be at the vertex β because β^* precedes β and β^* is not close to any vertex between x and y. It follows, that Blue must be at y on some subwalk $f(b)$

Now let without loss of generality Red continue the traversal in the direction of $\{A^{\{0\}}, A^{\{1\}}\}$. We can apply the same argument and conclude that Blue must also move to the unique corresponding vertex in the traversal in the direction of $\{B^{\{0\}}, B^{\{1\}}\}$ (where Blue must go towards $B^{\{1\}}$ if Red goes to $A^{\{0\}}$). We can continue to apply the same argument to show that these vectors a and b must be orthogonal. This concludes the proof. ◀

References

1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 434–443. IEEE, 2014.

2 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(01n02):75–91, 1995.

3 Alessandro Bombelli, Lluis Soler, Eric Trumbauer, and Kenneth D Mease. Strategic air traffic planning with Fréchet distance aggregation and rerouting. *Journal of Guidance, Control, and Dynamics*, 40(5):1117–1129, 2017.

4 Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In *Proceedings of the 31st international conference on Very large data bases*, pages 853–864, 2005.

5 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless seth fails. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*, pages 661–670. IEEE, 2014.

6 Karl Bringmann and Wolfgang Mulzer. Approximability of the discrete Fréchet distance. *Journal of Computational Geometry*, 7(2):46–76, 2016.

7 Kevin Buchin, Maike Buchin, David Duran, Brittany Terese Fasy, Roel Jacobs, Vera Sacristan, Rodrigo I Silveira, Frank Staals, and Carola Wenk. Clustering trajectories for map construction. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 1–10, 2017.

8 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four soviets walk the dog: Improved bounds for computing the Fréchet distance. *Discrete & Computational Geometry*, 58(1):180–216, 2017.

9 Kevin Buchin, Maike Buchin, and Yusu Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proceedings of the twentieth annual ACM-SIAM symposium on Discrete algorithms*, pages 645–654. SIAM, 2009.

10 Kevin Buchin, Tim Ophelders, and Bettina Speckmann. Seth says: Weak Fréchet distance is faster, but only if it is continuous and in one dimension. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2887–2901. SIAM, 2019.

11 Maike Buchin, Bernhard Kilgus, and Andrea Kölzsch. Group diagrams for representing trajectories. *International Journal of Geographical Information Science*, 34(12):2401–2433, 2020.

12 Erin Wolf Chambers, Eric Colin De Verdiere, Jeff Erickson, Sylvain Lazard, Francis Lazarus, and Shripad Thite. Homotopic Fréchet distance between curves or, walking your dog in the woods in polynomial time. *Computational Geometry*, 43(3):295–311, 2010.

13 Thomas Devogele. A new merging process for data integration based on the discrete Fréchet distance. In *Advances in spatial data handling*, pages 167–181. Springer, 2002.

14 Anne Driemel and Sariel Har-Peled. Jaywalking your dog: computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42(5):1830–1866, 2013.

15 Thomas Eiter and Heikki Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, 1994.

16 Minghui Jiang, Ying Xu, and Binhai Zhu. Protein structure–structure alignment with discrete Fréchet distance. *Journal of bioinformatics and computational biology*, 6(01):51–64, 2008.

17 Maximilian Konzack, Thomas McKetterick, Tim Ophelders, Maike Buchin, Luca Giuggioli, Jed Long, Trisalyn Nelson, Michel A Westenberg, and Kevin Buchin. Visual analytics of de-

lays and interaction in movement data. *International Journal of Geographical Information Science*, 31(2):320–345, 2017.

18 Anil Maheshwari, Jörg-Rüdiger Sack, Kaveh Shahbaz, and Hamid Zarrabi-Zadeh. Fréchet distance with speed limits. *Computational Geometry*, 44(2):110–120, 2011.

19 Ariane Mascret, Thomas Devogele, Iwan Le Berre, and Alain Hénaff. Coastline matching process based on the discrete Fréchet distance. In *Progress in Spatial Data Handling*, pages 383–400. Springer, 2006.

20 Jiri Matousek. *Lectures on discrete geometry*, volume 212. Springer Science & Business Media, 2013.

21 Roniel S. De Sousa, Azzedine Boukerche, and Antonio A. F. Loureiro. Vehicle trajectory similarity: Models, methods, and applications. *ACM Comput. Surv.*, 53(5), September 2020. `doi:10.1145/3406096`.

22 E Sriraghavendra, K Karthik, and Chiranjib Bhattacharyya. Fréchet distance based approach for searching online handwritten documents. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 1, pages 461–465. IEEE, 2007.

23 Han Su, Shuncheng Liu, Bolong Zheng, Xiaofang Zhou, and Kai Zheng. A survey of trajectory distance measures and performance evaluation. *The VLDB Journal*, 29(1):3–32, 2020.

24 Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005. `doi:10.1016/j.tcs.2005.09.023`.

25 Dong Xie, Feifei Li, and Jeff M Phillips. Distributed trajectory similarity search. *Proceedings of the VLDB Endowment*, 10(11):1478–1489, 2017.

Segment Visibility Counting Queries in Polygons[*]

Kevin Buchin[1], Bram Custers[†2], Ivor van der Hoog[‡3], Maarten Löffler[§4], Aleksandr Popov[¶2], Marcel Roeloffzen[‖2], and Frank Staals[4]

1 **Department of Computer Science, TU Dortmund, Germany**
`kevin.buchin@tu-dortmund.de`

2 **Department of Mathematics and Computer Science, TU Eindhoven, The Netherlands**
`{b.a.custers, a.popov, m.j.m.roeloffzen}@tue.nl`

3 **Department of Applied Mathematics and Computer Science, TU Denmark, Copenhagen, Denmark**
`vanderhoog@gmail.com`

4 **Department of Information and Computing Sciences, Utrecht University, The Netherlands**
`{m.loffler, f.staals}@uu.nl`

Abstract

Let P be a simple polygon with n vertices, and let A be a set of m points or line segments in P. We develop data structures that efficiently count the objects in A that are visible to a query point or segment. We obtain fast, $\mathcal{O}(\operatorname{polylog} nm)$, query times, while using as little space as possible, for all combinations of settings. In this abstract, we focus on the result where the query is a line segment and A contains only points, obtaining $\mathcal{O}(\log n \log nm)$ query time using only $\mathcal{O}(nm^{2+\varepsilon} + n^2)$ space.

Related Version *Full Version*: `https://arxiv.org/abs/2201.03490` [6]

1 Introduction

Let P be a simple polygon with n vertices, and let A be a set of m points or line segments inside P. We develop efficient data structures for *visibility counting queries* in which we wish to report the number of objects from A visible to some (constant-complexity) query object Q. An object X in A *sees* Q if there is a line segment connecting X and Q contained in P. We are mostly interested in the case when Q is a point or a line segment. Our aim is to obtain fast, $\mathcal{O}(\operatorname{polylog} nm)$, query times, using as little space as possible. Our work is motivated by problems in movement analysis where we have sets of entities, for example, an animal species and their predators, moving in an environment, and we wish to determine if there is mutual visibility between the entities of different sets and quantify it by counting.

There is a lot of work on visibility in polygons [16, 17, 23, 24, 25, 26] (and even on terrains [1, 14]); the data structure version has been considered, too, where the polygon is given in advance [3, 10, 20, 22]. There are efficient data structures for querying visibility between two points [22] and the visibility polygon of a point [3]. For *weak visibility polygons,* i.e. visibility polygons of line segments, there is algorithmic work [20] and work on the data

[*] Research on the topic of this paper was initiated at the 5th Workshop on Applied Geometric Algorithms (AGA 2020) in Langbroek, The Netherlands.

[†] Supported by the Dutch Research Council (NWO) under the project number 628.011.005.

[‡] Supported by the Dutch Research Council (NWO) under the project number 614.001.504.

[§] Partially supported by the Dutch Research Council (NWO) under the project numbers 614.001.504 and 628.011.005.

[¶] Supported by the Dutch Research Council (NWO) under the project number 612.001.801.

[‖] Supported by the Dutch Research Council (NWO) under the project number 628.011.005.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Table 1 Results in this paper and its full version [6]. • and / denote points and line segments, respectively.

A	Q		Data structure		Section
		Space	Preprocessing	Query	
•	•	$\mathcal{O}(nm^2)$	$\mathcal{O}(nm \log n + nm^2)$	$\mathcal{O}(\log nm)$	2
		$\mathcal{O}(n{+}m^{2+\varepsilon} \log n)$	$\mathcal{O}(n + m \log^2 n + m^{2+\varepsilon} \log n)$	$\mathcal{O}(\log n \log nm)$	2
/	•	$\mathcal{O}(nm^2)$	$\mathcal{O}(nm \log n + nm^2)$	$\mathcal{O}(\log nm)$	2
•	/	$\mathcal{O}(n^2 + nm^{2+\varepsilon})$	$\mathcal{O}(n^2 \log m + nm \log n + nm^{2+\varepsilon})$	$\mathcal{O}(\log n \log nm)$	3
/	/	$\mathcal{O}(n^2 + nm^{2+\varepsilon})$	$\mathcal{O}(n^2 \log m + nm \log n + nm^{2+\varepsilon})$	$\mathcal{O}(\log n \log nm)$	[6]

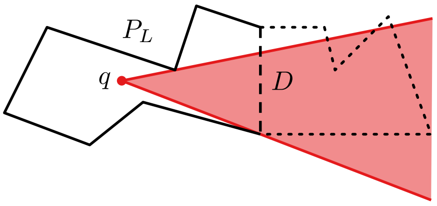


Figure 1 The filled shape is the cone $V(q, D, P_L)$.

structure version [12]. Eades et al. [15] and Aronov et al. [3] present data structures for visibility of moving points. Visibility counting queries have been studied before, as well, but mostly for counting the visible edges of the simple polygon containing the query point [5] or line segment [7]; there are also approximation algorithms [2, 18, 27]. In contrast, we count other visible line segments with visibility obstructed by the simple polygon. Our setting is closer to the problem of reporting all visible pairs of points in a simple polygon [4].

We discuss the following data structure question:[1] given a set of objects (points or line segments) A in a simple polygon P on n vertices, count the objects in A visible from a query object Q in P. We can answer such queries efficiently, i.e. in polylogarithmic time in n and $|A| = m$. In this abstract, we state the results, devoting extra attention to Section 3, as the machinery developed there is most useful to understand. The interested reader should refer the the full version [6] for details of the other settings. See Table 1 for an overview of the results. Next, we review some tools we use to build our data structures.

1.1 Preliminaries

A *cone* is a subspace of the plane enclosed by two rays from some point p, called the *apex*. Define the *visibility cone* of point $p \in P$ *through* a line segment $Q \subset P$ as the set of rays that intersect Q before properly crossing the boundary of P for the first time. A *visibility ray* is any ray in such a cone. Let D be a diagonal of P and let P_L and P_R be the two subpolygons we obtain when splitting P with D. For convenience, denote the *visibility cone* of point $p \in P_L$ *through* D by $V(p, D, P_L)$. It can be seen as the set of rays from p that intersect D and only cross the boundary of P_L at D (see Figure 1).

Cutting trees. We frequently use a *cutting tree,* a data structure based on recursively subdividing the plane to support fast half-plane range queries [9, 11, 13]. A benefit of

[1] The algorithmic version can be solved in optimal $\mathcal{O}(n + m \log n)$ time (see the full version).

that data structure is the ability to nest levels to support simplex range searching without increasing storage requirements or query time. Through the use of geometric dualisation, we can use variants of this data structure to support intersection and containment queries.

▶ **Lemma 1.** *Let $\mathcal{L}$ be a set of m lines. We can store $\mathcal{L}$ in a multilevel cutting tree, using $\mathcal{O}(m^{2+\varepsilon})$ space and preprocessing time, so that given a query line segment $\overline{pq}$, we can count the number of lines in $\mathcal{L}$ intersected by $\overline{pq}$ in time $\mathcal{O}(\log m)$.*

Proof. Observe that a line from $\mathcal{L}$ is either above or below point p. If a line is below both points p and q, then the line and $\overline{pq}$ do not intersect; same holds if it is above both points. So, to check that it does intersect $\overline{pq}$, it suffices to test, for lines above p, that they are also below q; and the other way around. This can be done with two queries to a two-level cutting tree in a standard way. In the first query, we find the canonical subsets representing lines that lie above p and then select the subsets representing lines that also lie below q. The second query handles the lines below p and above q. Summing up the counts from both queries gives the answer in the time stated in the lemma. ◀

Polygon decomposition. For a simple polygon P on n vertices, Chazelle [8] shows that we can construct a balanced hierarchical decomposition of P by recursively splitting the polygon into two subpolygons of approximately equal size. The polygon is split on diagonals between two vertices of the polygon. The recursion stops when reaching triangles. The decomposition can be computed in $\mathcal{O}(n)$ time and stored using $\mathcal{O}(n)$ space in a balanced binary tree.

Hourglasses and the shortest path data structure. Let P be a simple polygon. An *hourglass* for segments $\overline{pq}$, $\overline{rs}$ in P is the union of geodesic shortest paths in P from a point on $\overline{pq}$ to a point on $\overline{rs}$ [19]. There is a data structure to compute shortest paths in P [19, 21] using $\mathcal{O}(n)$ space and preprocessing time, based on the polygon decomposition [8] and the hourglasses between the diagonals of the decomposition. It can be queried in $\mathcal{O}(\log n)$ time:

Shortest path query. Given points $p, q \in P$, return the geodesic shortest path between p and q in P as a set of $\mathcal{O}(\log n)$ nodes of the decomposition.

Segment location query. Given a segment $\overline{pq}$, return the *polygon cover* of $\overline{pq}$, i.e. the two leaf triangles containing p and q in the decomposition and the $\mathcal{O}(\log n)$ pairwise disjoint open hourglasses, so that the leaf triangles and the hourglasses fully cover $\overline{pq}$.

Cone query. Return the visibility cone from a point s through the line segment $\overline{pq}$ in P.

2 Point Queries

Suppose the query is a point q. There is a simple arrangement-based solution using the fact that the desired count is the number of the (weak) visibility polygons of the objects in A that contain the query point. We can precompute the arrangement of the visibility polygons and preprocess it for point location; based on previous results on the complexity of such an arrangement [5], we arrive at the result. See the full version [6].

If the objects in A are also points, we get much better bounds on space and preprocessing time by employing the *polygon decomposition* by Chazelle [8] to hierarchically split the polygon on diagonals. We associate a data structure with every node in the decomposition tree. Assuming the data structure is associated with a split at diagonal D splitting P into P_L and P_R, and assuming $q \in P_L$, we can count the points in $A \cap P_R$ using the data structure; we store a symmetric data structure for $q \in P_R$. The key insight is that any point $a \in A \cap P_R$ sees q in P if and only if $q \in V(a, D, P_R)$ and $a \in V(q, D, P_L)$. See Figure 2.

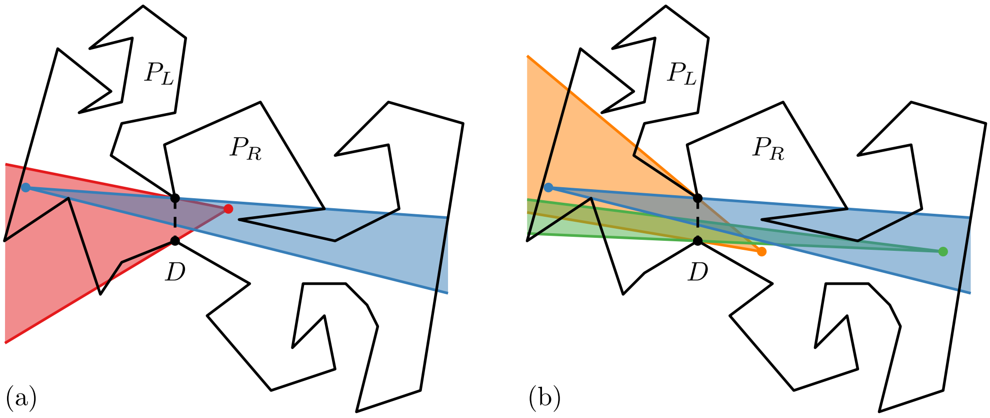


Figure 2 Visibility cones (coloured regions) of (coloured) points w.r.t. some diagonal D. (a) Blue and red are mutually visible. (b) Green and blue cannot see each other, nor can orange and blue.

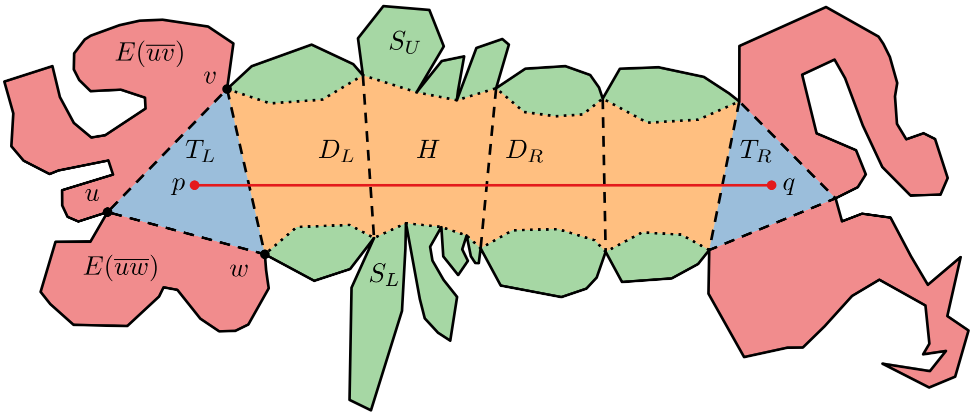


Figure 3 Partitioning of the polygon based on the polygon cover of $\overline{pq}$.

3 Segment Queries

Given a simple polygon P and a set A of points in P, we construct a data structure that efficiently counts points in A that see a query segment $\overline{pq}$. We use the data structure by Guibas and Hershberger [19] (GHDS) on P as the foundation. For a given query $\overline{pq}$, GHDS partitions P into four types of regions (Figure 3): hourglasses (orange); triangles that contain p or q, denoted by T_L and T_R (blue); regions that have as a border the upper or the lower chain of an hourglass, called *side polygons* (green); and regions that have as a border an edge of T_L or T_R, called *end polygons* (red). Each point of A belongs to only one region.

Given a segment $\overline{pq}$ at query time, we find its polygon cover. Counting the visible objects inside the relevant hourglasses and triangles is easy—all of them are visible. For the end polygons, we make a case distinction on the way the visibility cones of the objects cross the adjacent triangles, and use inclusion–exclusion-style arguments to obtain the correct count (see the full version [6]). For the side polygons, we check the conditions for an object *not* to be visible, and we subtract that from the overall count of points in the relevant side polygons.

3.1 The Data Structure

We begin by introducing a helper data structure.

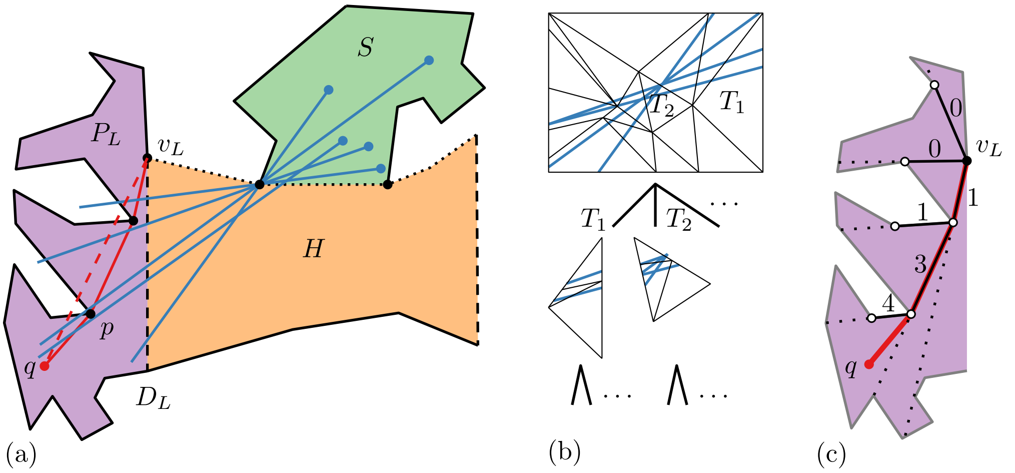


Figure 4 (a) We count the blue rays intersecting the shortest path between q and v_L. We store (b) a multilevel cutting tree to query ray intersections with $\overline{pq}$ and $\overline{qv_L}$ and (c) a shortest path map to count the rays intersecting the shortest path from v_L to p. In this case, we count $1/2 \cdot (3+1+4) = 4$.

▶ **Lemma 2.** *Let H be an hourglass bounding a side polygon S. Denote the left diagonal of H by D_L, and the polygon bounded by D_L by P_L. Let $\mathcal{R}$ be an arbitrary given set of visibility rays from objects in S into H that exit H through D_L. Denote the leftmost vertex of the convex chain separating H from S by v_L. (See Figure 4.) Given a query point $q \in P_L$ left of the supporting line of D_L, whose shortest path to v_L in P_L forms an upwards convex chain, we aim to count the rays in $\mathcal{R}$ that intersect this chain. In time $\mathcal{O}(|\mathcal{R}|^{2+\varepsilon} + |P_L| \log|\mathcal{R}|)$, we can compute a data structure of size $\mathcal{O}(|\mathcal{R}|^{2+\varepsilon} + |P_L|)$ that answers such queries in time $\mathcal{O}(\log|\mathcal{R}||P_L|)$.*

Proof sketch. Any edge of a shortest path between q and v_L is left of the supporting line of D_L, so we can use the data structure of Lemma 1 on $\mathcal{R}$ to count the rays intersecting an edge in $\mathcal{O}(|\mathcal{R}|^{2+\varepsilon})$ space and time. We compute a shortest path map with root v in P_L, where we store the number of rays intersected on the path to v_L for each edge. For a query, we can do two intersection queries and combine the counts (see Figure 4). ◀

We now introduce our Segment Query Data Structure (SQDS), based on the GHDS and augmented with extra data. It decomposes the polygon into hourglasses and triangles.

The data structure for hourglasses. Consider an hourglass H bounded by diagonals $D_L = \overline{v_L u_L}$, $D_R = \overline{v_R u_R}$, and the upper and the lower chains $\pi(v_L, v_R)$ and $\pi(u_L, u_R)$ in the GHDS. Let S_U be the (possibly degenerate) side polygon of H that is incident to the upper chain, and let $\mathcal{C}_U$ be the visibility cones of entities in S_U into H. Note that these are cones through the appropriate diagonal that forms part of $\pi(v_L, v_R)$; the cones can be computed using a cone query to the full convex chain. For the hourglass H itself, we store the number of objects in A that are contained in H. For ease of exposition, we refer to the boundaries of a cone $C \in \mathcal{C}_U$ as the *left* and the *right* boundary, when viewed from the apex of the cone in the direction of the cone. For the upper chain of H, we store in SQDS:

H1. The number of non-empty visibility cones in $\mathcal{C}_U$.
H2. The right cone boundaries of cones in $\mathcal{C}_U$ that exit H through D_R in the Lemma 2 DS.
H3. The left cone boundaries of cones in $\mathcal{C}_U$ that exit H through D_L in the Lemma 2 DS.
We store symmetrical data structures for the bottom chain of H.

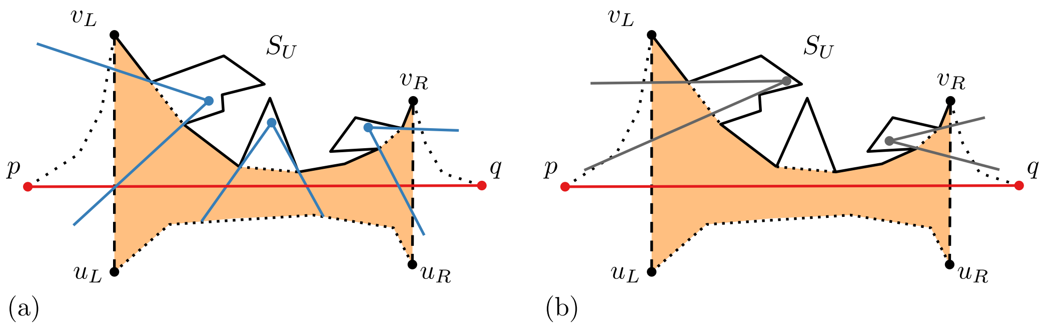


Figure 5 Cones entering from side polygon S_U that (a) see or (b) do not see $\overline{pq}$.

The data structure for triangles. We store data structures that let us count the visible objects in the end polygons and inside the triangles. Refer to the full version [6] for details.

▶ **Lemma 3.** *The SQDS requires $\mathcal{O}(nm^{2+\varepsilon} + n^2)$ space and can be constructed in time $\mathcal{O}(nm^{2+\varepsilon} + nm \log n + n^2 \log m)$.*

3.2 Counting Entities in Side Polygons

Let H be an hourglass that covers a part of query segment $\overline{pq}$ (see Figure 5).

▶ **Lemma 4.** *Let H be an hourglass with diagonals $D_L = \overline{u_L v_L}$ and $D_R = \overline{u_R v_R}$, let S_U be the (possible degenerate) side polygon bounded by the upper chain of H, and let $\overline{pq}$ be a segment that intersects both D_L and D_R, with p to the left of D_L and q to the right of D_R. Let $a \in S_U$ be a point with a non-empty visibility cone C into H. Then point a does not see $\overline{pq}$ if and only if either the right boundary R_R of C intersects $\pi(v_R, q)$, or the left boundary R_L of C intersects $\pi(v_L, p)$.*

Proof. First, assume that a does not see $\overline{pq}$. We argue that R_R intersects $\pi(v_R, q)$ or R_L intersects $\pi(v_L, p)$. Assume for the sake of contradiction that neither condition holds. Let I be the region bounded by $\overline{v_L v_R}$, $\pi(v_R, q)$, $\overline{pq}$, and $\pi(p, v_L)$. Since C can see points in H along R_R and R_L, R_R and R_L enter the region I through $\overline{v_L v_R}$, or a already lies inside I. Since R_R is a ray, it must also exit I, and by definition it cannot exit through $\overline{v_L v_R}$. It cannot exit I through $\overline{pq}$, either, as that would mean a can see $\overline{pq}$. Furthermore, by assumption, R_R does not intersect $\pi(v_R, q)$. Hence, R_R intersects $\pi(v_L, p)$. Using a similar argument, R_L intersects $\pi(v_R, q)$. It now follows that the intersection point $s = \overline{pq} \cap D_L$ lies inside the cone C, and must therefore be visible to a (i.e. nothing above H can intersect $\overline{ap}$, and inside H $\overline{ap}$ does not intersect any polygon vertices). Hence, a sees $\overline{pq}$. Contradiction.

Now assume that R_R intersects $\pi(v_R, q)$ (the case that R_L intersects $\pi(v_L, p)$ is symmetric). We now argue that a cannot see $\overline{pq}$. Let I_R be the region bounded by $\overline{pq}$, D_R, and $\pi(v_R, q)$. A point s on $\overline{pq}$ is visible via a ray R, entering via D_R, if it first exits the region I_R via $\overline{pq}$. Since R_R is not obstructed in H, it must enter I_R via D_R. In addition, by assumption, it first exits via $\pi(v_R, q)$. If R_R intersects $\pi(v_R, q)$ once, then by convexity of $\pi(v_R, q)$, it follows that q is below R_R and thus below any ray R in the cone C, thus it is not visible. If R_R intersects $\pi(v_R, q)$ twice, there is a subsegment of $\overline{pq}$ above R_R. The ray R_R now partitions I_R into three regions: one below R_R, containing points that cannot be visible, and two regions above R_R. The right region contains the subsegment of $\overline{pq}$ that is still above R_R. Consider now any ray R that could be a visibility ray to a point $x \in \overline{pq}$. This ray must be above R_R and must intersect $\overline{pq}$ at x. This means that it must traverse the region I_R from

D_R to x. But since R must be above R_R, it follows that it must cross the two regions above R_R, which are separated by a polygon boundary. Thus, no $x \in \overline{pq}$ is visible from a. ◀

▶ **Lemma 5.** *Using* **H1**, **H2**, *and* **H3** *stored with each chain of hourglass H in our SQDS, we can count the visible objects in the side polygons of H in time $\mathcal{O}(\log nm)$.*

Proof. By Lemma 4, we can count the number of visible objects from the upper side polygon by taking the total number of objects with non-empty visibility cones from the upper side polygon and subtracting those for which either R_R intersects $\pi(v_R, q)$ or R_L intersects $\pi(v_L, p)$. Counting for the lower side polygon is symmetrical. We store the number of entities with non-empty visibility cones from the side polygon in **H1**. Then, we query our **H2** and **H3** data structures to count the number of visibility cones that exit through D_L and D_R that do not see $\overline{pq}$. This requires $\mathcal{O}(\log nm)$ time per chain, yielding the total query time. ◀

There are $\mathcal{O}(\log n)$ hourglasses, so the query time is dominated by the side polygons.

▶ **Theorem 6.** *Let P be a simple polygon with n vertices, and let A be a set of m points in P. In time $\mathcal{O}(nm^{2+\varepsilon} + nm \log n + n^2 \log m)$, we can build a $\mathcal{O}(nm^{2+\varepsilon} + n^2)$-size data structure that can count the points from A that see a query segment $\overline{pq}$ in $\mathcal{O}(\log n \log nm)$ time.*

References

1 Pankaj K. Agarwal and Jiří Matoušek. Ray shooting and parametric search. *SIAM Journal on Computing*, 22(4):794–806, 1993. doi:10.1137/0222051.

2 Sharareh Alipour, Mohammad Ghodsi, Alireza Zarei, and Maryam Pourreza. Visibility testing and counting. *Information Processing Letters*, 115(9):649–654, 2015. doi:10.1016/j.ipl.2015.03.009.

3 Boris Aronov, Leonidas J. Guibas, Marek Teichmann, and Li Zhang. Visibility queries and maintenance in simple polygons. *Discrete & Computational Geometry*, 27:461–483, 2002. doi:10.1007/s00454-001-0089-9.

4 Boaz Ben-Moshe, Olaf Hall-Holt, Matthew J. Katz, and Joseph S. B. Mitchell. Computing the visibility graph of points within a polygon. In Jack S. Snoeyink and Jean-Daniel Boissonnat, editors, *Proceedings of the 20th Annual Symposium on Computational Geometry (SoCG 2004)*, pages 27–35, New York, NY, USA, 2004. ACM. doi:10.1145/997817.997825.

5 Prosenjit Bose, Anna Lubiw, and James Ian Munro. Efficient visibility queries in simple polygons. *Computational Geometry: Theory & Applications*, 23(3):313–335, 2002. doi:10.1016/S0925-7721(01)00070-0.

6 Kevin Buchin, Bram Custers, Ivor van der Hoog, Maarten Löffler, Aleksandr Popov, Marcel Roeloffzen, and Frank Staals. Segment visibility counting queries in polygons, 2022. arXiv:2201.03490.

7 Mojtaba Nouri Bygi, Shervin Daneshpajouh, Sharareh Alipour, and Mohammad Ghodsi. Weak visibility counting in simple polygons. *Journal of Computational and Applied Mathematics*, 288:215–222, 2015. doi:10.1016/j.cam.2015.04.018.

8 Bernard Chazelle. A theorem on polygon cutting with applications. In *Proceedings of the 23rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 1982)*, pages 339–349, Piscataway, NJ, USA, 1982. IEEE. doi:10.1109/SFCS.1982.58.

9 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9:145–158, 1993. doi:10.1007/BF02189314.

10 Bernard Chazelle and Leonidas J. Guibas. Visibility and intersection problems in plane geometry. *Discrete & Computational Geometry*, 4:551–581, 1989. doi:10.1007/BF02187747.

11 Bernard Chazelle, Micha Sharir, and Emo Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. *Algorithmica*, 8:407–429, 1992. doi:10.1007/BF01758854.

12 Danny Ziyi Chen and Haitao Wang. Weak visibility queries of line segments in simple polygons. *Computational Geometry: Theory & Applications*, 48(6):443–452, 2015. doi:10.1016/j.comgeo.2015.02.001.

13 Kenneth L. Clarkson. New applications of random sampling in computational geometry. *Discrete & Computational Geometry*, 2:195–222, 1987. doi:10.1007/BF02187879.

14 Mark de Berg, Dan Halperin, Mark H. Overmars, Jack S. Snoeyink, and Marc J. van Kreveld. Efficient ray shooting and hidden surface removal. *Algorithmica*, 12:30–53, 1994. doi:10.1007/BF01377182.

15 Patrick Eades, Ivor van der Hoog, Maarten Löffler, and Frank Staals. Trajectory visibility. In Susanne Albers, editor, *Proceedings of the 17th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2020)*, number 162 in Leibniz International Proceedings in Informatics (LIPIcs), pages 23:1–23:22, Dagstuhl, Germany, 2020. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.SWAT.2020.23.

16 Hossam El Gindy and David Avis. A linear algorithm for computing the visibility polygon from a point. *Journal of Algorithms*, 2(2):186–197, 1981. doi:10.1016/0196-6774(81)90019-5.

17 Subir Kumar Ghosh. *Visibility Algorithms in the Plane*. Cambridge University Press, Cambridge, UK, 2007. doi:10.1017/CBO9780511543340.

18 Joachim Gudmundsson and Pat Morin. Planar visibility: Testing and counting. In David G. Kirkpatrick and Joseph S. B. Mitchell, editors, *Proceedings of the 26th Annual Symposium on Computational Geometry (SoCG 2010)*, pages 77–86, New York, NY, USA, 2010. ACM. doi:10.1145/1810959.1810973.

19 Leonidas J. Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989. doi:10.1016/0022-0000(89)90041-X.

20 Leonidas J. Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2:209–233, 1987. doi:10.1007/BF01840360.

21 John Hershberger. A new data structure for shortest path queries in a simple polygon. *Information Processing Letters*, 38(5):231–235, 1991. doi:10.1016/0020-0190(91)90064-O.

22 John Hershberger and Subhash Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *Journal of Algorithms*, 18(3):403–431, 1995. doi:10.1006/jagm.1995.1017.

23 Barry Joe and Richard B. Simpson. Corrections to Lee's visibility polygon algorithm. *BIT Numerical Mathematics*, 27:458–473, 1987. doi:10.1007/BF01937271.

24 Der-Tsai Lee. Visibility of a simple polygon. *Computer Vision, Graphics, and Image Processing*, 22(2):207–221, 1983. doi:10.1016/0734-189X(83)90065-8.

25 Joseph O'Rourke. *Art Gallery Theorems and Algorithms*, volume 3 of *The International Series of Monographs on Computer Science*. Oxford University Press, Oxford, UK, 1987. URL: http://www.science.smith.edu/~jorourke/books/ArtGalleryTheorems/art.html.

26 Mark H. Overmars and Emo Welzl. New methods for computing visibility graphs. In Herbert Edelsbrunner, editor, *Proceedings of the 4th Annual Symposium on Computational Geometry (SoCG 1988)*, pages 164–171, New York, NY, USA, 1988. ACM. doi:10.1145/73393.73410.

27 Subhash Suri and Joseph O'Rourke. Worst-case optimal algorithms for constructing visibility polygons with holes. In Alok Aggarwal, editor, *Proceedings of the 2nd Annual Symposium on Computational Geometry (SoCG 1986)*, pages 14–23, New York, NY, USA, 1986. ACM. doi:10.1145/10515.10517.

Kinetic Group Density in 1D

Kevin Buchin[1], Max van Mulken[2], Bettina Speckmann[2], and Kevin Verbeek[2]

1 **Department of Computer Science, TU Dortmund, Germany**
kevin.buchin@tu-dortmund.de

2 **Department of Mathematics and Computer Science, TU Eindhoven, the Netherlands**
[m.j.m.v.mulken|b.speckmann|k.a.b.verbeek]@tue.nl

1 Introduction

Sets of moving entities can form groups that travel closely together for significant periods of time. There is a large body of work which describes methods to determine when and where groups are formed [3,6,7,8,10,14]. Analyzing the shape and the movement properties of such groups allows us to infer the underlying movement behavior. In this abstract we study how to characterize and kinetically maintain the density of a group. This density can capture, for example, the response of herd animals to predators or human intrusion [5,11].

Specifically, we focus on a set P of n points moving linearly in one dimension, that is, on a line. We assume that the points in P continuously form a single group. To model the density within this group we use the well-known concept of *kernel density estimation* (KDE). Given the points in P, KDE constructs a function KDE_P which estimates the density over the whole domain. KDE uses *kernels* which are functions $K : \mathbb{R} \to \mathbb{R}^+$ that capture the influence of a single point $p \in P$. The function KDE_P is the sum of the individual kernels, normalized to lie in the range $[0, 1]$. There are a variety of different kernels which are commonly used by KDE. We use the *triangular kernel function*, which is defined as $K^\Delta(x) = 1 - |x|/\sigma$ if $|x| < \sigma$, and $K^\Delta(x) = 0$ otherwise (see Figure 1). Here σ denotes the *kernel width*, which captures the influence of an individual point. We have [12,15]:

$$\text{KDE}_P^\Delta(x) = \frac{1}{n}\sum_{p\in P} K^\Delta(x-p) \qquad \text{for } x \in \mathbb{R}.$$

The local minima and maxima of KDE_P^Δ characterize the dense and sparse areas of the group P. In the remainder of this abstract we sketch how to maintain these critical points of KDE_P^Δ with the help of a *kinetic data structure* (KDS) [2] as the points in P move linearly.

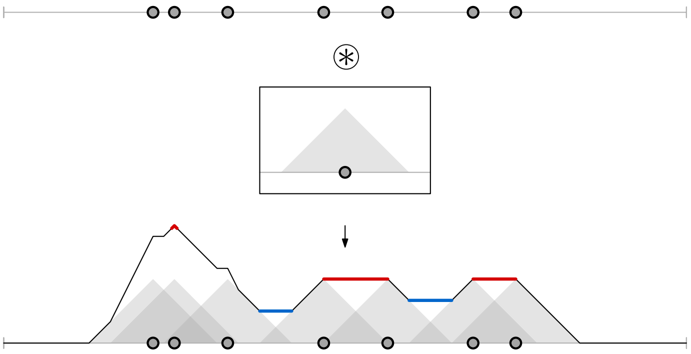


Figure 1 $\text{KDE}_P^\Delta(x)$: local maxima are indicated in red and local minima are indicated in blue.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Results. In Section 2 we present a KDS that maintains the set of critical points C of KDE_P^{Δ} of a set P of n linearly moving points in 1D. Our KDS is local, compact, responsive, and weakly efficient. However, the total number of events can be $\Theta(n^2)$. We hence show in Section 3 how to maintain an approximation of the set of critical points C via a *coreset* [13] of the 1D trajectories of the points in P. The coreset Q has size $O(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon})$, for $\varepsilon > 0$. Q does not change while the points move and hence requires no updates unless a point changes its trajectory. Using Q we can maintain a set of critical points C_Q, which approximate the critical points C well in a topological sense: we prove that C_Q contains all critical points with persistence at least 2ε. Omitted proofs can be found in the full version of the paper.

2 Exact KDS

In this section we present a KDS that maintains the critical points C of KDE_P^{Δ}. We first describe how to find the critical points of KDE_P^{Δ} for a static point set P without explicitly computing KDE_P^{Δ}. To do so, we first introduce some notation.

Let $P = \{p_1, \ldots, p_n\}$ and let $l_i = p_i - \sigma$ and $r_i = p_i + \sigma$ denote the left and right *boundary* of the kernel of $p_i \in P$. Since the kernel is triangular, the function KDE_P^{Δ} can have bends only at p_i, l_i, or r_i for $1 \le i \le n$; it must be linear in between. We refer to the bends in KDE_P^{Δ} correspondingly as bp_i, bl_i, and br_i (see Figure 2). The slope of KDE_P^{Δ} increases by $\frac{1}{\sigma n}$ at bends bl_i and br_i, and decreases by $\frac{2}{\sigma n}$ at bends bp_i. We use these slopes to determine the critical points. Note that critical points in KDE_P^{Δ} do not always consist of a single bend, but may consist of a horizontal segment, even in non-degenerate settings. These horizontal segments have three different types. If the slope of the function is increasing before and decreasing after the horizontal segment, we call the segment a *plateau* (Figure 3 left); a plateau is a type of maximum. Similarly, if the slope is decreasing before and increasing after the horizontal segment, it forms a type of minimum called a *valley* (Figure 3 center). Any other horizontal line segment is called a *flat* (Figure 3 right). The bends at the left and right endpoints of a horizontal segment are called *starting* and *ending h-bends*, respectively. We call a bend *regular* if it is neither a local maximum nor an h-bend.

For a coordinate $x \in \mathbb{R}$, we define the *left points* $P_\ell(x)$ and the *right points* $P_r(x)$ as those points of P that lie inside the open interval $(x - \sigma, x)$ and $(x, x + \sigma)$, respectively. If there is no bend located at x, then the slope of KDE_P^{Δ} at x is $\frac{|P_r(x)| - |P_\ell(x)|}{\sigma n}$. Hence, we can deduce from the number of left and right points if a bend is a local maximum or a starting or ending h-bend. Since the slope increases only by increments of $\frac{1}{\sigma n}$, a single bend can never be a local minimum, unless the point set is degenerate (two points or boundaries coincide). In the following we assume that the point set is non-degenerate.

▶ **Lemma 1.** *A bend bp_i with $p_i \in P$ is a local maximum in KDE_P^{Δ} iff $|P_\ell(p_i)| = |P_r(p_i)|$.*

Proof. Let $p_i^- = p_i - \varepsilon$ be a point just before p_i and $p_i^+ = p_i + \varepsilon$ be a point just after p_i, for some arbitrarily small $\varepsilon > 0$. Assume that bp_i is a local maximum. Then $|P_r(p_i^-)| > |P_\ell(p_i^-)|$

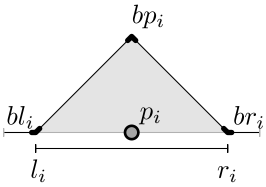


Figure 2 The triangular kernel.

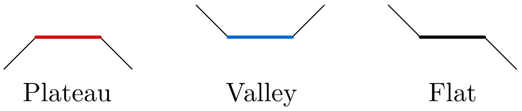


Figure 3 The three types of horizontal segments.

and $|P_r(p_i^+)| < |P_\ell(p_i^+)|$. Since $P_r(p_i^-) = P_r(p_i) \cup \{p_i\}$ and $P_\ell(p_i^+) = P_\ell(p_i) \cup \{p_i\}$ (and $P_\ell(p_i^-) = P_\ell(p_i)$ and $P_r(p_i^+) = P_r(p_i)$), we have $|P_\ell(p_i)| = |P_\ell(p_i^-)| < |P_r(p_i^-)| = |P_r(p_i)| + 1$ and $|P_r(p_i)| = |P_r(p_i^+)| < |P_\ell(p_i^+)| = |P_\ell(p_i)| + 1$. This implies that $|P_\ell(p_i)| = |P_r(p_i)|$.

Furthermore, if we assume that $|P_\ell(p_i)| = |P_r(p_i)|$, then we directly obtain that $|P_r(p_i^-)| - |P_\ell(p_i^-)| = 1$ and $|P_r(p_i^+)| - |P_\ell(p_i^+)| = -1$, and hence bp_i is a local maximum. ◀

We can determine starting and ending h-bends in a similar manner:

▶ **Lemma 2.** *For every point $p_i \in P$:*

- *bp_i is a starting h-bend if and only if $|P_r(p_i)| = |P_\ell(p_i)| + 1$*
- *bp_i is an ending h-bend if and only if $|P_\ell(p_i)| = |P_r(p_i)| + 1$*
- *bl_i is a starting h-bend if and only if $|P_\ell(l_i)| = |P_r(l_i)| + 1$*
- *bl_i is an ending h-bend if and only if $|P_\ell(l_i)| = |P_r(l_i)|$*
- *br_i is a starting h-bend if and only if $|P_\ell(r_i)| = |P_r(r_i)|$*
- *br_i is an ending h-bend if and only if $|P_\ell(r_i)| + 1 = |P_r(r_i)|$*

We can determine the type of a horizontal segment based on its starting and ending h-bends: bends bp_i decrease the slope and hence bound a plateau or one side of a flat, and bends bl_i and br_i increase the slope and hence bound a valley or the other side of a flat.

Kinetic Data Structure. We can maintain the set C of critical points of KDE_P^Δ by keeping track of the following information:

1. The number of left and right points for all points $p_i \in P$ and their boundaries l_i and r_i,
2. The order of all points p_i, l_i, and r_i in 1D (to match up starting and ending h-bends).

The left and right points of points and boundaries can change only when two points or boundaries coincide. Our KDS hence simply maintains the sorted order of all points and boundaries, and updates the left and right points accordingly (along with the classifications as critical points). In the following, let $p_i(t)$ (or $l_i(t), r_i(t)$) be the position of point $p_i \in P$ (or its boundaries) at time t. Let the time of an event be t_0. The following events can occur:

Point-Point collision ($p_i(t_0) = p_j(t_0)$). Note that we also have that $l_i(t_0) = l_j(t_0)$ and $r_i(t_0) = r_j(t_0)$. Essentially, points p_i and p_j (and their boundaries) switch places, and the critical points of KDE_P^Δ do not change. However, if p_i was a local maximum or a starting or ending h-bend before the event, then p_j takes over this role after the event, and vice versa. The same holds for the boundaries of p_i and p_j. We call this event a *shift event*.

Boundary-Boundary collision ($r_i(t_0) = l_j(t_0)$). Just before the event we have that the left and right points of r_i and l_j are the same. WLOG assume that $r_i(t) > l_j(t)$ for $t > t_0$. Hence $P_r(r_i)$ gains p_j and $P_\ell(l_j)$ gains p_i at $t = t_0$. Lemma 2 implies that if r_i is a starting (ending) h-bend before the event, then l_j becomes a starting (ending) h-bend after the event, and vice versa. Hence, this is another *shift event*.

Boundary-Point collision ($r_i(t_0) = p_j(t_0)$). We also have that $p_i(t_0) = l_j(t_0)$. WLOG assume that $r_i(t) > p_j(t)$ for $t > t_0$. Both the left- and the right points change: (1) $P_\ell(p_j)$ gains p_i, (2) $P_r(p_i)$ gains p_j, (3) p_j is removed from $P_r(r_i)$ and added to $P_\ell(r_i)$, and (4) p_i is removed from $P_r(l_j)$ and added to $P_\ell(l_j)$. Based on these changes, the types of bends and hence the set of critical points can change in various ways ; next to *shift events*, it can also happen that local maxima or h-bends are eliminated (the bends become regular). We call such events *death events*. Similarly, regular bends can become local maxima or h-bends, which we refer to as *birth events*. These updates can be processed using Lemma 1 and Lemma 2.

▶ **Theorem 3.** *Let P be a set of n linearly moving points in one dimension. The KDS described above maintains the set of critical points of $KDE_P^{\triangle}$ and is local, compact, responsive and weakly efficient.*

Proof sketch. The KDS requires certificates only for the order between the points $p_i \in P$ and their boundaries l_i and r_i. Since the set of critical points can change only when the order changes, no separate certificates are needed. Hence, we need 6 certificates for each point $p_i \in P$ (2 each for p_i, l_i, and r_i), and consequently the KDS is local and compact. When processing an event we first update the order of the points and boundaries. Next, since every event involves $O(1)$ bends, we can efficiently update the left and right points counts in $O(1)$ time. Further, we can use Lemma 1 and Lemma 2 to efficiently update the types of bends and the set C of critical points. Including the insertion of new events in the event queue, every event can be handled in $O(\log n)$ time, and hence the KDS is responsive. Finally, since we maintain the order of $3n$ elements, the total number of events is $\Theta(n^2)$ under linear motion. It is straightforward to construct an example where the number of external events (where the set of critical points changes) is also $\Theta(n^2)$: if σ is sufficiently small, then every time two points collide, two local maxima are first merged and then split again. Thus, the KDS is also weakly efficient. ◀

3 Approximation

The KDS we described in Section 2 is only weakly-efficient: there are examples where there are $\Theta(n^2)$ external events (when critical points change), but in general there might be much fewer such changes. Our KDS however tracks the sorted order of the points and hence always processes $\Theta(n^2)$ events. Furthermore, there are also external events of low significance: small "bumps" in the density function where critical points of low relevance appear and quickly disappear again. Here, we use the concept of *topological persistence* to quantify the relevance of critical points. We will briefly describe the concept of topological persistence for simple one-dimensional functions $f \colon \mathbb{R} \to \mathbb{R}$. This explanation is simplified; see e.g. [4] for a formal treatment of the subject.

For a function $f \colon \mathbb{R} \to \mathbb{R}$, let the sublevel set of f with respect to a value y be defined as $L_f(y) = \{x \mid f(x) \leq y\}$. Note that, for a "smooth" function, $L_f(y)$ generally consists of a number of intervals in the domain of f. Now consider $L_f(y)$ as we increase the value of y. If y is below the minimum of f, then $L_f(y)$ is empty. When we encounter a local minimum of f, say at x^-, then x^- is added to $L_f(y)$ when $y = f(x^-)$, and it grows into an interval of $L_f(y)$ as y is increased further. We refer to the local minimum at x^- as the *representative* of the corresponding interval. We can say that the *birth* of this interval was at $y = f(x^-)$. When we encounter a local maximum of f, say at x^+, then two intervals of $L_f(y)$ are merged into one at $y = f(x^+)$. Let x_1^- and x_2^- be the representatives of the two intervals, and assume WLOG that $f(x_1^-) < f(x_2^-)$. Then x_1^- becomes the representative of the merged interval (this is called the *elder rule*), and x_2^- and x^+ will form a new *persistence pair* between a local minimum and a local maximum. The difference $f(x^+) - f(x_2^-)$ is called the *persistence* of the persistence pair (x_2^-, x^+). We also say that the *death* of the interval of x_2^- was at $y = f(x^+)$.

We continue increasing y until $L_f(y)$ contains all points in $\mathbb{R}$. The persistence pairs generated in this process describe a *topological signature* of the function f. The idea is that persistence pairs with small persistence are mostly caused by noise, and persistence pairs with large persistence describe the main behavior of the function. Thus, we can say that critical points that are involved in a persistence pair with high persistence are more relevant for the behavior of the function than critical points that are involved in a persistence pair with low persistence. For simplicity we refer to the persistence of a critical point as the persistence of the corresponding persistence pair in which the critical point is involved.

In the following, we describe how to use *coresets* to approximate the set of critical points in such a way that (1) our KDS processes fewer events, and (2) the approximate critical points represent those with high persistence. Specifically, we replace the point set P by a smaller point set Q—the coreset. In our context, a coreset Q is an ε-approximation, for some $\varepsilon > 0$, if the following holds:

$$\max_{x\in\mathbb{R}} |\mathrm{KDE}_P^{\Delta}(x) - \mathrm{KDE}_Q^{\Delta}(x)| \leq \varepsilon.$$

KDE_Q^{Δ} may not have the same critical points as KDE_P^{Δ}. However, Cohen-Steiner *et al.* [4] show that it preserves the critical points with topological persistence at least 2ε. KDE_Q^{Δ} might also preserve some critical points of KDE_P^{Δ} with low persistence, however, the total number of critical points is bounded by $O(|Q|)$.

In one dimension we can easily compute an ε-approximation Q of P of size $O(\frac{1}{\varepsilon})$ using various methods. However, we also need to maintain this approximation as the points move, while keeping the current performance levels of the KDS with respect to locality, compactness, and responsiveness. Therefore we propose to use a coreset Q which remains fixed over the entire linear motion of the points in P. We can then build the KDS directly on Q instead of on P; we only need to update the KDS if the flight plan of one of the points in P changes.

Consider now the set of linear functions that describe the linear motion of the points in P, where the horizontal dimension is time, and the vertical dimension describes the position in 1D. These functions form a linear arrangement where each moving point is represented by a line. Every vertical slice represents the point set P at some time t and every vertical segment corresponds to an interval in 1D at some time t. The pair $(S, \mathcal{R})$, where S is the set of lines, and $\mathcal{R}$ contains all subsets of lines in S intersected by a single vertical segment, forms a *range space*. The following theorem by Agarwal *et al.* [1] gives us the tools to efficiently compute and maintain an ε-approximation of this range space.

▶ **Theorem 4** ([1, Theorem 4]). *Given a range space $X = (S, \mathcal{R})$ of VC-dimension d and a parameter ε, one can (deterministically) maintain an ε-approximation of X of size $O(\frac{1}{\varepsilon^2}\log(\frac{1}{\varepsilon}))$, in $O(\frac{\log^{2d+3} n}{\varepsilon^{2d+2}}(\log(\log(n)/\varepsilon))^{2d+2})$ time per insertion and deletion.*

The ε-approximation from Theorem 4 does not directly imply an ε-approximation for our kernel density estimation, since it applies to a uniform kernel instead of a triangular kernel. However, Joshi *et al.* [9] show that the ε-approximation of a range space is also an ε-approximation for well-behaved kernel functions. It remains to determine the VC-dimension of our range space. Using geometric point-line duality which replaces a line $ax - b$ by a point (a, b) and vice versa, our range space is equivalent to the range space on a set of points in 2D, where each range is represented by an infinite strip (with arbitrary width and orientation). The VC-dimension of this range space is 5.

We now build the KDS of Theorem 3 on the coreset Q obtained via Theorem 4. This allows us to maintain the critical points of KDE_P^{Δ} that have persistence at least 2ε. The number of points in the KDS is reduced from n to $O(\frac{1}{\varepsilon^2}\log(\frac{1}{\varepsilon}))$; it follows directly that the number of events is reduced from $\Theta(n^2)$ to $O(\frac{1}{\varepsilon^4}\log^2(\frac{1}{\varepsilon}))$. If the flight plan of a point in P changes, then we have to update coreset Q (and hence the KDS); this can be done in $O(\frac{\log^{13} n}{\varepsilon^{12}}(\log(\log(n)/\varepsilon))^{12})$ time by Theorem 4.

Future work. Our approach cannot explicitly track function values at critical points. Although this allows us to maintain the set of critical points, we cannot track relations between critical points, like persistence or contour trees. We plan to extend our approach to maintain such information efficiently. Furthermore, we intend to extend our approach to higher dimensions, specifically to 2D, and to apply our algorithms to real-world data.

References

1 Pankaj Agarwal, Mark de Berg, Jie Gao, Leonidas Guibas, and Sariel Har-Peled. Staying in the Middle: Exact and Approximate Medians in R1 and R2 for Moving Points. *Proceedings of the 17th Canadian Conference on Computational Geometry*, pages 43–46, 01 2005.

2 Julien Basch, Leonidas Guibas, and John Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1):1–28, 1999.

3 Kevin Buchin, Maike Buchin, Marc van Kreveld, Bettina Speckmann, and Frank Staals. Trajectory grouping structure. *Journal of Computational Geometry*, 6(1):75–98, 2015.

4 David Cohen-Steiner, Herbert Edelsbrunner, and John Harer. Stability of persistence diagrams. *Discrete & Computational Geometry - DCG*, 37:263–271, 2005. `doi:10.1007/s00454-006-1276-5`.

5 Jasper A.J. Eikelboom. *Sentinel animals: Enriching artificial intelligence with wildlife ecology to guard rhinos*. PhD thesis, Wageningen University, 2021.

6 Joachim Gudmundsson, Marc J. van Kreveld, and Bettina Speckmann. Efficient detection of patterns in 2D trajectories of moving points. *GeoInformatica*, 11(2):195–215, 2007. `doi:10.1007/s10707-006-0002-z`.

7 Yan Huang, Cai Chen, and Pinliang Dong. Modeling herds and their evolvements from trajectory data. *Geographic Information Science*, pages 90–105, 2008.

8 San-Yih Hwang, Ying-Han Liu, Jeng-Kuen Chiu, and Ee-Peng Lim. Mining mobile group patterns: A trajectory-based approach. *Advances in Knowledge Discovery and Data Mining*, pages 713–718, 2005.

9 Sarang Joshi, Raj Varma Kommaraji, Jeff M. Phillips, and Suresh Venkatasubramanian. Comparing distributions and shapes using the kernel distance. *Proceedings of the Twenty-Seventh Annual Symposium on Computational Geometry*, page 47–56, 2011.

10 Panos Kalnis, Nikos Mamoulis, and Spiridon Bakiras. On discovering moving clusters in spatio-temporal data. *Advances in Spatial and Temporal Databases*, pages 364–381, 2005.

11 Anders Nilsson. Predator behaviour and prey density: evaluating density-dependent intraspecific interactions on predator functional responses. *Journal of Animal Ecology*, 70(1):14–19, 2001.

12 Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, 33(3):1065–1076, 1962.

13 Jeff M. Phillips. ε-samples for kernels. In *Proceedings of the twenty-fourth annual ACM-SIAM symposium on Discrete algorithms*, pages 1622–1632. SIAM, 2013.

14 Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, 1987.

15 Murray Rosenblatt. Remarks on Some Nonparametric Estimates of a Density Function. *The Annals of Mathematical Statistics*, 27(3):832 – 837, 1956. `doi:10.1214/aoms/1177728190`.

Finding a Battleship of Uncertain Shape*

Eva-Maria Hainzl[1], Maarten Löffler[2], Daniel Perz[3], Josef Tkadlec[4], and Markus Wallinger[5]

1 **Institute of Discrete Mathematics and Geometry, TU Wien**
`eva-maria.hainzl@tuwien.ac.at`

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

3 **Institute of Software Technology, TU Graz**
`daperz@ist.tugraz.at`

4 **Department of Mathematics, Harvard University**
`tkadlec@math.harvard.edu`

5 **Algorithms and Complexity Group, TU Wien**
`mwallinger@ac.tuwien.ac.at`

Abstract

Motivated by a game of *Battleship*, we consider the problem of efficiently hitting a ship of an uncertain shape within a large playing board. Formally, we fix a dimension $d \in \{1,2\}$. A ship is a subset of $\mathbb{Z}^d$. Given a family $\mathcal{F}$ of ships, we say that an infinite subset $X \subset \mathbb{Z}^d$ of the cells *pierces* $\mathcal{F}$, if it intersects each translate of each ship in $\mathcal{F}$ (by a vector in $\mathbb{Z}^d$). In this work, we study the lowest possible (asymptotic) density $\pi(\mathcal{F})$ of such a piercing subset. To our knowledge, this problem has previously been studied only in the special case $|\mathcal{F}| = 1$ (a single ship). As our main contribution, we present a formula for $\pi(\mathcal{F})$ when $\mathcal{F}$ consists of 2 ships of size 2 each, and we identify the toughest families in several other cases. We also implement an algorithm for finding $\pi(\mathcal{F})$ in 1D.

Related Version arXiv:2202.08747

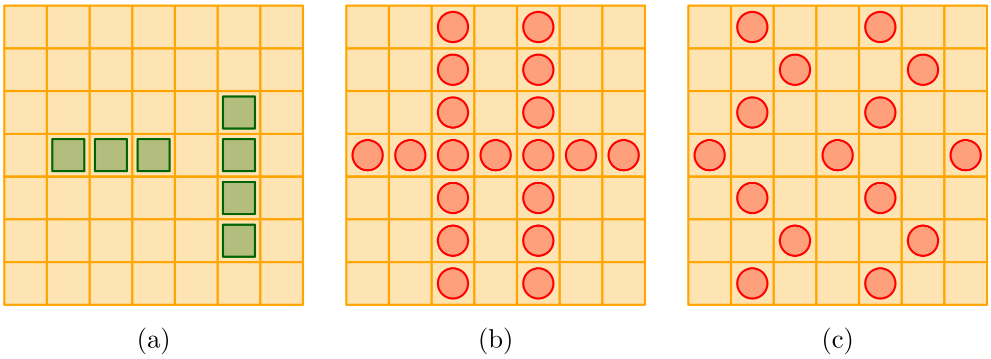


Figure 1 (a) A game with two ships: a 3×1 rectangle and a 1×4 rectangle. Note that we **do not** allow individual ships to be rotated. (b) A shooting pattern that is certain to hit both ships, no matter where they are. (c) A sparser (in fact, optimal) shooting pattern.

* This work was initiated at the 2nd Austrian Computational Geometry Reunion Workshop in Strobl, June 2021. E.-M. H. supported by the Austrian Science Foundation FWF, project F55-02. D.P. partially supported by FWF within the collaborative DACH project *Arrangements and Drawings* as FWF project I 3340-N35. M.L. partially supported by the Dutch Science Foundation (NWO) under grant number 614.001.504

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

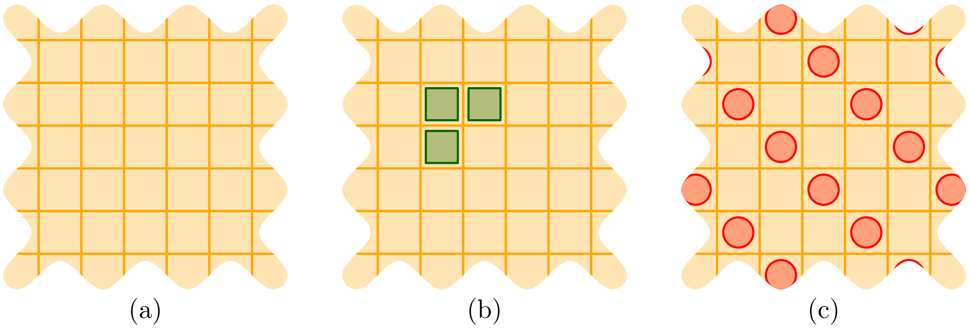


Figure 2 (a) An infinite playing board. (b) A single L-shaped ship. It may be translated, but not rotated. (c) An optimal shooting pattern with density $\frac{1}{3}$ hitting every possible translation.

1 Introduction

In a game *Battleship*, two players first secretly place a family $\mathcal{F}$ of *ships* (often rectangles) on a *domain* (an integer grid), and then they aim to locate the opponent's ships by querying individual grid cells. Inspired by the game, we consider the problem of finding a sparse *shooting pattern*: a subset of the grid cells that is guaranteed to hit at least one cell of each ship, no matter how the ships are translated within the domain. See Figure 1 for an example and Section 1.3 for a formal problem statement.

This problem is surprisingly intricate. In this note, we make two simplifying assumptions.

Infinite domains. First, in order to avoid boundary effects, we assume the domain is an infinite grid $\mathbb{Z}^d$. Since any shooting pattern on an infinite domain is also infinite, we measure its quality using the (asymptotic) *density*, refer to Figure 2. Note that the problem is subtle; for instance, for two L-shaped triominoes as ships, the lowest possible density of a shooting pattern depends on the relative orientation of the ships (see Figure 3).

▶ **Lemma 1.1.** *Let $\mathcal{F}_{180}$ and $\mathcal{F}_{90}$ be families of two L-shaped triominoes from Fig. 3. Then $\pi(\mathcal{F}_{180}) = \frac{1}{3}$ and $\pi(\mathcal{F}_{90}) = \frac{1}{2}$.*

Proof. The shooting patterns shown in Fig. 3 imply $\pi(\mathcal{F}_{180}) \le \frac{1}{3}$ and $\pi(\mathcal{F}_{90}) \le \frac{1}{2}$. For $\mathcal{F}_{180}$ the matching lower bound is trivial, since $\pi(\mathcal{F}_{180}) \ge \pi(\mathcal{F}) \ge \frac{1}{3}$, where $\mathcal{F}$ consists of a single L-shaped triomino. It remains to prove $\pi(\mathcal{F}_{90}) \ge \frac{1}{2}$. Split the plane into infinite vertical slabs of width 2. We argue for each slab separately. Fix a row. If neither cell is shot, then in the row just above it, both cells must be shot. Hence the overall density is at least $\frac{1}{2}$. ◀

One dimension. Second, for the remainder of this note we focus on the case $d = 1$, which is far from trivial if we consider not only *connected* ships, but arbitrary finite subsets of integers (see Section 1.3). In the full version, we describe how our results extend to higher dimensions. Thus, our problem is: Given a family $\mathcal{F} = \{S_1, \dots, S_n\}$ of n ships in $\mathbb{Z}$, find the minimum density $\pi(\mathcal{F})$ of a shooting pattern that hits each translate of each ship $S_i \in \mathcal{F}$.

1.1 Related work

The game *Battleship* spawned research along several fronts [3, 5]. Here we review the 1D case.

We say that a family $\mathcal{F}$ of ships in $\mathbb{Z}$ is of type $(k_1, \dots, k_n)$ if it consists of n ships with sizes $k_1 \le \dots \le k_n$, respectively. Previous work has studied families consisting of a single

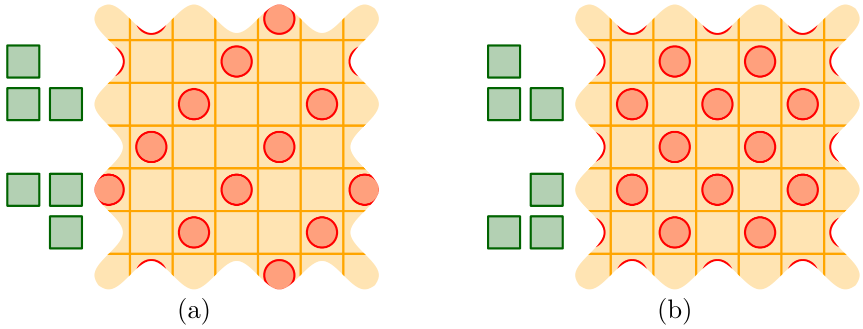


Figure 3 Two sets of two L-shaped ships of size 3. (a) Two ships, rotated 180°. The same optimal shooting pattern with density $\frac{1}{3}$ as for a single ship still works. (b) Two ships, rotated 90°. The optimal shooting pattern has density $\frac{1}{2}$ (see Lemma 1.1).

(not necessarily connected) ship, that is, families of type (k) for some $k \in \mathbb{N}$. It is easy to see that $\pi(\mathcal{F}) = 1/k$ for any (k)-family $\mathcal{F}$ with $k \in \{1,2\}$. In 2008, Schmidt and Tuller conjectured a formula for $\pi(\mathcal{F})$ for any (3)-family $\mathcal{F}$ [7], but as of now its validity is still open.

Given this difficulty, other works studied the toughest instances of a given type. Formally, given a type t, let $M_t = \sup_{\mathcal{F} \text{ has type } t}\{\pi(\mathcal{F})\}$ be the smallest density that suffices to hit any family of type t. Already in 1967, Newman [6] showed that $M_{(3)} = \frac{2}{5}$ (one toughest instance is the ship in Fig. 4(a)) and that $M_k = \Theta(\log k/k)$ as $k \to \infty$. Recently, it was shown that $M_{(4)} = \frac{1}{3}$ [1]. Also, given a ship S, the density $\pi(\{S\})$ can be found using a "sliding window" algorithm [2]. The algorithm can be used to establish lower bounds such as $M_{(5)} \geq \frac{3}{11}$.

To our knowledge, the problem for multiple ships has not been studied; however, we point out the work [4] which addresses an analogous question for rectangles in two dimensions in the continuous setting, and also explains the connection to covering density in the case of a single ship ($|\mathcal{F}| = 1$).

1.2 Our contribution

We propose to study the problem for multiple ships or, equivalently, for a single ship of uncertain shape. (Note that by considering suitable families, we can model mirrored or reflected ships on top of translated ships, cf. Fig. 3.) Apart from Lemma 1.1 above, we present results in 1D (see the full version for extensions to 2D). First, we note that the sliding window algorithm of [2] can be adapted to families of multiple ships in a straightforward way, see Theorem 1.3. We implement the algorithm and use it to obtain lower bounds such as $M_{(2,3)} \geq \frac{3}{5}$ (due to e.g. $\mathcal{F} = \{[0,1],[0,2,4]\}$). As our main contribution, we present three results for families of ships of small size k (k-ships). First, for any family $\mathcal{F}$ of two 2-ships, we find an explicit formula for $\pi(\mathcal{F})$, see Theorem 2.1. Second, we determine $M_{(2,\ldots,2)}$, that is, we identify the toughest instances for families consisting of any number of 2-ships, see Theorem 2.2. Third, we determine the toughest instances for families consisting of any 3-ship together with its reflection, see Theorem 2.3. Finally, we present bounds for the density of the toughest instances of n ships of size k each (with proofs in the full version).

1.3 Preliminaries

A ship of size k (a k-*ship*) is a k-tuple $[a_1, a_2, \ldots a_k]$ with $a_i \in \mathbb{Z}, i \geq 1$. A *span* of a ship S is $\mathrm{sp}(S) = a_k - a_1 + 1$. See Fig. 4 for an illustration. A finite family of ships $\mathcal{F} = \{S_1, S_2, \ldots S_n\}$

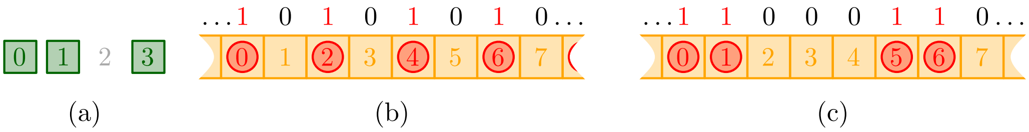


Figure 4 (a) A single disconnected 1-dimensional ship $S = [0, 1, 3]$ of size $k = 3$ and span $\mathrm{sp}(S) = 4$. (b-c) Possible periodic shooting patterns for $\{S\}$, with densities $\frac{1}{2}$ and $\frac{2}{5}$, respectively.

has a span $\mathrm{sp}(\mathcal{F}) = \max_{S\in\mathcal{F}} \mathrm{sp}(S)$. A *shooting pattern* is a 01-sequence $X = (x_i)_{i\in\mathbb{Z}}$. We say that a shooting pattern *hits* a k-ship $S = [a_1, a_2, \dots a_k]$ (or that X is a shooting pattern for S) if $\sum_{i=1}^{k} x_{n+a_i} \geq 1$, $\forall n \in \mathbb{Z}$. The density of a shooting pattern X is defined as $\pi(X) = \lim_{N\to\infty} \frac{\sum_{|i|\leq N} x_i}{2N+1}$. The density of a ship S and of a family $\mathcal{F}$ of ships is then defined as

$$\pi(S) = \inf_{X \text{ hits } S} \pi(X) \quad \text{and} \quad \pi(\mathcal{F}) = \inf_{X \text{ hits each } S\in\mathcal{F}} \pi(X).$$

Further, we define $m_k^n = \inf\{\pi(\{S_1, \dots, S_n\}) \mid |S_i| = k \text{ for } i = 1, \dots, n\}$ and $M_k^n = \sup\{\pi(\{S_1, \dots, S_n\}) \mid |S_i| = k \text{ for } i = 1, \dots, n\}$. That is, m_k^n is the required density for the simplest instances, whereas M_k^n is the required density for the toughest instances, among families that consist of n ships of size k each. Regarding m_k^n, it is straightforward to prove $m_k^n = \frac{1}{k}$ (even when no two ships in the family are translates of each other). Regarding M_k^n, in the full version we establish non-trivial upper and lower bounds. Here, we just state those results without proof, together with two other auxiliary results (whose proofs can be found in the full version too).

▶ **Theorem 1.2.** *Let $n \geq 1$ and $k \geq 2$ be integers. Then $m_k^n = \frac{1}{k}$ and*

$$1 - \frac{e}{\sqrt[k-1]{n}} \leq M_k^n \leq \min\left\{\frac{n}{n+1}, \frac{1 + \log(kn)}{k}\right\}.$$

▶ **Theorem 1.3** (Sliding window algorithm). *Given a family $\mathcal{F}$ with span $s = \mathrm{sp}(\mathcal{F})$, the density $\pi(\mathcal{F})$ can be computed in time polynomial in 2^s.*

Finally, for an integer d and a ship $S = [a_1, \dots, a_k]$ let $dS = [da_1, \dots, da_k]$, and likewise for a family $\mathcal{F} = \{S_1, \dots, S_n\}$ let $d\mathcal{F} = [dS_1, \dots, dS_n]$.

▶ **Lemma 1.4.** *Let d be a positive integer and $\mathcal{F}$ any family. Then $\pi(\mathcal{F}) = \pi(d\mathcal{F})$.*

2 Families of 2-ships and 3-ships

Here we study families $\mathcal{F}$ that consist of n ships of small size $k \leq 3$ each. Note that when $k = 1$, we obviously have $\pi(\mathcal{F}) = 1$ (for all $n \geq 1$). Also, for a single 2-ship S it is straightforward to show that $\pi(\{S\}) = 1/2$. Our first non-trivial result is an explicit formula for $\pi(\mathcal{F})$ when $\mathcal{F}$ consists of two 2-ships.

▶ **Theorem 2.1** (Formula for two 2-ships). *Let $\mathcal{F} = \{[0, da], [0, db]\}$, a, b coprime and $d \geq 1$.*

$$\pi(\mathcal{F}) = \begin{cases} 1/2 & \text{if both } a \text{ and } b \text{ are odd,} \\ \frac{a+b+1}{2(a+b)} & \text{otherwise.} \end{cases}$$

Proof. Let $\mathcal{F}' = \{[0, a], [0, b]\}$. By Lemma 1.4, it suffices to determine $\pi(\mathcal{F}')$. Clearly, $\pi(\mathcal{F}') \geq \pi(\{[0, a]\} = 1/2$. When both a and b are odd, a shooting pattern X defined by

"$x_i = 1$ if and only if i is even" provides a matching construction. From now on, assume that precisely one of a, b is odd (that is, $a + b$ is odd).

Split $\mathbb{Z}$ into blocks of $a + b$ consecutive integers. Let $S = \{0, \dots, a + b - 1\}$ be one such block and let X' be a shooting pattern for $\mathcal{F}'$ on S (instead of on $\mathbb{Z}$). We will argue that S needs to be hit at least $(a + b + 1)/2$ times (that is, $\sum_{i \in S} x'_i \geq (a + b + 1)/2$). Consider a graph $G = (S, E)$ with nodes S and directed edges $E = \{(u, v) \mid v - u \in \{a, -b\}\}$. This graph records the constraints on the shooting pattern: For every edge $(u, v) \in E$, we must have $x_u + x_v \geq 1$. Since $|S| = a + b$, each node in G has indegree 1 and outdegree 1. Moreover, since a and b are coprime, the graph G is connected. Hence it is a directed cycle on an odd number $a + b$ of nodes. Its minimum vertex cover has size $(a + b + 1)/2$, thus $\pi(\mathcal{F}') \geq (a + b + 1)/2$.

To prove that this bound is tight, consider any vertex cover $C \subseteq S$ of G of the minimum size $(a + b + 1)/2$. Then the $(a + b)$-periodic shooting pattern X^C defined by "$x_i^C = 1$ if and only if $i \pmod{(a + b)} \in C$" hits $\mathcal{F}'$ on $\mathbb{Z}$: Indeed, consider any translate $(n, n + a)$ of the ship $[0, a]$. Suppose $n \equiv r \pmod{(a + b)}$ for some $0 \leq r < a + b$. If $r < b$ then n and $n + a$ both belong to the same block, thus $x_n + x_{n+a} \geq 1$, since C is a vertex cover. On the other hand, if $r \geq b$ then by the $(a + b)$-periodicity of the shooting pattern we have $x_{n+a} = x_{n-b}$. Since $r \geq b$, both $n - b$ and n belong to the same block, so we conclude as before. For translates of the ship $[0, b]$ we argue analogously. ◀

As a corollary, we have $M_2^2 = 2/3$, as witnessed by families $\{[0, d], [0, 2d]\}$ for any $d \geq 1$.

Next, we study the toughest instances in two other cases, namely for any number of 2-ships, and for a 3-ship together with its reflection.

▶ **Theorem 2.2** (Toughest families of 2-ships). *For any $n \geq 1$ we have $M_2^n = n/(n + 1)$.*

Proof. For $n = 1$ the claim is trivial. For $n = 2$ it follows from Theorem 2.1. Assume $n \geq 3$.

First, note that for a family $\mathcal{F}_n = \{[0, 1], \dots, [0, n]\}$ we have $\pi(\mathcal{F}_n) = n/(n + 1)$: Indeed, split $\mathbb{Z}$ into blocks of $n + 1$ consecutive integers. Then any shooting pattern X_n for $\mathcal{F}_n$ may miss at most 1 number from each block. On the other hand, the pattern X_n defined by "$x_i = 0$ if and only if i is a multiple of $n + 1$" hits $\mathcal{F}_n$.

To prove the upper bound, consider any n positive integers $a_1 < \dots < a_n$, and the corresponding family $\mathcal{F} = \{[0, a_1], \dots, [0, a_n]\}$. We construct a shooting pattern X for $\mathcal{F}$ with density at most $n/(n + 1)$. We proceed in steps. Initially, we set $x_t = 1$ for all t with $|t| \leq a_n$. Then, we process integers $t > a_n$ in increasing order. Whenever x_t is not yet set, we set $x_t = 0$ and $x_{t+a_i} = 1$ for each $i = 1, \dots, n$. (Note that some of x_{t+a_i} might have already been set to 1, due to some $t' < t$.) By construction, X hits all translates of $\mathcal{F}$ within the interval $[a_n + 1, \infty)$. Moreover, since for every x_t set to 0 there are at most n values newly set to 1, in the limit $t \to \infty$ we obtain $\pi(X[a_n + 1, \infty)) \leq n/(n + 1)$. Similarly, we process integers $t < -a_n$ in decreasing order and get $\pi(X(-\infty, -a_n - 1]) \leq n/(n + 1)$. Together with the finite initial segment $X[-a_n, a_n]$ this gives $\pi(X) = \pi(X(-\infty, \infty)) \leq n/(n + 1)$. ◀

Given a ship $S = [a_1, \dots, a_k]$, let $\overline{S} = [-a_k, \dots, -a_1]$ be its reflection.

▶ **Theorem 2.3** (Toughest 3-ship with its reflection). *Let S be a 3-ship and let $\mathcal{F} = \{S, \overline{S}\}$. Then $\pi(\mathcal{F}) \leq \frac{2}{5}$, with equality if and only if $S \in \{[0, 2d, 3d], [0, 3d, 4d]\}$ (or their reflections) for some $d \geq 1$.*

Proof. First, we argue that for a single 3-ship S, the toughest instance has a density of $\frac{2}{5}$; that is, $M_3^1 = \frac{2}{5}$. This fact was first proven by Newman [6]. Here, we present a geometric proof, which we then extend to the case of two symmetric 3-ships.

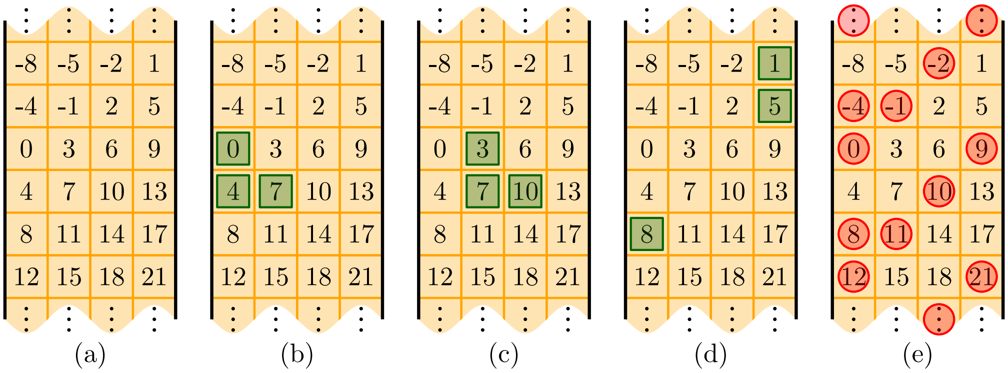


Figure 5 Illustration of the solution for $S = [0, 4, 7]$. (a) A layout of the integers into an infinite vertical slab of width 4. (b-c) Every translation of S corresponds to a triple of cells that form an L-shape. (d) If the L-shape falls on the edge of the slab, the pieces "wrap around" but are shifted vertically. (e) A valid shooting pattern.

Consider a 3-ship $S = [0, a, a+b]$ for positive integers a and b with $\mathrm{GCD}(a, b) = 1$ and $a \geq b$. We arrange the integers into a 2-dimensional grid $\{0, \ldots, a-1\} \times \mathbb{Z}$ by the bijection $(i, j) \mapsto ib + ja$. Refer to Figure 5(a). Most translations of S now correspond to an L-triomino with the same orientation; therefore, we can hit all of them with a shooting pattern with density $\frac{1}{3}$ using the same solution as in Figure 2. However, this misses exactly the translations by an amount that is congruent to $-b \mod a$; those translations correspond to a triomino that "wraps around" (Figure 5(d)). To hit these it is sufficient to increase the density of the first column to $\frac{2}{3}$. This gives $\pi(\{S\}) \leq (a+1)/(3a)$, which is strictly less than $2/5$ for $a \geq 6$. For the remaining 10 cases with $b < a \leq 5$, we find an optimal solution by Theorem 1.3. The toughest instances, yielding $\pi(S) = \frac{2}{5}$, turn out to be $S \in \{[0, 2, 3], [0, 3, 4]\}$ as claimed.

Now, fix S and consider a family $\mathcal{F} = \{S, \overline{S}\}$. We claim that $\pi(\mathcal{F}) \leq 2/5$ as well. Indeed, as in Lemma 1.1, when $a \geq 6$, the solution described above hits not only every translate of S but also every translate of $\overline{S}$. For the 10 cases with $b < a \leq 5$, using Theorem 1.3 we again verify that all such families $\mathcal{F} = \{S, \overline{S}\}$ satisfy $\pi(\mathcal{F}) \leq \frac{2}{5}$, with equality for $S \in \{[0, 2, 3], [0, 3, 4]\}$. ◀

We note that Theorems 2.1, 2.2 and 2.3 can be generalized to higher dimensions. Notes on these extensions can be found in the full version.

	M_k^n	# ships, n 1	2	3
ship size, k	1	$= 1$	$= 1$	$= 1$
	2	$= 1/2 = 0.5$	$= 2/3$ [Thm 2.1]	$= 3/4$ [Thm 2.2]
	3	$= 2/5 = 0.4$ [6]	$\geq 1/2 = 0.5$	$\geq 5/9 \doteq 0.56$
	4	$= 1/3 \doteq 0.33$ [9]	$\geq 2/5 = 0.4$	$\geq 4/9 \doteq 0.44$
	5	$\geq 3/11 \doteq 0.27$ [2]	$\geq 1/3 \doteq 0.33$	$\geq 1/3 \doteq 0.33$
	6	$\geq 1/4 = 0.25$ [2]	$\geq 3/11 \doteq 0.27$	$\geq 3/11 \doteq 0.27$

Figure 6 Bounds on the density M_k^n of the toughest instances among the families with n ships of size k each, found by running the algorithm in Theorem 1.3 for all families with span up to $11 - n$.

3 Conclusions

We introduced the problem of locating a battleship of an uncertain shape. Given the difficulty of the problem in general, we focused on the simplest possible setting, namely ships of size 2 or 3 in 1D (see the full version for extensions to 2D). We also implemented an algorithm for computing $\pi(\mathcal{F})$ in 1D and used it to compute lower bounds on the minimum density M_k^n required for the toughest families of $n \leq 3$ ships of size $k \leq 6$ each, see Fig. 6. Many open problems arise, e.g.:

1. Which values in Fig. 6 are tight? For instance, is it true that $M_3^2 = 1/2$?
2. What are the asymptotics of $1 - M_k^n$ for fixed small $k \geq 3$?
3. In 2D, is there an algorithm for computing $\pi(\mathcal{F})$?

References

1 M Axenovich, J Goldwasser, B Lidicky, R Martin, D Offner, J Talbot, and M Young. Polychromatic colorings on the integers. *Integers: Electronic Journal of Combinatorial Number Theory*, 19(A18), 2019.

2 Béla Bollobás, Svante Janson, and Oliver Riordan. On covering by translates of a set. *Random Structures & Algorithms*, 38(1-2):33–67, 2011.

3 Loïc Crombez, Guilherme D. da Fonseca, and Yan Gerard. Efficient Algorithms for Battleship. In Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara, editors, *10th International Conference on Fun with Algorithms (FUN 2021)*, volume 157 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 11:1–11:15, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum für Informatik. URL: https://drops.dagstuhl.de/opus/volltexte/2020/12772, doi:10.4230/LIPIcs.FUN.2021.11.

4 Adrian Dumitrescu and Josef Tkadlec. Piercing all translates of a set of axis-parallel rectangles. *arXiv preprint arXiv:2106.07459*, 2021.

5 Amos Fiat and Adi Shamir. How to find a battleship. *Networks*, 19(3):361–371, 1989.

6 DJ Newman. Complements of finite sets of integers. *Michigan Mathematical Journal*, 14(4):481–486, 1967.

7 Wolfgang M Schmidt and David M Tuller. Covering and packing in zn and rn,(i). *Monatshefte für Mathematik*, 153(3):265–281, 2008.

Linear size universal point sets for classes of planar graphs*

Stefan Felsner[1], Hendrik Schrezenmaier[1], Felix Schröder[1], and Raphael Steiner[2]

1 **Institut für Mathematik,**
Technische Universität Berlin, Germany
`{felsner,fschroed,schrezen}@math.tu-berlin.de`

2 **Institut für Theoretische Informatik,**
Eidgenössische Technische Hochschule Zürich, Switzerland
`raphaelmario.steiner@inf.ethz.ch`

Abstract

A finite point set $P \subseteq \mathbb{R}^2$ is n-universal with respect to a class $\mathcal{G}$ of planar graphs if every n-vertex-graph $G \in \mathcal{G}$ admits a crossing-free straight-line drawing with vertices being placed at points of P. A widely studied problem in graph drawing is to identify small universal point sets.

For the class of all planar graphs the best known upper bound on the size of a universal point set is quadratic and the best known lower bound is linear in n. One of the classical results in the area is that every set of n points in general position (no three collinear) is n-universal for outerplanar graphs. While some other classes are known to admit universal point sets of near linear size, we are not aware of truly linear bounds for interesting classes beyond outerplanar graphs.

In this paper we study a specific ordered point set H (the exploding double chain) and show that all planar graphs G on $n \geq 2$ vertices which are subgraphs of a planar graph admitting a one-sided Hamiltonian cycle have a straight-line drawing on the initial piece H_n of size $2n-2$ in H. Let $\mathcal{H}'$ be the class of all subgraphs of planar graphs admitting a one-sided Hamiltonian cycle. It had been conjectured that all 4-connected triangulations belong to $\mathcal{H}'$. While the conjecture has been disproved, it is still true that H_n is n-universal for a large class $\mathcal{H}$ of planar graphs. We show that all bipartite plane graphs and all cubic plane graphs belong to $\mathcal{H}' \subseteq \mathcal{H}$. Remarkably, however, not all 2-trees are in $\mathcal{H}'$.

1 Introduction

Given a family $\mathcal{G}$ of planar graphs and a positive integer n, a point set $P \subseteq \mathbb{R}^2$ is called an *n-universal point set* for the class $\mathcal{G}$ or simply *n-universal* for $\mathcal{G}$ if for every graph $G \in \mathcal{G}$ on n vertices there exists a straight-line crossing-free drawing of G such that every vertex of G is placed at a point of P. It is a widely studied and fundamental open problem in geometric graph theory (compare also the entry [16] in the Open Problem Garden) to determine, given a class of graphs $\mathcal{G}$, (the asymptotics of) the minimum size $f_{\mathcal{G}}(n)$ of an n-universal point set for $\mathcal{G}$. If $\mathcal{G}$ is the class of all planar graphs we simply write $f(n) := f_{\mathcal{G}}(n)$.

Schnyder [20] showed that for $n \geq 3$ the $[n-1] \times [n-1]$-grid forms an n-universal point set for planar graphs, even if the combinatorial embedding of the planar graph is prescribed. This shows that $f(n) < n^2 = O(n^2)$. Asymptotically, the quadratic upper bound on $f(n)$ remains the state of the art. However, the multiplicative constant in this bound has been improved, see [4, 5]. The current upper bound is $f(n) \leq \frac{1}{4}n^2 + O(n)$ by Bannister et al. [4].

[*] This research was supported by the German Research Foundation DFG Project FE 340/12-1. R. Steiner was funded by DFG-GRK 2434 Facets of Complexity.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

For several subclasses $\mathcal{G}$ of planar graphs, better upper bounds are known: A classical result by Pach et al. [18] is that every outerplanar n-vertex graph embeds straight-line on *any* set of n points in general position, and hence $f_{\text{out-pl}}(n) = n$. Near-linear upper bounds of $f_{\mathcal{G}}(n) = O(n \operatorname{polylog}(n))$ are known for 2-outerplanar graphs, simply nested graphs, and for the classes of bounded pathwidth [3, 4]. Finally, for the class $\mathcal{G}$ of planar 3-trees (also known as Apollonian networks or stacked triangulations), an upper bound of $f_{\mathcal{G}}(n) = O(n^{3/2} \log n)$ has been proved by Fulek and Tóth [12].

As for lower bounds, the trivial bounds $n \leq f_{\mathcal{G}}(n) \leq f(n)$ hold for all $n \in \mathbb{N}$ and all planar graph classes $\mathcal{G}$. The currently best lower bound $f(n) \geq 1.293n - o(n)$ from [19] has been shown using planar 3-trees, we refer to [6, 14, 8, 9] for earlier work on lower bounds.

It seems that in order to improve the quadratic upper bound on $f(n)$ to $o(n^2)$, the considered point sets should be not too uniformly distributed. Indeed, Choi, Chrobak and Costello [7] recently proved that point sets chosen uniformly at random from the unit square must have size $\Omega(n^2)$ in order to be universal for n-vertex planar graphs, with high probability.

In this paper we study a specific ordered point set H (the exploding double chain) and let H_n be the initial piece of size $2n - 2$ in H (for $n \geq 2$). Throughout the paper, let $\mathcal{H}$ be the class of all planar graphs G which have a plane straight line drawing on the point set H_n where $n = |V(G)|$. That is, H_n forms an n-universal point set for $\mathcal{H}$.

A graph is POSH (partial one-sided Hamiltonian) if it is a spanning subgraph of a graph admitting a plane embedding with a one-sided Hamiltonian cycle[1]). Our main result (Theorem 2.1) is that every POSH graph is in $\mathcal{H}$. We let $\mathcal{H}' := \{G : G \text{ is POSH}\}$.

Theorem 2.1 motivates the study of $\mathcal{H}'$. This class of planar graphs seems to be quite large, e.g., the smallest 4-connected triangulation which is known not to be POSH has 113 vertices [2]. On the positive side we show that every bipartite plane graph is POSH (proof sketch in Section 3). In the full paper we use the construction for bipartite graphs to show that cubic plane graphs are POSH; Section 4 gives an overview of the proof method. The full paper also contains a negative result, namely that not all 2-trees are POSH. We conclude with some conjectures and open problems in Section 5.

An exploding double chain was previously used by Löffler and Tóth [15]. They show that every planar graph with n vertices has a 1-bend drawing on a subset S_n of H with $|S_n| = 6n - 10$. Note that our result about bipartite graphs implies a better bound: The dual of a plane triangulation has a perfect matching. Hence, subdividing at most $n - 2$ edges is enough to make any planar graph on n vertices bipartite, therefore, a subset of H of size $2(n + n - 2) - 2 = 4n - 6$ is large enough to accommodate 1-bend drawings of all planar graphs with n vertices. Universality for 1-bend and 2-bend drawings has been studied by Kaufmann and Wiese [13], they show that every n element point set is universal for 2-bend drawings of planar graphs.

2 The point set and the embedding strategy

In this section we define an ordered point set H and a class $\mathcal{H}'$ of planar graphs and show that for every $n \geq 2$ the initial part H_n of size $2n - 2$ is n-universal for the class $\mathcal{H}'$.

A sequence $Y = (y_i)_{i \geq 1}$ of real numbers satisfying $y_1 = 0$, $y_2 = 0$, and $y_{i+1} > 2y_i + y_{i-1}$ for all $i \geq 2$ is called *exploding*. Note that if $\alpha > 1 + \sqrt{2}$, then $y_1 = y_2 = 0$ and $y_i = \alpha^{i-3}$ for $i \geq 3$ is an exploding sequence. Given an exploding sequence Y let $P(Y) = (p_i)_{i \geq 1}$ be the set of points with $p_i = (i, y_i)$ and let $\bar{P}(Y) = (q_i)_{i \geq 1}$ be the set of points with

[1] The precise definition of a one-sided Hamilton cycle is given below Figure 1 on page 3.

$q_i = (i, -y_i)$, i.e., the point set reflected at the x-axis, and note that $p_1 = q_1$ and $p_2 = q_2$. Let $H(Y) = P(Y) \cup \bar{P}(Y)$ and $H_n(Y) = \{p_i, q_i | 1 \leq i \leq n\}$ so that $|H_n(Y)| = 2n - 2$. Figure 1 illustrates $H_6(Y)$.

Let $H = H(Y)$ for some exploding sequence Y. For two points p and q let $H(p, q)$ be the set of points of H in the open right half-plane of the directed line $\overrightarrow{pq}$. Note that[2]

$$H(p_i, q_j) = \begin{cases} (p_k)_{k \leq j} \cup (p_k)_{k > i} \cup (q_\ell)_{\ell < j} & \text{if} \quad i > j \\ (p_k)_{k < i} \cup (q_\ell)_{\ell < i} & \text{if} \quad i = j \\ (p_k)_{k < i} \cup (q_\ell)_{\ell \leq i} \cup (q_\ell)_{\ell > j} & \text{if} \quad i < j \end{cases}$$

Moreover, if $i < j$ then $H(q_i, q_j) = H(p_i, q_j) \backslash \{q_i\}$ and if $i > j$ then $H(p_i, p_j) = H(p_i, q_j) \backslash \{p_j\}$. These sidedness conditions characterize the order type of H. A point set $A = \{p_i, q_i | i \geq 1\}$ is an *exploding double chain* if it has the order type of H.

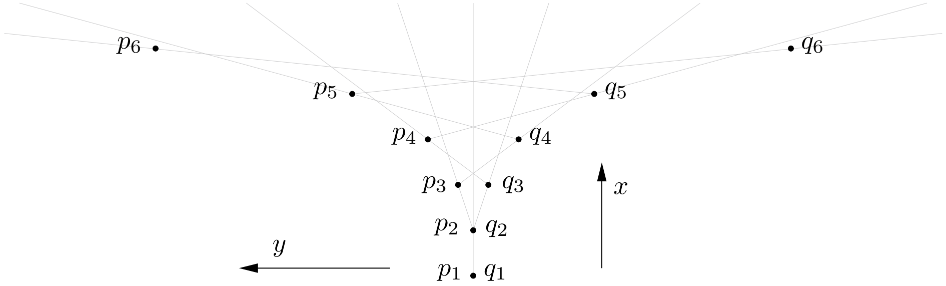


Figure 1 An example of a point set H_6 in a rotated coordinate system ($p_i = q_i$ for $i = 1, 2$).

A plane graph G has a *one-sided Hamiltonian cycle* with reverse edge vu if it has a Hamiltonian cycle $(v = v_1, v_2, \ldots, v_n = w)$ such that uv is incident to the outer face and for every $j = 2, \ldots, n$ in the induced subgraph $G[v_1, \ldots, v_j, v_{j+1}]$ of G the edges $v_{j-1}v_j$ and $v_{j+1}v_j$ are consecutive in the rotation of v_j. A more visual reformulation of the second condition is that in the embedding of G for every j either all the back-edges v_iv_j with $i < j$ are drawn inside the closed bounded region D whose boundary is the Hamiltonian cycle or they are all drawn in the closed region outside of the cycle. We let V_I be the set of vertices v_j which have a back-edge v_iv_j with $i < j - 1$ drawn inside D and $V_O = V \setminus V_I$.

In the context of cartograms Alam et al. [1] conjectured that every plane 4-connected triangulation has a one-sided Hamiltonian cycle. Later Alam and Kobourov [2] found a plane 4-connected triangulation on 113 vertices which has no one-sided Hamiltonian cycle.

Recall that $\mathcal{H}'$ is the class of POSH graphs, i.e., of planar graphs which are spanning subgraphs of plane graphs admitting a one-sided Hamiltonian cycle. Our interest in this class is motivated by the following theorem.

▶ **Theorem 2.1.** *Let G' be POSH and let $v_1, \ldots, v_n$ be a one-sided Hamiltonian cycle of a plane supergraph G of G' on the same vertex set. Then there is a crossing-free embedding of G' on H_n with the property that v_i is placed on either p_i or q_i.*

Proof. It is sufficient to describe the embedding of the supergraph G on H_n. For the proof we assume that in the plane drawing of G the sequence $v_1, \ldots, v_n$ traverses the boundary

[2] In cases where i or j are in $\{1, 2\}$ the following may list one of the two points defining the halfspace with its second name as member of the halfspace. For correctness such listings have to be ignored.

of D in counter-clockwise direction. For each i vertex v_i is embedded at $\bar{v}_i = p_i$ if $v_i \in V_I$ and at $\bar{v}_i = q_i$ if $v_i \in V_O$.

Let $G_i = G[v_1, \dots, v_i]$ be the subgraph of G induced by $\{v_1, \dots, v_i\}$. The path $\Lambda_i = v_1, \dots, v_i$ separates G_i, the *left part* GL_i consists of the intersection of G_i with D, the *right part* GR_i is G_i minus all edges which are interior to D. The intersection of GL_i and GR_i is Λ_i and their union is G_i. The counter-clockwise boundary walk of G_i consists of a path ∂R_i from v_1 to v_i which is contained in GR_i and a path from v_i to v_1 which is contained in GL_i, let ∂L_i be the reverse of this path.

Let $\bar{G}_i$ be the straight line drawing of the plane graph G_i induced by placing each vertex v_j at the corresponding $\bar{v}_j$. A vertex $\bar{v}$ of $\bar{G}_i$ is said to *see a point* p if there is no crossing between the segment $\bar{v}p$ and an edge of $\bar{G}_i$. By induction on i we show:

1. The drawing of $\bar{G}_i$ is plane, i.e., non-crossing.
2. $\bar{G}_i$ and G_i have the same outer boundary walks.
3. Every vertex of ∂L_i in $\bar{G}_i$ sees all the points p_j with $j > i$ and every vertex of ∂R_i in $\bar{G}_i$ sees all the points q_j with $j > i$.

For $i = 2$ the graph G_i is just an edge and the three claims are immediate, for property 3 just recall that the line spanned by p_1 and p_2 separates the p-side and the q-side of H_n.

Now assume that $i \in \{3, \dots, n\}$, the properties are true for $\bar{G}_{i-1}$ and suppose that $v_i \in V_I$ (the argument in the case $v_i \in V_O$ works symmetrically). This implies that all the back-edges of v_i are in the interior of D whence all the neighbors of v_i belong to ∂L_{i-1}. Since $v_i \in V_I$ we have $\bar{v}_i = p_i$ and property 3 of $\bar{G}_{i-1}$ implies that the edges connecting to $\bar{v}_i$ can be added to $\bar{G}_{i-1}$ without introducing a crossing. This is property 1 of $\bar{G}_i$.

Since G_{i-1} and $\bar{G}_{i-1}$ have the same boundary walks and v_i respectively $\bar{v}_i$ belong to the outer faces of G_i and $\bar{G}_i$ and since v_i has the same incident edges in G_i as $\bar{v}_i$ in $\bar{G}_i$, the outer walks of G_i and $\bar{G}_i$ again equal each other, i.e., property 2.

Let j be minimal such that $v_j v_i$ is an edge and note that ∂L_i is obtained by taking the prefix of ∂L_{i-1} whose last vertex is v_j and append v_i. The line spanned by $\bar{v}_j$ and $\bar{v}_i = p_i$ separates all the edges incident to $\bar{v}_i$ in $\bar{G}_i$ from all the segments $\bar{v}_\ell p_k$ with $\ell < j$ and $\bar{v}_\ell \in \partial L_i$ and $k > i$. This shows that every vertex of ∂L_i in $\bar{G}_i$ sees all the points p_k with $k > i$. For the proof of the second part of property 3 we refer to Figure 2, it shows that the new edges $\bar{v}_j p_i$ do not obstruct the visibility between vertices of ∂R_i and any q_k with $k > i$. Of course this can also be derived formally by translating the condition for a crossing between two segments into sidedness conditions and then compare with the sidedness conditions given for the order type of H. This completes the proof of property 3 and thus the inductive step.

Finally, property 1 for G_n implies the theorem. ◀

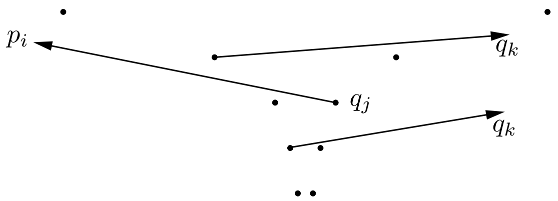


Figure 2 Vertices from ∂R_i see q_k

3 Plane bipartite graphs

In this section we consider bipartite plane graphs and sketch a proof that they are POSH.

▶ **Theorem 3.1.** *Every bipartite plane graph $G = (V, E)$ is a spanning subgraph of a plane graph G' on the same vertex set V which has a one-sided Hamiltonian cycle, i.e., G is POSH.*

Quadrangulations are the plane graphs with all faces of degree four. Equivalently they are edge-maximal plane bipartite graphs. Every connected bipartite plane graph with at least two vertices in each color class is a spanning subgraph of a plane quadrangulation. Therefore it suffices to prove the theorem for plane quadrangulations.

A *separating decomposition* of a quadrangulation is an orientation and 2-coloring of the edges, such that two vertices s and t which are diagonally opposite on the outer 4-face only have incoming edges in red and blue respectively, while each other vertex has outgoing edges in both colors as shown in Figure 3.

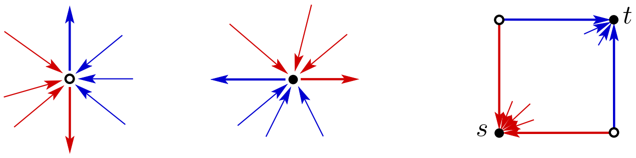


Figure 3 The local conditions at black and white vertices and at s and t.

Every quadrangulation admits a separating decomposition [11, 17]. The equatorial line of the separating decomposition separates the red and blue edges which form trees rooted in s and t respectively, see [10]. Figure 4 shows that the equatorial line yields a one-sided Hamiltonian cycle with reverse edge ts: Along the equatorial line white vertices have red and black vertices have blue backward edges. This shows that quadrangulations and hence bipartite graphs are POSH.

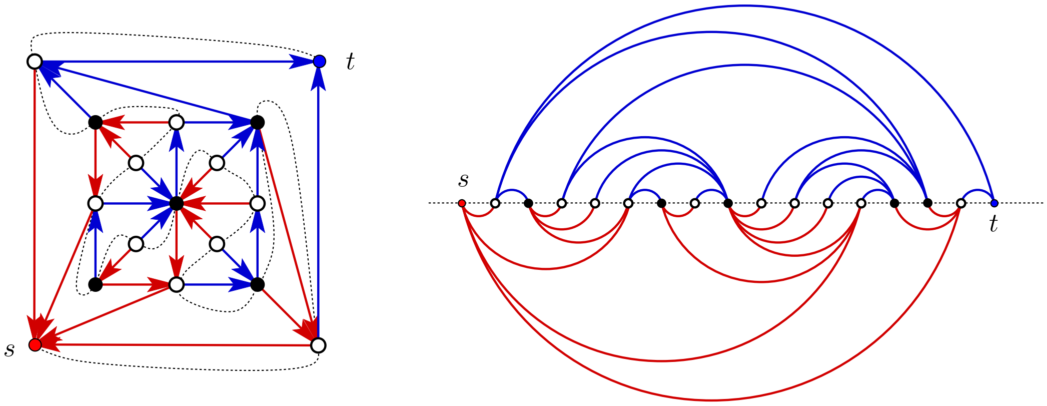


Figure 4 A quadrangulation with a separating decomposition, the equatorial line (dotted), and the induced drawing with a one-sided Hamiltonian cycle.

4 Plane cubic graphs

This section is devoted to a sketch of the proof of the following theorem:

▶ **Theorem 4.1.** *Every plane cubic graph G is a spanning subgraph of a plane graph G' on the same vertex set V which has a one-sided Hamiltonian cycle, i.e., G is POSH.*

To prove this, we use Theorem 3.1 and the following lemma:

▶ **Lemma 4.2.** *Let G be a cubic graph. Then G admits a matching M such that contracting all the edges of M results in in a bipartite multi-graph.*

The technique used to prove the theorem using the lemma is to do vertex splits carefully. We distinguish between *local* and *far* splits, some of which are illustrated in Figure 5 and Figure 6.

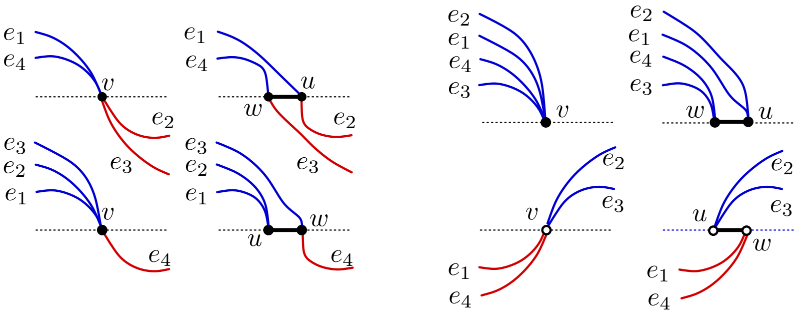


Figure 5 Four cases for the local split of a vertex v.

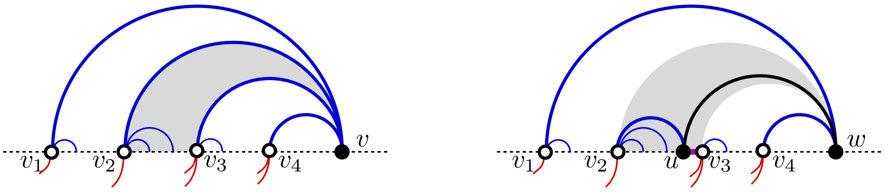


Figure 6 Far split within the gray region of vertex v with edges to the left in the upper half-plane.

5 Concluding remarks

We have examined the exploding double chain as a special point set (order type) and shown that the initial part H_n of size $2n-2$ is n-universal for graphs on n vertices which are POSH.

▶ **Conjecture 1.** *Every triangle-free plane graph is POSH.*

▶ **Conjecture 2.** *Every 5-connected planar triangulation is POSH.*

We have shown that 2-trees and their superclasses series-parallel and planar Laman graphs are not contained in the class $\mathcal{H}'$ of POSH graphs. The question whether these classes admit universal point sets of linear size remains intriguing.

References

1 M. J. Alam, T. C. Biedl, S. Felsner, M. Kaufmann, S. G. Kobourov, and T. Ueckerdt, *Computing cartograms with optimal complexity*, Discret. Comput. Geom., 50 (2013), 784–810.

2 M. J. Alam and S. G. Kobourov, *Proportional contact representations of 4-connected planar graphs*, in Graph Drawing, vol. 7704 of LNCS, Springer, 2012, pp. 211–223.

3 P. Angelini, T. Bruckdorfer, G. Di Battista, M. Kaufmann, T. Mchedlidze, V. Roselli, and C. Squarcella, *Small universal point sets for k-outerplanar graphs*, Discrete & Computational Geometry, (2018), 1–41.

4 M. J. Bannister, Z. Cheng, W. E. Devanny, and D. Eppstein, *Superpatterns and Universal Point Sets*, Journal of Graph Algorithms and Applications, 18 (2014), 177–209.

5 F. J. Brandenburg, *Drawing planar graphs on $\frac{8}{9}n^2$ area*, Electronic Notes in Discrete Mathematics, 31 (2008), 37–40.

6 J. Cardinal, M. Hoffmann, and V. Kusters, *On Universal Point Sets for Planar Graphs*, Journal of Graph Algorithms and Applications, 19 (2015), 529–547.

7 A. Choi, M. Chrobak, and K. Costello, *An $\Omega(n^2)$ lower bound for random universal sets for planar graphs*, arXiv preprint, arXiv1908.07097, (2019).

8 M. Chrobak and H. J. Karloff, *A Lower Bound on the Size of Universal Sets for Planar Graphs*, ACM SIGACT News, 20 (1989), 83–86.

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

10 S. Felsner, Éric. Fusy, M. Noy, and D. Orden, *Bijections for Baxter families and related objects*, Journal of Combinatorial Theory, Series A, 118 (2011), 993–1020.

11 S. Felsner, C. Huemer, S. Kappes, and D. Orden, *Binary labelings for plane quadrangulations and their relatives*, Discrete Mathematics and Theoretical Computer Science, 12:3 (2010), 115–138.

12 R. Fulek and C. D. Tóth, *Universal point sets for planar three-trees*, Journal of Discrete Algorithms, 30 (2015), 101–112.

13 M. Kaufmann and R. Wiese, *Embedding vertices at points: Few bends suffice for planar graphs*, Journal of Graph Algorithms and Applications, 6 (2002), 115–129.

14 M. Kurowski, *A 1.235n lower bound on the number of points needed to draw all n-vertex planar graphs*, Information Processing Letters, 92 (2004), 95–98.

15 M. Löffler and C. D. Tóth, *Linear-size universal point sets for one-bend drawings*, in Graph Drawing, vol. 9411 of LNCS, Springer, 2015, pp. 423–429.

16 B. Mohar, Universal point sets for planar graphs, Open Problem Garden, 2007. `http://www.openproblemgarden.org/op/small_universal_point_sets_for_planar_graphs`.

17 P. Ossona de Mendez and H. de Fraysseix, *On topological aspects of orientations*, Discrete Mathematics, 229 (2001), 57–72.

18 J. Pach, P. Gritzmann, B. Mohar, and R. Pollack, *Embedding a planar triangulation with vertices at specified points*, American Mathematical Monthly, 98 (1991), 165–166.

19 M. Scheucher, H. Schrezenmaier, and R. Steiner, *A note on universal point sets for planar graphs*, Journal of Graph Algorithms and Applications, 24 (2020), 247–267.

20 W. Schnyder, *Embedding Planar Graphs on the Grid*, in Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, 1990, pp. 138–148.

Fast Reconfiguration for Programmable Matter

Irina Kostitsyna[1], Tom Peters[1], and Bettina Speckmann[1]

1 TU Eindhoven, the Netherlands
[i.kostitsyna|t.peters1|b.speckmann]@tue.nl

Related Version A full version of the paper is available at arxiv.org/abs/2202.11663.

1 Introduction

The concept of programmable matter envisions a very large number of tiny and simple robot particles forming a smart material that can change its physical properties and shape based on the outcome of computation and movement performed by the individual particles in a concurrent manner. The ultimate goal is to have programmable matter that is indistinguishable from any other material. Thus, when modeling it, we assume a very small size of the particles and greatly restrict their computation, communication, and movement capabilities. Shape assembly and reconfiguration of particle systems have attracted a lot of interest in the past decade and a variety of specific models have been proposed [1, 8, 11, 13, 7, 3, 10, 12]. Here we focus on the *amoebot* model which was introduced in [4] and refined in [2]. Refer to [2] for additional details on the model description. For reconfiguration in the amoebot model, the approach taken by existing solutions is to build the target shape from scratch, ignoring any possible similarities between the initial shape and the target shape [5, 6]. However, in some scenarios (e.g. shape repair) where the initial and target shapes are similar, this might not be the most efficient strategy. We focus on an approach for reconfiguration that takes this similarity into account. In the worst case our algorithm works as well as the existing solutions, but it is natural to expect our approach to be more advantageous in the case where there are only small changes necessary in the system.

Amoebot model. Particles occupy nodes of a plane triangular grid G. A particle can occupy one (contracted particle) or two (expanded particle) adjacent nodes of the grid, and can communicate with its neighboring particles. The particles have constant memory space, and thus have limited computational power. They have no common notion of orientation, and no common notion of clockwise or counter-clockwise order. The particles are identical, i.e., they have no IDs and execute the same algorithm, but they can locally distinguish between neighbors using six (for contracted) or ten (for expanded particles) *port identifiers* (see Figure 1 (left)). Ports are labeled in order (either cw or ccw) modulo six or ten, respectively. Particles communicate by sending messages to the neighbors using the ports.

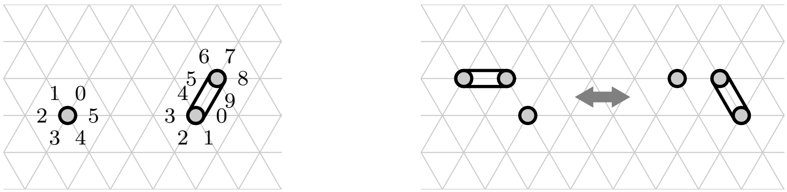


Figure 1 Left: particles with ports labeled, in contracted and expanded state. Right: handover operation between two particles.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

Particles can move in two different ways: a contracted particle can *expand* into an adjacent empty node of the grid, and an expanded particle can *contract* into one of the nodes it currently occupies. Each node of G can be occupied by at most one particle, and we require that the particle system stays connected at all times. To preserve connectivity more easily, we allow a *handover* variant of both move types, a simultaneous expansion and contraction of two neighboring particles using the same node (see Figure 1 (right)). The handover can be initiated by any of the two particles: an expanded particle can *pull* its contracted neighbor, and a contracted particle can *push* its expanded neighbor.

Particles operate in activation cycles: when activated, they can read from the memory of their immediate neighbors, compute, send constant size messages to their neighbors, and perform a move operation. Particles are activated by an asynchronous adversarial but fair scheduler (at any moment in time t, for any particle, it must be activated at some time in the future $t' > t$). If two particles are attempting at conflicting actions (e.g., expanding into the same node), the conflict is resolved by the scheduler arbitrarily, and exactly one of these actions succeeds. We perform running time analysis in terms of the number of *rounds*: the time intervals in which all particles have been activated at least once.

We call the set of particles and their internal states a *particle configuration* $\mathcal{P}$. Let $G_{\mathcal{P}}$ be the subgrid of G induced by the nodes occupied by particles in $\mathcal{P}$. We say that $\mathcal{P}$ is *connected* if there is a path in $G_{\mathcal{P}}$ between any two particles in $\mathcal{P}$. A *hole* in $\mathcal{P}$ is an interior face of $G_{\mathcal{P}}$ with more than three vertices. A particle configuration $\mathcal{P}$ is *simply connected* if it is connected and has no holes.

Problem description. An instance of the *reconfiguration problem* consists of a pair of simply connected shapes (I, T) embedded in the grid G. We assume that I and T have the same number of nodes, and that $I \cap T$, which we call the *core*, is non-empty and simply connected. Initially, all particles in I are contracted. The problem is solved when every node of T contains a contracted particle.

We call the particles in $I \setminus T$ the *supply* particles (see Figure 2), and assume that every connected component of $I \setminus T$ has a designated particle in the core $I \cap T$ adjacent to it, which we call the *root* of that component. Similarly, we say that $T \setminus I$ are *demand* nodes. For every connected component D of $T \setminus I$, we designate one particle from the core $I \cap T$ adjacent to D as the *demand root* of D. We assume that each demand root d stores a spanning tree of the corresponding component D in its memory. The root d will pull supply particles through the core $I \cap T$ to fill D.

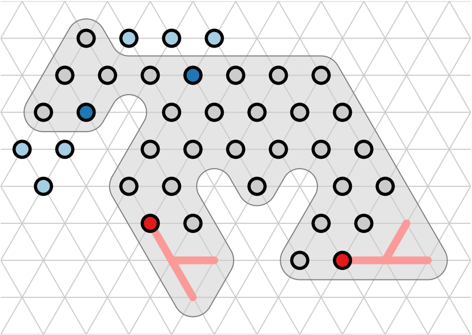


Figure 2 Initial shape I (formed by the particles), target shape T (gray), supply particles (blue), supply roots (dark blue). Demand roots (red) store a spanning tree of their demand component.

Contribution and organization. We propose a new approach for fast shape reconfiguration in the amoebot model, based on the symmetric difference between the initial and the target shapes. Our goal is to design a reconfiguration algorithm, for the case when the initial and target shapes are similar, that is faster and more natural than constructing the target shape from scratch. To this extent, we propose a new primitive: a special case of the *shortest path tree* (SP-tree) which we call a *feather tree*. We use feather trees to construct a graph, which the particles can use to move along shortest paths and reconfigure the particle system.

2 Feather trees

To solve the particle reconfiguration problem, we need to coordinate the movement of the particles. Among the previously proposed primitives for amoebot coordination is the *shortest path tree* (SP-tree) primitive [9] which facilitates movement of particles between the root and the leaves along shortest paths. Our approach to reconfiguration is to use multiple overlapping trees to guide the particles between the supply and demand regions. To do so we need trees with a more restricted shape than arbitrary SP-trees. In this section we hence introduce *feather trees* which are a special case of SP-trees (Figure 3).

Feather trees largely follow the same construction as the SP-trees. A feather tree consists of *shafts* and *branches*. Shafts are straight connections emanating from the root (and sometimes reflex nodes) that grow branches on either side. Branches are straight connections in the tree that do not branch further. To grow a feather tree, the root particle chooses a maximal independent set of neighbors; this set contains at most three particles. The particles in the independent set grow the shafts (in red) emanating from the root. All other neighbors of the root are the beginning of a blue branch. If a particle p at the end of a shaft or branch activates, it first extends the tree straight. Specifically, if i is the port from p to its parent, p extends the tree into the direction $i+3$. Recall that all arithmetic on ports and directions is modulo six. The particle q in direction $i+3$ becomes a child of p and p becomes the parent of q. Next, if p lies on a shaft, it starts branches in the directions $i+2$ and $i+4$.

To reach all the particles of $\mathcal{P}$ with a feather tree, and not just those within one bend from the root, we extend our construction around reflex vertices on the boundary of $\mathcal{P}$. If for a particle p, direction i is the direction to its parent, and the direction $i+1$ (or $i-1$) does not contain a particle, while the direction $i+2$ (or $i-2$) does, then p lies on a reflex vertex of the boundary of $\mathcal{P}$. If in addition p lies on a branch, p starts a new shaft in the direction $i+2$ (or $i-2$), see Figure 3 (right). We hence have the following lemma:

▶ **Lemma 1.** *Given a simply connected particle configuration $\mathcal{P}$ with n particles and a*

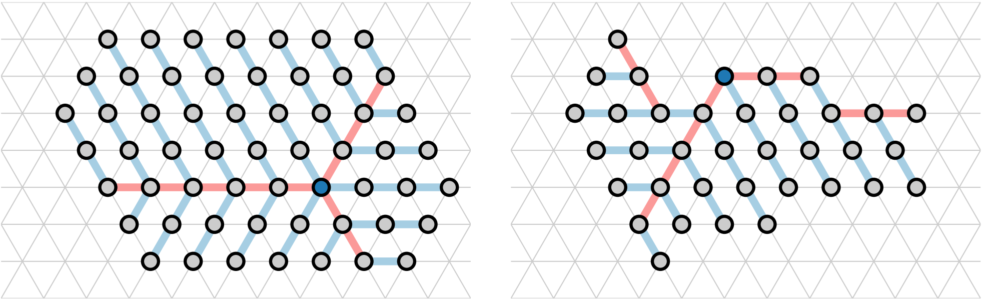


Figure 3 Two feather trees growing from the dark blue root. Shafts are red and branches are blue. Left: every particle is reachable by the initial feathers; Right: additional feathers are necessary.

particle $r \in \mathcal{P}$, we can grow a feather tree from r in $O(n)$ rounds.

Feather trees are unique, which helps with navigating the particles. Next we describe how to navigate multiple overlapping feather trees. First we identify a useful property of shortest paths in feather trees.

We say that a vertex v of $G_{\mathcal{P}}$ is an *inner vertex*, if v and its six neighbors lie in the core $I \cap T$. All other vertices of the core are *boundary vertices.* A *bend* in a path is formed by three consecutive vertices that form an angle of $120°$. We say that a bend is an *inner bend* if the middle vertex is an inner vertex; otherwise the bend is a *boundary bend.*

▶ **Definition 2** (Feather Path)**.** A path in $G_{\mathcal{P}}$ is a *feather path* if it does not contain two consecutive inner bends.

We now argue that every path from the root to a leaf in a feather tree is a feather path. A root-to-leaf path bends either on a shaft or on a branch; the path can transition from a shaft to a branch or vice versa only at a bend. If the bend occurs on a shaft, then it can be either an inner or a boundary bend; the path leaves the shaft and continues on a branch. Branches grow straight with one exception: if they detect a reflex vertex on the boundary of $\mathcal{P}$. In this case a bend occurs on the branch. This bend is always a boundary bend; the path leaves the branch and continues on the new shaft.

▶ **Lemma 3.** *Every path from the root to a leaf in a feather tree is a feather path.*

Navigating feather trees. Due to its limited memory, a particle cannot identify different trees. However, particles can navigate feather trees by counting inner bends, even in presence of multiple overlapping trees. Therefore, while moving down a specific tree, a particle always knows if it is on a shaft or a branch and which directions belong to the tree. Starting from the root of a feather tree, a particle can always reach one of its leaves. We cannot control which leaf it reaches, but it will do so along a shortest path from the root. In particular, if feasible, it is always a valid choice for the particle to continue straight ahead. A left or right $120°$ bend is a valid choice if the particle is on a shaft, or if this is a boundary bend.

When moving up from leaves, we cannot control which root of which feather tree a particle will reach, but it will always travel along a shortest path. In particular, if the particle is moving along a shaft, then its only valid choice is to continue straight ahead. Otherwise, all three options (straight ahead or a $120°$ left or right turn) are valid.

3 Supply and demand

We now explain how to use feather trees to create a *supply graph* in the core $I \cap T$ of the particle system that connects supply roots and demand roots along shortest paths. This graph serves as a navigation network for the particles moving from supply to the demand. Let $G_{I \cap T}$ be the subset of G induced by the nodes of $I \cap T$. We say a *supply graph* S is a directed subgraph of $G_{I \cap T}$ connecting every supply root s to every demand root d such that the following three *supply graph properties* hold:

1. for every pair (d, s) a shortest path from d to s in S is also a shortest path in $G_{I \cap T}$,
2. for every pair (d, s) there exists a shortest path from d to s in S that is a feather path,
3. every particle p in S lies on a shortest path for some pair (d, s).

We construct supply graph S from feather trees as follows. First, every demand root initiates the growth of a feather tree. When a feather tree reaches a supply root s, a *supply* *found token* is sent back to the root of the tree. Note that if several feather trees overlap, a node in charge of forwarding the token up the tree cannot determine which specific tree the token belongs to. However, it can identify and forward the token to all *valid* parents that lie on valid feather paths for this token. To count inner bends, the supply found token carries a flag β which is updated at each bend. Specifically, a particle p that receives a supply found token t does the following:

1. p marks itself as part of the supply graph S,
2. p stores the direction i that t came from as a valid child in S,
3. from all of its parents in all the feather trees, p computes the set U of valid parents by checking the flag β of t,
4. p adds U to the set of its parents in S,
5. p forwards t to the particles in U, updating the flag β if necessary, and
6. p stores the complete information about t for the future.

Note that a particle p receives at most two supply found tokens from each direction, one for each value of β. Hence p can store the corresponding information in its memory. When a feather tree F reaches a particle p that is already marked as part of the supply graph S then p first checks if F would have been included in U for at least one of the supply found tokens t stored in p. If that is the case, p sends a copy of t towards the root of F, and sets its parent in F to be a parent in S as well. Otherwise, p grows F as normal.

▶ **Lemma 4.** *Given a simply connected particle configuration $\mathcal{P}$ with n particles, a set of particles marked as supply roots, and a set of particles marked as demand roots, we can create a supply graph using $O(n)$ rounds.*

Algorithm. We present a high level overview of our algorithm. A complete description and analysis can be found in the full version.

In the first phase of the algorithm, the particles form the supply graph, by creating feather trees starting from the demand roots, and sending tokens back up the trees if a branch finds supply. Each supply root organizes the corresponding supply component into a tree; these supply trees are connected to the supply graph S via the supply roots. After the supply graph S has been formed, demand roots pull particles from S and fill the demand components; the pulling of particles propagates through S to the supply components. Particles moving in S store the number of inner bends they take. An extended particle checks the number of inner bends of its children in S to determine which of them are valid choices to pull. Eventually, pulling propagates to the supply particles at the leaves of the corresponding spanning trees. These particles simply contract, reducing the outstanding supply. If a supply component becomes empty, some extended particles may need to revert and pull back against the prescribed direction in S. Then the corresponding edges get removed from S, thus rerouting the movement of extended particles towards supply that still exists.

Recall that the particles follow feather paths through S. This ensures that even if the particles themselves do not know which supply component they are pulling from, they do so via a shortest path. Moreover, even in the case where particles have to move back because a supply component was already empty, the total path taken by each particle will be at most linear in the size of the particle configuration.

▶ **Theorem 5.** *The particle reconfiguration problem can be solved using $O(n)$ rounds of activation.*

4 Conclusion

We have presented a fast reconfiguration algorithm for a system of amoebot particles. Our solution currently only works on simply-connected particle systems with a simply-connected intersection of the initial and the target shapes. An interesting direction to explore is to extend our results for a larger class of shapes. Furthermore, our worst-case running time matches the efficiency bounds of existing solutions, however we expect our algorithm to be more advantageous for the case when the initial and target shapes are similar. Thus it would be interesting to evaluate our solution experimentally on realistic scenarios.

References

1 Kenneth C. Cheung, Erik D. Demaine, Jonathan R. Bachrach, and Saul Griffith. Programmable Assembly With Universally Foldable Strings (Moteins). *IEEE Transactions on Robotics*, 27(4):718–729, 2011. doi:10.1109/TRO.2011.2132951.

2 Joshua J. Daymude, Andréa W. Richa, and Christian Scheideler. The Canonical Amoebot Model: Algorithms and Concurrency Control. In *35th International Symposium on Distributed Computing (DISC)*, volume 209 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 20:1–20:19, 2021. doi:10.4230/LIPIcs.DISC.2021.20.

3 Erik D. Demaine, Jacob Hendricks, Meagan Olsen, Matthew J. Patitz, Trent A. Rogers, Nicolas Schabanel, Shinnosuke Seki, and Hadley Thomas. Know When to Fold 'Em: Self-assembly of Shapes by Folding in Oritatami. In *DNA Computing and Molecular Programming*, pages 19–36, 2018. doi:10.1007/978-3-030-00030-1_2.

4 Zahra Derakhshandeh, Shlomi Dolev, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Brief announcement: Amoebot—A New Model for Programmable Matter. In *Proc. 26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 220–222, 2014. doi:10.1145/2612669.2612712.

5 Zahra Derakhshandeh, Robert Gmyr, Andréa W. Richa, Christian Scheideler, and Thim Strothmann. Universal Shape Formation for Programmable Matter. In *Proc. 28th Annual ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 289–299, 2016. doi:10.1145/2935764.2935784.

6 Giuseppe A. Di Luna, Paola Flocchini, Nicola Santoro, Giovanni Viglietta, and Yukiko Yamauchi. Shape formation by programmable particles. *Distributed Computing*, 33:69–101, 2020. doi:10.1007/s00446-019-00350-6.

7 Cody Geary, Paul W. K. Rothemund, and Ebbe S. Andersen. A single-stranded architecture for cotranscriptional folding of RNA nanostructures. *Science*, 345(6198):799–804, 2014. doi:10.1126/science.1253920.

8 Robert Gmyr, Kristian Hinnenthal, Irina Kostitsyna, Fabian Kuhn, Dorian Rudolph, Christian Scheideler, and Thim Strothmann. Forming Tile Shapes with Simple Robots. In *Proc. International Conference on DNA Computing and Molecular Programming (DNA)*, pages 122–138, 2018. doi:10.1007/978-3-030-00030-1_8.

9 Irina Kostitsyna, Tom Peters, and Bettina Speckmann. Coordinating Programmable Matter via Shortest Path Trees. In *Book of Abstracts, 37th European Workshop on Computational Geometry*, pages 32:1–32:7, 2021.

10 Andre Naz, Benoit Piranda, Julien Bourgeois, and Seth Copen Goldstein. A distributed self-reconfiguration algorithm for cylindrical lattice-based modular robots. In *Proc. 2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*, pages 254–263, 2016. doi:10.1109/NCA.2016.7778628.

11 Matthew J. Patitz. An introduction to tile-based self-assembly and a survey of recent results. *Natural Computing*, 13(2):195–224, 2014. doi:10.1007/s11047-013-9379-4.

12 Benoit Piranda and Julien Bourgeois. Designing a quasi-spherical module for a huge modular robot to create programmable matter. *Autonomous Robots*, 42(8):1619–1633, 2018. doi:10.1007/s10514-018-9710-0.

13 Damien Woods, Ho-Lin Chen, Scott Goodfriend, Nadine Dabby, Erik Winfree, and Peng Yin. Active self-assembly of algorithmic shapes and patterns in polylogarithmic time. In *Proc. 4th Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 353–354, 2013. doi:10.1145/2422436.2422476.

Short topological decompositions of non-orientable surfaces

Niloufar Fuladi[1], Alfredo Hubard[1], and Arnaud de Mesmay[1]

1 Univ Gustave Eiffel, CNRS, LIGM, F-77454 Marne-la-Vallée, France

Abstract

We provide a polynomial-time algorithm that for any graph embedded on a non-orientable surface computes a canonical non-orientable system of loops so that any loop from the canonical system intersects any edge of the graph in at most 30 points. The existence of such short canonical systems of loops was known in the orientable case and an open problem in the non-orientable case. Our techniques combine recent work of Schaefer-Štefankovič with ideas from computational biology.

1 Introduction

Lazarus, Pocchiola, Vegter and Verroust [7] (see also [6]) were the first to design an algorithm that finds, for any graph G embedded in a closed orientable surface S a *canonical system of loops* C such that no edge of C intersects[1] any edge of G more than a constant number of times. By a canonical system of loops we mean a one-vertex one-face embedded graph in which the cyclic ordering of the edges around the vertex is $a_1 b_1 a_1^{-1} b_1^{-1} \dots a_g b_g a_g^{-1} b_g^{-1}$. This system of loops has been widely used, both in applied and theoretical works, as it provides a short and canonical way to cut a surface into a disk.

In the non-orientable case, no instance of such short decompositions seems to be known. Even for the *non-orientable canonical system of loops*, that is, a system of one-sided loops with the cyclic ordering $a_1 a_1 a_2 a_2 \dots a_g a_g$ around the vertex, the best known algorithm requires $O(g|E(G)|)$ crossings for each loop (see [6]). Non-orientable surfaces have been often neglected in computational topology, but there are many reasons to want to correct this: natural models of random surfaces yield non-orientable surfaces with large probability, they appear as configuration spaces in diverse contexts [3, 11], and insights garnered from non-orientable surfaces can sometimes be applied to the orientable ones; e.g. see [9].

In this article, we prove the following theorem providing, to the best of our knowledge, the first known case of a short canonical topological decomposition for non-orientable surfaces.

▶ **Theorem 1.1.** *There exists a polynomial time algorithm that, given a graph cellularly embedded on a non-orientable surface, computes a non-orientable canonical system of loops such that each loop in the system has multiplicity at most* 30.

We first point out that the techniques used to prove the orientable version in [7] incur an overhead of $O(g)$ in the multiplicity of the resulting curves (see [6, Theorem 4.3.9]). Instead, our proof of Theorem 1.1 builds on important recent work of Schaefer and Štefankovič [10], who showed that any graph embedded on a non-orientable surface can be represented with a cross-cap drawing so that each edge uses each cross-cap at most twice (see the second picture of Figure 1 for an example). Our main technical contribution is to upgrade their

[1] Throughout the article, we decompose surface-embedded graphs by cutting them along embedded graphs which are *transverse* to the original graph, and count the number of intersections. This is equivalent to the primal setting studied in Lazarus, Pocchiola, Vegter and Verroust [7] by graph duality.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

construction so that the cross-caps can be connected to each other so as to yield a non-orientable canonical system of loops, intersecting the edges of the one-vertex graph with multiplicity at most 30 (see Figure 1).

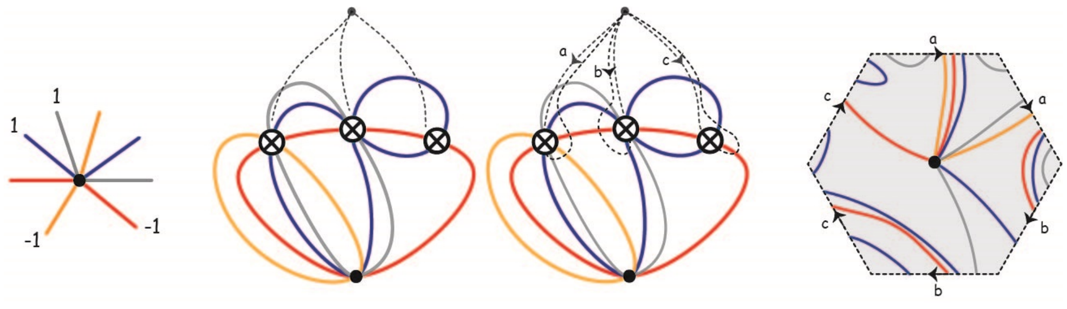


Figure 1 *From left to right: 1) The combinatorial information of a one-vertex graph. 2) A cross-cap drawing of this graph, with cross-caps connected to a base-point. 3) A joint drawing of the graph and a canonical system of loops. 4) A different representation: decomposing the graph along the canonical system of loops.*

Due to line limitations, the proofs are only sketched and can be found in the full version.

2 Preliminaries

An *embedding* of a graph G on a surface S is informally a crossing-free drawing of G on S. We treat (G, S) as a *cross-metric surface* [2], i.e., the objects we define and work with will be in general position with respect to G. The *multiplicity* of such a curve embedded on S is the maximum number of times it intersects with an edge of the underlying graph G. As in many similar works, in our proofs we first contract a spanning tree of the underlying graph, reducing the problem to the setting of a one-vertex graph embedded on a non-orientable surface. The combinatorics of a one-vertex embedded graph are completely described by an *embedding scheme*, i.e., by the circular order of the edges around the vertex, and a signature for each loop indicating whether it is one-sided or two-sided. We simply use *scheme* to refer to a graph with an embedding scheme. A one-vertex scheme is *orientable* if all its loops are two-sided and *non-orientable* otherwise. We denote by $eg(G)$ the Euler genus of an embedding scheme G, i.e., the Euler genus of the surface obtained by gluing a topological disk on each face. A loop e in a scheme divides the half-edges around the vertex into two parts, called *wedges* of e.

We represent an embedding of a graph on a non-orientable surface by drawing it on the plane with a finite number of *cross-caps*. A family of edges entering a cross-cap emerges on the other side with a reversed order, and a loop is one-sided (two-sided) if it enters odd (even) number of cross-caps. See Figure 2 for an example of two cross-cap drawings of the same scheme on a non-orientable surface of genus 3.

Short Orienting Curves. A simple but important object that we rely on extensively is an *orienting curve*, i.e., a closed curve on a non-orientable surface such that cutting along it produces an orientable surface with boundary. The following lemma is a restatement of [8, Proposition 5.5].

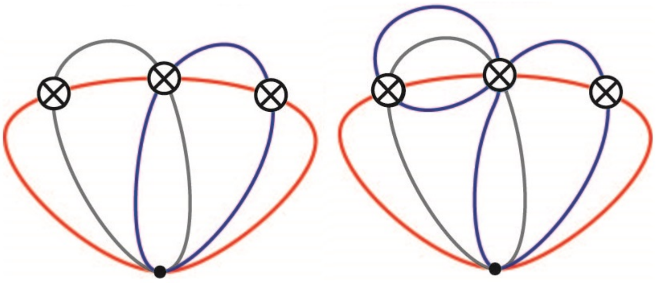


Figure 2 *Two cross-cap drawings for the same scheme.*

▶ **Lemma 2.1.** *Let N be a non-orientable surface without boundary and with genus g and G be a graph embedded on N. Then there exists an orienting curve of multiplicity at most 2.*

3 The Schaefer Štefankovič Algorithm

Schaefer and Štefankovič proved the following theorem.

▶ **Theorem 3.1.** *[10, Lemma 9] If G is a one-vertex non-orientable (respectively orientable) scheme, then it admits a cross-cap drawing with $eg(G)$ (respectively $eg(G)+1$) cross-caps in which every edge passes through every cross-cap at most twice.*

The proof of the theorem uses an inductive algorithm, which distinguishes cases depending on the topological type of the next loop to be drawn. This algorithm works by providing a way to draw a loop at each inductive step, assuming that some simpler one-vertex graph without that loop can be drawn. Here, we only elaborate on techniques to deal with two special type of curves; we refer to the full version for complete description of the algorithm.

One-sided loop move. Let r be a one-sided loop in the scheme G. We remove r and flip one of its wedges, i.e., we reverse the order of the edges within this wedge and change the signature of the loops that alternate with r. The new scheme $G^{'}$ has Euler genus $eg(G)-1$. Let us assume inductively that we are given a drawing for $G^{'}$. We add r to this drawing by adding a cross-cap near the vertex and the flipped wedge and dragging r and every edge in the flipped wedge in it; see Figure 3.

Dragging move. Let s be a separating loop in a scheme G, such that cutting along it results in two subschemes G_1 and G_2 in which G_2 is orientable. We add a one-sided loop o with consecutive ends in G_2 in the wedge where s used to be. Having a drawing for G_1 and $G_2+\{o\}$, we can draw the loop s in the drawing for $G_2+\{o\}$ as follows: we start next to an end of o, follow o through all the cross-caps, except that after coming out of the last cross-cap, we go back to the first one entered, and traverse all of the cross-caps again. Finally we end up next to the other end of o; see Figure 4, left. Denote this drawing of $G_2+\{o\}+\{s\}$ by $H_2^{'}$ and the drawing we obtain for G_1 by H_1. By gluing H_1 to $H_2^{'}$, we get a drawing $H^{'}$ for $G+\{o\}+\{s\}$ but the drawing is not using the minimum number of cross-caps. However we can eliminate one cross-cap in $H^{'}$ by dragging some curves of G_1 through the crosscaps of G_2: this is pictured in Figure 4, right.

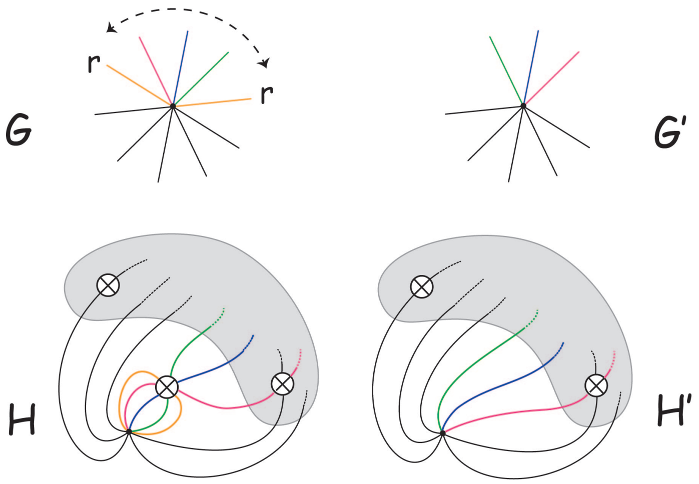


Figure 3 The one-sided loop move on the loop r.

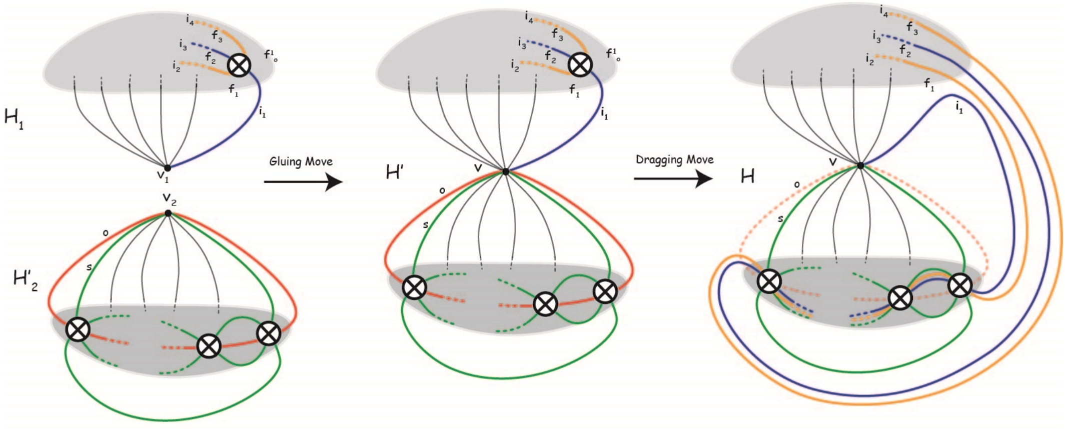


Figure 4 The dragging move.

4 From cross-cap drawings to canonical systems of loops

The complexity of the drawings provided by the proof of Schaefer and Štefankovič increases too fast to directly yield a short non-orientable canonical system of loops. Therefore, we modify their algorithm by enforcing more specific rules to simplify some of the steps and provide additional structure to the inductive argument. First, we reduce the given embedded graph to a scheme with an orienting loop.

▶ **Lemma 4.1.** *Given a graph G embedded on a non-orientable surface N, there exists a one-vertex scheme $\hat{G}$ such that $\hat{G}$ has an orienting loop, and if $\hat{G}$ has a non-orientable canonical system of loops of multiplicity at most k, then G has a non-orientable canonical system of loops of multiplicity at most $3k$.*

The idea of the proof is to add an orienting curve (Lemma 2.1) and contract a spanning tree. Second, we impose a certain order to choose the one-sided and separating loops.

An order to choose one-sided loops. This order comes from a seemingly unrelated problem in computational biology, precisely genome rearrangements. Given a word with signatures (a bit assigned to each letter), a signed reversal consists in choosing a subword, and

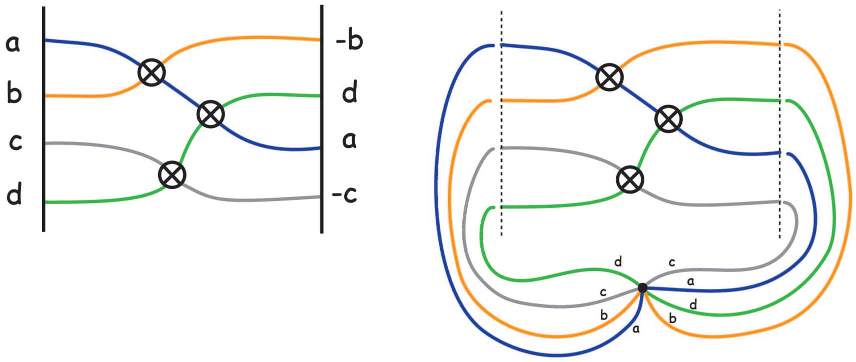


Figure 5 *Left: a representation of three signed reversals bringing the signed permutation on the left to the one on the right. Right: Attaching the two permutations to a common basepoint yields a one-vertex graph with an embedding scheme, for which signed reversals provide a cross-cap drawing.*

reversing it as well as the signatures of its letters. The *signed reversal distance* between two signed words is the minimum number of signed reversals needed to go from one signed word to the other one. As we show in Figure 5, (see also [1, 5]), there is a strong similarity between computing the signed reversal distance between two signed permutations and embedding a one-vertex graph built from these two permutations with a minimum number of cross-caps. Hannenhalli and Pevzner [4] provided a polynomial time algorithm to compute the signed reversal distance between two signed permutations. In this algorithm they introduce an optimal choice for choosing a subword to reverse, which in our setting, translates to an order for choosing between the one-sided loops in the induction: we choose the one-sided loop such that flipping its wedge maximizes the number of new one-sided loops.

Saturating the scheme and an order to choose separating loops. Then, we *saturate* the scheme with auxiliary separating loops. That is, whenever possible, we add a two-sided loop so that its ends interleave with no other loop in the scheme and that is not homotopic to a loop that was already present. These loops do not interfere with the genus and can be removed at the end. We choose a non-contractible separating loop that divides an orientable scheme that does not contain a separating loop, from the rest of the scheme.

The Modified Algorithm. We are now ready to describe our algorithm. By Lemma 4.1, we can reduce an embedded graph G to a scheme that contains an orienting loop, and we saturate it. Then we apply the following steps inductively in the following order:

- If there is a **contractible loop**, remove it and recurse on the new scheme. We can then draw it in the obtained drawing without using any of the cross-caps.
- If there is a **non-contractible separating loop**, choose one as described above, recurse on the subschemes and apply a dragging move.
- If there is a **one-sided non-orienting loop**, choose one as described above and apply the one-sided loop move on this loop.
- If all one-sided loops are **orienting** and there are two-sided loops, pick an orienting loop o adjacent to a two-sided loop t, recurse on the scheme in which o is replaced by the concatenation of o and t. Finally drag the concatenation of o and t back along t.
- If all one-sided loops are orienting and there are no two-sided loops, one cross-cap is sufficient to draw all the loops.

The following lemma guarantees that our algorithm provides a good cross-cap drawing.

▶ **Lemma 4.2.** *Applying the modified algorithm on a graph G embedded on a non-orientable surface S yields a cross-cap drawing of G with $eg(S)$ cross-caps such that each loop of G enters each cross-cap at most 6 times.*

The proof is similar to the original proof of Schaefer and Štefankovič algorithm but heavily relies on the presence of the orienting loop to reduce the number of cases. Our main technical result is then to show that the modified algorithm outputs a cross-cap drawing where cross-caps are not too far from the vertex.

▶ **Lemma 4.3.** *For any saturated one-vertex scheme G with an orienting loop o, in the cross-cap drawing H output by the modified algorithm, there is a path from every cross-cap to a face incident to the vertex with multiplicity at most two.*

The tension in the proof of this lemma lies in the fact that long dual paths are added to the cross-cap drawings when doing one-sided loop moves and dragging moves, making it delicate to track the diameter of the graph dual to the cross-cap drawings throughout the recursive calls. This is handled by tracking specific paths, whose lengths are controlled using two different strategies. For these strategies to succeed, it is crucial that those moves do not alternate during recursive calls, and that the separating curves are chosen in the appropriate order. These properties are guaranteed by the specific orders in which we choose one-sided loop and separating loops.With this lemma, we can finally sketch the proof of Theorem 1.1.

Sketch of proof of Theorem 1.1. The modified algorithm on G constructs a cross-cap drawing in which, by Lemma 4.2, the number of cross-caps is minimal and the loops enter each cross-cap at most twice. By Lemma 4.3, there exist paths $\{p_j\}$ of multiplicity two from a face incident to each cross-cap to a face incident to the vertex in this cross-cap drawing. We follow these paths to construct loops based at a common vertex surrounding each cross cap, see Figure 1, third picture. It can be seen that the system of loops that we obtain is canonical. Using Lemma 4.1 and adding different bounds on the multiplicity, we obtain a canonical system of loops where each loop has multiplicity 30. ◀

Acknowledgments. We are grateful to Marcus Schaefer and Daniel Štefankovič for providing us the full version of [10], and to Francis Lazarus for helpful discussions. This work was partially supported by the grant(s) ANR-17-CE40-0033 of the French National Research Agency ANR (project SoS) and INTER/ANR/16/11554412/SoS of the Luxembourg National Research fund FNR.

References

1 Andrei C Bura, Ricky XF Chen, and Christian M Reidys. On a lower bound for sorting signed permutations by reversals. *arXiv preprint arXiv:1602.00778*, 2016.

2 Éric Colin De Verdière and Jeff Erickson. Tightening nonsimple paths and cycles on surfaces. *SIAM Journal on Computing*, 39(8):3784–3813, 2010.

3 Robert Ghrist. Barcodes: the persistent topology of data. *Bulletin of the American Mathematical Society*, 45(1):61–75, 2008.

4 Sridhar Hannenhalli and Pavel A Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. *Journal of the ACM (JACM)*, 46(1):1–27, 1999.

5 Fenix WD Huang and Christian M Reidys. A topological framework for signed permutations. *Discrete Mathematics*, 340(9):2161–2182, 2017.

6 Francis Lazarus. Combinatorial graphs and surfaces from the computational and topological viewpoint followed by some notes on the isometric embedding of the square flat torus. *Mémoire d'HDR*, 2014. Available at `http://www.gipsa-lab.grenoble-inp.fr/~francis.lazarus/Documents/hdr-Lazarus.pdf`.

7 Francis Lazarus, Michel Pocchiola, Gert Vegter, and Anne Verroust. Computing a canonical polygonal schema of an orientable triangulated surface. In *Proceedings of the seventeenth annual symposium on Computational geometry*, pages 80–89, 2001.

8 Jiří Matoušek, Eric Sedgwick, Martin Tancer, and Uli Wagner. Untangling two systems of noncrossing curves. In *International Symposium on Graph Drawing*, pages 472–483. Springer, 2013.

9 Marcus Schaefer and Daniel Štefankovič. Block additivity of $\mathbb{Z}_2$-embeddings. In *International Symposium on Graph Drawing*, pages 185–195. Springer, 2013.

10 Marcus Schaefer and Daniel Štefankovič. The degenerate crossing number and higher-genus embeddings. *Journal of Graph Algorithms and Applications*, 26(1):35–58, 2022. `doi:10.7155/jgaa.00580`.

11 James P Sethna. Order parameters, broken symmetry, and topology. In *1991 Lectures in Complex Systems*. Addison-Wesley, 1992.

Time and Space Efficient Collinearity Indexing*

Boris Aronov[1], Esther Ezra[2], Micha Sharir[3], and Guy Zigdon[4]

1 **Department of Computer Science and Engineering, Tandon School of Engineering, New York University, Brooklyn, NY 11201, USA**
boris.aronov@nyu.edu

2 **School of Computer Science, Bar Ilan University, Ramat Gan, Israel**
ezraest@cs.biu.ac.il

3 **School of Computer Science, Tel Aviv University, Tel Aviv, Israel**
michas@tauex.tau.ac.il

4 **School of Computer Science, Bar Ilan University, Ramat Gan, Israel**
guy.zigdon@live.biu.ac.il

Abstract

The *collinearity testing* problem is a basic problem in computational geometry, in which the task at hand is the following: Given three sets of n planar points A, B and C, detect a collinear triple of points in $A \times B \times C$ or report there is no such triple. In this paper we consider a preprocessing variant of collinearity testing, namely, the *collinearity indexing* problem, in which we are given two sets A and B, each of n points in the plane, and our goal is to preprocess A and B into a data structure, so that, for any query point $q \in \mathbb{R}^2$, we can determine whether q is collinear with a pair of points $(a, b) \in A \times B$. We provide a solution to the problem for the case where the points of A, B lie on an integer grid, and the query points lie on a vertical line, with a data structure of subquadratic storage and sublinear query time.

1 Introduction

Let A, B, C be three sets of points in the plane. The *collinearity testing* problem is to determine whether there exists a collinear triple $a \in A$, $b \in B$, $c \in C$. This problem is *3SUM-hard* [10],[1] and recently several input-restricted variants of collinearity testing have been studied in the decision tree model and also in the RAM model [3, 4, 6]. In this paper we study a preprocessing variant of collinearity testing, referred to as *collinearity indexing*.

1.1 Collinearity Indexing

Let A and B be two sets, each of n points in the plane. The *collinearity indexing* problem is to preprocess A and B into a data structure, so that, given any query point $q \in \mathbb{R}^2$, we can determine whether there exists a pair $(a, b) \in A \times B$ such that a, b and q are collinear. The goal is to come up with such a structure that uses subquadratic storage and answers queries in sublinear time, but see a finer discussion of this issue shortly below. Notice that in

[*] Work by Boris Aronov was partially supported by NSF grants CCF-15-40656 and CCF-15-40656 and by grant 2014/170 from the US-Israel Binational Science Foundation. Work by Esther Ezra was partially supported by NSF CAREER under grant CCF:AF-1553354 and by Grant 824/17 from the Israel Science Foundation. Work by Micha Sharir was partially supported by ISF Grant 260/18.

[1] In fact, Gajentaan and Overmars [10], who introduced this concept, initially called this problem (as well as other related geometric problems) "n^2-hard," as it was strongly believed that it cannot be solved in subquadratic time (whereas it has a simple $O(n^2)$ time solution).

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

this formulation we ignore the preprocessing time, which can be $\Omega(n^2)$, but only care about storage and query time.[2]

Near-linear query time is easy to obtain, with no auxiliary storage. To do so, one simply sorts the points of $A \cup B$ in angular order around the query point q, and checks every consecutive pair, in this order, of an A-point and a B-point, for collinearity with q. The cost of the query is $O(n \log n)$.

We can improve the query time to $O(n)$ if we are allowed to use quadratic storage. To do so, we pass to the dual plane, and get a set A^* of n lines dual to the points of A, and a set B^* of n lines dual to the points of B. We compute the arrangement $\mathcal{A}$ of $A^* \cup B^*$, and store its DCEL representation. Then, given a query point q, we take its dual line q^* and trace the faces of $\mathcal{A}$ that q^* crosses, checking whether q^* passes through a "bichromatic" vertex of $\mathcal{A}$, incident to a line of A^* and a line of B^*, which is the dual interpretation of the property that q is collinear with a point of A and a point of B. Using the DCEL representation and the *zone theorem* (i.e., that the overall zone complexity of a line in a planar arrangement of lines is linear in the number of lines), this takes $O(n)$ time, see, e.g., [7].

We do not know whether the query cost can be made sublinear, still using quadratic storage, nor whether the storage can be reduced to subquadratic, still allowing linear query cost. These two problems are discussed below, and are left as challenging, seemingly hard, open problems.

We can obtain sublinear query time with superquadratic storage. There are several equivalent ways to describe such a procedure, but one of the simpler ways is to take the set V of the $O(n^2)$ bichromatic vertices of $\mathcal{A}$, and preprocess it for halfplane range searching (see, e.g., [1, 2]). It is easy to adapt the resulting procedure so that it can detect whether the query line q^* passes through a vertex in V. With s storage, a query takes $O^*(n^2/\sqrt{s})$ time, which is sublinear when s is superquadratic. In the extreme case, when $s = \Theta(n^4)$, the query time becomes $O(\log n)$. Indeed, in this case, the above approach essentially constructs the primal arrangement of the $O(n^2)$ bichromatic lines, each connecting an A-point with a B-point, and preprocesses the arrangement for fast point location, using $O(n^4)$ storage and $O(\log n)$ query time.

In view of this discussion, the following problems arise:

(i) Preprocess A and B into a data structure that requires $O(n^2)$ storage and can answer a collinearity query in $o(n)$ time.

(ii) Preprocess A and B into a data structure that requires $o(n^2)$ storage and can answer a collinearity query in $O(n)$ time.

(iii) Preprocess A and B into a data structure that requires $o(n^2)$ storage and can answer a collinearity query in $o(n)$ time.

Of course, a solution to Problem (iii) will automatically solve the other two problems, but it is conceivable, and very likely, that the first two problems are easier to solve, although at the moment the solution to any of problems (i)–(iii) seems to be elusive.

Our result. In general, A and B may contain arbitrary points in the plane, and the queries are also arbitrary points. We can formulate special instances of these problems in which the locations of either the points of $A \cup B$ and / or of the query points are restricted. In this paper we consider the case where the points of A, B lie on an integer grid, and the points of

[2] We note that a related problem, referred to as "3POL-indexing", was defined in [11] in a somewhat different context.

C lie *anywhere* on the y-axis. For this case we show (in what follows, $O^*(\cdot)$ hides a factor of n^ε, for any $\varepsilon > 0$):

▶ **Theorem 1.1.** *Let A, B be two sets of n planar points each, lying on an integer grid. For any $0 < \delta < 1$ there exists a data structure of overall storage complexity $O^*(n^{2-\delta/3})$, which answers collinearity-indexing queries in $O^*(n^\delta)$ time. In particular, when $\delta = 1/4$ the storage and query bounds are $O^*(n^{7/4})$ and $O^*(n^{3/4})$. The overall preprocessing time of this data structure is $O^*(n^2)$.*

2 A Detailed Description of the Data Structure

A naive solution for the case where A and B are unrestricted. As discussed in the introduction, our main result holds for the case where A, B lie on an integer grid, but we first discuss a more general setup.

Let A and B be two sets, each of n (unrestricted) points in the plane, and let γ be a vertical line containing the points of C. Without loss of generality, we assume that γ is the y-axis. The collinearity indexing problem, restricted to γ, is to preprocess A and B into a data structure, so that, given any query point $q \in \gamma$, we can determine whether there exists a pair $(a, b) \in A \times B$ such that a, b and q are collinear.

The problem is easy to solve using $O(n^2 \log n)$ preprocessing time, $O(n^2)$ storage, and $O(\log n)$ query time, as follows. We pass to the dual plane, construct the arrangement of $A^* \cup B^*$, where A^* (resp., B^*) is the set of lines dual to the points of A (resp., of B), collect all the "bichromatic" vertices (those formed by a line of A^* and a line of B^*), and sort them by their y-coordinates. All this takes $O(n^2 \log n)$ time. Given a query point $q = (0, \eta) \in \gamma$, we need to determine whether the horizontal dual line $q^* : \; y = \eta$ passes through a bichromatic vertex of the arrangement, and this takes $O(\log n)$ time, as is easily verified. At the other extreme setup, as already described in the introduction, we can solve the problem with no preprocessing, using $O(n)$ storage, and then a query with a point c (not necessarily on γ) is answered by sorting $A \cup B$ in the angular order around q, and then by checking all consecutive bichromatic pairs in this ordering for collinearity with q. This takes $O(n \log n)$ time. Our goal, however, is to obtain a data structure that uses subquadratic storage and sublinear query time.

2.1 The case where A and B lie on an integer grid

We next present a more efficient solution to this problem, under the following further restrictions. We assume that the points of A and of B lie on (some of the) vertices of a $k \times k$ integer grid G, with integer coordinates (so $n \leq k^2$), where k is bounded by a polynomial function of n. For concreteness, and without loss of generality, assume that $G = [t, k + t - 1] \times [0, k - 1]$ (t is an integer parameter that indicates where the y-axis is located with respect to G). We continue to denote by γ the y-axis, and assume the points of C lie on γ.

Our solution is an adaptation of the technique of Golovnev *et al.* [11] (see also Kopelowitz and Porat [12]), which solves the simpler *3SUM-indexing* problem: That is, the goal is preprocess two sets A, B of n real numbers each, such that given a real number c, one can quickly determine whether there is a pair $a \in A$, $b \in B$ such that $a + b + c = 0$ (see also [8] for a related problem). As in [11, 12], our solution is based on the function inversion technique of Fiat and Naor [9].

We first assume, without loss of generality, that no point of $A \cup B$ lies on γ. If γ contains a point of A and a point of B then every query has a positive outcome and the problem becomes trivial. If γ contains points of only A, say, then these points can form collinear triples only when q coincides with one of them, a situation that is easy to detect.

Our strategy is to consider all bichromatic lines that connect a point of A with a point of B. Let $a = (x_a, y_a) \in G$ and $b = (x_b, y_b) \in G$ be two grid points. The line $\lambda_{a,b}$ that passes through a and b has the equation

$$\frac{y - y_a}{x - x_a} = \frac{y_b - y_a}{x_b - x_a}, \qquad \text{or}$$
$$(y_b - y_a)(x - x_a) - (x_b - x_a)(y - y_a) = 0, \qquad \text{or}$$
$$(y_b - y_a)x - (x_b - x_a)y = x_a y_b - y_a x_b.$$

Since a and b belong to G, we have

$$y_b - y_a,\ x_b - x_a \in [-2(k-1), 2(k-1)], \qquad \text{and} \qquad x_a y_b - y_a x_b \in [-2t(k-1) - 2(k-1)^2, 2t(k-1) + 2(k-1)^2].$$

Let Λ denote the set of the lines $\lambda_{a,b}$. Note that $|\Lambda| = O(k^4)$. A line $\lambda \in \Lambda$ intersects the y-axis at the point with y-coordinate

$$y = \frac{x_a y_b - y_a x_b}{x_a - x_b},$$

which is a rational with denominator in $[-2(k-1), 2(k-1)]$ and numerator in $[-2t(k-1) - 2(k-1)^2, 2t(k-1) + 2(k-1)^2]$.

Set

$$U = \left\{ \left(0, \frac{i}{4k^2}\right) \mid |i| \le 8tk^3 + 8k^4 \right\}.$$

It is easily checked that (i) each intercept of a line of Λ with the y-axis lies in an interval delimited by two consecutive points of U, and (ii) each of these intervals contains at most one such intercept. Indeed, (i) follows from the definition of U and the respective values of the y-coordinates, and (ii) holds since the smallest difference between two such intercepts is at least $1/(2k-2) - 1/(2k-1)$, which is larger than $1/(4k^2)$. (An intercept can be shared by many lines of Λ, but distinct intercepts lie in distinct intervals.) Let V denote the (multi)set of these intercepts.

Preparing for the Fiat-Naor setup. Write $A = \{a_1, a_2, \ldots, a_n\}$ and $B = \{b_1, b_2, \ldots, b_n\}$. Following the considerations in [11, 12], we define a function $g :\ [n] \times [n] \mapsto \gamma$ by setting $g(i,j)$ equal to the (unique) intersection point of γ with the line $\ell_{i,j} = \lambda_{a_i, b_j}$. The image of g is the multi-set V.

Let $h :\ U \mapsto [n] \times [n]$ be a *universal hashing function*. That is, for each pair $x \ne y \in U$ we have $\Pr[h(x) = h(y)] \le \frac{1}{n^2}$, where probability is with respect of the random choice of h. One can construct such a function by encoding U as a range of integers (using only the numerators of its elements), and encoding $[1,n] \times [1,n]$ as $[1, n^2]$, using, e.g., the lexicographical order, and use the class of functions of the form $h(i) = (ci + d \pmod p) \pmod{n^2}$, where $p > n^2$ is a prime number and c, d are random integers modulo p, with $c \ne 0$. Finally, define a function $f :\ [n] \times [n] \mapsto [n] \times [n]$ by

$$f(i,j) = h(\mathrm{flr}_\delta(g(i,j))),$$

for $(i,j) \in [n] \times [n]$, where $\delta = \frac{1}{4k^2}$ is the separation parameter between any pair of consecutive elements of U, and flr_δ maps each point $q \in \gamma$ to the largest element of U that is smaller than or equal to q.

Building the data structure. We now use the trade-off result of Fiat and Naor [9] for inverting a function. It implies that, for any choice of values (S,T) that obey $TS^3 = n^6$, one can preprocess f, by a randomized algorithm, in time $O^*(n^2)$ into a data structure that uses $O^*(S)$ storage, so that, for any (i,j) in the range of f, one can compute, in $O^*(T)$ time, with probability $1 - \frac{1}{n^2}$, a pair in $f^{-1}(i,j)$. Since f may be many-to-one, $f^{-1}(i,j)$ may consist of many elements; in this case, the Fiat-Naor procedure returns just one of them. This is sufficient for our analysis as such a scenario implies that we may have several potential pairs $(a,b) \in A \times B$, which are collinear with the query point q—see below.

An element z of the range $R \subseteq U$ of $\mathrm{flr}_\delta \circ g$ is said to be an *h-singleton* if for any $z' \neq z$ in R we have $h(z) \neq h(z')$. Standard properties of universal hash functions (see once again [11, 12]) ensure that the expected number of singleton elements in R is at least some constant fraction of $|R|$. Hence, if we repeat this scheme $\Theta(\log n)$ times, drawing h independently at each incarnation, the expected number of elements that are not singletons in all the schemes becomes smaller than 1, and we can therefore assume that our structure has the property that every element of R is singleton in at least one instance.

We query with a point $q \in \gamma$ in each of these $O(\log n)$ structures, collect $O(\log n)$ potential inverses of f at $h(\mathrm{flr}_\delta(q))$, and claim, using the above reasoning, that if $q \in R$ then at least one of these candidates is a pair (i,j) such that a_i, b_j and q are collinear. We go over the candidates and select the one that satisfies this property. If such a candidate exists then we found a pair in $(a,b) \in A \times B$ with which q is collinear. Otherwise, if none of the candidates satisfies the property, we conclude that q is not collinear with any pair in $A \times B$.

In summary, we obtain $O(\log n)$ structures, whose overall space complexity is $O^*(S)$, and the total query time is $O^*(T)$ (where $TS^3 = n^6$), as is easily verified. This implies that by setting $T = n^\delta$, we obtain $S = n^{2-\delta/3}$, for any $0 < \delta < 1$. In particular, we can choose S and T so that S is subquadratic and T is sublinear. For example, we can choose $T = n^{3/4}$ and $S = n^{7/4}$. This at last completes the proof of Theorem 1.1.

Acknowledgments. We would like to thank Tsvi Kopelowitz for helpful discussions.

References

1 P. K. Agarwal, Simplex range searching and its variants: A review, in *Journey through Discrete Mathematics: A Tribute to Jiří Matoušek* (M. Loebl, J. Nešetřil and R. Thomas, editors), Springer Verlag, Berlin-Heidelberg, 2017, pages 1–3.

2 P. K. Agarwal and J. Erickson, Geometric range searching and its relatives, in *Advances in Discrete and Computational Geometry* (B. Chazelle, J. E. Goodman and R. Pollack, editors), Contemp. Math. 223, AMS Press, Providence, RI, 1999, pages 1–56.

3 B. Aronov, E. Ezra, and M. Sharir. Testing polynomials for vanishing on Cartesian products of planar point sets. *Proc. 36th Sympos. on Computational Geometry* (2020), 8:1–8:14. Also in arXiv:2003.09533.

4 L. Barba, J. Cardinal, J. Iacono, S. Langerman, A. Ooms, and N. Solomon, Subquadratic algorithms for algebraic 3Sum, *Discrete Comput. Geom.* 61 (2019), 698–734. Also in *Proc. 33rd Inter. Sympos. Comput. Geom.* (2017), 13:1–13:15.

5 Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics).* Springer-Verlag, Berlin, Heidelberg, 2006.

6 T. M. Chan, More logarithmic-factor speedups for 3Sum, (median,+)-convolution, and some geometric 3Sum-hard problems, *ACM Trans. Algorithms* 16 (2020), 7:1–7:23.

7 M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry, Algorithms and Applications*, 3rd edition, Springer Verlag, 2008.

8 A. Dumitrescu and W. L. Steiger, Space-time trade-offs for some ranking and searching queries. *Inf. Process. Lett.*, 79(5):237-241 (2001).

9 Amos Fiat and Moni Naor. Rigorous time/space tradeoffs for inverting functions. In *SIAM Journal on Computing*, pages 534–541. ACM, ACM Press, 2000.

10 A. Gajentaan and M. H. Overmars, On a class of $O(n^2)$ problems in computational geometry, *Comput. Geom. Theory Appl.* 5 (1995), 165–185.

11 A. Golovnev, S. Guo, T. Horel, S. Park and V. Vaikuntanathan, Data structures meet cryptography: 3SUM with preprocessing. *Proc. 52nd Annu. ACM SIGACT Sympos. Theory Comput., STOC*, (2020) pp. 294–307. Also in arXiv:1907.08355.

12 Tsvi Kopelowitz and Ely Porat. The Strong 3SUM-INDEXING Conjecture is False. *arXiv e-prints*, page arXiv:1907.11206, July 2019. `arXiv:1907.11206`.

13 Dumitrescu, A. & Steiger, W. Space-time trade-offs for some ranking and searching queries. *Information Processing Letters.* **79**, 237-241 (2001,9)

Universal Lower Bounds on the Segment Number of Some Classes of Planar Graphs

Jonathan Klawitter[1], Boris Klemz[1], Felix Klesen[1], Stephen Kobourov[2], Myroslav Kryven[2], Alexander Wolff[1], and Johannes Zink[1]

1 **Universität Würzburg**
`firstname.lastname@uni-wuerzburg.de`

2 **University of Arizona**
`firstname.lastname@email.arizona.edu`

Abstract

The *segment number* of a planar graph G is the smallest number of line segments needed for a planar straight-line drawing of G. Dujmović, Eppstein, Suderman, and Wood [Comp. Geom., 2007] introduced this measure for the *visual complexity* of a graph. Several upper bounds on the segment number have been established, e.g., Dujmović et al. gave an optimal algorithm for trees and worst-case optimal algorithms for outerplanar graphs, 2-trees, and planar 3-trees. We prove the first linear *universal* lower bounds for maximal outerpaths, maximal outerplanar graphs, and 2-trees. This makes the corresponding algorithms of Dujmović et al. constant-factor approximation algorithms. For maximal outerpaths, our bound is best possible and can be generalized to circular arcs.

1 Introduction

A drawing of a given graph can be evaluated by various quality measures depending on the concrete purpose of the drawing. Classic examples of such measures include drawing area, number of edge crossings, neighborhood preservation, and stress of the embedding. More recently, Schulz [5] proposed the *visual complexity* of a drawing, determined by the number of geometric objects (such as line segments or circular arcs) that the drawing consists of. It has been experimentally verified that people without mathematical background tend to prefer drawings with low visual complexity [4]. The visual complexity of a graph drawing depends on the drawing style, as well as on the underlying graph properties. A well-studied measure of the visual complexity of a graph is its segment number, introduced by Dujmović, Eppstein, Suderman, and Wood [1]. It is defined as follows. Recall that a *straight-line drawing* of a graph maps (i) the vertices of the graph injectively to points in the plane and (ii) the edges of the graph to straight-line segments that connect the corresponding points. A *segment* in such a drawing is a maximal set of edges that together form a line segment. For a given straight-line drawing Γ of a graph, the set of segments it induces is unique. The cardinality of that set is the *segment number* of Γ. The *segment number*, $\operatorname{seg}(G)$, of a planar graph G is the smallest segment number over all crossing-free straight-line drawings of G.

Previous work. Dujmović et al. [1] pointed out two natural lower bounds for the segment number: (i) $\eta(G)/2$, where $\eta(G)$ is the number of odd-degree vertices of G, and (ii) the slope number $\operatorname{slope}(G)$ of G, which is defined as follows. The slope number $\operatorname{slope}(\Gamma)$ of a straight-line drawing Γ of G is the number of different slopes used by any of the straight-line edges in Γ. Then $\operatorname{slope}(G)$ is the minimum of $\operatorname{slope}(\Gamma)$ over all straight-line drawings Γ of G. Dujmović et al. also showed that any tree T admits a drawing with $\operatorname{seg}(T) = \eta(T)/2$

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

segments and $\text{slope}(T) = \Delta(T)/2$ slopes, where $\Delta(T)$ is the maximum degree of a vertex in T. These drawings, however, use exponential area. Further, Dujmović et al. showed that every maximal outerplanar graph G with n vertices admits an *outerplanar* straight-line drawing with at most n segments. They showed that this is worst-case optimal. They also gave (asymptotically) worst-case optimal algorithms for 2-trees and plane (where the combinatorial embedding and outer face is fixed) 3-trees. Finally, they showed that every triconnected planar graph with n vertices can be drawn using at most $5n/2 - 3$ segments. For the special cases of triangulations and 4-connected triangulations, Durocher and Mondal [2] improved the upper bound of Dujmović et al. to $(7n - 10)/3$ and $(9n - 9)/4$, respectively. The former bound implies a bound of $(16n - 3m - 28)/3$ for arbitrary planar graphs with n vertices and m edges. The *arc number*, $\text{arc}(G)$, of a graph G is the smallest number of circular arcs in any circular-arc drawings of G. It has been introduced by Schulz [5], who gave algorithms for drawing series-parallel graphs, planar 3-trees, and triconnected planar graphs with few circular arcs. For trees, he used the additional flexibility of circular arcs to trade an increase in the visual complexity for a reduction in the size of the drawing area (from exponential to a grid of size $O(n^{2.81})$).

Contribution and outline. We prove the first linear universal lower bound of $\lfloor n/2 \rfloor + 2$ for the segment number of maximal outerpaths with n vertices; see Sec. 3. We obtain our results by using more general *pseudo k-arcs*, that is, an arrangement of curves in the plane such that each pair of curves intersects at most k times. For $k = 1$ these are pseudo segments and for $k = 2$ these are pseudo (circular) arcs. This also yields a universal lower bound of $\lceil 2n/7 \rceil$ for the arc number of maximal outerpaths. We present infinite families of maximal outerpaths using few segments, arcs, and pseudo arcs, which show that our bound on the segment number is tight. Via considering maximal outerpaths, we also obtain a universal lower bound of $(n + 7)/5$ for the segment number of 2-trees (which contain the maximal outerplanar graphs); see Sec. 4. This makes the corresponding algorithms of Dujmović et al. constant-factor approximation algorithms.

Results marked with a "$\star$" are available in the full version [3].

2 Notation and Terminology

Let G be a planar and connected graph and let Γ be a planar drawing of G. The unique unbounded face of Γ is called its *outer* face; the remaining faces are called *internal*. Vertices (edges) belonging to the boundary of the outer face are called *outer* vertices (edges); the remaining vertices (edges) are called *internal*. A *plane* graph is a planar graph equipped with a combinatorial embedding and a distinguished outer face.

Let s be a segment in a straight-line drawing Γ, and let v be an endpoint of s. Geometrically speaking, we could extend s at v into a face f. We say that s has a *port* at v in f. We call v *open* if v has at least one port and *closed* otherwise. Let $\text{port}(\Gamma)$ be the number of ports in Γ, and let $\text{port}(G)$ be the minimum number of ports over all straight-line drawings of G. Observe that, for any planar graph G, it holds that $\text{seg}(G) = \text{port}(G)/2$.

An *outerplanar graph* is a plane graph with all vertices outer. The *weak dual graph* of an outerplane graph is its dual graph without the vertex corresponding to the outer face; it is known to be a tree. An outerplane graph whose weak dual is a path is called an *outerpath*. A *maximal outerplanar graph* is an outerplanar graph with the maximum number of edges. Similarly, we define a *maximal outerpath*. A *2-tree* is a graph that can be constructed by starting with a K_3 and iteratively adding vertices that are adjacent to both endpoints of an existing edge of the current 2-tree. Clearly, all 2-trees are planar and all maximal outerplanar graphs are 2-trees. We call the sequence of vertices $v_1, v_2, \ldots, v_n$ of a 2-tree G its *stacking order* if, for each $i \in \{3, \ldots, n\}$, the graph G_i induced by the vertices $v_1, v_2, \ldots, v_i$ is a 2-tree. If G is a maximal outerpath, each G_i is a maximal outerpath.

3 Maximal Outerpaths

In this section, we generalize the concept of segments and arcs to pseudo k-arcs and give a universal lower bound for the number of pseudo k-arcs in drawings of maximal outerpaths. An arrangement of *pseudo k-arcs* is a set of curves in the plane such that any two of the curves intersect at most k times. (If two curves share a tangent, this counts as two intersections.) We forbid self-intersections, but for $k \geq 2$ we allow a pseudo k-arc to be closed.

To show the bound, we present a charging scheme that assigns internal edges to pseudo k-arcs. Any outerpath drawing has exactly $n-3$ internal edges. A pseudo k-arc is *long* if it contains at least $k+1$ internal edges; otherwise it is *short*. Let arc_k denote the number of pseudo k-arcs, and let arc_k^i denote the number of pseudo k-arcs with i internal edges. The internal edges of a long arc α subdivide the outerpath into subgraphs $H_0, H_1, \ldots, H_\ell$ called *bays*; see Fig. 1. Since an outerpath can be constructed by a sequence of (outer) 2-tree stacking operations, we have the partial drawings $P_3, P_4, \ldots, P_n$. A pseudo k-arc α is *incident* to a face f if α contains an edge incident to a vertex of f. We say that α is *active* in P_i if α is incident to the last face that has been added.

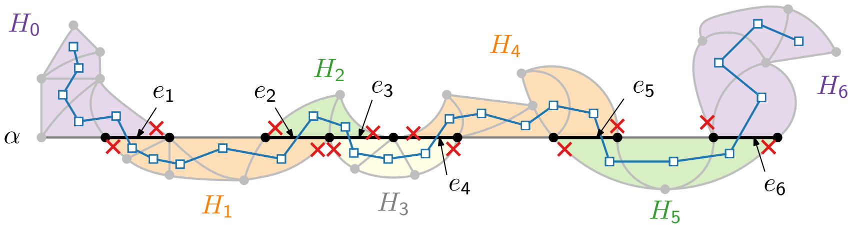


Figure 1 An outerpath represented by a pseudo 2-arc arrangement. The internal edges $e_1, \ldots, e_6$ of arc α subdivide the outerpath into bays $H_0, \ldots, H_6$. For our charging, we count crossings of α with other arcs (indicated by red crosses).

▶ **Lemma 1** (⋆). *For any $i \in \{3, \ldots, n\}$, a partial outerpath drawing P_i contains at most one active long pseudo k-arc.*

We do a 2-round assignment to assign each internal edge to a pseudo k-arc. We start with the **round-1 assignment**. Let I denote the set of internal edges of long pseudo k-arcs starting at the $(k+1)$-th internal edge (as for the first k internal edges an arc is still short). We assign all $n-3$ internal edges except for the edges in I to their own pseudo k-arcs:

$$(n-3) - |I| = k\,\mathrm{arc}_k^{\geq k} + (k-1)\,\mathrm{arc}_k^{k-1} + \cdots + \mathrm{arc}_k^1 = k\,\mathrm{arc}_k - \sum_{i=0}^{k} (k-i)\,\mathrm{arc}_k^i \quad (1)$$

Now we describe the **round-2 assignment**. There, we charge internal edges to crossings, which we can charge in turn to pseudo k-arcs. A *crossing* is a triplet (α, β, p) that consists of two pseudo k-arcs α and β and a point p at which α and β intersect. If there is a tangential point between α and β, we count two crossings at that tangential point.

Next we consider the number of crossings in an outerpath drawing that involve the long arcs. In the round-2 assignment we charge the surplus internal edges of the long arcs to the other pseudo k-arcs that are involved in the crossings we counted. Suppose α has ℓ internal edges ($\ell > k$). For each bay $H \in \{H_1, \dots, H_{\ell-1}\}$, there are ≥ 2 crossings of α with other pseudo k-arcs – one at the first and one at the last vertex of H; see the red crosses in Fig. 1.

These vertices are individual for each pair of consecutive bays. Hence, the crossings must all be different as well. Note that a tangential point may be shared by some H_j and H_{j+2} (for $j \in \{1, \dots, \ell-3\}$); see H_2 and H_4 in Fig. 1 for an example. However, we can still charge a crossing to each of H_j and H_{j+2} since a tangential point counts for two crossings. An exception are H_0 and H_ℓ. They may share a crossing with H_1 and $H_{\ell-1}$, respectively, as H_6 and H_5 do in Fig. 1. Hence, we count only one crossing for H_0 and H_ℓ.

Therefore, for each internal edge e of I we have two individual crossings of the preceding bay, e.g., in Fig. 1 H_2 provides two crossings for e_3. We denote the set of these crossings by C. Note that the crossings of $H_0, H_1, \dots, H_{k-1}$, and H_ℓ are not included in C since the succeeding internal edges are not contained in I. Clearly, we know that $2|I| = |C|$.

Next, we give an upper bound for $|C|$ in terms of arc_k. The main argument we exploit is that, by definition, each pseudo k-arc can participate in at most k crossings with the (current) long arc. However, we need to be a bit careful for the case when one long pseudo k-arc becomes inactive and a new pseudo k-arc becomes long, i.e., we consider the transition between one long arc to a new long arc. For each long arc, we count the crossings in $H_k, H_{k+1}, \dots, H_\ell$ as described before. We may get additional crossings because potentially any pseudo k-arc could intersect each long arc k times. To compensate for the double counting at transitions, we introduce the *transition loss* t_k. Moreover, we cannot count crossings of the first long arc with (other) long arcs, and we do not count the crossings of the very first bay and the very last bay. This yields the following.

$$2|I| = |C| \leq \underbrace{k \cdot (\mathrm{arc}_k}_{\substack{\text{each pseudo } k\text{-arc intersects the current} \\ \text{long pseudo } k\text{-arc at most } k \text{ times}}} \overbrace{-1)}^{\substack{\text{the first long pseudo } k\text{-arc does not provide} \\ \text{crossings with another long pseudo } k\text{-arc}}} \underbrace{-(2k-1)}_{\substack{\text{the crossings of } H_0, H_1, \dots, H_{k-1} \text{ of the first} \\ \text{long pseudo } k\text{-arc are not counted}}} \overbrace{-1}^{\substack{\text{the crossing of } H_\ell \text{ of the last long} \\ \text{pseudo } k\text{-arc is not counted}}} \underbrace{+t_k}_{\substack{\text{transition} \\ \text{loss}}} \tag{2}$$

Plugging Eq. (2) into Eq. (1), we obtain the following general formula, which gives a lower bound on the number of pseudo k-arcs for any outerpath relative to n and k.

$$(n-3) - \frac{k(\mathrm{arc}_k - 1) - (2k-1) - 1 + t_k}{2} \leq k\,\mathrm{arc}_k - \sum_{i=0}^{k} (k-i)\,\mathrm{arc}_k^i$$
$$\Leftrightarrow \mathrm{arc}_k \geq \frac{2n - 6 + 2\sum_{i=0}^{k}(k-i)\,\mathrm{arc}_k^i - t_k}{3k} + 1 \tag{3}$$

Since this formula still contains unresolved variables, we now resolve t_k.

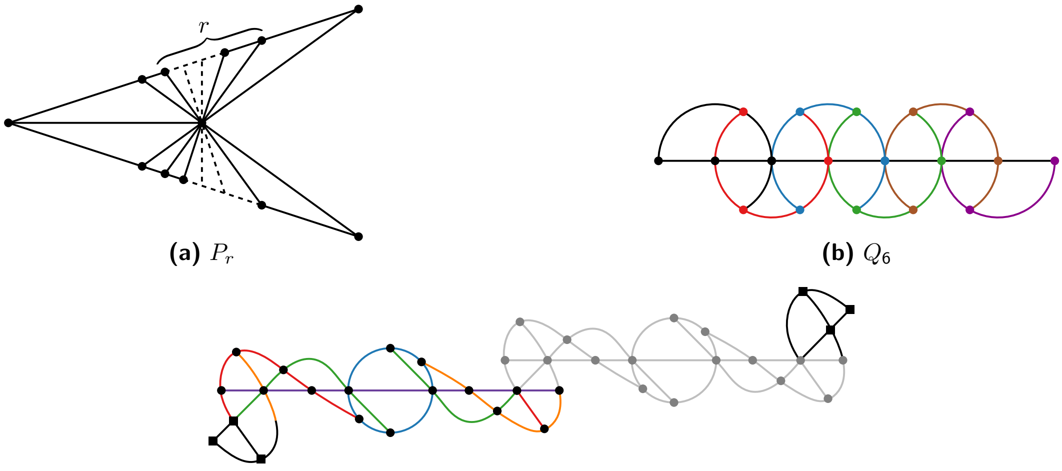
▶ **Lemma 2** ($\star$). *There is a loss of at most one crossing per transition from one long pseudo k-arc to another long pseudo k-arc in any outerpath drawing. Hence,* $t_k \leq \max\{0, \mathrm{arc}_k^{>k} - 1\} \leq \mathrm{arc}_k^{>k} = \mathrm{arc}_k - \sum_{i=0}^{k} \mathrm{arc}_k^i$ *where* $\mathrm{arc}_k^{>k}$ *is the number of long pseudo k-arcs.*

Applying this insight to Eq. (3), we get

$$\mathrm{arc}_k \geq \frac{2n + 3k - 6 + \sum_{i=0}^{k}(2k - 2i + 1)\,\mathrm{arc}_k^i}{3k+1}. \tag{4}$$

Since this general formula is hard to grasp and still contains the unresolved variables arc_k^i, we next investigate this formula for specific values of k and prove lower bounds on arc_k^i.

Let us start with $k = 1$, i.e, outerpath drawings on pseudo segments. We use the following lemma to fill the unresolved values in Eq. (4).



(a) P_r

(b) Q_6

(c) U_2

Figure 2 Three families of maximal outerpaths with (a) $n/2 + 2$ segments (hence, matching the lower bound from Thm. 1), (b) $n/3 + 1$ circular arcs, and (c) $(5n + 18)/16 < n/3$ pseudo 2-arcs.

▶ **Lemma 3** ($\star$). *For $k = 1$ and $n \geq 3$, in any outerpath drawing either $\text{arc}_1^0 \geq 3$ or $(\text{arc}_1^0 \geq 2$ and $\text{arc}_1^1 \geq 3)$.*

Now we can use Lem. 3 to fill the gaps in Eq. (4) for $k = 1$:

▶ **Theorem 1.** *For any maximal outerpath P with n vertices, $\text{seg}(P) \geq \text{arc}_1(P) \geq \lfloor \frac{n}{2} \rfloor + 2$.*

Proof. We plug the result from Lem. 2 into Eq. (4) for $k = 1$ and use Lem. 3 to observe that $3\,\text{arc}_1^0 + \text{arc}_1^1 \geq 9$:

$$\text{arc}_1 \geq \frac{2n - 3 + 3\,\text{arc}_1^0 + \text{arc}_1^1}{4} = \frac{n}{2} + \frac{3\,\text{arc}_1^0 + \text{arc}_1^1 - 3}{4} \geq \frac{n+3}{2}$$

As we cannot have partial (pseudo) segments, we can round up to $\lceil \frac{n+3}{2} \rceil = \lfloor \frac{n}{2} \rfloor + 2$. ◀

For $k = 2$, i.e, for (pseudo) circular arcs, Eq. (4) leads to the following lower bound.

▶ **Theorem 2.** *For any maximal outerpath P with n vertices, $\text{arc}(P) \geq \text{arc}_2(P) \geq \lceil \frac{2n}{7} \rceil$.*

Proof. We plug the result from Lem. 2 into Eq. (4) for $k = 2$:

$$\text{arc}_2 \geq \frac{2n + 5\,\text{arc}_2^0 + 3\,\text{arc}_2^1 + 1\,\text{arc}_2^2}{7} \geq \frac{2n}{7}$$

◀

For $k > 2$, it is not obvious how to generalize circular arcs. However, we can make a similar statement for curve arrangements. This follows directly from Eq. (4).

▶ **Proposition 1.** *Let P be a maximal outerpath with n vertices drawn on an arrangement of curves in the plane where curves intersect pairwise at most k times, can be closed, but do not self-intersect. Then, the number $\text{arc}_k(P)$ of curves required is $\lceil \frac{2n+3k-6}{3k+1} \rceil$.*

We show by three infinite families of examples in Fig. 2 that our bounds for segments and arcs are tight. This implies, somewhat surprisingly, that, at least for worst-case instances, using pseudo segments requires as many elements as using straight line segments. Whether this also holds for pseudo circular arcs and circular arcs is an open question. With circular arcs, we could not beat a bound of $n/3$, which we could do for pseudo circular arcs.

▶ **Proposition 2** ($\star$). *For every $r \in \mathbb{N}$, there exist maximal outerpaths P_r, Q_r, and U_r with (i) P_r has $2r+6$ vertices and $\operatorname{seg}(P_r) \le r+5 = \frac{n}{2}+2$, (ii) Q_r has $3r$ vertices and $\operatorname{arc}(Q_r) \le r+1 = \frac{n}{3}+1$, (iii) U_r has $16r+6$ vertices and $\operatorname{arc}_2(U_r) \le 5r+3 = \frac{5n+18}{16} \approx 0.3125\,n$.*

4 2-Trees

The main idea for a universal lower bound for 2-trees (and for its subclass of maximal outerplanar graphs) is that a 2-tree G either has many degree-2 vertices and thus requires many segments (recall that, in a 2-tree, we stack vertices onto edges and hence degree-2 vertices cannot be closed) or G can be obtained by "gluing" few maximal outerpaths for which we know (tight) universal lower bounds on the segment number; see Thm. 1.

Unfortunately, for gluing maximal outerpaths, we cannot directly employ Thm. 1 because it does not tell us how many ports we lose when gluing. Therefore, we first investigate the distribution of ports within a straight-line drawing of a maximal outerpath. In particular, for any straight-line drawing of any maximal outerpath, we can find an injective assignment of ports to vertices such that every port is assigned to its own vertex or to a neighboring vertex.

Using this assignment, we can show that we lose at most seven (assigned) ports by gluing an outerpath to a 2-tree. Consequently, one can see that any 2-tree drawing has at least $2(n+7)/5$ ports, which yields the following result.

▶ **Theorem 3** ($\star$). *Let G be a 2-tree with n vertices. Then $\operatorname{seg}(G) \ge (n+7)/5$.*

5 Discussion

Circular-arc drawings are a natural generalization of straight-line drawings. What is the maximum ratio between the segment number and the arc number of a graph? Note that $\operatorname{seg}(K_3)/\operatorname{arc}(K_3) = 3$, but it is open whether it can be larger. What is the complexity of deciding whether the arc number of a given graph is strictly smaller than its segment number?

References

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

2 Stephane Durocher and Debajyoti Mondal. Drawing plane triangulations with few segments. *Comput. Geom. Theory Appl.*, 77:27–39, 2019. doi:10.1016/j.comgeo.2018.02.003.

3 Ina Goeßmann, Jonathan Klawitter, Boris Klemz, Felix Klesen, Stephen Kobourov, Myroslav Kryven, Alexander Wolff, and Johannes Zink. The segment number: Algorithms and universal lower bounds for some classes of planar graphs. *arXiv preprint*, 2022. URL: https://arxiv.org/abs/2202.11604.

4 Philipp Kindermann, Wouter Meulemans, and André Schulz. Experimental analysis of the accessibility of drawings with few segments. *J. Graph Alg. Appl.*, 22(3):501–518, 2018. doi:10.7155/jgaa.00474.

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

Outside-Obstacle Representations with All Vertices on the Outer Face

Oksana Firman[1], Philipp Kindermann[2], Jonathan Klawitter[1], Boris Klemz[1], Felix Klesen[1], and Alexander Wolff[1]

1 **Universität Würzburg, Germany**
`firstname.lastname@uni-wuerzburg.de`

2 **Universität Trier, Germany**
`kindermann@uni-trier.de`

Abstract

An *obstacle representation* of a graph G consists of a set of polygonal obstacles and a drawing of G as a *visibility graph* with respect to the obstacles: vertices are mapped to points and edges to straight-line segments such that each edge avoids all obstacles whereas each non-edge intersects at least one obstacle. Obstacle representations have been investigated quite intensely over the last few years. Here we focus on *outside-obstacle representations* that use only one obstacle in the outer face of the drawing. It is known that every outerplanar graph admits such a representation [Alpert, Koch, Laison; DCG 2010]. We strengthen this result by showing that every partial 2-tree has an outside-obstacle representation. We also consider a restricted version of outside-obstacle representations where the vertices lie on a regular polygon. We construct such regular representations for partial outerpaths, partial cactus graphs, and partial grids.

1 Introduction

Recognizing graphs that have a certain type of geometric representation is a well-established field of research dealing with, for example, interval graphs, unit disk graphs, coin graphs, and visibility graphs. Given a set $\mathcal{C}$ of *obstacles* (in our case: polygons) and a set P of points in the plane, the *visibility graph* $G_{\mathcal{C}}(P)$ has a vertex for each point in P and an edge pq for any two points p and q in P that can *see* each other, that is, the line segment $\overline{pq}$ that connects p and q does not intersect any obstacle in $\mathcal{C}$. An *obstacle representation* of a graph G consists of a set $\mathcal{C}$ of obstacles in the plane and a mapping of the vertices of G to a set P of points such that $G = G_{\mathcal{C}}(P)$. Drawing the edges of the visibility graph as straight-line segments allows us to differentiate between two types of obstacles: *outside* obstacles lie in the outer face of the drawing, and *inside* obstacles lie in the complement of the outer face; see Fig. 1.

Every graph trivially admits an obstacle representation: take an arbitrary drawing and "fill" each face with an obstacle. However, this can lead to a large number of obstacles. Hence, it makes sense to consider the optimization problem of finding an obstacle representation

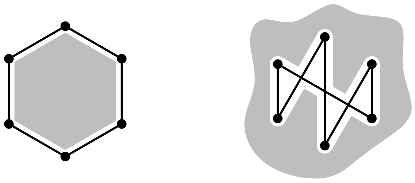


Figure 1 Two representations of C_6: with an inside obstacle (left) and an outside obstacle (right).

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

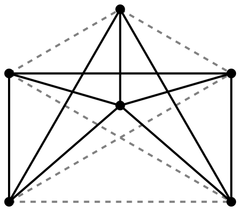


Figure 2 The wheel graph W_6 admits an outside-obstacle representation – but not a convex one (see full version [6]). Non-edges are dashed.

with the minimum number of obstacles. For a graph G, the *obstacle number* $\operatorname{obs}(G)$ is the smallest number of obstacles that suffice to represent G as a visibility graph.

In this paper, we focus on *outside-obstacle representations*, that is, obstacle representations with a single outside obstacle and without any inside obstacles. For such a representation, it suffices to specify the position of the vertices. The outside obstacle is simply the whole outer face of the straight-line drawing of the graph. We also consider three special types: In a *convex outside-obstacle representation*, the vertices must be in convex position; in a *circular outside-obstacle representation*, the vertices must lie on a circle; and in a *regular outside-obstacle representation*, the vertices must form a regular n-gon.

In general, the class of graphs representable by outside obstacles is not closed under taking subgraphs, but the situation is different for graphs admitting an outside-obstacle representation that is *reducible*, meaning that all of its edges are incident to the outer face:

▶ **Observation 1.** *If a graph G admits a reducible outside-obstacle representation, then every subgraph of G also admits such a representation.*

Previous Work. The notion of the obstacle number of a graph has been introduced by Alpert et al. [1]. They also introduced inside-obstacle representations, i.e., representations without an outside-obstacle. They characterized the class of graphs that have an inside-obstacle representation with a single convex obstacle and showed that every outerplanar graph has an outside-obstacle representation. They showed, for any $m \le n$, that $\operatorname{obs}(K^*_{m,n}) \le 2$, where $K^*_{m,n}$ with $m \le n$ is the complete bipartite graph minus a matching of size m. They also proved that $\operatorname{obs}(K^*_{5,7}) = 2$. Pach and Sarıöz [10] showed that $\operatorname{obs}(K^*_{5,5}) = 2$. More recently, Berman et al. [3] suggested some necessary conditions for a graph to have obstacle number 1, which they used to find a *planar* 10-vertex graph that has no 1-obstacle representation.

Obviously, any n-vertex graph has obstacle number $O(n^2)$. Balko et al. [2] improved this to $O(n \log n)$. For the lower bound, Dujmović and Morin [5] showed that there are n-vertex graphs whose obstacle number is $\Omega(n/(\log \log n)^2)$, improving on previous results [1,8,9].

Chaplick et al. [4] proved that the class of graphs with an inside-obstacle representation is incomparable with the class of graphs with an outside-obstacle representation. They showed that any graph with at most seven vertices has an outside-obstacle representation, which does not hold for a specific graph with eight vertices.

Our Contribution. We first establish two combinatorial conditions for convex outside-obstacle representations (see Section 2) that we later use to establish our main results. In particular, we introduce a necessary condition that can be used to show that a given graph does *not* admit a convex representation as, e.g., the graph in Fig. 2. We construct *regular* reducible outside-obstacle representations for outerpaths, grids, and cacti; see Section 3. Finally, we strengthen the result of Alpert et al. [1] about outside-obstacle representations of outerplanar graphs by showing that every (partial) 2-tree admits a reducible outside-obstacle representation with all vertices on the outer face; see Section 4. We remark that outerplanar graphs and series-parallel graphs are partial 2-trees.

2 Conditions for Convex Outside-Obstacle Representations

We start with a sufficient condition. Suppose that we have a convex outside-obstacle representation of a graph G. Let σ be the clockwise circular order of the vertices of G along the convex hull. If all neighbors of a vertex v of G are consecutive in σ, we say that v has the *consecutive-neighbors property*, which implies that all non-edges incident to v trivially intersect the outer face in the immediate vicinity of v; see Fig. 3a.

▶ **Lemma 2** (Consecutive-neighbors property). *For a graph G, a circular vertex order σ admits a convex outside-obstacle representation if a subset of $V(G)$ that covers all non-edges has the consecutive-neighbors property.*

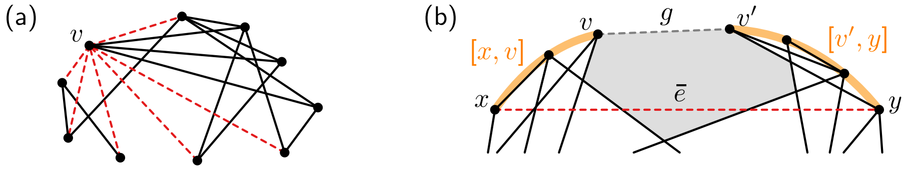


Figure 3 (a) Vertex v has the consecutive-neighbors property; (b) gap g is a candidate gap for $\bar{e}$.

Next, we derive a necessary condition. For any two consecutive vertices v and v' on the convex hull that are not adjacent in G, we say that the line segment $g = \overline{vv'}$ is a *gap*. Then the *gap region* of g is the inner face of $G + vv'$ incident to g; see the gray region in Fig. 3b. (We consider the gap region open, but add to it the relative interior of the line segment $\overline{vv'}$, so that the non-edge vv' actually intersects its own gap region.) Observe that each non-edge $\bar{e} = xy$ that intersects the outer face has to intersect some gap region in an outside-obstacle representation. For vertices a and b, the set $[a, b] \subseteq V(G)$ consists of a and b and all vertices that succeed a and precede b in σ. Suppose that g lies between x and y with respect to σ, that is, $[v, v'] \subseteq [x, y]$. We say that g is a *candidate gap* for $\bar{e}$ if there is no edge that connects a vertex in $[x, v]$ and a vertex in $[v', y]$. (Otherwise $\bar{e}$ cannot intersect the gap region of g.)

▶ **Lemma 3** (Gap condition). *For a graph G, a circular vertex order σ admits a convex outside-obstacle representation only if there exists a candidate gap for each non-edge of G.*

It remains an open problem whether the gap condition is also sufficient. We can use the gap condition for no-certificates. To this end, we derived a SAT formula from the following expression, which checks the gap condition for every non-edge of a graph G:

$$\bigwedge_{xy \notin E(G)} \left[\bigvee_{v \in [x,y)} \left(\bigwedge_{u \in [x,v], w \in (v,y]} uw \notin E(G) \right) \vee \bigvee_{v \in [y,x)} \left(\bigwedge_{u \in [y,v], w \in (v,x]} uw \notin E(G) \right) \right]$$

We have used this formula to test whether all small cubic graphs (with up to 16 vertices) admit convex outside-obstacle representations. The only counterexample we found was the Petersen graph. The so-called Blanusa snarks, the Pappus graph, the dodecahedron, and the generalized Peterson graph $G(11, 2)$ satisfy the gap condition. The latter three graphs do admit convex outside-obstacle representations [7]. This motivates the following conjecture.

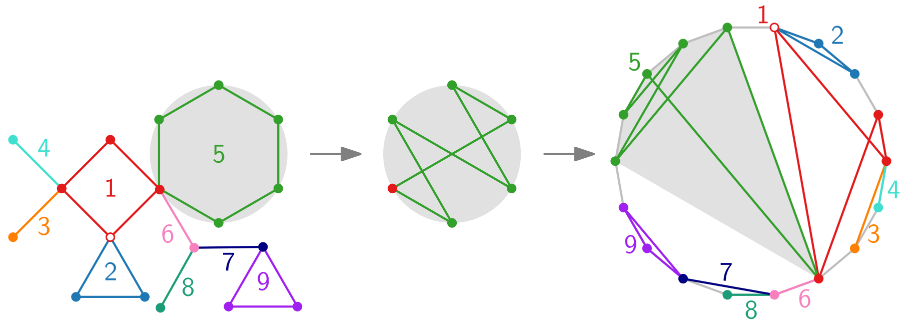


Figure 4 Constructing a reducible regular outside-obstacle representation of a cactus.

▶ **Conjecture 4.** *Every connected cubic graph* except the Peterson graph *admits a convex outside-obstacle representation.*

The smallest graph (and only graph with six vertices) that does not satisfy the gap condition is the wheel graph W_6 with six vertices (see the full version [6]). Obviously, W_6 does not admit a *convex* outside-obstacle representation, but it does admit a (non-convex) outside-obstacle representation; see Fig. 2.

3 Regular Outside-Obstacle Representations

In this section, we show that some graph classes admit regular outside-obstacle representations. A *cactus* is a connected graph where every edge is contained in at most *one* simple cycle. An *outerpath* is an outerplanar graph that admits a drawing whose weak dual is a path. The constructions for the following result are rather simple; see Figs. 4–6.

▶ **Theorem 5.** *The following graphs have reducible regular outside-obstacle representations:* ***1.*** *every cactus;* ***2.*** *every grid;* ***3.*** *every outerpath.*

Proof sketch. We sketch our algorithms. For correctness and reducibility, see [6].

1. Let G be a cactus, and let T be the *block-cut tree* of G, which has a vertex for each *block* (i.e., a biconnected component) and for each cut vertex. There is an edge in the block-cut tree for each pair of a block and a cut vertex that belongs to it. We root T in an arbitrary block vertex. We construct a drawing of G on a circle, starting with the root block and then inserting the other blocks in the order of a BFS traversal of T; see Fig. 4. We insert the vertices of a block B as an interval between the cut vertex that connects B to its parent in T and its clockwise successor in the circular order. The resulting drawing has the consecutive-neighbors property. Hence, by Lemma 2, it is an outside-obstacle representation.

2. Given the graph $P_k \times P_\ell$ of a square $k \times \ell$ grid, we order the vertices of each copy of the path P_k in a zig-zag mannar as shown in Fig. 5. We place the copies one after the other around the circle such that the vertices of each copy form an interval.

3. Let G be an outerpath with n vertices. Since our representation will be reducible, we can assume that G is a *maximal outerpath*, i.e., for any vertex pair $\{u, v\}$, $G + uv$ is not outerplanar. Let $\langle v_1, v_2, \dots, v_n \rangle$ be a *stacking order* of G, that is, for each $i \in \{3, \dots, n\}$, the graph $G_i = G[v_1, v_2, \dots, v_i]$ is a maximal outerpath. Vertex v_j $(3 < j < n)$ is incident to an inner edge $v_i v_j$. We place v_j cyclically next to v_i, avoiding the (empty) arc of the circle that corresponds to the previous inner edge; see Fig. 6. ◀

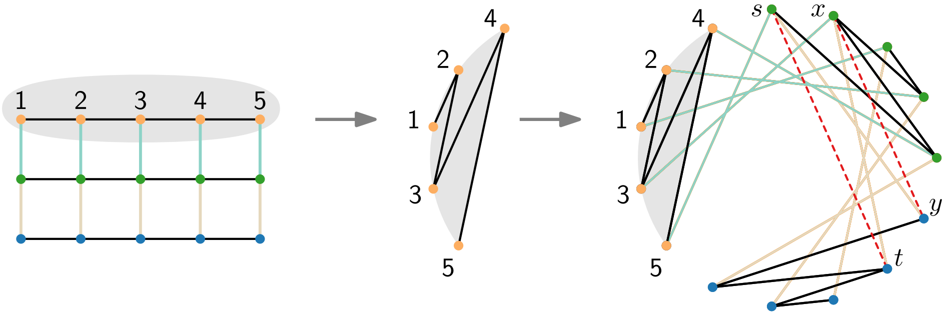


Figure 5 Constructing a reducible regular outside-obstacle representation of the grid $P_5 \square P_3$.

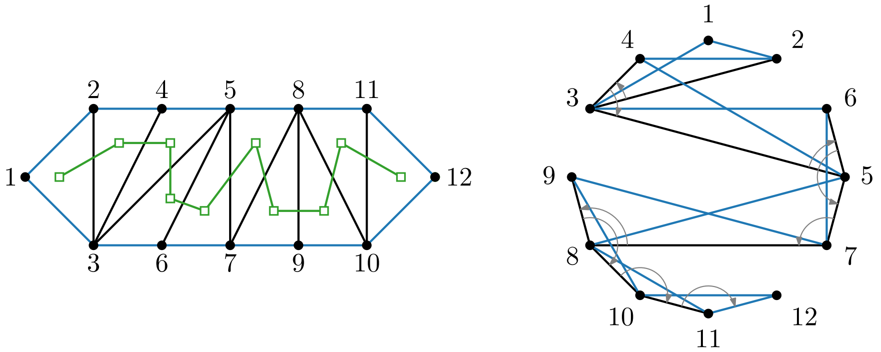


Figure 6 A maximal outerpath and its reducible regular outside-obstacle representation with inner edges (black), outer edges (blue), weak dual (green). Vertices are numbered in stacking order.

Our representations for cacti and outerpaths depend only on the vertex order rather than the exact positions. Hence, for such graphs every cocircular point set is *universal*, i.e., every set of n points on a circle can be used for the vertices of an outside-obstacle representation.

Every graph with up to six vertices – except for the graph in Fig. 2 – admits a regular outside-obstacle representation [6]. The 8-vertex outerplanar graph in Fig. 7, however, does not admit any regular outside-obstacle representation [6].

4 Outside-Obstacle Representations for Partial 2-Trees

The graph class of 2-trees is recursively defined as follows: K_2 is a 2-tree. A graph obtained from a 2-tree G by adding a new vertex x with exactly two neighbors u, v that are adjacent

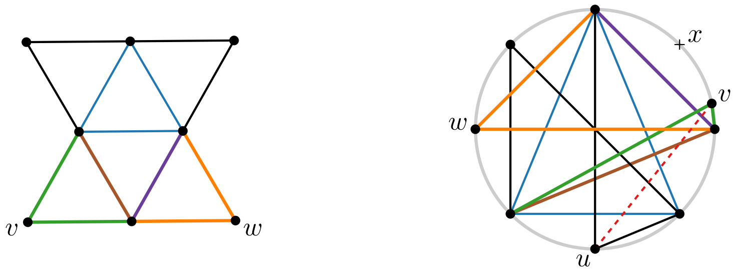


Figure 7 An outerplanar graph G' and a circular outside-obstacle representation of G'. The dashed red non-edge uv will stop intersecting the outer face of G' if we move v towards the point x.

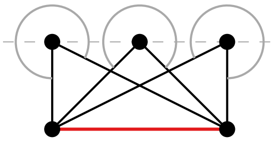


Figure 8 Step (1) in the proof of Theorem 6.

in G is a 2-tree. We say that x is *stacked* on the edge uv. The edges xu and xv are called the *parent edges* of x. For the full proof of the following theorem, see [6].

▶ **Theorem 6.** *Every 2-tree admits a reducible outside-obstacle representation with all vertices on the outer face.*

Proof sketch. Every 2-tree T can be constructed through the following iterative procedure: (1) We start with one edge, called the *base* edge and mark its vertices as *inactive*. We stack any number of vertices onto the base edge and mark them as *active*. During the entire procedure, every present vertex is marked either as active or inactive. Moreover, once a vertex is inactive, it remains inactive for the remainder of the construction. (2) We pick one active vertex v and stack any number of vertices onto each of its two parent edges. All the new vertices are marked as active and v is marked as inactive. (3) If there are active vertices remaining, repeat step (2). We construct a drawing of T by geometrically implementing this iterative procedure, so that after every step of the algorithm the present part of the graph is realized as a straight-line drawing satisfying the following invariants:

(i) Each vertex v not incident to the base edge is associated with an open circular arc C_v that lies completely in the outer face and whose endpoints belong to the two parent edges of v. Moreover, v is located at the center of C_v and the parent edges of v are below v.

(ii) Each non-edge passes through the circular arc of at least one of its incident vertices.

(iii) For each active vertex v, the region R_v enclosed by C_v and the two parent edges of v is *empty*, meaning that R_v is not intersected by any edges, vertices, or circular arcs.

(iv) Every vertex is incident to the outer face.

It is easy to see that once the procedure terminates with a drawing that satisfies invariants (i)–(iv), we have indeed obtained the desired representation (in particular, the combination of invariants (i) and (ii) implies that each non-edge passes through the outer face).

Construction. To carry out step (1), we draw the base edge horizontally and place the stacked vertices on a common horizontal line above the base edge, see Fig. 8. Circular arcs that satisfy the invariants are now easy to define. Suppose we have obtained a drawing Γ of the graph obtained after step (1) and some number of iterations of step (2) such that Γ is equipped with a set of circular arcs satisfying the invariants (i)–(iv). We describe how to carry out another iteration of step (2) while maintaining the invariants. Let v be an active vertex. By invariant (i), both parent edges of v are below v. Let e_ℓ and e_r be the left and right parent edge, respectively. Let $\ell_1, \ell_2, \dots, \ell_i$ and $r_1, r_2, \dots, r_j$ be the vertices stacked onto e_ℓ and e_r, respectively. We refer to $\ell_1, \ell_2, \dots, \ell_i$ and $r_1, r_2, \dots, r_j$ as the *new* vertices; the vertices of Γ are called *old*. We place all the new vertices on a common horizontal line h that intersects R_v above v, see Fig. 9. The vertices $\ell_1, \ell_2, \dots, \ell_i$ are placed inside R_v, to the right of the line $\overline{e_\ell}$ extending e_ℓ. Symmetrically, $r_1, r_2, \dots, r_j$ are placed inside R_v, to the left of the line $\overline{e_r}$ extending e_r. We place $\ell_1, \ell_2, \dots, \ell_i$ close enough to e_ℓ and $r_1, r_2, \dots, r_j$

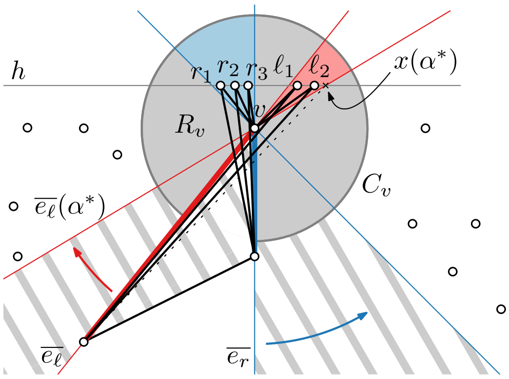


Figure 9 Step (2) in the proof of Theorem 6. The shaded areas do not contain any vertices.

close enough to e_r such that the following properties are satisfied: (a) None of the parent edges of the new vertices intersect C_v. (b) For each new vertex, the unbounded open cone obtained by extending its parent edges to the bottom does not contain any vertices.

Each of the old vertices retains its circular arc from Γ. By invariants (i) and (iii) for Γ, it is easy to define circular arcs for the new vertices that satisfy invariant (i). Using invariants (i)–(iv) for Γ and properties (a) and (b), it can be shown that all invariants are satisfied. ◀

5 Open Problems

(1) What is the complexity of deciding whether a given graph admits an outside-obstacle representation? (2) Does every graph that admits a circular vertex order satisfying the gap condition admit a convex outside-obstacle representation? (3) Does every graph that admits a *convex* outside-obstacle representation also admit a *circular* outside-obstacle representation? (4) Does every outerplanar graph admit a (reducible) convex outside-obstacle representation? (5) Which other classes of graphs admit regular or circular outside-obstacle representations?

References

1 Hannah Alpert, Christina Koch, and Joshua D. Laison. Obstacle numbers of graphs. *Discrete Comput. Geom.*, 44(1):223–244, 2010. doi:10.1007/s00454-009-9233-8.

2 Martin Balko, Josef Cibulka, and Pavel Valtr. Drawing graphs using a small number of obstacles. *Discrete Comput. Geom.*, 59(1):143–164, 2018. doi:10.1007/s00454-017-9919-2.

3 Leah Wrenn Berman, Glenn G. Chappell, Jill R. Faudree, John Gimbel, Chris Hartman, and Gordon I. Williams. Graphs with obstacle number greater than one. *J. Graph Algorithms Appl.*, 21(6):1107–1119, 2017. doi:10.7155/jgaa.00452.

4 Steven Chaplick, Fabian Lipp, Ji-won Park, and Alexander Wolff. Obstructing visibilities with one obstacle. In Yifan Hu and Martin Nöllenburg, editors, *Proc. 24th Int. Symp. Graph Drawing & Network Vis. (GD)*, volume 9801 of *Lect. Notes Comput. Sci.*, pages 295–308. Springer-Verlag, 2016. URL: http://arxiv.org/abs/1607.00278, doi:10.1007/978-3-319-50106-2_23.

5 Vida Dujmović and Pat Morin. On obstacle numbers. *Electr. J. Combin.*, 22(3):paper #P3.1, 7 pages, 2015. See also arxiv.org/1308.4321. URL: http://www.combinatorics.org/ojs/index.php/eljc/article/view/v22i3p1.

6 Oksana Firman, Philipp Kindermann, Jonathan Klawitter, Boris Klemz, Felix Klesen, and Alexander Wolff. Outside-obstacle representations with all vertices on the outer face. Arxiv report, 2022. URL: http://arxiv.org/abs/2202.13015.

7 Christian Goldschmied. 1-Hindernis-Sichtbarkeitsgraphen von kubischen Graphen. Bachelor's thesis, Institut für Informatik, Universität Würzburg, 2021. URL: http://www1.pub.informatik.uni-wuerzburg.de/pub/theses/2021-goldschmied-bachelor.pdf.

8 Padmini Mukkamala, János Pach, and Dömötör Pálvölgyi. Lower bounds on the obstacle number of graphs. *Electr. J. Combin.*, 19(2):paper #P32, 8 pages, 2012. URL: http://www.combinatorics.org/ojs/index.php/eljc/article/view/v19i2p32.

9 Padmini Mukkamala, János Pach, and Deniz Sarıöz. Graphs with large obstacle numbers. In Dimitrios M. Thilikos, editor, *Proc. Conf. Graph-Theoretic Concepts Comput. Sci. (WG)*, volume 6410 of *Lect. Notes Comput. Sci.*, pages 292–303. Springer-Verlag, 2010. doi:10.1007/978-3-642-16926-7_27.

10 János Pach and Deniz Sarıöz. On the structure of graphs with low obstacle number. *Graphs & Combin.*, 27(3):465–473, 2011. doi:10.1007/s00373-011-1027-0.

Linear Time Point Location in Delaunay Simplex Enumeration over all Contiguous Subsequences

Felix Weitbrecht

Universität Stuttgart, Germany
weitbrecht@fmi.uni-stuttgart.de

Abstract

Given a spatio-temporal point set $P = \{p_1, p_2, \dots, p_n\} \subset \mathbb{R}^d$ with timestamps $t_1 < t_2 < \dots < t_n$, we want to compute T, the set of all distinct Delaunay d-simplices over all Delaunay triangulations of contiguous subsequences $\{p_i, p_{i+1}, \dots, p_j\}$ of P. We extend the output-sensitive approach of [5] into arbitrary dimensions, improve its runtime to $\mathcal{O}(|T|)$, and introduce various other improvements.

1 Introduction

The Delaunay triangulation and many of its subcomplexes, such as α-shapes [4], are valuable tools to study point set data. If points are associated with timestamps $t_1 < t_2 < \dots < t_n$, considering points with contiguous subsequences of timestamps $(t_i, t_{i+1}, \dots, t_j)$ can provide additional insight [2]. For example, one can visualize the shape of storm events within a certain time window [3], or reconstruct animated meshes from a series of time-deforming point clouds [1, 6]. For this we need not only the Delaunay d-simplices of the full point set, but also those of all time windows within the point set. We call the set of these d-simplices T and present an algorithm to compute T in optimal time $\mathcal{O}(|T|)$.

This result might be surprising since we implicitly compute the Delaunay triangulation in linear time when $|T|$ is $\mathcal{O}(n)$, but this can only happen in special cases. Intuitively, $|T|$ can only be $\mathcal{O}(n)$ if the spatial order and temporal order of the points are strongly correlated.

We extend the *hole triangulation* framework of [5], which computes T, into arbitrary dimensions. For this purpose we maintain for every hole triangulation the star of the point whose removal is represented by that hole triangulation. Together with a series of pointers, this allows accomplishing faster point location, resulting in an overall output-sensitive linear runtime of $\mathcal{O}(|T|)$. A sample implementation of our algorithm is available on GitHub [7].

We go over notation and prior work in Section 2, and we present new work in Section 3.

2 Preliminaries

We are given a point set $P = \{p_1, p_2, \dots, p_n\} \subset \mathbb{R}^d$ in general position with timestamps $t_1 < t_2 < \dots < t_n$. We consider the d-dimensional Delaunay triangulation $DT(P)$, i.e. that (unique) subdivision of the convex hull of P which consists only of d-simplices whose open circumhyperspheres contain no points from P in their interior. We will call the set $\{p_i, p_{i+1}, \dots, p_j\}$ $P_{i,j}$, and we will say $T_{i,j}$ for its Delaunay triangulation. The dimension d is arbitrary but fixed, so we will use the term *triangulation* to refer to the d-dimensional Delaunay triangulation. We refer to open circumhyperspheres as *circumspheres*, to Delaunay d-simplices as *simplices*, and to $(d-1)$-simplices as *faces*. We denote by T the set of all Delaunay d-simplices occurring over all contiguous subsequences $P_{i,j}$. The set of simplices incident to some point p_i is called the *star* of p_i. The faces of simplices in the star which are not incident to p_i are called the *link* of p_i.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

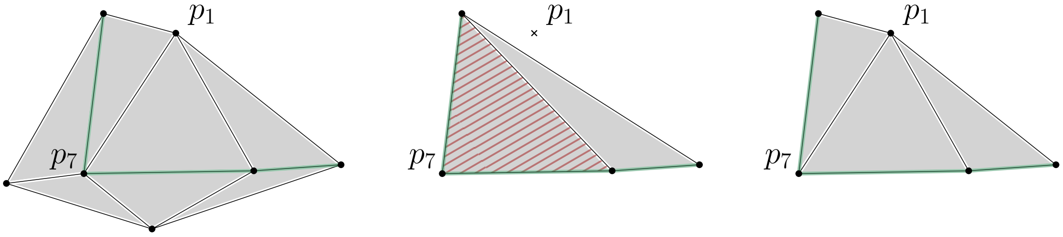


Figure 1 The data structures in our algorithm. The link of p_1 in $T_{1,7}$ is highlighted in green. The link faces are shared between the full triangulation $T_{1,7}$, the hole triangulation $H_{2,7}$, and the star $star_{1,7}$. **Left:** $T_{1,7}$, the incremental construction of $DT(P)$ after inserting p_7. **Middle:** The hole triangulation $H_{2,7}$, which contains the simplices of $T_{2,7}$ with p_1 in their circumspheres. The position of p_1 is shown only for reference. The contribution set $C_{2,7}$ is the triangle highlighted in red. **Right:** The corresponding star $star_{1,7}$, containing the simplices incident to p_1 in $T_{1,7}$.

The hole triangulation framework was introduced in [5] compute T without asymptotic overhead for structural changes. The idea is to execute an incremental construction (IC) for $T_{1,n}$, one for $T_{2,n}$, $\dots$, and one for $T_{n-d+1,n}$. This process encounters the Delaunay triangulation of every time window as an intermediate state of one of these ICs, so it finds every Delaunay simplex of T. To avoid the overhead of computing simplices multiple times in multiple triangulations of similar time windows, all ICs but the one for $T_{1,n}$ use the hole triangulation data structure: $H_{i,j}$ maintains the simplices of $T_{i,j}$ which contain p_{i-1} in their circumsphere, see Figure 1, left and middle. So $H_{i,j}$ represents the difference from $T_{i-1,j}$ to $T_{i,j}$. Arrange these data structures into a matrix for visual aid:

$$\begin{array}{ccccccccc}
T_{1,d} & T_{1,d+1} & \dots & \dots & \dots & T_{1,j} & \dots & \dots & T_{1,n} \\
 & H_{2,d+1} & \dots & \dots & \dots & H_{2,j} & \dots & \dots & H_{2,n} \\
 & & & & & \vdots & & & \\
 & & & H_{i,i+d-1} & \dots & H_{i,j} & \dots & \dots & H_{i,n} \\
 & & & & & \vdots & & & \\
 & & & & & & & H_{n-d,n-1} & H_{n-d,n} \\
 & & & & & & & & H_{n-d+1,n}
\end{array}$$

A hole triangulation $H_{i,j-1}$ is only changed by the insertion of p_j if p_j is adjacent to p_{i-1} in $T_{i-1,j}$. So updates to hole triangulations can be triggered by the discovery of new edges, which avoids $\Theta(n^2)$ checks for whether a hole triangulation needs to be updated. Conceptually, the matrix is computed left to right, going top to bottom within each column. The *contribution set* $C_{i,j}$ are the simplices which appear first in $T_{i,j}$ by this order. So the simplices created by the insertion of p_j into $T_{1,j-1}$ or $H_{i,j-1}$ are $C_{1,j}$ or $C_{i,j}$, respectively. Since structural changes correspond exactly to contribution sets and contribution sets exactly partition T, structural changes have $\mathcal{O}(|T|)$ overall cost. However, point location still causes logarithmic overhead per simplex. We improve that in Section 3.

3 Efficient Point Location

An important difference to the original hole triangulation framework [5] is how hole triangulation updates are triggered. Originally, the hole triangulation update from $H_{i+1,j-1}$ to $H_{i+1,j}$ could be executed immediately upon finding finding the edge $\{p_i, p_j\}$. In Section 3.1 we will see that now all simplices incident to both p_i and p_j in $T_{i,j}$ are necessary for this update because we maintain the star of p_i as an auxiliary data structure for $H_{i+1,j}$.

So, upon finding the edge $\{p_i, p_j\}$, we wait for all new simplices of the star to be found and then we update the star $star_{i,j-1}$ to $star_{i,j}$. We update the hole triangulation $H_{i+1,j-1}$ to $H_{i+1,j}$ immediately after this star update. This way, the star update provides all information about how the link changes for the hole triangulation update.

The new star simplices can come from more than one contribution set, so the update from $H_{i,j-1}$ to $H_{i,j}$ depends on one or more contribution sets $C_{i',j}$ with $i' < i$. These dependencies impose a partial order on the order in which hole triangulations are updated within each matrix column, implicitly forming a dependency graph. This graph is a directed acyclic graph (DAG) since all its edges go towards higher indices. All updates (transitively) depend on the insertion of p_j into $T_{1,j-1}$ and there are no cyclic dependencies, so the algorithm can proceed as follows. We compute the incremental construction step by step $(T_{1,d}, T_{1,d+1} \ldots, T_{1,n})$. After every step, a cascade of star and hole triangulation updates is triggered in which each update is computed once all its predecessors in the DAG have been computed. Note that, depending on the structure of the DAG, there may actually be multiple valid computation orders within a column.

3.1 Stars for Point Location in Hole Triangulations

Before inserting p_j into $H_{i+1,j-1}$, we need to locate p_j in $H_{i+1,j-1}$. For this purpose we maintain the closed star of point p_i w.r.t. $T_{i,j}$ as $star_{i,j}$. It is that subcomplex of $T_{i,j}$ which contains the simplices which are incident to p_i, and their faces. Similar to how $H_{i+1,j}$ is the difference from $T_{i,j}$ to $T_{i+1,j}$, $star_{i,j}$ is the difference from $T_{i+1,j}$ to $T_{i,j}$. The link of p_i (w.r.t. $T_{i,j}$) exists in the hole triangulation $H_{i+1,j}$ and in the star $star_{i,j}$, as shown by the green highlights in Figure 1. Since we do not maintain $T_{i,j}$ explicitly for $i > 1$, the simplices incident to p_i in $T_{i,j}$ are distributed across $T_{1,j}$ and $H_{2,j}, H_{3,j}, \ldots, H_{i,j}$. We store a copy of these simplices in $star_{i,j}$ so they can be traversed efficiently. Every copy gets an $original$-pointer to its original instance in the IC or in some hole triangulation.

Observe that every Delaunay face can be uniquely associated with the first $T_{i,j}$ in which it is a Delaunay face. For example, the link faces in the hole triangulation $H_{2,7}$ in Figure 1, middle, first exist in $T_{1,7}$, and the interior faces first exist in $T_{2,7}$. We can follow how faces are propagated through the incremental construction, hole triangulations and stars:

▶ **Lemma 3.1.** *Every instance of a face created from a d-subset of points $F \subseteq P$ in the algorithm is either the first time a face is created from F, or a copy of an existing instance of f. This allows matching up simplices from different data structures which share a face.*

Proof. Faces created in the incremental construction are always the first instance. Faces in stars are copied from the IC and hole triangulations. Link faces in hole triangulations are copied from the corresponding stars. Every other face in a hole triangulation $H_{i,j}$ triangulates the interior of the link of p_{i-1}. Such a face f cannot exist in earlier triangulations because any hypersphere passing through all points of f must contain p_{i-1}, or the point which forms the coface of f on the side of f opposite p_{i-1} in $H_{i,j}$, or both. So f is the first instance of that face.

We create an attribute object O with every newly created face. Every time we create a copy of that face, we store a reference to O with the new copy. Any two simplices created in different data structures (for example the IC and a hole triangulation) that have a face defined by the same d points can then be matched up by registering with that face's O. ◀

When inserting point p_j into $star_{i,j-1}$, we wait for all new simplices (they originate in $T_{1,j}$ and $H_{2,j}, H_{3,j}, \ldots, H_{i,j}$) to be available before executing the update. We can recognize that all new simplices have been found once both cofaces of all new faces incident to both p_i and p_j are available. We then use Lemma 3.1 to sew copies of these new simplices into the existing star. So stars can be updated in overall $\mathcal{O}(|T|)$ time simply by copying new simplices and setting some pointers.

After the star $star_{i,j-1}$ is updated to $star_{i,j}$ with the insertion of p_j, we update its corresponding hole triangulation $H_{i+1,j-1}$ to $H_{i+1,j}$. The star update reveals exactly how the link changes. By exploring the old hole triangulation from where the link changes, we can find all simplices of $H_{i+1,j-1}$ which need to be destroyed in overall $\mathcal{O}(|T|)$ time. We can then compute the new simplices similar to how a regular IC would, and set *link*-pointers between matching instances of link faces in $star_{i,j}$ and $H_{i+1,j}$. To ensure that the hole triangulation stays restricted to the interior of the new link, we must pay special attention when creating new simplices between p_j and *ex-link faces*, i.e. those faces that were part of the link until this update but are no longer part of the link after this update. If p_i and p_j are on the same side of the supporting hyperplane of an ex-link face f, we must not create a simplex between p_j and f because f is now outside the link. We create a simplex between p_j and all other ex-link faces if their existing coface inside the link in $H_{i+1,j-1}$ is not destroyed.

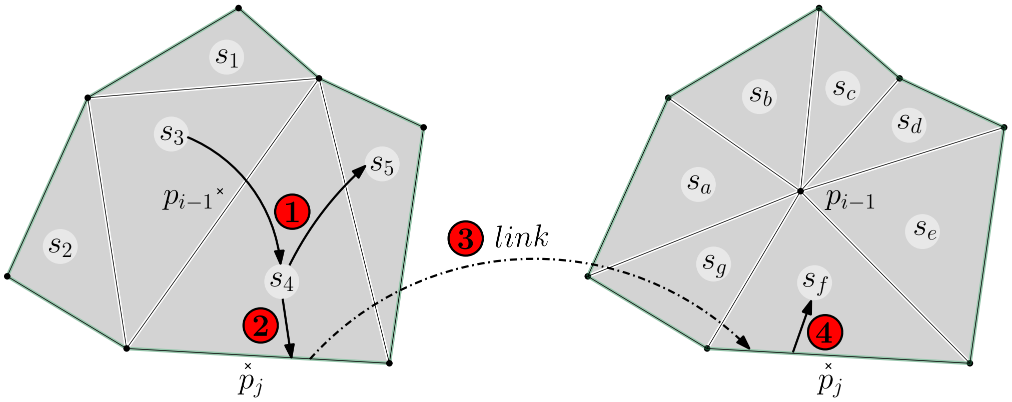


Figure 2 Point location process for the IC finding a simplex destroyed by p_j in $star_{i-1,j-1}$ (right), based on a destroyed simplex in the corresponding hole triangulation $H_{i,j-1}$ (left). The link is highlighted in green. The positions of p_j, and p_{i-1} in the hole triangulation, are shown only for reference. We proceed in four steps. 1: Given an initial simplex $curr = s_3$ with p_j in its circumsphere, we explore simplices destroyed by p_j in $H_{i,j-1}$; we find s_3, s_4 and s_5. 2: We consider all link faces which have one of these simplices as a coface. 3: For each such link face, we follow its *link*-pointer into the star. 4: We test whether its coface contains p_j in its circumsphere. Here, only the link face of s_4 leads to a destroyed star simplex, s_f.

3.2 Point Location in the Incremental Construction

In Section 3.1 we presented stars to facilitate point location in hole triangulations in overall $\mathcal{O}(|T|)$ time. Now we will explain how to use stars and the pointers introduced in Section 3.1 to also accomplish point location in the IC in overall $\mathcal{O}(|T|)$ time.

Accomplishing point location this fast using only the IC data structure seems unlikely because $|T|$ can be as small as $\mathcal{O}(n)$. Our approach exploits that p_j is inserted not only into $T_{1,j-1}$, but also into some hole triangulations and stars. The order of these insertions follows the DAG underlying the computation order. Before computing the updates in column j, we will trace back a path to the root of the DAG of column j, $T_{1,j-1}$ (i.e. the IC), locating p_j in every data structure we visit. To navigate between these data structures, we can use *original*-pointers to get from star simplices to their original instances in hole triangulations or in the IC, and we can use *link*-pointers to jump from the link of hole triangulations to the link of their corresponding stars.

More specifically, given a simplex s with p_j in its circumsphere in some star, we follow its *original*-pointer to, say, *curr*. If *curr* is part of the IC, we have successfully found a simplex with p_j in its circumsphere in the IC. Otherwise, *curr* is part of some hole triangulation, and we need to find another simplex s with p_j in its circumsphere in the corresponding star. Figure 2 shows how, given a simplex *curr* with p_j in its circumsphere in some hole triangulation, we are able to find a destroyed simplex in the corresponding star. We iterate this process of finding a destroyed simplex s in the star and following its *original*-pointer until that pointer goes into the IC.

By following only *original*-pointers of simplices with p_j in their circumsphere, we ensure that we only visit data structures p_j will later be inserted into. We only explore simplices that will be destroyed by p_j, so the total cost of these explorations is bounded by the structural changes of inserting p_j, i.e. $\mathcal{O}(|T|)$ overall.

3.3 Putting it all together

In contrast to [5], our approach computes the IC in its natural order, so our algorithm can actually be executed in an online manner if points arrive in their temporal order.

▶ **Theorem 3.2.** *There exists an algorithm to compute the set T of all Delaunay d-simplices over all contiguous subsequences of a set of timestamped points in $\mathbb{R}^d$ in output-sensitive linear time $\mathcal{O}(|T|)$ for arbitrary fixed d. This algorithm can be executed in an online manner.*

As a result of Lemma 3.1, we can also construct a simplicial complex representing any $T_{i,j}$ in linear time given the Delaunay simplices of the time window $[i, j]$. Any rectangle stabbing data structure can identify these simplices using the lifetime attributes computed for every simplex by the hole triangulation framework, as explained in Section 3.6 of [5].

3.4 Experimental Results

We benchmarked a prototypical Java implementation [7] of our enumeration algorithm in one to six dimensions on an Intel® Xeon® E5-2650 v4 CPU, see Figure 3. The results

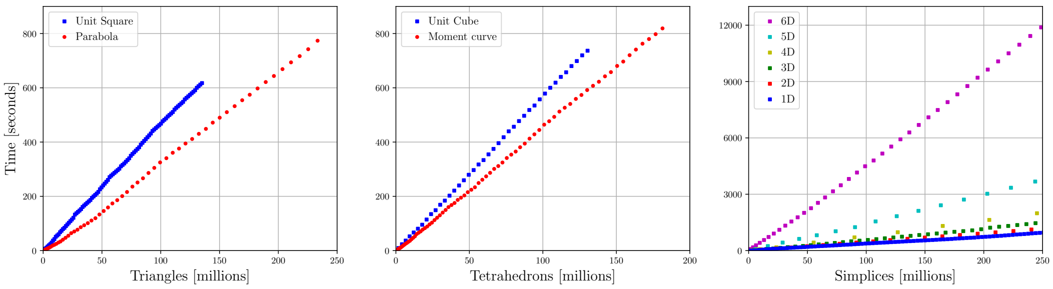


Figure 3 Benchmarking time per simplex creation. **Left:** $1 \cdot 10^4, 2 \cdot 10^4, \ldots, 100 \cdot 10^4$ 2D points sampled u.a.r. from the unit square, and $1 \cdot 10^3, 2 \cdot 10^3, \ldots, 60 \cdot 10^3$ 2D points drawn from the normal parabola $y = x^2$. **Middle:** $5 \cdot 10^3, 10 \cdot 10^3, \ldots, 200 \cdot 10^3$ 3D points sampled u.a.r. from the unit cube, and $1 \cdot 10^2, 2 \cdot 10^2, \ldots, 80 \cdot 10^2$ 3D points sampled from the moment curve $x(t) = (t, t^2, t^3)^T$. **Right:** Points drawn u.a.r. from the unit hypercube in one to six dimensions, enough to get 250 million simplices each.

show that our algorithm is practical, and even our simple implementation creates well over 150,000 tetrahedrons per second. In the left and middle diagram, we draw points from the unit square/cube and the moment curve, resulting in $|T| \in \mathcal{O}(n \log n)$ and $|T| \in \Omega(n^2)$, respectively. Time per simplex creation is indeed constant. On the right, we can see the runtime grow exponentially with increasing dimension, but it is still linear for fixed d.

4 Outlook

We have presented an algorithm to compute all Delaunay simplices over all contiguous subsequences of a point set in arbitrary dimension in output-sensitive linear time. We showed how to efficiently construct Delaunay triangulations of arbitrary time windows from the algorithm output. Experimental results showed practicality and indeed linear runtime.

Our enumeration algorithm can serve as a basis for constructing data structures allowing efficient retrieval of time-windowed Delaunay triangulations or their subcomplexes. We look forward to seeing its uses for fast interactive visualization of spatio-temporal data. It would also be interesting to see for which other subcomplexes of the Delaunay triangulation all elements over all contiguous subsequences can be computed in an equally efficient manner.

References

1 Ehsan Aganj, Jean-Philippe Pons, Florent Ségonne, and Renaud Keriven. Spatio-temporal shape from silhouette using four-dimensional Delaunay meshing. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE, 2007. doi:10.1109/ICCV.2007.4409016.

2 Michael J. Bannister, William E. Devanny, Michael T. Goodrich, Joseph A. Simons, and Lowell Trott. Windows into geometric events: Data structures for time-windowed querying of temporal point sets. In *Proceedings of the 26th Canadian Conference on Computational Geometry (CCCG)*, pages 11–19, 2014. URL: https://www.cccg.ca/proceedings/2014/papers/paper02.pdf.

3 Annika Bonerath, Benjamin Niedermann, and Jan-Henrik Haunert. Retrieving α-shapes and schematic polygonal approximations for sets of points within queried temporal ranges. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 249–258. ACM, 2019. doi:10.1145/3347146.3359087.

4 Herbert Edelsbrunner, David G. Kirkpatrick, and Raimund Seidel. On the shape of a set of points in the plane. *IEEE Transactions on Information Theory*, 29(4):551–559, 1983. doi:10.1109/TIT.1983.1056714.

5 Stefan Funke and Felix Weitbrecht. Efficiently computing all Delaunay triangles occurring over all contiguous subsequences. In *31st International Symposium on Algorithms and Computation (ISAAC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ISAAC.2020.28.

6 Jochen Süßmuth, Marco Winter, and Günther Greiner. Reconstructing animated meshes from time-varying point clouds. In *Computer Graphics Forum*, volume 27, pages 1469–1476. Wiley Online Library, 2008. doi:10.1111/j.1467-8659.2008.01287.x.

7 Felix Weitbrecht. DelaunayEnumerator, a GitHub repository. https://github.com/felixweitbrecht/DelaunayEnumerator, 2021.

Approximating Multiplicatively Weighted Voronoi Diagrams: Efficient Construction with Linear Size

Joachim Gudmundsson[1], Martin P. Seybold[2], and Sampson Wong[3]

1 **University of Sydney, Australia**
joachim.gudmundsson@gmail.com

2 **University of Sydney, Australia**
mpseybold@gmail.com

3 **University of Sydney, Australia**
swon7907@sydney.edu.au

Abstract

Given a set of n sites from $\mathbb{R}^d$, each having some positive weight factor, the Multiplicative Weighted Voronoi Diagram is a subdivision of space that associates each cell to the site whose weighted Euclidean distance is minimal for its points.

We give an approximation algorithm that outputs a subdivision such that the weighted distance of a point with respect to the associated site is at most $(1+\varepsilon)$ times the minimum weighted distance, for any fixed parameter $\varepsilon \in (0,1)$. The diagram size is $\mathcal{O}(n \log(1/\varepsilon)/\varepsilon^{d-1})$ and the construction time is within a factor $\mathcal{O}(1/\varepsilon^{(d+1)d} + \log(n)/\varepsilon^{d+2})$ of the output size. As a by-product, we obtain $\mathcal{O}(\log(n/\varepsilon))$ point-location query time in the subdivision.

The key ingredients of the proposed method are the study of convex regions that we call *cores*, an adaptive refinement algorithm to obtain small output size, and a combination of Semi-Separated Pair Decomposition (SSPD) and conic space partitions to obtain efficient runtime.

Related Version https://arxiv.org/abs/2112.12350

1 Introduction

Let $\{s_1, \dots, s_n\}$ be a set of n sites in Euclidean space $\mathbb{R}^d$, where d is a constant. Let $w_i > 0$ be the weight of s_i. The weighted Euclidean distance from any point p to s_i is given by the formula $d_i(p) = \|p - s_i\|/w_i$. For a point $q \in \mathbb{R}^d$, a site s_i that minimizes $d_i(q)$ is called a *weighted nearest-neighbor* of q. A Multiplicative Weighted Voronoi Diagram (MWVD) is a subdivision of space that associates each cell to the site that is weighted nearest-neighbor for all points in that cell. The *Voronoi region* of a site is the union of all cells that are associated to the site. See also [2, Chapter 7.4.2].

We give an ε-approximation algorithm that computes a subdivision of $\mathbb{R}^d$ into cells. Each cell is associated with one site that is a weighted nearest-neighbor for all points in it, up to a factor of $(1+\varepsilon)$, where $\varepsilon \in (0, 1/2^6]$. Each cell is a d-cube or the set difference of two d-cubes. As a by-product of the construction, we get a d-dimensional compressed QuadTree that can answer approximate nearest-neighbor queries using point-location in $\mathcal{O}(\log(n/\varepsilon))$ time[1].

Our algorithm assembles the diagram based on *cores* – see Section 2. To guarantee small output size, we propose an Adaptive Refinement algorithm that ε-approximates each core with a set of d-cubes. Its runtime is proportional to the number of output cubes, multiplied by a factor that is polynomial in the number of balls that define the core (see Section 3). Our final bound improves on the state-of-the-art for ε-AVD size (see Table 1) and is within a $\mathcal{O}(\log 1/\varepsilon)$-factor of the best known $\Omega(1/\varepsilon^{d-1})$ lower bound [1].

[1] We assume d is a small constant and the $\mathcal{O}$-notation hides constant factors that are exponential in d.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

In Section 4, we show that it is possible to ε-approximate each core with $\mathcal{O}(1/\varepsilon)^{d+1}$ balls. Moreover, we give an efficient algorithm that computes the ε-approximations of all cores in $\mathcal{O}(n\log(n)/\varepsilon^{2d+1})$ time. The method uses a Semi-Separated Pair Decomposition of the input sites and conic subdivisions of $\mathbb{R}^d$.

Diagram	Technique	Size	Runtime (factor over size)
ε-AVD	Triangle ineq., PLEB [4]	$\mathcal{O}\left(n\frac{\log n}{\varepsilon^d}\log\frac{n}{\varepsilon}\right)$	$\times\mathcal{O}\left(\log\frac{n}{\varepsilon}\right)$
ε-AVD	Triangle ineq., ε-PLSB [7]	$\mathcal{O}\left(n\frac{1}{\varepsilon^d}\right)$	$\times\mathcal{O}\left(\log^2\frac{n}{\varepsilon}\right)$
ε-AVD	Triangle ineq., 8-WSPD [1]	$\mathcal{O}\left(n\frac{1}{\varepsilon^d}\right)$	$\times\mathcal{O}\left(\log\frac{n}{\varepsilon}\right)$
ε-AMWVD	Clustering, Sketches [6]	$\mathcal{O}\left(n\left(\frac{\log^{d+2}(n)}{\varepsilon^{2d+2}}+\frac{1}{\varepsilon^{d(d+1)}}\right)\right)$	
ε-AMWVD	Cores, Adaptive Refinement, $(1+\frac{32}{\varepsilon})$-SSPD, Cones	$\mathcal{O}\left(n\frac{\log 1/\varepsilon}{\varepsilon^{d-1}}\right)$	$\times\mathcal{O}\left(\frac{1}{\varepsilon^{d(d+1)}}+\frac{\log n}{\varepsilon^{d+2}}\frac{1}{\log 1/\varepsilon}\right)$

Table 1 Overview of constructions of ε-AVDs and the proposed method for ε-AMWVDs. All methods, including the proposed, have $\mathcal{O}(\log\frac{n}{\varepsilon})$ point-location query time. Note that ε-AMWVDs are more general than the unweighted ε-AVDs, that have size $\Omega(n/\varepsilon^{d-1})$. The runtime for the ε-AMWVD in [6] is $\mathcal{O}\left(n\log^{2d+3}(n)/\varepsilon^{2d+2}+n/\varepsilon^{d(d+1)}\right)$.

2 Preliminaries: Voronoi Maps, Apollonian Balls, and the Core

Mapping $\ell:\mathbb{R}^d\to\{1,\ldots,n\}$ is called a Voronoi Map, if $\|p-s_{\ell(p)}\|/w_{\ell(p)}\le\min_i\|p-s_i\|/w_i$, for all points $p\in\mathbb{R}^d$. The site with index $\ell(p)$ is called a weighted nearest-neighbor of point p. We call a subdivision of $\mathbb{R}^d$ a MWVD if every cell in the subdivision is associated to one of the input sites, and mapping the points in a cell to the associated site is a Voronoi Map.

For the MWVD, the cell boundaries occur where the weighted distance to two or more sites are equal. For $d=2$, cell boundaries occur along an Apollonian circle. For general d, we define the Apollonian sphere between s_i and s_j to be $\{p\in\mathbb{R}^d : ||p-s_i||/w_i=||p-s_j||/w_j\}$.

A subdivision of $\mathbb{R}^d$ is an ε-AMWVD if every cell in the subdivision is associated to one of the input sites, and $\|p-s_{\ell(p)}\|/w_{\ell(p)}\le(1+\varepsilon)\cdot\min_i\|p-s_i\|/w_i$, where $\ell(p)$ is the index of the input site associated to the region in the subdivision containing the point p.

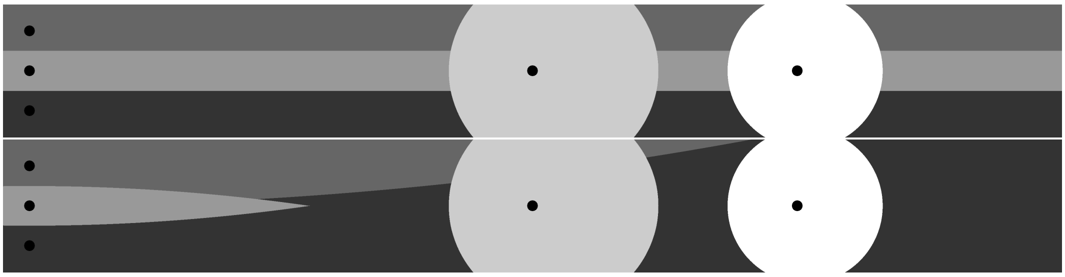


Figure 1 The top shows an example of a MWVD of five sites. The bottom shows an ε_S-AMWVD of the same instance obtained from cores with $\varepsilon_S=0.01$.

Our approach is to use canonical cubes, or the set difference between two canonical cubes, as the cells in our ε-AWMVD. A canonical cube in $\mathbb{R}^d$ has the form $[2^kx_1,2^k(x_1+1)]\times\ldots\times[2^kx_d,2^k(x_d+1)]$ for integers $k,x_1,\ldots,x_d$. Recall that for the exact MWVD, the region boundaries are Apollonian spheres. If $w_j=w_i$, the Apollonian sphere becomes a $(d-1)$-dimensional hyperplane. We introduce a constant $\varepsilon_S\in(0,\varepsilon)$, and we ε_S-approximate the

$(d-1)$-dimensional hyperplane with a sphere, to make it easier to approximate with canonical cubes. Sort the sites by weight, so that $w_1 \leq \ldots \leq w_n$. Then for all $i < j$, define

$$ball(i,j) = ball(s_i, s_j, \gamma_{i,j}) = \left\{p \in \mathbb{R}^d : \|p - s_i\| \leq \gamma_{ij} \cdot \|p - s_j\|\right\} \tag{1}$$

where $\gamma_{ij} = \max(w_j/w_i, 1 + \varepsilon_S)$. Since $\gamma_{ij} \geq 1 + \varepsilon_S$, $ball(i,j)$ is not a hyperplane. Note that the arrangement of $ball(i,j)$ for $i < j$ forms an ε_S-AMWVD. Next, we define the core. We denote the set of balls of site s_i with any other partner site s_j of higher weight by $B_i := \left\{(i,j) \ : \ i < j\right\}$. For a subset $A \subseteq B_i$, define the convex region $core(A) := \cap_{(i,j)\in A} ball(i,j)$.

We define $t^*(s_i, s_j, \gamma_{ij})$ to be the closest distance from s_i to a point on the surface of $ball(i,j)$, and $t^\dagger(s_i, s_j, \gamma_{ij})$ to be the furthest distance from s_i to a point on the surface of $ball(i,j)$. The formulas for each of these values are

$$t^*_{ij} = t^*(s_i, s_j, \gamma_{ij}) = \|s_j - s_i\|/(\gamma_{ij} + 1) \tag{2}$$

$$t^\dagger_{ij} = t^\dagger(s_i, s_j, \gamma_{ij}) = \|s_j - s_i\|/(\gamma_{ij} - 1) \ . \tag{3}$$

3 Basic AMWVD in $\mathbb{R}^d$ using $\binom{n}{2}$ bisectors

For a convex region R, and point s in its interior, we define an ε_A-approximation of (R, s). Intuitively, an ε_A-approximation of (R, s) is a set of canonical cubes whose union covers R, but is not too much larger than R, where all distances are measured from the point s. See Figure 2. Formally, we define a set of interior disjoint canonical cubes L to be an ε_A-approximation of (R, s) if for any point p in $\mathbb{R}^d$, p is in the union of L if and only if either p is in R, or the canonical cube of L containing p intersects the boundary of R and has side length at most ε_A times the distance between p and s.

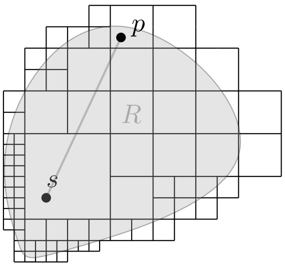


Figure 2 A convex region R, a point in its interior s, and an ε_A-approximation of (R, s).

With this definition of an ε_A-approximation in mind, we now state a recursive, adaptive refinement algorithm for computing the set of canonical cubes L. Our algorithm starts with a canonical cube that completely contains our convex region R. We then recursively split the canonical cube into 2^d cubes, until it satisfies one of the three following halting conditions:

- Halting condition 1: The canonical cube is entirely inside R.
- Halting condition 2: The canonical cube is entirely outside R.
- Halting condition 3: The canonical cube intersects the boundary of R, and has side length at most ε_A times its distance to s.

If the canonical cube halts on condition 2, we discard the cube. Otherwise, if the canonical cube halts on condition 1 or 3, we add the cube to L. This completes the construction.

▶ **Theorem 1.** *Let R be a convex region that is the intersection of k balls in $\mathbb{R}^d$. Let s be a point in the interior of R. Then one can construct a set of canonical cubes L in time $\mathcal{O}(|L| \cdot k^d)$ so that L is an ε_A-approximation of (R, s).*

Next, we apply Theorem 1 to construct a cube-based ε_A-approximation of $(core(B_i), s_i)$. Recall from Section 2 that $core(B_i)$ is the intersection of $ball(i, j)$ for $j > i$.

▶ **Theorem 2.** *One can construct an ε_A-approximation of $(core(B_i), s_i)$ that has total size $\mathcal{O}(\log(1/\varepsilon_S)/\varepsilon_A^{d-1})$. The construction time is $\mathcal{O}(|B_i|^d)$ times the output size.*

Finally, we combine the ε_A-approximations of each of the regions $core(B_i)$ to construct an ε-AMWVD, where $\varepsilon = (1 + \varepsilon_S)(1 + \varepsilon_A) - 1$. For each $1 \leq i < n$, we construct the ε_A-approximate cubes for $(core(B_i), s_i)$ using Theorem 2. Each cube in the ε_A-approximation of $(core(B_i), s_i)$ is given the label i. We collect all cubes for labels $1 \leq i < n$ in this way. For $i = n$, we construct a canonical cube that contains all other canonical cubes for $1 \leq i < n$, and give this canonical cube the label n. We construct a compressed QuadTree from this set of canonical cubes. Sort the canonical cubes by their z-order. Iterate over the sorted list, and remove duplicate cubes by retaining only the cube with the minimum label. Construct a compressed QuadTree from the set of canonical cubes, via Lemma 2.11 in [5]. The compressed QuadTree induces a subdivision of $\mathbb{R}^d$, where each cell in the subdivision is either a canonical cube, or the set difference between two canonical cubes. We label all cells in the compressed QuadTree. Cubes in the sorted list have their initial label, and the root has initial label n. Starting at the root, if a child is unlabeled, or the child has larger label than its parent, then the child replaces its label with its parent's label. We repeat this process for all nodes in the compressed QuadTree, in a top-down fashion. This completes the construction of the AMWVD.

▶ **Theorem 3.** *Given $\varepsilon_S, \varepsilon_A > 0$ and a set of balls B_i for each $i < n$, one can compute an ε-AMVWD, where $\varepsilon = (1+\varepsilon_S)(1+\varepsilon_A) - 1$, with total size $\mathcal{O}\left(n \frac{\log 1/\varepsilon_S}{\varepsilon_A^{d-1}}\right)$. The construction time is $\mathcal{O}\left(\frac{1}{n}\sum_i |B_i|^d + \log \frac{n \log 1/\varepsilon_S}{\varepsilon_A}\right)$ times the output size. The point-location query time in the ε-AMWVD is $\mathcal{O}\left(\log \frac{n \log 1/\varepsilon_S}{\varepsilon_A}\right)$.*

The proofs of Theorems 1, 2 and 3 can be found in the full version of the paper [3].

4 Reducing the number of balls to $\mathcal{O}(1/\varepsilon^{d+1})$

Let α-*ball*(i, j) denote the enlarged ball obtained by setting the effective weight $w_j/\alpha w_i$ in the Apollonian bisector, i.e. $\alpha\text{-}ball(i, j) = ball(s_i, s_j, \gamma_{ij}/\alpha)$. For $\alpha \geq 1$, we define a relation between every two subsets $X, Y \subseteq B_i$ as

$$X \prec_\alpha Y \quad \Longleftrightarrow \quad \forall\ (i,k) \in Y \ :\ core(X) \subseteq \alpha\text{-}ball(i,k)\ ,$$

and say for such a pair that X is an α-cover of Y. Given a subset $X \subseteq B_i$, we call the largest subset $Y \subseteq B_i$ with $X \prec_\alpha Y$ the set of balls that are α-covered by X. Moreover, X is called an α-cover if it covers *all balls* in B_i, i.e. $X \prec_\alpha B_i$, and we have

$$core(B_i) \quad \subseteq \quad core(X) \quad \subseteq \quad \alpha\text{-}core(B_i) := \bigcap_{(i,j) \in B_i} \alpha\text{-}ball(i,j) \quad . \tag{4}$$

The goal of our algorithm is to compute a subset $A_i \subseteq B_i$, so that A_i is an α-cover of B_i, and A_i has constant size. Let parameters β, σ, and ε_C be constants. We show how to choose these constants in the full version of the paper [3].

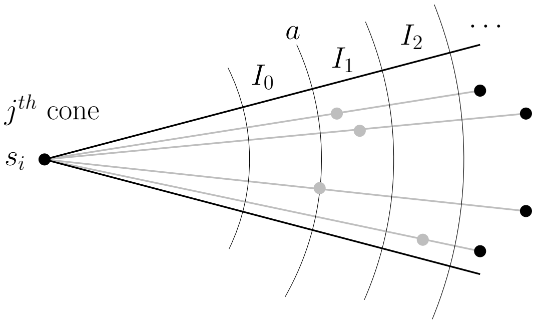


Figure 3 The values t^*_{ij} in $[a,b]$ are partitioned by intervals $I_0,\dots,I_m$ of length $a\varepsilon_C/2$.

Let $\mathcal{P}$ be a σ-Semi Separated Pair Decomposition (SSPD) of the input sites S. For a pair $(L,H) \in \mathcal{P}$, we call L the 'light set' and H the 'heavy set' if s_ℓ is the site with maximum index in L, s_h is the site with the maximum index in H, and $\ell < h$.

A β-cone around s_i is an angular domain of the spherical coordinate system around s_i. In each of the $(d-1)$-dimensions in the spherical coordinate system, the angular domain is partitioned into intervals of at most 2β radians. For each s_i, we assign each β-cone a unique array index j, where $j = \mathcal{O}(1/\beta^{d-1})$. E.g. a rotation of at most β radians suffices to rotate any site in the cone onto the cone's central ray.

Our algorithm maintains the following data structure: for each site $s_i \in S$, and for each β-cone around s_i with array index j, the data structure stores a set of partner sites A_{ij}. Our algorithm populates the data structure in three passes. In our first pass, for each $(L,H) \in \mathcal{P}$, we reduce the size of H to a subset H'. In our second pass, we iterate over $\mathcal{P}$ to initialize each of the sets A_{ij}. Finally, the sets are populated in the third pass.

In our first pass, for each $(L,H) \in \mathcal{P}$, we construct a subset H' of H. If the diameter of H is at most the diameter L, we set $H' := \{\}$. If the diameter of H is larger than the diameter of L, we construct H' as follows. Let $s_\ell \in L$ with ℓ maximal. For the j^{th} cone around s_ℓ, we let the sites of H contained in this cone be $C_{\ell j}$. We use the following function:

```
SCAN-CONE-SITES($i$, $C$, $\varepsilon_C$):
    Let $C' := \emptyset$, $a = min\{t^*_{ij} : s_j \in C\}$, and $b = min\{t^\dagger_{ij} : s_j \in C\}$
    Let $I_k = (x_k, x_{k+1}]$, with length $a\varepsilon_C/2$ and $x_1 = a$, cover $[a,b]$.
    Every interval $I_k$ holds one pointer.
    FOR $s_j \in C$ DO
        Compute the index $k$ with $t^*_{ij} \in I_k$.
        If diameter $(t^*_{ij} + t^\dagger_{ij})$ is smaller than that of $I_k$'s reference,
            then set $I_k$'s pointer on $s_j$.
    FOR interval $I_k$ DO
        Add the kept disc to result set $C'$.
    return $C'$
```

See Figure 3 for an illustration of the intervals I_k. We set, for the j^{th} cone, $C'_{\ell j} =$ SCAN-CONE-SITES$(\ell, C_{\ell j}, \varepsilon_C)$ and let $H' = \cup_j C'_{\ell j}$. This completes the construction of H'.

In our second pass, we initialize each cone of each site in our data structure to store an interval $[a,b]$. We iterate over all pairs $(L,H) \in \mathcal{P}$ and all $s_i \in L \cup H$, and store for j^{th} cone of s_i, variables a and b equal to the minimum values of t^*_{ik} and $t^\dagger_{ik}$ respectively, where the minimum is taken over all sites $s_k \in H' \cup \{s_\ell, s_h\}$ that are in the j^{th} cone of s_i and have $k > i$. This gives us the interval $[a,b]$. After the pass over $\mathcal{P}$ is completed, we iterate over each cone of each site and partition the interval $[a,b]$ into disjoint intervals $I_k = (x_k, x_{k+1}]$

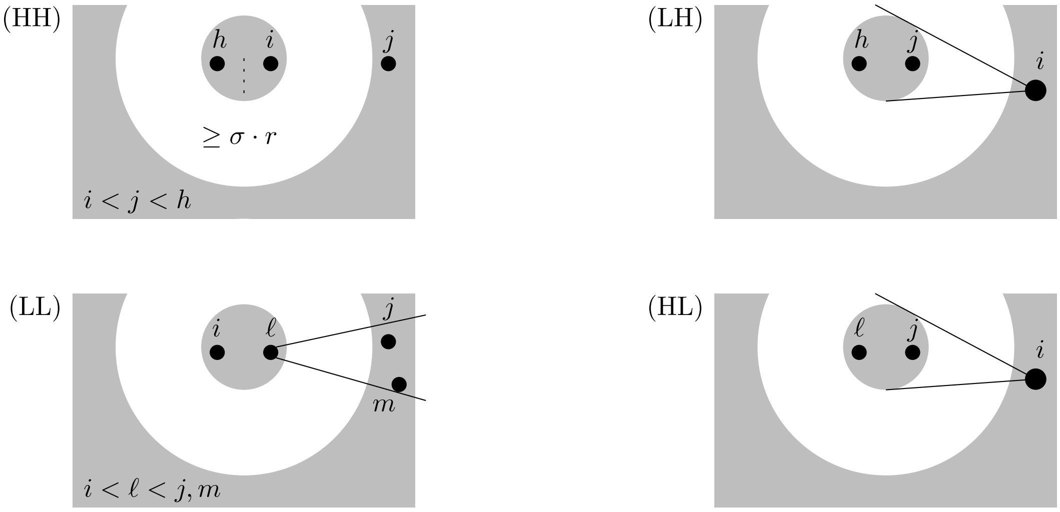


Figure 4 Cases (HH), (LH), (HL), and (LL), for covering $ball(i, j)$.

of length $a\varepsilon_C/2$ that cover $[a, b]$, i.e. $x_{k+1} - x_k = a\varepsilon_C/2$ and $x_1 = a$.

In our third pass, we populate the sets A_{ij} based on the intervals $\{I_k\}$ of the j^{th} cone of s_i. We iterate over all pairs $(L, H) \in \mathcal{P}$ and maintain a reference from I_k to the site that realized a minimum diameter. For $s_i \in L \cup H$, and for the j^{th} cone around s_i, we let the sites $s_m \in H' \cup \{s_\ell, s_h\}$ with $m > i$ that are contained in this cone be C_{ij}. For each $s_m \in C_{ij}$, we locate the interval I_k of the cone that contains t^*_{im} and compare the diameter of $ball(i, m)$ with the smallest diameter of I_k that we have encountered so far. If the diameter of $ball(i, m)$ is smaller, we set s_m to be the site of I_k realizing the minimum diameter. After the pass over all pairs is completed, for the j^{th} cone of site s_i, and for all intervals I_k, we add the site that realized the minimum diameter for I_k into the set A_{ij}. This completes our three passes that construct the cone sets. Finally, we set $A_i = \cup_j A_{ij}$, and then apply Theorem 3 to the set of balls A_i. This gives us the following theorem.

▶ **Theorem 4.** *Given n sites in $\mathbb{R}^d$, each with a positive weight factor, one can compute an ε-AMVWD with total size $\mathcal{O}\left(n \frac{\log 1/\varepsilon}{\varepsilon^{d-1}}\right)$. The construction time is $\mathcal{O}\left(\frac{1}{\varepsilon^{d(d+1)}} + \frac{\log n}{\varepsilon^{d+2}} \frac{1}{\log 1/\varepsilon}\right)$ times the output size. The point location query time in the ε-AMWVD is $\mathcal{O}\left(\log \frac{n}{\varepsilon}\right)$.*

To prove Theorem 4, we show that every $(i, j) \in B_i \setminus A_i$ is α-covered. The main idea is to let $(L, H) \in \mathcal{P}$ be the separating pair for (i, j), and to consider the four possible cases of (L, H) as shown in Figure 4. For the full proof see the full version of the paper [3].

Acknowledgement

This work was supported under the Australian Research Council Discovery Projects funding scheme (project number DP180102870).

References

1 Sunil Arya and Theocharis Malamatos. Linear-size approximate Voronoi diagrams. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'02)*, pages 147–155, 2002. URL: http://dl.acm.org/citation.cfm?id=545381.545400.

2 Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific, 2013. doi:10.1142/8685.

3 Joachim Gudmundsson, Martin P. Seybold, and Sampson Wong. Approximating multiplicatively weighted voronoi diagrams: Efficient construction with linear size. *CoRR*, abs/2112.12350, 2021. URL: https://arxiv.org/abs/2112.12350, arXiv:2112.12350.

4 Sariel Har-Peled. A replacement for Voronoi diagrams of near linear size. In *Proc. 42nd Annual Symposium on Foundations of Computer Science (FOCS'01)*, pages 94–103, 2001. doi:10.1109/SFCS.2001.959884.

5 Sariel Har-Peled. *Geometric approximation algorithms*. Number 173 in Mathematical Surveys and Monographs. American Mathematical Society, 2011.

6 Sariel Har-Peled and Nirman Kumar. Approximating minimization diagrams and generalized proximity search. *SIAM J. Comput.*, 44(4):944–974, 2015. doi:10.1137/140959067.

7 Yogish Sabharwal, Nishant Sharma, and Sandeep Sen. Nearest neighbors search using point location in balls with applications to approximate Voronoi decompositions. *J. Comput. Syst. Sci.*, 72(6):955–977, 2006. doi:10.1016/j.jcss.2006.01.007.

An Optimal Algorithm for the Weighted Center Problem on Cycle Graphs[*]

Taekang Eom[1] and Hee-Kap Ahn[2]

1 **Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang, Korea**
tkeom0114@postech.ac.kr

2 **Department of Computer Science and Engineering, Graduate School of Artificial Intelligence, Pohang University of Science and Technology, Pohang, Korea**
heekap@postech.ac.kr

Abstract

We study the problem of computing the weighted center of cycle graphs whose vertices are weighted. The distance from a vertex to a point of the graph is defined as the weight of the vertex times the length of the shortest path between the vertex and the point. The weighted center of the graph is a point of the graph such that the maximum distance of the vertices of the graph to the point is minimum among all points of the graph. We present an $O(n)$-time algorithm for cycle graphs with n vertices, which improves upon the best-known running time $O(n \log n)$.

1 Introduction

For a graph $G = (V, E)$, the k centers of G consist of k points $c_1, \ldots, c_k$ of G lying on vertices or edges such that the maximum distance of the vertices in V to their closest points in $\{c_1, \ldots, c_k\}$ is minimum among all sets of k points of G.

Each edge e of G has length $\lambda(e)$, and each vertex v of G is assigned a positive weight, denoted by $w(v)$. A graph is said to be *unweighted* if $w(v) = 1$ for all vertices, and it is *weighted* otherwise. For any two points p and q of G, the length of the shortest path between p and q is the sum of the lengths $\lambda(e)$ of the edges e contained in the path. If p or q lies in the interior of some edge e', the portion of the path appearing in e' has length proportional to the ratio of the length of the portion to $\lambda(e')$. The (weighted) distance from a vertex $v \in V$ to a point p of G is defined as $w(v)$ times the length of the shortest path between v and p. For unweighted graphs where $w(v) = 1$ for all $v \in V$, the distance is simply the length of the shortest path between v and p. There are two versions of the problem depending on the locations of the centers. In the *discrete* version, the centers must be chosen from V, while in the *continuous* version, the center can be placed anywhere in the graph, including edges.

There have been works on various types of graphs. Kariv and Hakimi [4] showed NP-hardness of the continuous k-center problem for weighted graphs and gave an $O(m^k n^k \log n\alpha(n))$-time algorithm, where n and m are the numbers of vertices and edges, respectively, and $\alpha(n)$ is the inverse Ackermann function. Frederickson [3] considered the k-center problem for unweighted trees with n vertices, and gave an $O(n)$-time algorithm using $O(n)$ space for both discrete and continuous problems. The algorithm uses parametric search. For weighted trees with n vertices, Wang and Zhang [8] gave an $O(n \log n)$-time algorithm for both continuous

[*] This research was partly supported by the Institute of Information & communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (No. 2017-0-00905, Software Star Lab (Optimal Data Structure and Algorithmic Applications in Dynamic Geometric Environment)) and (No. 2019-0-01906, Artificial Intelligence Graduate School Program(POSTECH)).

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

and discrete versions of the k-center problem. Bhattacharya et al. [2] gave an $O(2^{k^2}n)$-time algorithm for the continuous version of the problem. For weighted cactus graphs with n vertices, Ben-Moshe et al. [1] gave an $O(n\log^2 n)$-time algorithm for the discrete version, and an $O(n^2)$-time algorithm for the continuous version.

For the case of computing one center for the discrete and continuous versions, Kariv and Hakimi [4] gave an $O(mn\log n)$-time algorithm on undirected weighted graphs, where n and m are the numbers of vertices and edges of a graph, respectively. Megiddo [6] gave an $O(n)$-time algorithm for weighted trees with n vertices. Lan et al. [5] gave an $O(n)$-time algorithm on unweighted cactus graphs with n vertices. Ben-Moshe et al. [1] gave an $O(n\log n)$-time algorithm for weighted cactus graphs with n vertices. For weighted cycle graphs with n vertices, the best known algorithm is given by Rayco et al. which runs in $O(n\log n)$ time [7].

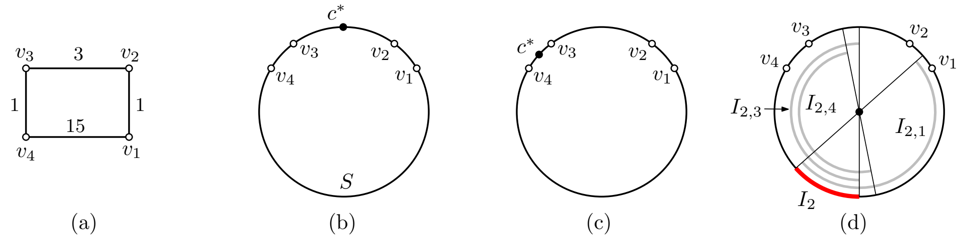


Figure 1 (a) A cycle graph with four vertices. (b) An embedding of the graph in (a) onto S. The center of the graph is c^* for $w(v_j) = 1$ for all $j = 1, \dots, 4$. (c) The weighted center is c^* when $w(v_4) > w(v_j)$ for all $j = 1, 2, 3$. (d) $I_2 = I_{2,1} \cap I_{2,3} \cap I_{2,4}$ for $w(v_j) = 1$ for all $j = 1, \dots, 4$.

In this paper, we give an $O(n)$-time algorithm for the discrete and continuous weighted center problem on cycle graphs with n vertices, which is optimal for the weighted center problem of cycle graphs. Since the algorithm for the continuous version also works for the discrete version with little modification, we focus on the algorithm for the continuous version.

2 Preliminaries

Observe that any cycle graph G can be embedded in a unit circle $S \subset \mathbb{R}^2$ such that each vertex of G corresponds to a point in S, appearing in S in the order along G, and each edge of G corresponds to a circular arc between two points of S corresponding to its end vertices. The lengths of the circular arcs are proportional to the lengths of the edges in G. See Figure 1 (a) and (b) for an illustration. For any three disjoint subsets S_1, S_2, S_3 of S, we use $S_1 \prec S_2 \prec S_3$ if S_1, S_2, S_3 appear in counterclockwise order along S.

We assume that the input graph $G = (V, E)$ is embedded in a unit circle S. Let $v_1, \dots, v_n \in V$ be the vertices of G appearing in counterclockwise order on S. For any two points $x, y \in S$, let $\ell(x, y)$ denote the length of a shortest path between x and y in S. Let $w_i = w(v_i)$ denote the weight assigned to each vertex $v_i \in V$. The distance of a vertex v_i to a point $x \in S$ is defined as $d_i(x) := w_i \cdot \ell(x, v_i)$. Then, the weighted center problem finds a point $c^* = \arg\min_{c\in S} \max_{1\le i\le n} d_i(c)$ and a distance value $r^* = \min_{c\in S} \max_{1\le i\le n} d_i(c)$. Figure 1 (b) and (c) show the weighted centers of a cycle graph.

Let $I_{i,j} := \{x \mid d_i(x) > d_j(x)\} \subset S$. Clearly, $I_{i,i} = \emptyset$ and $I_{i,j}$ is connected for any $i, j \in \{1, \dots, n\}$. For a subset $W \subseteq V$, let $I_i^W := \cap_{v_j \in W\setminus\{v_i\}} I_{i,j}$. We denote I_i^V by I_i. See Figure 1 (d) for an illustration.

▶ **Lemma 2.1.** *The followings hold.*

$I_i^W \subseteq I_i^U$ if $U \subseteq W$ for any two subsets U and W of V.

- *I_i^W and I_j^W are disjoint for any two distinct indices $i, j \in W \subseteq V$.*
- *I_i^W is connected for any subset $W \subseteq V$.*

Proof. For two subsets U and W with $U \subseteq W$, $I_i^W = \cap_{v_j \in W \setminus \{v_i\}} I_{i,j} \subseteq \cap_{v_j \in U \setminus \{v_i\}} I_{i,j} = I_i^U$.

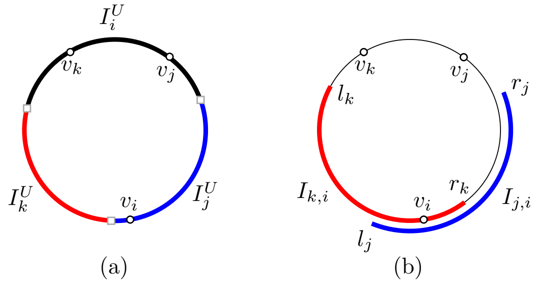
For any two distinct indices $i, j \in W \subseteq V$, $I_i^W \subseteq I_{i,j}$ and $I_j^W \subseteq I_{j,i}$. Since $I_{i,j}$ and $I_{j,i}$ are disjoint, I_i^W and I_j^W are disjoint.

Let W be any subset of V. If $I_i^W = \emptyset$, it is connected. Assume $I_i^W \neq \emptyset$. Since $I_{i,j}$ is connected, $S \setminus I_{i,j}$ is also connected for any $v_j \in W \setminus \{v_i\}$. Since $d_i(v_i) = 0$, $v_i \in S \setminus I_{i,j}$ for every $v_j \in W \setminus \{v_i\}$. So, $S \setminus I_i^W = \cup_{v_j \in W \setminus \{v_i\}} (S \setminus I_{i,j})$ is connected, implying that I_i^W is also connected. ◀

For any two points $x, y \in S$, we denote by (x, y) the open interval from x to y in counterclockwise order. We say a vertex v_i is *active* in $W \subset V$ if $v_i \in W$ and $I_i^W \neq \emptyset$.

▶ **Lemma 2.2.** *If v_i, v_j, v_k are active in $W \subset V$ and $v_i \prec v_j \prec v_k$, then $I_i^W \prec I_j^W \prec I_k^W$.*

Proof. Let v_i, v_j, v_k be three active vertices in W satisfying $v_i \prec v_j \prec v_k$. By Lemma 2.1, each of I_i^W, I_j^W, I_k^W is connected, and they are disjoint. Assume to the contrary that $I_i^W \prec I_k^W \prec I_j^W$. Let $I_{j,i} = (l_j, r_j)$ and $I_{k,i} = (l_k, r_k)$. For the subset $U = \{v_i, v_j, v_k\} \subseteq W$, I_i^U, I_j^U, I_k^U are disjoint, and $I_i^W \subseteq I_i^U, I_j^W \subseteq I_j^U, I_k^W \subseteq I_k^U$ by Lemma 2.1. Thus, $I_i^U \prec I_k^U \prec I_j^U$. Moreover, $v_i \in I_{j,i} \cap I_{k,i}$. Thus l_k, l_j, v_i, r_k, r_j appear in counterclockwise order along S. See Figure 2. Observe that $v_k, v_j \in I_i^U$ since $v_i \prec v_j \prec v_k$, $v_k \notin I_{k,i}$ and $v_j \notin I_{j,i}$. By the same argument, $v_i, v_k \in I_j^U$. This contradicts that I_i^U and I_j^U are disjoint. ◀



(a) (b)

Figure 2 (a) $I_i^U \prec I_k^U \prec I_j^U$. (b) l_k, l_j, v_i, r_k, r_j are in counterclockwise order along S.

3 Algorithm

We present an algorithm for computing the weighted center of a cycle graph $G = (V, E)$ embedded in a unit circle S. Our algorithm works as follows. In the first step, it finds the index $i^* = \arg\max d_i(x)$ for any fixed position $x \in S$, sets v_{i^*} to v_1, and relabels all vertices in counterclockwise order along S. Then v_1 is active in the relabeled list V. In the second step, it iterates over the vertices one by one in order from v_1, and updates the list of active vertices, sorted in increasing order of indices, at each iteration. After the iteration finishes, it has the final list of active vertices. In the third step, it computes the intervals I_i for each active vertex v_i in V in counterclockwise order using the list of active vertices. Then it finds the minimum of $d_i(x)$ among points x in the closure of I_i for each active vertex v_i in constant time. So, this step takes $O(n)$ time. In the fourth step, it returns the point $c^* = \arg\min_{c \in S} \max_{1 \leq i \leq n} d_i(c)$, achieving the minimum among the minimum distance values $r^* = \min_{c \in S} \max_{1 \leq i \leq n} d_i(c)$ as the weighted center of G.

We focus on the second step, the iteration part, of the algorithm for computing the list of active vertices in $O(n)$ time. Before describing the algorithm, we introduce some terms and technical lemmas. We say a vertex v_i is *dominated by* $W \subset V$ *if* $I_i^W = \emptyset$, and we denote this by $v_i \lhd W$. If v_i is not dominated by W, we denote this by $v_i \ntriangleleft W$.

▶ **Lemma 3.1.** *If* $v_k, v_\ell \lhd \{v_i, v_j\}$ *for* $i < k < \ell < j$*, then* $v_k \lhd \{v_i, v_\ell\}$ *or* $v_\ell \lhd \{v_k, v_j\}$*.*

Proof. Assume to the contrary that (1) $v_k, v_\ell \lhd \{v_i, v_j\}$, but (2) $v_k \ntriangleleft \{v_i, v_\ell\}$ and (3) $v_\ell \ntriangleleft \{v_k, v_j\}$. We first consider the case that (4) $v_\ell \ntriangleleft \{v_i, v_k\}$. Let $U = \{v_i, v_k, v_\ell\}, W = \{v_k, v_\ell, v_j\}$ and $X = \{v_i, v_k, v_\ell, v_j\}$. Then all vertices in U are active in U: $I_i^U \neq \emptyset$ because $v_j \in I_i^X$ by (1) and $I_i^X \subset I_i^U$ by Lemma 2.1, $I_k^U \neq \emptyset$ by (2), and $I_\ell^U \neq \emptyset$ by (4). By Lemma 2.2, $I_i^U \prec I_k^U \prec I_\ell^U$.

Let x be the clockwise boundary point of $I_{k,\ell}$ and y be the other boundary point of $I_{k,\ell}$. Then, $x \in (v_k, v_\ell) \subset (v_k, v_j)$. We have $I_{k,i} \subset (v_j, v_k)$ because $d_k(v_k) < d_i(v_k)$ and $d_k(v_j) \leq d_i(v_j)$ by (1). So, $x \in (v_k, v_j) \subset I_{i,k}$. Moreover, $x \in I_{i,\ell}$ because $d_i(x) > d_k(x) = d_\ell(x)$. Thus $x \in I_i^U$. Also $y \notin I_i^U$ because $I_k^U \neq \emptyset$ and $I_\ell^U \neq \emptyset$. See Figure 3(a), (b) and (c).

Observe that $y \in I_j^W$, $I_k^W \subset I_{k,\ell}$ and $I_\ell^W \subset I_{\ell,k}$. Thus, if all vertices in W are active in W, the three intervals I_k^W, I_ℓ^W, I_j^W must appear in clockwise order along S. See Figure 3(d). This contradicts Lemma 2.2, so k or ℓ is not active in W, i.e. $v_k \lhd \{v_\ell, v_j\}$ or $v_\ell \lhd \{v_k, v_j\}$.

Since $v_\ell \ntriangleleft \{v_k, v_j\}$ by (3), $v_k \lhd \{v_\ell, v_j\}$. Then, $d_k(v_j) \leq \max\{d_j(v_j), d_\ell(v_j)\} = d_\ell(v_j)$ and $d_k(v_k) < d_\ell(v_k)$ i.e. $I_{k,\ell} \subset (v_k, v_j)$. Also, $I_{k,i} \subset (v_j, v_k)$ by the previous argument. $I_k^U = I_{k,i} \cap I_{k,\ell} \subset (v_j, v_k) \cap (v_k, v_j) = \emptyset$, it contradicts the assumption.

Now consider the case that $v_\ell \lhd \{v_i, v_k\}$, instead of (4). Observe that this is symmetric to $v_k \lhd \{v_\ell, v_j\}$, and we can achieve a contradiction $I_\ell^W = \emptyset$ similarly. ◀

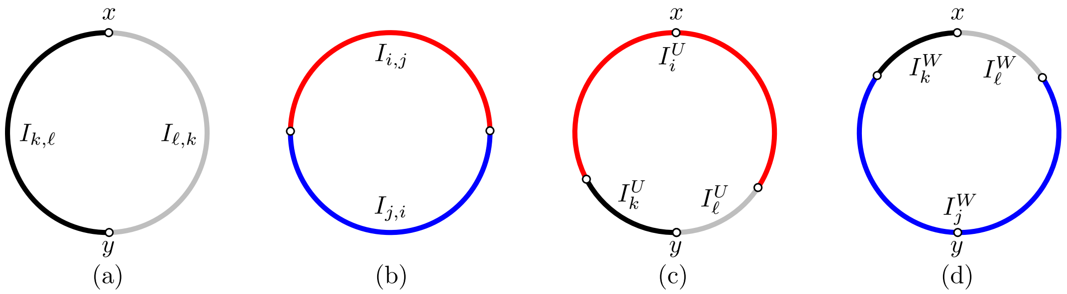


Figure 3 (a) Intervals on S for v_k and v_ℓ. (b) Intervals for v_i and v_j. (c) Intervals for v_i, v_k and v_ℓ. (d) Intervals for v_k, v_ℓ and v_j.

We denote by V_i the sequence $\langle v_1, v_2, \ldots, v_i \rangle$ of vertices of V.

▶ **Lemma 3.2.** *For any two vertices* v_i *and* v_j *that are consecutive in the (cyclic) list of active vertices of* V_m*, the followings hold.*

(1) $v_k \lhd \{v_i, v_j\}$ *if* $v_i \prec v_k \prec v_j$*.*

(2) For any subsequence V' *of* V_m*, containing* v_i *and* v_j*, there are three vertices* v, v', v'' *consecutive in* V' *and* $v_i \prec v' \prec v_j$ *such that* $v' \lhd \{v, v''\}$*.*

Proof. For ease of description, assume that $1 \leq i < j \leq m$. For Claim (1), assume to the contrary that there exists an index k with $i < k < j$ such that $v_k \ntriangleleft \{v_i, v_j\}$. Let $W = \{v_i, v_k, v_j\}$. Then $I_k^W \neq \emptyset$. Moreover, $I_i^W \neq \emptyset$ and $I_j^W \neq \emptyset$ because $I_i^{V_m} \neq \emptyset$ and $I_j^{V_m} \neq \emptyset$ (because v_i and v_j are active in V_m), and $I_i^{V_m} \subseteq I_i^W$ and $I_j^{V_m} \subseteq I_j^W$ (because $W \subseteq V_m$) by Lemma 2.1. By Lemma 2.2, $I_i^W \prec I_k^W \prec I_j^W$. However, $I_\ell^{V_m} = \emptyset$ for all ℓ with

$i < \ell < j$, and thus $I_j^{V_m}$ appears next to $I_i^{V_m}$ consecutively in counterclockwise order along S by Lemma 2.2. Since $I_i^{V_m} \subset I_i^W, I_j^{V_m} \subset I_j^W$, we obtain $I_k^W = \emptyset$, a contradiction.

We prove Claim (2) by induction on $j-i$. When $j-i=2$, the claim holds by Lemma 3.2(1). When $j-i=3$, the claim holds by Lemma 3.1 and Lemma 3.2(1). Assume to the contrary that the claim holds for all $j-i \geq 2$ values up to t for some integer t with $3 \leq t < n$, but it does not hold for $t+1$. Then $v_k \not\lhd \{v_{k-1}, v_{k+1}\}$ for all k with $i < k < j$ for $j-i = t+1$.

- Let $V^{i+1} = \langle v_1, \dots, v_i, v_{i+2}, \dots, v_j, \dots, v_m \rangle$ be the list obtained by removing v_{i+1} from V_m. Then, by the induction hypothesis, $v_{i+2} \lhd \{v_i, v_{i+3}\}$ or there exists an index ℓ with $i+2 < \ell < j$ such that $v_\ell \lhd \{v_{\ell-1}, v_{\ell+1}\}$. Thus, assume $v_{i+2} \lhd \{v_i, v_{i+3}\}$.
- Similarly, let V^{i+2} be the list obtained by removing v_{i+2} from V_m. Then we can deduce $v_{i+1} \lhd \{v_i, v_{i+3}\}$ or $v_{i+3} \lhd \{v_{i+1}, v_{i+4}\}$ by the same argument. If $v_{i+1} \lhd \{v_i, v_{i+3}\}$, we have $v_{i+1} \lhd \{v_i, v_{i+2}\}$ or $v_{i+2} \lhd \{v_{i+1}, v_{i+3}\}$ by Lemma 3.1, a contradiction. For $v_{i+3} \lhd \{v_{i+1}, v_{i+4}\}$, we can obtain $v_{j-1} \lhd \{v_{j-3}, v_j\}$ by applying the same argument repeatedly.
- Similarly, let V^{j-1} be the list obtained by removing v_{j-1} from V_m. Then we can obtain $v_{j-2} \lhd \{v_{j-3}, v_j\}$. From $v_{j-1} \lhd \{v_{j-3}, v_j\}$ and $v_{j-2} \lhd \{v_{j-3}, v_j\}$, we obtain $v_{j-2} \lhd \{v_{j-3}, v_{j-1}\}$ or $v_{j-1} \lhd \{v_{j-2}, v_j\}$ by Lemma 3.1, a contradiction.

Thus, the claim also holds for $t+1 = j-i$. ◀

In the second step, the algorithm updates the active vertices incrementally as follows. Given the list of all active vertices in V_{i-1}, it computes the list of all active vertices in V_i. Let L_a be the list of the active vertices in counterclockwise order from v_1 along S computed so far. Thus, it is computed for V_{i-1}. The algorithm tests if v_i is dominated by v_1 and the last vertex of L_a. If v_i is dominated by both the vertices, L_a is the set of vertices active in V_i by Lemma 3.2(2). So, the algorithm proceeds to the next vertex v_{i+1}. Otherwise, the algorithm does the followings. It appends v_i to L_a since v_i is active in V_i by Lemma 3.2(2). Then it deletes $L_a[-2]$ if $|L_a| \geq 3$ and $L_a[-2] \lhd \{L_a[-3], L_a[-1]\}$, where $L_a[-j]$ is the j-th last element in L_a for $j > 0$. It repeats this until $|L_a| < 3$ or $L_a[-2] \not\lhd \{L_a[-3], L_a[-1]\}$. At the end of the repetition, L_a is the list of active vertices in V_i by Lemma 3.2(2). Then the algorithm proceeds to the next vertex v_{i+1}. After v_n is handled, L_a is the list of active vertices in V_n.

Now we analyze the running time of the algorithm. At the iteration for V_i, vertex v_i is tested for its dominance once. If v_i passes the test, it is appended to L_a. If it fails the test, the algorithm proceeds to the next iteration for V_{i+1}. Observe that a vertex v contained in L_a can be tested more than once. If it passes the test at the iteration for V_i, v remains in L_a but the algorithm proceeds to the next iteration for V_{i+1}. If it fails the test, it is removed from L_a and will never be inserted to L_a again. Since each test can be done in $O(1)$ time and the algorithm iterates n times, the final list of active vertices can be computed in $O(n)$ time.

▶ **Theorem 3.3.** *The weighted center problem on cycle graphs can be solved in $O(n)$ times using $O(n)$ spaces.*

References

1 Boaz Ben-Moshe, Binay Bhattacharya, Qiaosheng Shi, and Arie Tamir. Efficient algorithms for center problems in cactus networks. *Theoretical Computer Science*, 378(3):237–252, 2007.

2 Binay Bhattacharya, Sandip Das, and Subhadeep Ranjan Dev. The weighted k-center problem in trees for fixed k. In *30th International Symposium on Algorithms and Computation (ISAAC 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

3 Greg N Frederickson. Parametric search and locating supply centers in trees. In *Workshop on Algorithms and Data Structures*, pages 299–319. Springer, 1991.

4 Oded Kariv and S Louis Hakimi. An algorithmic approach to network location problems. I: The p-centers. *SIAM Journal on Applied Mathematics*, 37(3):513–538, 1979.

5 Yu-Feng Lan, Yue-Li Wang, and Hitoshi Suzuki. A linear-time algorithm for solving the center problem on weighted cactus graphs. *Information Processing Letters*, 71(5-6):205–212, 1999.

6 Nimrod Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.

7 M Brenda Rayco, Richard L Francis, and Arie Tamir. A p-center grid-positioning aggregation procedure. *Computers & Operations Research*, 26(10-11):1113–1124, 1999.

8 Haitao Wang and Jingru Zhang. An $O(n \log n)$-time algorithm for the k-center problem in trees. *SIAM Journal on Computing*, 50(2):602–635, 2021.

Intersections of Double-Wedge Arrangements

Daniel Bertschinger[1], Henry Förster[2], and Birgit Vogtenhuber[3]

1 **ETH Zürich, Zürich, Switzerland**
`daniel.bertschinger@inf.ethz.ch`

2 **University of Tübingen, Tübingen, Germany**
`henry.foerster@uni-tuebingen.de`

3 **TU Graz, Graz, Austria**
`bvogt@ist.tugraz.at`

Abstract

We study the common intersection of arrangements of double-wedges. In contrast to earlier studies in the past, we consider arrangements where double-wedges may or may not include a vertical line. This changes the setting drastically, in particular with respect to all arguments involving the point-line duality. We show that in this setting, the intersection of n double-wedges may consist of $\Omega(n^2)$ interior-disjoint regions. Further, we provide algorithms for computing the intersection of such arrangements with worst-case optimal running time.

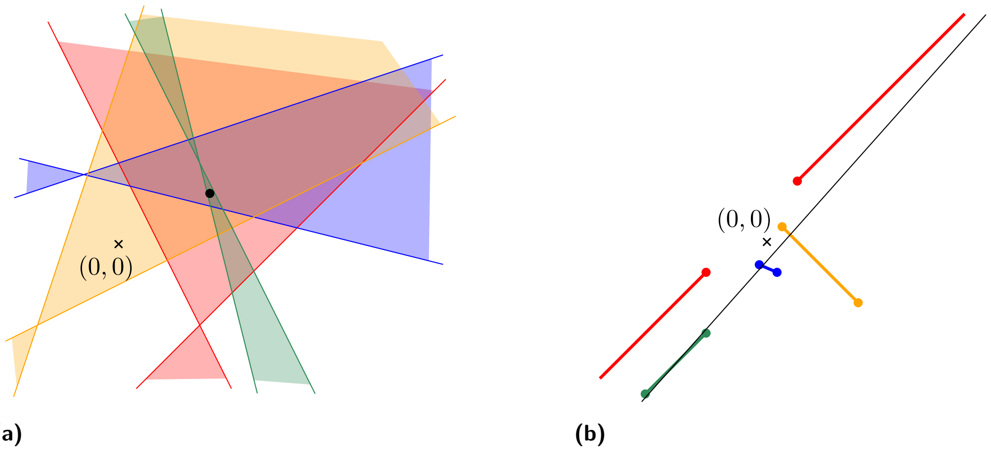


Figure 1 (a) An arrangement of double-wedges and (b) its projective dual. A point contained in every double-wedge dualizes to a line stabbing all segments corresponding to bow-ties and stabbing all anti-segments corresponding to hour-glasses.

1 Introduction

Two non-parallel lines ℓ_1 and ℓ_2 subdivide the Euclidean plane into four different wedges. The union of two opposite wedges forms a so-called *double-wedge* [4, 10]. In other words, it is the closure of the symmetric difference of two half-planes delimited by ℓ_1 and ℓ_2. We distinguish two different types of double-wedges, namely, those that do not contain any vertical line, which we call *bow-ties*, and those that do contain a vertical line, which we call *hour-glasses*. If neither ℓ_1 nor ℓ_2 are vertical, then the closure of the complement of the bow-tie spanned by them is an hour-glass and vice versa. For the remainder of this work, we assume w.l.o.g. that no bounding line of any double-wedge is vertical or horizontal.

It is noteworthy that bow-ties are the projective dual of a non-vertical line segment in the Euclidean plane; see Figures 1 and 2a. Hence, each point in the common intersection of

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

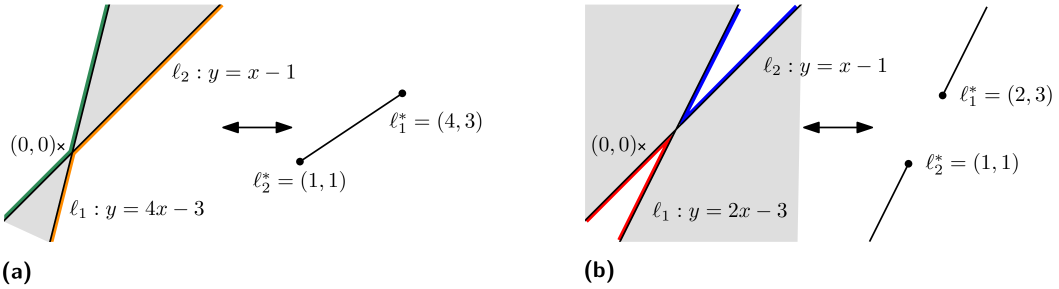


Figure 2 (a) A bow-tie and its dual line segment. (b) An hour-glass and its dual anti-segment. Left, right, upper and lower traces are colored red, blue, green and orange, respectively.

a family of bow-ties corresponds to a stabbing line of the dual family of line segments. This duality has been used for the efficient stabbing of line segments by settling the computation of the intersection of bow-tie arrangements in time $\mathcal{O}(n \log n)$ [8]. Coming from this line of reasoning, that is, about the stabbing of line segments, double-wedges are sometimes also defined as what we call bow-ties [1, 7].

In this work however, we study more general arrangements of double-wedges, namely, arrangements that can contain both bow-ties and hour-glasses. In contrast to bow-ties, hour-glasses are the projective dual of *anti-segments*, that is, a straight line minus a segment contained in this line; see Figures 1 and 2b. Thus, the common intersection of a general double-wedge arrangement corresponds to (i) the stabbing lines of an arrangement of segments and anti-segments, or, equivalently, (ii) the lines that stab all segments corresponding to the bow-ties while avoiding the segments dual to the inverse of each hour-glass, see also Figure 1.

The remainder of this paper is structured as follows. First, we give formal definitions and preliminary observations in Section 2. Then, we prove that the intersection of double-wedge arrangements with both bow-ties and hour-glasses may consist of $\Omega(n^2)$ interior-disjoint regions in Section 3. Finally, in Section 4 we describe efficient algorithms for the computation of the intersection before concluding the paper with open problems in Section 5.

2 Preliminaries

Let h_1 and h_2 be two half-planes in $\mathbb{R}^2$ that are bounded by non-parallel lines ℓ_1 and ℓ_2, respectively. We denote the double-wedge formed by the closure of the symmetric difference of h_1 and h_2 as $\langle h_1, h_2 \rangle$ and the intersection point of ℓ_1 and ℓ_2 as the *origin* of $\langle h_1, h_2 \rangle$.

For a double-wedge $d = \langle h_1, h_2 \rangle$, let ℓ be the line with slope $a_\ell = (a_1 + a_2)/2$ through the origin o_d of d, where a_1 and a_2 are the slopes of the bounding lines ℓ_1 of h_1 and ℓ_2 of h_2, respectively. We refer to the two rays of ℓ_1 and ℓ_2 emerging from o_d that are located above ℓ as the *upper trace* of d and to the other two rays of ℓ_1 and ℓ_2 as the *lower trace* of d. Similarly, we call the two rays of ℓ_1 and ℓ_2 emerging from the origin o_d of d that are located to the left (and right) of the vertical line through o_d the *left trace* of d (and *right trace* of d, respectively); see Figure 2.

Note that the upper and lower trace coincide with the upper and lower envelope of the arrangement $\mathcal{A}$ of ℓ_1 and ℓ_2. In contrast, the left and right trace only coincide with the left and right envelope of $\mathcal{A}$ if the slopes of ℓ_1 and ℓ_2 have different signs.

We say that a point $x \in \mathbb{R}^2$ is *below* the upper trace (or *above* the lower trace) if and only if a vertical ray emerging from x in positive (or negative, respectively) y-direction

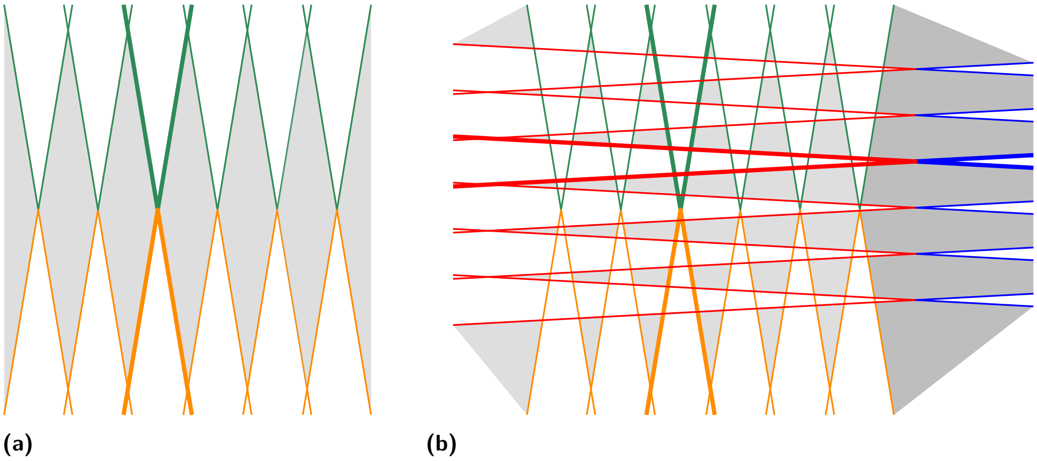


Figure 3 (a) A vertical grating consisting of $k = 6$ bow-ties. Their intersection consists of $k + 1 = 7$ interior-disjoint regions (gray). (b) A combined grating consisting of a vertical grating of $k = 6$ bow-ties and a horizontal grating consisting of $k = 6$ hour-glasses. Their intersection consist of $(k + 1)^2 = 49$ interior-disjoint regions (gray). (For convenience one bow-tie in (a) and two double-wedges in (b) are drawn bold.)

hits the upper trace (or lower trace, respectively). Similarly, we say x is *to the left* of the right trace (or *to the right* of the left trace) if and only if a ray orthogonal to ℓ emerging from x in positive (or negative, respectively) x-direction hits the right trace (or left trace, respectively).

With these definitions at hand, we can characterize the intersection of a double-wedge arrangement $\mathcal{A}$ consisting of both bow-ties and hour-glasses in the following way:

▶ **Observation 1.** *A point $p \in \mathbb{R}^2$ is part of the intersection of a double-wedge arrangement $\mathcal{A}$ if and only if (i) p is below the upper trace but above the lower trace of each bow-tie in $\mathcal{A}$, and (ii) p is to the right of the left trace but to the left of the right trace of each hour-glass in $\mathcal{A}$.*

In order to emphasize the connection to the stabbing of line arrangements discussed before, we also mention the transformation T between straight lines and points used for establishing the projective duality [2, 8]. Namely, the standard *projective duality* T transforms the point $p = (p_x, p_y)$ to the non-vertical line $p^* : y = p_x \cdot x - p_y$ and, vice-versa, the non-vertical line $\ell : y = mx + b$ to the point $\ell^* = (m, -b)$; see again Figure 2.

3 Combinatorial complexity of double-wedge intersections

In this section we prove the following result concerning the complexity of the intersection of a double-wedge arrangement:

▶ **Theorem 3.1.** *For every $k \in \mathbb{N}$, there exists a double-wedge arrangement $\mathcal{A}_n$ consisting of $n = 2k$ double-wedges, so that its intersection consists of $(n/2 + 1)^2$ interior-disjoint regions.*

Proof. We prove the statement by an explicit construction. We first take k bow-ties. Their origins are located on a common horizontal line and their bounding lines are chosen to be parallel, so that they create a *vertical grating* as shown in Figure 3a. The intersection of the vertical grating consists of a 4-gon between each pair of consecutive (along the x-axis) double-wedges plus the two unbounded regions at the left and right boundary. Thus, the vertical grating defines an intersection consisting of $k+1$ interior-disjoint regions.

The remaining k double-wedges are a copy of the vertical grating that is rotated by $\pi/2$, yielding a *horizontal grating* of hour-glasses.

The entire arrangement $\mathcal{A}_n$ is a combined grating consisting of both a vertical and a suitably stretched and translated horizontal grating. The origins of the latter are located in the right unbounded region of the former. Further, the gratings overlap in such a way that no origin of the vertical grating lies in the intersection of the horizontal grating and that each region in the intersection of the vertical grating intersects each region in the intersection of the horizontal grating; see Figure 3b. Hence, the intersection of $\mathcal{A}_n$ consists of $(k+1)^2$ interior-disjoint regions (all but $k+1$ of which are pairwise disjoint entirely; indicated in darker gray in Figure 3b), and the statement of the theorem follows. ◀

Note that this construction yields double-wedges in non-general position. However, slightly wiggling everything allows to also get double-wedge arrangements in general position with $\Omega(n^2)$ many interior-disjoint intersection regions. Moreover, rotating the arrangement of Figure 3b by $\pi/4$ yields an arrangement that only consists of hour-glasses.

We remark that in contrast to the intersection of general double-wedge arrangements discussed in this paper, the intersection of a double-wedge arrangement consisting purely of bow-ties consists only of $\mathcal{O}(n)$ regions [8]. This increase in complexity results in an increased worst-case time complexity for the computation of general double-wedge arrangements.

4 Computing the intersection of double-wedges

In this section, we present two algorithms for computing the intersection of an arrangement $\mathcal{A}$ of double-wedges. The first algorithm in Section 4.1 assumes that the double-wedges do not cover all slopes in $\mathbb{R} \cup \{\infty, -\infty\}$ and runs in time $\mathcal{O}(n \log n)$. The second algorithm in Section 4.2 requires time $\mathcal{O}(n^2)$ but works for all double-wedge arrangements. We will also discuss that both algorithms have worst-case optimal running time.

4.1 The double-wedges in $\mathcal{A}$ do not cover all slopes

As pointed out before, an $\mathcal{O}(n \log n)$-time algorithm [8] is known for the case where all double-wedges are bow-ties. This algorithm can also be applied in the case where the double-wedges do not cover all slopes in the plane. In order to achieve this, we identify a slope a not covered by any double-wedge and rotate $\mathcal{A}$ such that lines with slope a become vertical. It is noteworthy that if a exists, we may assume w.l.o.g. that a is rational under the assumption that all lines bounding the double-wedges have rational slope. This procedure results in a pure bow-tie arrangement.

It remains to argue that a can be computed in time $\mathcal{O}(n \log n)$. We first sort the bounding lines of all double-wedges by slope in time $\mathcal{O}(n \log n)$ and check for slope $a^* = -\infty$ in how many double-wedges it lies. In fact, this number is equal to the number of hour-glasses and it can be found in linear time by checking for every double-wedge individually whether it is an hour-glass or not. If it is zero, then we can directly apply the algorithm from [8]. Otherwise, we iteratively increase a^* and whenever a^* becomes larger than the slope of a line bounding a double-wedge d, we update the number of double-wedges a^* is in. Note that the number of double-wedges covering a slope only changes by ±1 on each such event (and does not change between two neigbhoring slopes in the sorted slope list). Hence, going once through the sorted list of slopes, each update can be done in constant time (by checking the double-wedge d). This procedure takes $\mathcal{O}(n)$ time and identifies two slopes of bounding lines a_1 and a_2 so that no slope between a_1 and a_2 is covered by any double-wedge in $\mathcal{A}$. We choose any slope between a_1 and a_2 for a, e.g., $a = (a_1 + a_2)/2$.

In total, we can summarize as follows:

▶ **Theorem 4.1.** *Let $\mathcal{A}$ be an arrangement of double-wedges so that $\mathcal{A}$ does not cover all slopes in $\mathbb{R} \cup \{\infty, -\infty\}$. Then, the intersection of $\mathcal{A}$ can be computed in time $\mathcal{O}(n \log n)$.*

It is worth noting that Edelsbrunner et al. [8] also state that the computation of the intersection of n bow-ties requires time $\Omega(n \log n)$. Since a pure bow-tie arrangement is a special case of the instances discussed in this section, the running time of the procedure described above is worst case-optimal. We further emphasize that the algorithm for identifying a can also be used to check whether $\mathcal{A}$ covers all slopes in time $\mathcal{O}(n \log n)$. Namely, if a_1 and a_2 do not exist, we can instead proceed with the algorithm in the next subsection.

4.2 The double-wedges in $\mathcal{A}$ cover all slopes

In this scenario, we first compute the line arrangement formed by the lines bounding the n double-wedges in $\mathcal{A}$ and compute its dual (graph) G. This can be done in time $\mathcal{O}(n^2)$ [3, 7, 9]. The resulting dual graph G has $\mathcal{O}(n^2)$ edges and nodes. Each edge of G corresponds to an edge in the arrangement (and lies on a bounding line of some double-wedge). Each node of G corresponds to one cell in the arrangement, which might be in the intersection of all double-wedges.

We choose an arbitrary node f_0 of G and compute the number of double-wedges of $\mathcal{A}$ in which it is contained. This can be done in constant time per double-wedge and hence in $\mathcal{O}(n)$ time in total.

Finally, we traverse G starting from f_0 using breadth-first search. Note that traversing an edge of G corresponds to either leaving or entering a single double-wedge of $\mathcal{A}$ as characterized in Observation 1. Hence, whenever we visit a new node (corresponding to a cell in the arrangement) during this traversal, we can determine the number of double-wedges in $\mathcal{A}$ in which it lies in constant time. Thus we can perform this last step in which we determine all cells in the intersection of $\mathcal{A}$ in $\mathcal{O}(n^2)$ time.

We summarize this result as follows:

▶ **Theorem 4.2.** *Let $\mathcal{A}$ be an arrangement of double-wedges. Then the intersection of $\mathcal{A}$ can be computed in time $\mathcal{O}(n^2)$.*

By Theorem 3.1, the intersection of $\mathcal{A}$ can consist of $\Omega(n^2)$ interior-disjoint regions. Thus enumerating all those regions takes time $\Omega(n^2)$ and our algorithm is worst-case optimal.

5 Conclusion and open problems

In this work, we considered the problem of determining the intersection of an arrangement of double-wedges. We gave a tight bound on the worst-case combinatorial complexity of this intersection and presented worst-case optimal algorithms for computing it. The considered problem is the dual formulation of the problem of finding all lines that stab certain line segments while keeping other line segments untouched. We conclude with an open problem and further research directions.

▶ **Open Problem 1.** *Given two sets of line segments in $\mathbb{R}^2$, how fast is it possible to compute a line that stabs all segments of the one set while avoiding all segments of the other set or conclude that no such line exists?*

This question translates to asking, how fast it is possible to compute a single point in the intersection of a double-wedge arrangement or decide that this intersection is empty. While we showed that $\Omega(n^2)$ time is required to compute the entire intersection, this lower bound does not carry over to finding only one point. For the special case of all double-wedges being hour-glasses, the intersection is always non-empty and a point in this intersection can easily be found in linear time. However, for the mixed setting, a solution might not be so straight-forward.

Further, it may be interesting to study more restricted double-wedge arrangements. For instance, arrangements of only bow-ties, where each k-tuple of double-wedges have a common intersection, display interesting properties. Namely, it is known [5, 6] that for any such arrangement with $k = 4$, there always exist two points in the plane such that any double-wedge contains at least one of them (while one single point is not always sufficient). Similarly, for $k = 3$ there always exist four points such that any double-wedge contains at least one of them (but it is unclear whether this bound is tight). Can the bound for $k = 3$ be reduced? And can these results for bow-tie arrangements be generalized to arrangements containing both bow-ties and hour-glasses?

Finally, one may investigate generalizations of double-wedge arrangements to higher dimensions and study their intersections.

Acknowledgements. This work was initiated at the 6th D-A-CH Workshop on Arrangements and Drawings in Stels, Switzerland. We thank the organizers for making this workshop possible, especially in these times, and the other participants for fruitful discussions. H.F. is partially supported by DFG grant KA812-18/2. B.V. is partially supported by the Austrian Science Fund within the collaborative D-A-CH project *Arrangements and Drawings* as FWF project I 3340-N35.

References

1 M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. 3rd edition, 2008.

2 K. Q. Brown. *Geometric Transforms for Fast Geometric Algorithms*. PhD thesis, USA, 1979. AAI8012772.

3 B. Chazelle, L. J. Guibas, and D. T. Lee. The power of geometric duality. *BIT Numerical Mathematics*, 25:76 – 90, 1985. doi:https://doi.org/10.1007/BF01934990.

4 M. Claverol, D. Garijo, C. I. Grima, A. Márquez, and C. Seara. Stabbers of line segments in the plane. *Computational Geometry*, 44(5):303–318, 2011. doi:https://doi.org/10.1016/j.comgeo.2010.12.004.

5 J. Eckhoff. Transversalenprobleme in der Ebene. *Archiv der Mathematik*, 24:195–202, 1973.

6 J. Eckhoff. A Gallai-type transversal problem in the plane. *Discret. Comput. Geom.*, 9:203–214, 1993. doi:10.1007/BF02189319.

7 H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10. 1st edition, 1987.

8 H. Edelsbrunner, H. A. Maurer, F. P. Preparata, A. L. Rosenberg, E. Welzl, and D. Wood. Stabbing line segments. *BIT*, 22(3):274–281, 1982. doi:10.1007/BF01934440.

9 H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM J. Comput.*, 15(2):341–363, 1986. doi:10.1137/0215024.

10 F. Hurtado, M. Mora, P. A. Ramos, and C. Seara. Separability by two lines and by nearly straight polygonal chains. *Discrete Applied Mathematics*, 144(1):110–122, 2004. Discrete Mathematics and Data Mining. doi:https://doi.org/10.1016/j.dam.2003.11.014.

Free Space Realizability for Curves in 1D

Hugo A. Akitaya[1], Maike Buchin[2], Majid Mirzanezhad[3], Leonie Ryvkin[2], and Carola Wenk*[4]

1 **Department of Computer Science, University of Massachusetts Lowell**
hugo_akitaya@uml.edu

2 **Faculty of Computer Science, Ruhr University Bochum**
maike.buchin|leonie.ryvkin@rub.de

3 **Transportation Research Institute, College of Engineering, University of Michigan – Ann Arbor**
miirza@umich.edu

4 **Department of Computer Science, Tulane University**
cwenk@tulane.edu

Abstract

The Fréchet distance is a well-established distance measure for comparing curves, and the free space diagram is the main tool to efficiently compute it. Inspired by [7], we consider the following inverse problem: Given a diagram of size $m \times n$, we ask whether it can be *realized* by a pair of curves on a real line. We prove weak NP-completeness and present an FPT algorithm that runs in $O(mn2^k)$ time where k is an implicit parameter that relates to the complexity of the diagram.

1 Introduction

Buchin, Ryvkin and Wenk [7] introduced the problem of realizing a given (free space) diagram by polygonal curves in the plane. We build upon their results by focusing on curves in one dimension. The main objective of studying the realizability problem is to gain a better understanding of the *Fréchet distance*, which is a popular distance and similarity measure used in various applications. The Fréchet distance of two curves $P, Q \colon I \to \mathbb{R}^d$, where $I \subseteq \mathbb{R}$, is given by $\delta_\mathrm{F}(P,Q) = \inf_{(\sigma,\theta)} \; \max_{t\in[0,1]} \; \|P(\sigma(t)) - Q(\theta(t))\|$, where the pair of reparameterizations (σ,θ) are continuous non-decreasing functions. Intuitively, one can picture a person walking their dog, each of them walking on one of the curves. The Fréchet distance equals the length of the shortest leash allowing both to fully traverse their curves continuously, choosing their speed independently.

The *free space diagram* $D_\varepsilon(P,Q)$ is the most important tool for efficiently computing the Fréchet distance, see [1]. It is defined as the cross-product $I \times I$ of the parameter spaces of the curves partitioned into *free space* and its complement. For a given $\varepsilon > 0$, the free space is defined as $F_\varepsilon(P,Q) = \{(r,t) : \|P(r) - Q(t)\| \leq \varepsilon\}$. For given ε, it holds that $\delta_\mathrm{F}(P,Q) \leq \varepsilon$ iff there exists a monotone path through the free space of $D_\varepsilon(P,Q)$ that connects bottom left and top right corner of the diagram. For polygonal curves defined by segment endpoints $p_0, p_1, \ldots, p_n, q_0, \ldots, q_m$, respectively, the free space diagram can be subdivided into $n \times m$ *cells* C_{ij}, where the cell boundaries have the same lengths $\|p_i - p_{i-1}\|$ and $\|q_j - q_{j-1}\|$ as the corresponding segments. It takes $O(nm)$ time to compute a monotone path verifying that $\delta_\mathrm{F}(P,Q) \leq \varepsilon$, see [1], implying that the complexity of the free space diagram determines the runtime of the standard algorithm for computing the Fréchet distance. For curves in 1D and curves of ply k the Fréchet distance can be computed in $O(nk \log n)$ time [5].

* Partially supported by the National Science Foundation, grant CCF-2107434.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

It is known that the Fréchet distance of curves in 2D cannot be computed in subquadratic time unless SETH fails [4], but it is possible to exploit the decreased complexity of the free space diagram to obtain faster algorithms for some curve classes [2, 9]. Some variants of the Fréchet distance are even NP-hard to decide [6, 11], and the reductions build specific instances of free space diagrams. To increase the understanding of these diagrams, we study the following problem:

Realizability in 1D. Given a diagram D_ε of size $n \times m$ and a parameter $\varepsilon > 0$, can we find 1-dimensional curves P and Q such that $D_\varepsilon = D_\varepsilon(P, Q)$?

Further notation. We define a polygonal curve $P \colon I \to \mathbb{R}$ by vertices $p_0, \dots, p_n \in \mathbb{R}$, and call the line segment connecting consecutive vertices $s_i^P = \overline{p_{i-1}p_i}$. If two consecutive segments s_i^P, s_{i+1}^P have different orientations (the segments are placed on top of each other), we say the curve *folds* at the common *folding vertex* p_i. A cell C_{ij} is called *empty* (or *gray*) if $C_{ij} \cap F_\varepsilon = \emptyset$, *full* (or *white*) if $C_{ij} \cap F_\varepsilon = C_{ij}$, and *partially full* if $\emptyset \neq C_{ij} \cap F_\varepsilon \neq C_{ij}$. An empty cell corresponds to a pair of non-overlapping segments whose endpoints have pairwise distances $> \varepsilon$; a full cell stems from segments where any interval of length ε centered at an arbitrary point of either segment fully contains the other segment. Partially full cells correspond to segments where at least one endpoint has distance $< \varepsilon$ to some point of the other segment. The free space $F_\varepsilon(s_i^P, s_j^Q)$ in a cell is a slab, i.e., the space between two diagonal lines tilted by $\pm 45°$, cropped at cell boundaries, see Figure 1. The horizontal and vertical lines bounding cells C_{ij} in the diagram are denoted as *grid lines.*

Note that free space (diagram) is a term defined by the corresponding curves. Thus we call the considered diagrams *input* or *given diagrams* D_ε and ask whether there exist curves P and Q such that $D_\varepsilon = D_\varepsilon(P, Q)$, moreover *white space* in D_ε equals free space in $D_\varepsilon(P, Q)$.

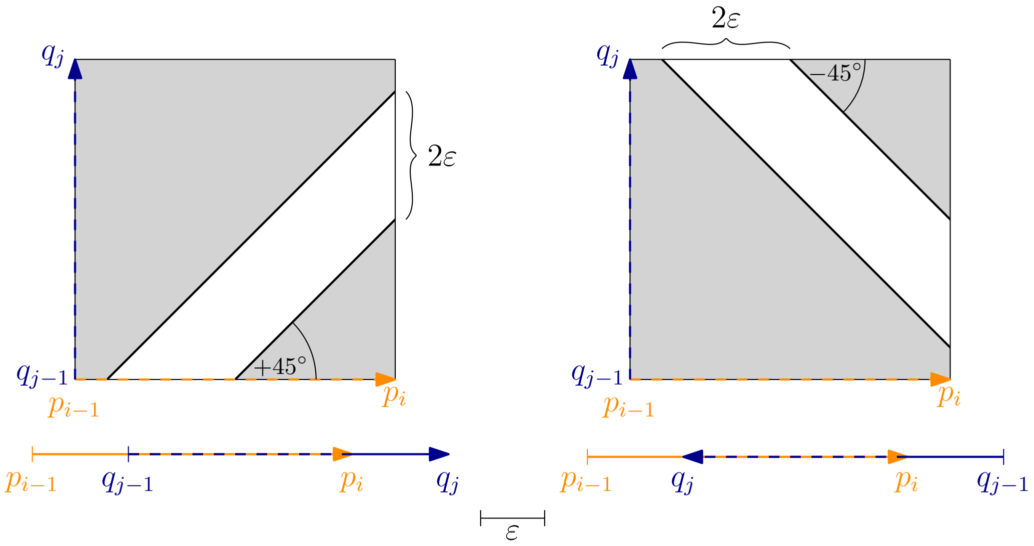


Figure 1 Given a free space in white and its complement in grey, partially full cells C_{ij} corresponding to segment s_j^Q, oriented in the same or in opposite direction as fixed segment s_i^P.

Next, we prove NP-hardness in Section 2, and give our FPT-algorithm in Section 3.

2 Hardness of Realizability for general diagrams

We prove that realizability in 1D is weakly NP-hard by reducing from the Partition problem.

Partition problem. Given a set of positive integers $A = \{a_1, \dots, a_n\}$, decide whether there exist two sets A_1, A_2, such that $\sum_{a_i \in A_1} a_i = \sum_{a_j \in A_2} a_j$, where $A_1 \cap A_2 = \emptyset$ and $A_1 \cup A_2 = A$.

▶ **Theorem 2.1.** *It is weakly NP-complete to decide whether a given diagram is realizable through curves in 1D.*

Proof. Given partition instance $\{a_1, \dots, a_n\}$, we set $S = \sum_{i=1}^{n} a_i$. We construct an input diagram of size $(n+2) \times 1$, see Figure 2, and simplify our notation of cells $C_{i1} = C_i$. Realizing this constructed diagram through curves P, defined by endpoints $p_0, \dots p_{n+2}$, and Q, consisting of a single segment $s^Q = \overline{q_0 q_1}$, has to correspond to partitioning our integers into two sets of equal "weight" $S/2$. The constructed diagram is a strip of height $|s^Q|$, which we set to 1. The total width of our diagram is set to $2(1+S)+S$: Each cell width corresponds to the length of a segment in P, and we choose the length of a segment s_i^P to be $|a_{i-1}|$, for $i = 2, \dots, n+1$. Segments s_1^P and s_{n+2}^P both have length $1 + \sum_{i=0}^{n} |a_i|$, and we set $\varepsilon = 1$.

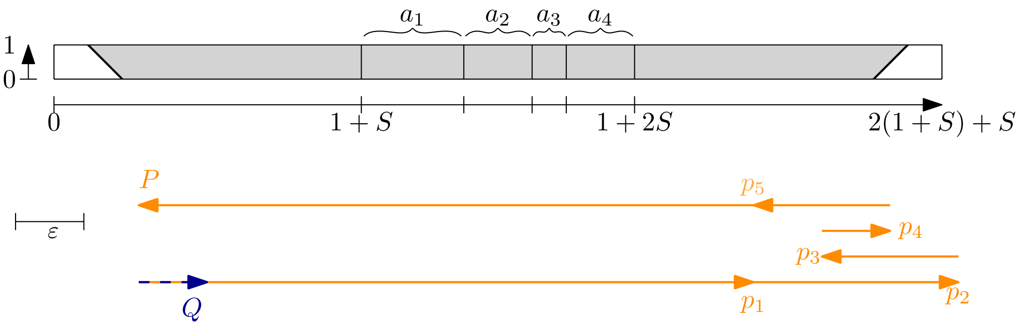


Figure 2 Reduction for partition instance $\{a_1, \dots, a_4\} = \{3, 2, 1, 2\}$. The corresponding segments are scaled by factor 2 and placed parallel instead of on top of each other to increase readability.

Now, the first and last cell of our constructed diagram are set to be partially full, the bottom left and top right corner being contained in white space. All intermediate cells $C_2, \dots C_{n+1}$ are empty. For fixed position of s^Q, we construct the first and last cell such that segments s_1^P and s_{n+2}^P need to be aligned with s^Q; namely, we force $q_0 = p_0 = p_{n+2}$. Consequently, $p_1 = p_{n+1}$. For the remaining segments $s_2^P, \dots, s_{n+1}^P$, we compare their orientation with placing the corresponding integer in either of the two sets A_1, A_2. Starting with s_2^P, we can decide to place it on top of its predecessor, such that $\|p_2 - p_0\| = \|p_1 - p_0\| - a_1$, or facing the same direction, such that $\|p_2 - p_0\| = \|p_1 - p_0\| + a_1$. If we choose the first option, we say s_1^Q is *oriented to the left*, otherwise it is *oriented to the right.*

Assume we are given an instance of the partition problem. The constructed strip is realizable if and only if we can place the curve P such that points p_1 and p_{n+1} coincide. This is the case if and only if the total length of segments oriented to the right equals the total length of segments oriented to the left. Thus, the information on orientations of $s_2^P, \dots s_{n+1}^P$ directly encodes a partition of integers $a_1, \dots, a_n$ into sets A_1 and A_2. We conclude

$$\sum_{a_i \in A_1} a_i = \sum_{a_j \in A_2} a_j = \frac{S}{2} \iff p_1 = p_{n+1}.$$

This proves NP-hardness. The problem lies in NP since the realization of a polygonal curve with n segments can be described by a bit sequence of length $n-1$, specifying for each vertex whether the incident segments have the same orientation. ◀

3 Deciding realizability for 1-dimensional curves

Assume that we are given an input diagram where the lengths of cell boundaries as well as intersection points of white space with grid lines are part of the input.

In comparison to curves in 2D [7], we observe that a free space diagram corresponding to curves in 1D has limited "configurations". We still face empty, full or partially full cells, where white space is bounded by the cell boundaries and parallel line segments tilted by ±45°, see [5, 10]. Additionally, we state the following:

▶ **Lemma 3.1.** *For each partially full cell C_{ij} corresponding to a pair of segments s_i^P, s_j^Q in $1D$, there exists at least one point x on the intersection of the free space boundary and the cell boundary. Its position fixes the distance of two endpoints $(p, q) \subset \{p_i, p_{i+1}\} \times \{q_j, q_{j+1}\}$, and thus allows to fully determine the relative positions of s_i^P and s_j^Q.*

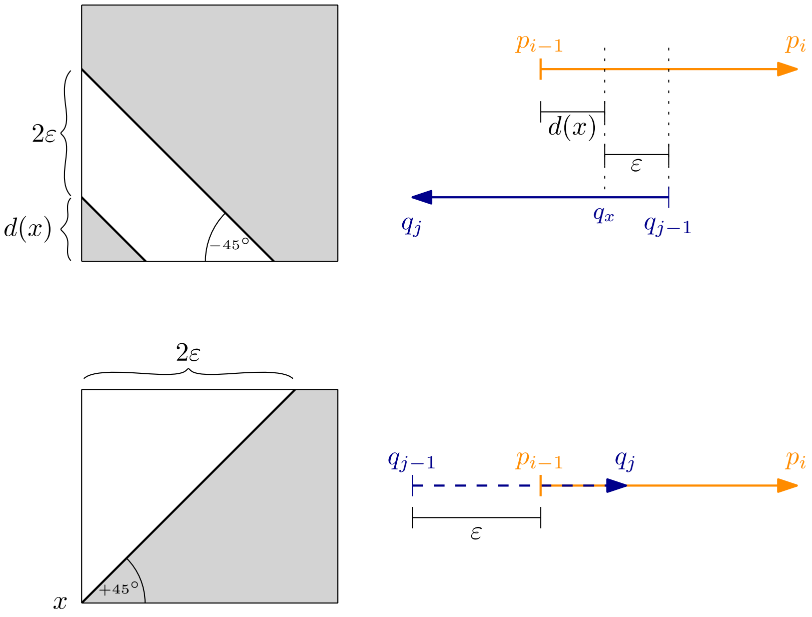


Figure 3 Placing s_j^Q for fixed s_i^P to realize a corresponding partially full cell.

Proof. As both segments are placed on the real line, at least one endpoint lies with ε distance of a point of the other segment. W.l.o.g., we fix the position of s_i^P such that $p_{i-1} = 0$, $p_i = |s_i^P|$, where $|s_i^P|$ denotes the segment's length. Let x lie on the left boundary of C_{ij}, which corresponds to $p_{i-1} \times s_j^Q$, and we call $d(x)$ the distance between x and the bottom left corner $p_{i-1} \times q_{j-1}$, see Figure 3. For $d(x) = 0$, we have that $\|p_{i-1} - q_{j-1}\| = \varepsilon$, for $d(x) = |s_j^Q|$ it holds that $\|p_{i-1} - q_j\| = \varepsilon$. In both cases, the orientation of s_j^Q depends on whether C_{ij} contains a free space region. Iff this is the case both segments have the same orientation. If $0 < d(x) < |s_j^Q|$ there could be one such intersection point, or two at distance 2ε, see Figure 1. Assuming x denotes the lower one, i.e., the interval between bottom left corner and x is not contained in free space, it holds that $\|p_{i-1} - q_{j-1}\| = d(x) + \varepsilon$, because the point $q_x \in s_j^Q$ at distance $d(x)$ from q_{j-1} has distance exactly ε to p_{i-1}. The mirrored case holds for x denoting the upper intersection point and endpoints p_{i-1}, q_j. The orientation of s_j^Q depends on the angle of the free space boundary within the cell; for +45°, both segments s_i^P, s_j^Q face in the same direction. ◀

▶ **Observation 3.2.** *Free space diagrams of curves in 1D are in some sense symmetrical, as described in [5]: Consider an endpoint $p_i \in P$ at which the curve folds, and some point $q \in Q$. We choose points $p \in s_i^P, p' \in s_{i+1}^P$ that are equidistant to p_i, so $p = p' \in \mathbb{R}$. Now $\|p - q\| \leq \varepsilon$ holds iff $\|p' - q\| \leq \varepsilon$. Thus, the strip to the right of the grid line $p_i \times Q$ is a reflection of the strip to the left of that line. For consecutive partially full cells, this implies that a curve folds at the common endpoint p_i iff the incident portions of free space have alternating slopes, see Figure 4.*

We borrow some definitions from computational origami, giving intuitive descriptions. We refer to [8] for formal definitions. The *crease pattern* $C(D_\varepsilon)$ of a diagram D_ε is the crease pattern obtained by considering grid lines that correspond to folding vertices as creases. The *folded state* of a crease pattern $C(D_\varepsilon)$ is a continuous function that maps each face isometrically and reflects adjacent faces. From Observation 3.2, we conclude

▶ **Corollary 3.3.** *A given diagram D_ε is realizable iff there is an assignment of the grid lines to* {`fold`, `straight`}*, such that overlapping white space aligns in the folded state $C(D_\varepsilon)$.*

Algorithm. We use Corollary 3.3 to obtain an algorithm for the 1D realizability problem. The algorithm runs in exponential time for general inputs; more precisely, it runs in $O(mn2^k)$ time where k is the number of rows of D_ε that do not intersect the boundary of the white space, i.e., the number of vertical or horizontal strip "gaps" (completely gray or completely white) in the diagram. Intuitively, gaps correspond to segments of one curve that are either too far or too close to the other curve, and do not offer us direct information about the placement of the curves. In particular, if there is a white gap in D_ε (e.g. a row of only full cells) then the diameter of one of the curves is smaller than 2ε. For inputs where $k \in O(\log(mn))$, the algorithm thus runs in polynomial time in the size of the diagram.

Our algorithm is inspired by the algorithm for *simple-foldability in 1D* [3] by Arkin et al., which asks whether a 1D crease pattern can be folded through a sequence of simple folds (±180° rotations of a portion of the paper). This paper provides a linear-time greedy algorithm to constructively decide whether a given crease pattern admits a sequence of simple folds. Although it is meant to be applied to crease pattern where each crease has a *mountain/valley* assignment (whether the crease should be folded by +180° or −180°), for our problem this assignment is irrelevant. Without a mountain/valley assignment, every crease pattern is simple-foldable. We use Arkin et al.'s algorithm for the properties that it maintains during a linear-sized sequence of operations (simple folds). The algorithm identifies one of two sufficient operations that can be greedily applied: (i) an *end fold* which is a simple fold applied to either the first or last crease; and (ii) a *crimp* which folds through two consecutive creases. They show that these operations are *safe* if

(⋆) the resulting folded state after each operation does not cause a portion of the paper without creases to overlap with a not yet folded crease.

After applying the operation, they reduce the problem to a smaller crease pattern obtained by "gluing" the overlapping layers. They show that if the crease pattern is not trivial (no creases), there is always a safe operation, which they can find in $O(1)$ time with $O(n)$-time preprocessing. We call any operation (end fold or crimp) *valid* if the resulting folded state

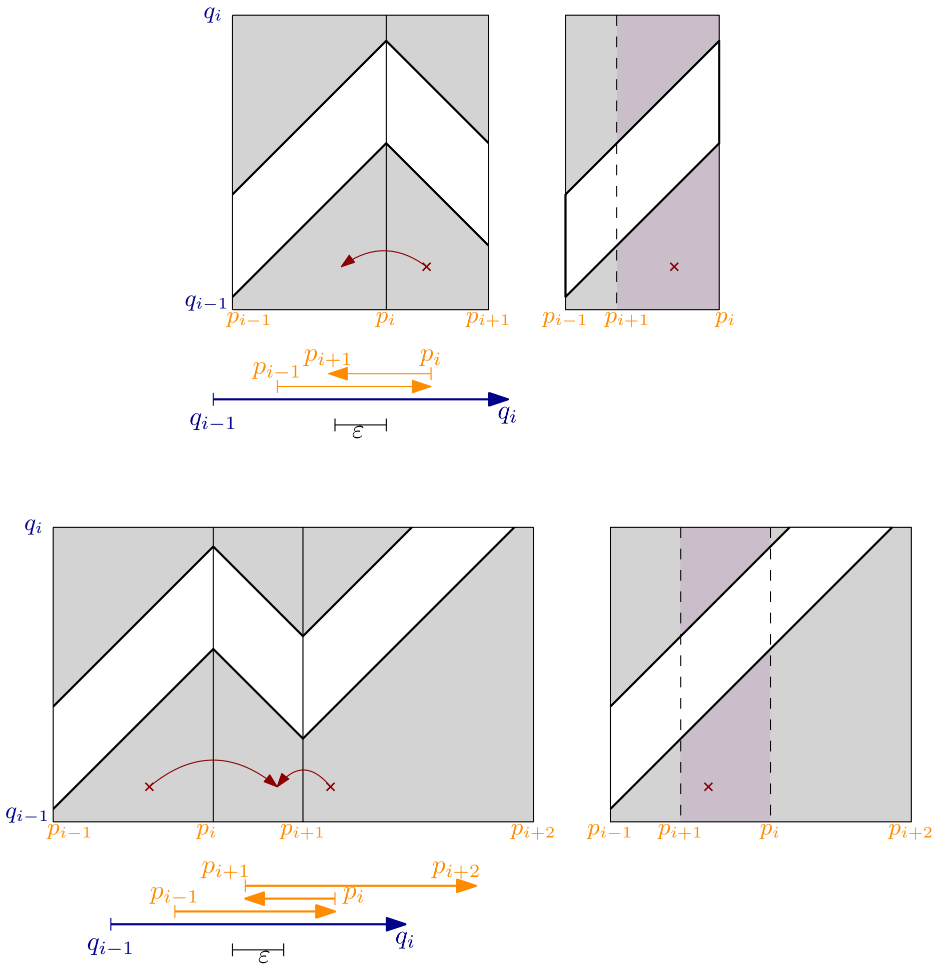


Figure 4 An end fold and a crimp of two diagrams along the grid lines of folding vertices.

aligns the white space patterns, see Figure 4. We use the above mentioned algorithm to efficiently check if a given crease pattern induces a realizable input space diagram.

▶ **Theorem 3.4.** *Given an $m \times n$ diagram D_ε, we can find 1-dimensional curves P and Q such that $D_\varepsilon = D_\varepsilon(P, Q)$, if they exist, in $O(mn2^k)$ time, where k is the total number of (vertical and horizontal) grid lines of D_ε that do not intersect the white space.*

Proof. We describe a constructive algorithm. The main idea is to test every possible corresponding crease pattern that is compatible to D_ε. Note that if a grid line intersects the white space, we can determine if it corresponds to a folding vertex or not as follows. If two adjacent cells align the white space after a reflection through the grid line, then the grid line should be assigned to `fold`; otherwise, we assign it to `straight`. If there is an inconsistency of the assignment given by different pairs of adjacent cells incident to a grid line, the instance is not realizable, and we return "no". For the remaining k grid lines that do not intersect the white space, we try all possible assignments to $\{$`fold`, `straight`$\}$. We delete all grid lines that are assigned `straight`, merging pairs of adjacent cells through the deleted line. That defines a crease pattern and allows us to check realizability using Corollary 3.3.

It remains to show how we efficiently check whether a given crease pattern satisfies Corollary 3.3. Note that we could also pay an extra linear factor using brute force to check for every pair of overlapping cells of D_ε whether their white space aligns, leading to an $O(m^2n^2)$ time algorithm. We use the simple-foldability algorithm [3] to obtain an $O(mn)$-time algorithm as follows. We fold one dimension at a time. W.l.o.g., we focus on the horizontal dimension, corresponding to P, which contains $O(n)$ creases. Identify a safe operation and apply the corresponding fold(s). Note that each operation causes at most three layers to overlap. Recall that we merge these layers into a single layer by "gluing", and apply induction. It suffices to check the alignment of the white space in the overlapping layers. By property ($\star$), the number of cells in these layers is $O(m)$, so the check can be performed in $O(m)$ time. In future operations, if the white space of a merged cell and the white space of another cell align, the white space of all original overlapping cells align since alignment is transitive. This proves the induction step. After $O(m)$ operations, we obtain a single segment in the horizontal dimension, and we can apply the same algorithm for the vertical dimension. The runtime of the check step is then $O(mn)$, proving our claim.

Finally we address the edge cases when the input diagram is completely empty or completely full. These cases trivially satisfy Corollary 3.3. In case the diagram is completely empty, we just place the curves sufficiently far apart from each other (farther than ε). In the latter case, we can compute the intersection of the ε-neighborhoods of every segment of one curve, and check whether we can place the other curve in this intersection. Recall that with a fixed crease pattern, the diameters of the curves are deterministically defined. ◀

▶ **Corollary 3.5.** *If all vertices are folding vertices then it takes $O(mn)$ time to compute respective curves in 1D and verify whether these curves correspond to the input diagram.*

4 Conclusion

Buchin, Ryvkin and Wenk [7] give results for the realizability of free space diagrams for curves in 2D. This paper proves NP-hardness and gives an FPT-algorithm for curves in 1D. We note that there are no known results for curves in higher dimensions.

References

1 Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *Internat. J. Comput. Geom. Appl.*, 5(1-2):75–91, 1995.

2 Helmut Alt, Christian Knauer, and Carola Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2004.

3 Esther M. Arkin, Michael A. Bender, Erik D. Demaine, Martin L. Demaine, Joseph S.B. Mitchell, Saurabh Sethia, and Steven S. Skiena. When can you fold a map? *Computational Geometry*, 29(1):23–46, 2004. Special Issue on the 10th Fall Workshop on Computational Geometry, SUNY at Stony Brook. URL: https://www.sciencedirect.com/science/article/pii/S0925772104000483.

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

5 Kevin Buchin, Jinhee Chun, Maarten Löffler, Aleksandar Markovic, Wouter Meulemans, Yoshio Okamoto, and Taichi Shiitada. Folding free-space diagrams: Computing the Fréchet distance between 1-dimensional curves (multimedia contribution). In *33rd International Symposium on Computational Geometry, (SoCG'17)*, volume 77 of *LIPIcs*, pages 64:1–64:5. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. URL: https://doi.org/10.4230/LIPIcs.SoCG.2017.64.

6 Maike Buchin, Anne Driemel, and Bettina Speckmann. Computing the Fréchet distance with shortcuts is NP-hard. In *Proceedings of the Thirtieth Annual Symposium on Computational Geometry*, page 367–376, 2014.

7 Maike Buchin, Leonie Ryvkin, and Carola Wenk. On the realizability of free space diagrams. In *37th European Workshop on Computational Geometry (EuroCG)*, pages 377–383, 2021. URL: `http://eurocg21.spbu.ru/wp-content/uploads/2021/04/proceedings.pdf`.

8 Erik D. Demaine and Joseph O'Rourke. *Geometric Folding Algorithms: Linkages, Origami, Polyhedra.* Cambridge University Press, 2008.

9 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete Comput. Geom.*, 48(1):94–127, 2012.

10 Günter Rote. Lexicographic Fréchet matchings. In *30th European Workshop on Computational Geometry*, 2014.

11 Leonie Ryvkin. *On distance measures for polygonal curves bridging between Hausdorff and Fréchet distance.* doctoralthesis, Ruhr-Universität Bochum, Universitätsbibliothek, 2021. `doi:10.13154/294-8275`.

Querying the Hausdorff Distance of a Line Segment

Frank Staals[1], **Jérôme Urhausen**[1], **and Jordi L. Vermeulen**[1]

[1] **Utrecht University, the Netherlands**
`{F.Staals, J.E.Urhausen, J.L.Vermeulen}@uu.nl`

Abstract

We consider the problem of preprocessing a set of points or segments R such that we can quickly determine the Hausdorff distance between a query segment and R. For $|R| = n$ and parameters $k \in [1 \dots n]$, $\delta \in (0, 1)$ and $\varepsilon > 0$, in expected $O(nk^{1+\varepsilon} + n^{1+\delta})$ time we can store R in a data structure of size $O(nk^{1+\varepsilon} + n^{1+\delta})$ such that given a query line segment b we can compute the Hausdorff distance $\mathcal{D}(b, R)$ in $O((n/k) \log k + \log^3 k + 2^{1/\delta} \log n)$ time.

1 Introduction

The Hausdorff distance is one of the most well-known distance measures between (sets of) geometric objects. Given a set R of "red" objects, and a set B of "blue" objects, their Hausdorff distance $\mathcal{D}(R, B) = \max\{\overrightarrow{\mathcal{D}}(B, R), \overrightarrow{\mathcal{D}}(R, B)\}$ is the maximum of the two *directed* Hausdorff distances defined as

$$\overrightarrow{\mathcal{D}}(B, R) = \max_{b \in \bigcup B} \min_{r \in \bigcup R} d(b, r),$$

where $d(b, r)$ denotes the Euclidean distance between two points b and r. We are interested in computing the Hausdorff distance efficiently, in particular for objects in the plane. In case R and B are both sets of points in $\mathbb{R}^2$, of sizes n and m, respectively, it is easy to compute the (directed) Hausdorff distance in $O((n + m) \log(n + m))$ time. We simply build the Voronoi diagram of one set, and query it with the other. In case R and B are sets of disjoint line segments (in $\mathbb{R}^2$) the problem can be solved in the same time [1]. When R and B are convex polygons, their Hausdorff distance can even be computed in linear time [2].

The above algorithms are very good if B and R have similar sizes. However, when we wish to compute the Hausdorff distance between one "large" set, say R, and many much smaller sets $B_1, \dots, B_k$, we wish to avoid the costly linear dependence on n for every B_i. That is, we wish to build a data structure on R that can be efficiently queried for the Hausdorff distance between R and some query object B. This setting appears naturally in a range of applications, for example when querying a shape database (e.g. find all hand written characters similar to some low complexity "sketch" of a character), trajectory clustering (in which a cluster is represented by a low complexity representative) [8], or polyline simplification (test if some candidate shortcut segment is "good enough") [4, 5, 10]. Furthermore, we are actually interested in maintaining the Hausdorff distance between two sets of line segments that are subject to updates. Efficiently computing the Hausdorff distance between subsets of varying sizes is one of the (many) challenging subproblems that arise in this setting.

Problem Statement and Results. We focus on the cases where R is a set of n disjoint line segments in $\mathbb{R}^2$, and B is a single line segment b. We develop a data structure storing R that can efficiently be queried for the Hausdorff distance between b and R. Computing

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

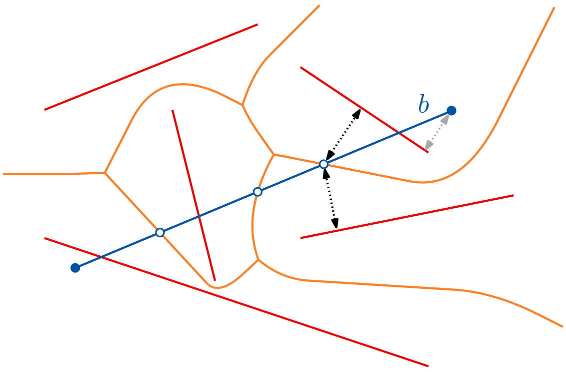


Figure 1 The directed Hausdorff distance $\overrightarrow{\mathcal{D}}(b, R)$ is realized either at an endpoint of b, or at an intersection of b with an edge of the Voronoi diagram of R.

$\overrightarrow{\mathcal{D}}(R, b)$ turns out to be relatively straightforward, as the maximum distance from R to b is realized by an endpoint of a segment in R. Using known results, e.g. furthest point Voronoi diagrams this then yields an $O(n^{1+\delta})$ space $O(2^{1/\delta} \log n)$ time solution, for some parameter $\delta \in (0, 1)$. The hard part is in computing $\overrightarrow{\mathcal{D}}(b, R)$. In Section 3, we first present an $O(n^{2+\varepsilon})$ size data structure that realizes $O(\log^3 n)$ query time when R is a set of n points. Here, and throughout the rest of the paper, $\varepsilon > 0$ is an arbitrarily small constant. We then generalize this solution to the case of line segments in Section 4. This thus gives us an $O(n^{2+\varepsilon})$ size data structure to query the directed Hausdorff distance in $O(\log^3 n)$ time. Finally, in Section 5 we show how, for any parameter $k \in [1 \ldots n]$, we can decrease the space usage to $O(nk^{1+\varepsilon})$ at the cost of increasing the query time to roughly $O(n/k)$.

2 Preliminaries

For a set of points or segments R in $\mathbb{R}^2$, let Vor_R be their Voronoi Diagram. We consider Vor_R as a set of edges. Each edge is either a segment or a parabolic arc. Each edge $e \in \mathrm{Vor}_R$ is equally close to two objects p_e and q_e of R that induce e. For a line segment b, the directed Hausdorff distance $\overrightarrow{\mathcal{D}}(b, R)$ is realized either by an endpoint of b, or a point on the intersection $b \cap e$, for an edge $e \in \mathrm{Vor}_R$ [1], see Figure 1. That is, $\overrightarrow{\mathcal{D}}(b, R) = \max\{\textsc{EndDi}(b, R), \textsc{IntDi}(b, R)\}$ with the following definitions: for $b = \overline{b_1 b_2}$, $\textsc{EndDi}(b, R) = \max_{i \in \{1,2\}} \overrightarrow{\mathcal{D}}(b_i, R) = \max_{i \in \{1,2\}} \min_{r \in R} d(b_i, r)$ is the *endpoint-distance* and $\textsc{IntDi}(b, R) = \max_{e \in \mathrm{Vor}_R} \{d(u, p_e) \mid \{u\} = b \cap e\}$ is the *intersection-distance*. We can easily compute the endpoint-distance using two $O(\log n)$ time nearest neighbor queries on R. Hence, our main task is to develop a data structure that allows us to efficiently compute $\textsc{IntDi}(b, R)$.

For a point $p = (p_x, p_y) \in \mathbb{R}^2$, let $p^* \equiv y = p_x x - p_y$ be the line dual to p. And for a line $\ell \equiv y = sx + t$, let $\ell^* = (s, -t)$. The set of lines dual to the points on a line ℓ all intersect in ℓ^*. For a segment $b = \overline{uv}$, let b^* be the wedge that is the set of points between the lines u^* and v^*, see Figure 2. For a set R of objects, we define $R^* = \{r^* \mid r \in R\}$. Note that for a line ℓ, a point p, and a segment b, we have $p \in \ell \iff \ell^* \in p^*$ and $\ell \cap b \neq \emptyset \iff \ell^* \in b^*$.

3 A Datastructure for Red Points

We first explore how to determine $\textsc{IntDi}(b, R)$, when R is a set of points. We start with the case where b is a line, and then extend to the case where b is a line segment.

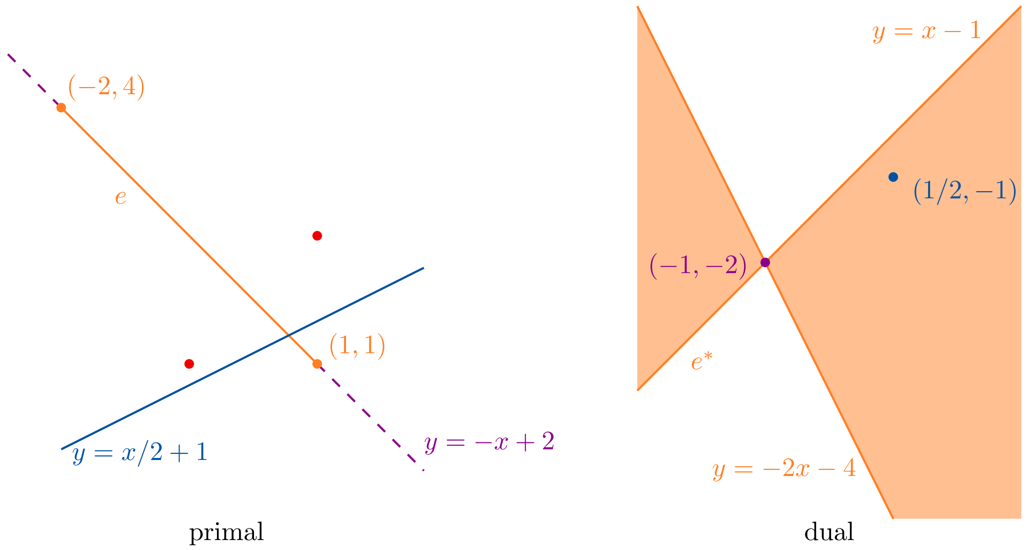


Figure 2 The Voronoi edge between two red points and the corresponding dual wedge. The blue line intersects the edge and therefore its dual point is within the wedge.

3.1 Querying with a Line

We want to store R so that we can efficiently compute $\textsc{IntDi}(b, R)$ for a query line b. For each edge e in the Voronoi Diagram Vor_R, we lift the wedge e^* to the surface $e^\#$ such that a point $\ell^* \in e^*$ is lifted to a height equal to the squared distance between the red Voronoi site p_e and the intersection of ℓ and e. See Figure 3. We square to simplify the derivations. Formally, $e^\# = \{(s, t, d^2(p_e, \ell \cap e)) \mid \ell^* := (s,t) \in e^*\}$. For a set E, we set $E^\# = \{e^\# \mid e \in E\}$. As a result, for a wedge e^* and a line p^* through the center of the wedge, all points ℓ^* on p^* will be lifted to the same height, meaning $e^\#$ is a ruled surface, as shown in Figure 4. As we want to determine the Hausdorff distance, for each line ℓ, we are interested in $\max\{z \mid \exists e \in \mathrm{Vor}_R : (s,t) = \ell^* \wedge (s,t,z) \in e^\#\}$, that is, we care about the upper envelope of these surfaces. Using [9] to be able to quickly query the upper envelope, we get:

▶ **Lemma 3.1.** *Let R be a set of n points in $\mathbb{R}^2$. In $O(n^{2+\varepsilon})$ time we can build an $O(n^{2+\varepsilon})$ size data structure that can compute $\textsc{IntDi}(b, R)$ for a query line b in $O(\log n)$ time.*

Lemma 3.1 is illustrated in Figure 5. We also state the following corollary:

▶ **Corollary 3.2.** *Let $E \subseteq \mathrm{Vor}_R$ be a set of k Voronoi edges. In $O(k^{2+\varepsilon})$ time we can build an $O(k^{2+\varepsilon})$ size data structure that can compute $\max\{z \mid \exists e \in E : (s,t) = \ell^* \wedge (s,t,z) \in e^\#\}$ for a query segment b intersecting all edges E with supporting line ℓ in $O(\log k)$ time.*

3.2 Querying with a Line Segment

We now extend the data structure to support queries with a line segment, see Figure 6.

For a set of n hyperplanes H in $\mathbb{R}^d$ and $r \in [1 \ldots n]$, a $1/r$-*cutting* of H is a partition of $\mathbb{R}^d$ into cells with disjoint interiors, each of which is intersected by at most n/r hyperplanes from H [6]. The *conflict list* of a cell ∇ is the set of the hyperplanes intersecting the interior of ∇.

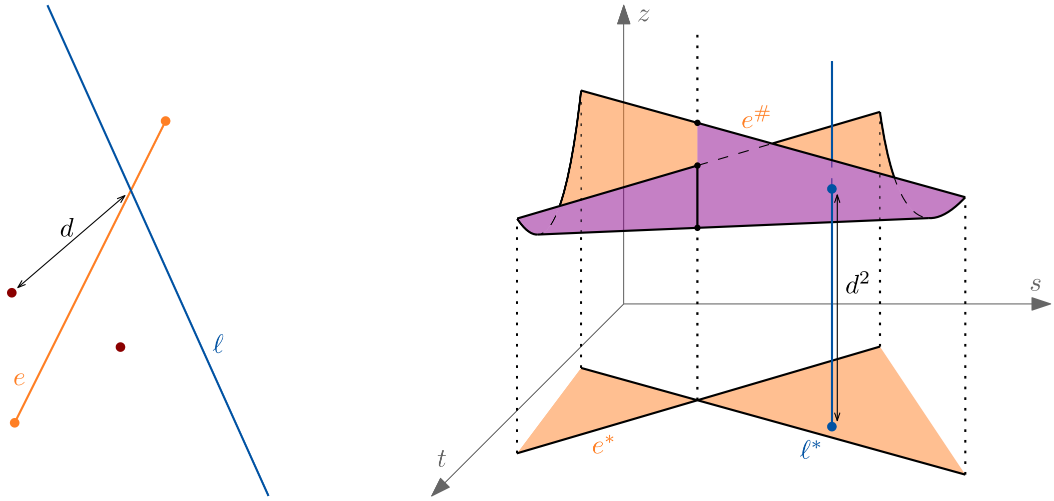


Figure 3 The Voronoi edge e, its corresponding dual e^* and its corresponding lifted surface $e^{\#}$. For a line ℓ intersecting e, the squared distance between the intersection $\ell \cap e$ and a corresponding red point p_e equals the height at which a vertical line above ℓ^* punctures $e^{\#}$.

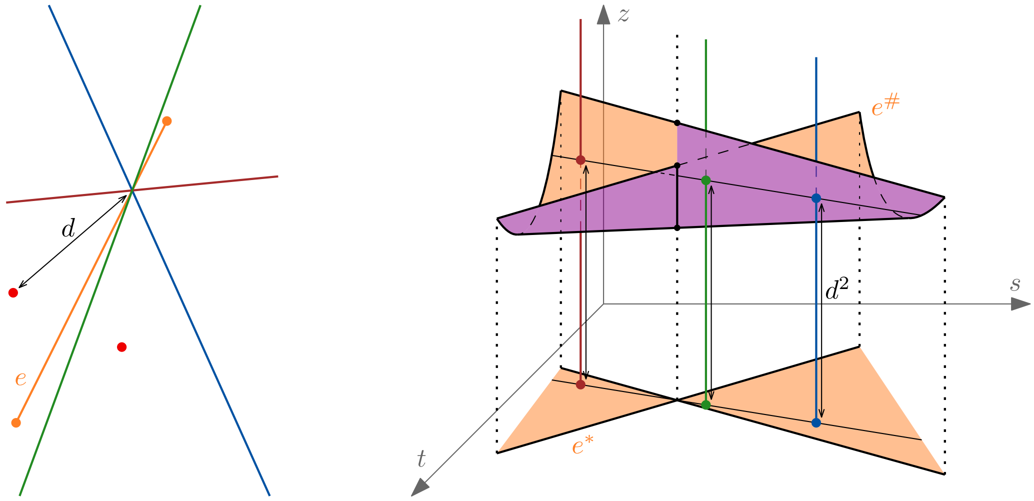


Figure 4 The points dual to lines intersecting a Voronoi edge e in the same point, are aligned and are lifted to the same height in $e^{\#}$.

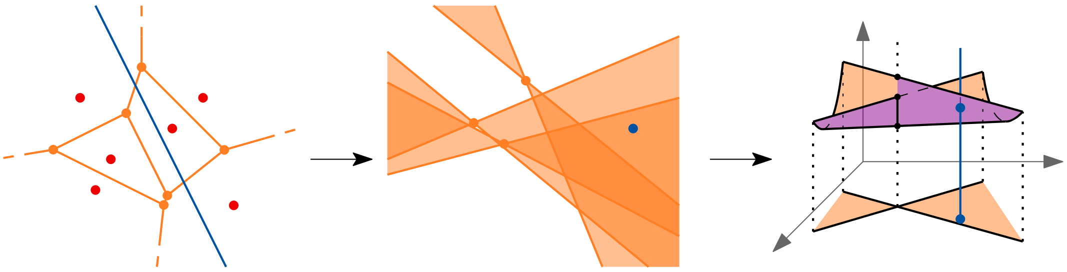


Figure 5 The overview of Section 3.1. To query the intersection distance for a line ℓ, we query the upper envelope above the point ℓ^*.

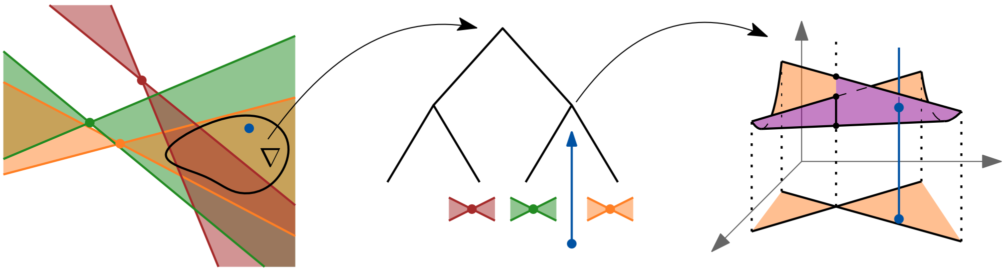


Figure 6 The overview of the datastructure of Section 3.2. Each cell of the cutting stores a balanced binary search tree whose internal nodes store an upper envelope.

Let $\mathcal{A}$ be the arrangement induced by the wedges Vor_R^*. We can cut the $O(n^2)$ faces of $\mathcal{A}$ into $O(r^2)$ cells where each cell is intersected by at most n/r boundaries of wedges in $O(nr)$ time [6]. This also computes the conflict list for each cell. Let $E_\nabla^{\mathrm{int}} = \{e \in \mathrm{Vor}_R \mid \emptyset \subsetneq e^* \cap \nabla \subsetneq \nabla\}$ be the edges whose dual wedges have boundaries in the conflict list of ∇.

For each cell ∇, we compute the following: let $E_\nabla^{\mathrm{elem}} \subseteq \mathrm{Vor}_R$ be the set of edges e whose dual wedges contain ∇, that is $\nabla \subseteq e^*$. We sort the edges E_∇^{elem} by the order in which a line ℓ with $\ell^* \in \nabla$ intersects them, and build a balanced binary search tree T_∇ on E_∇^{elem} whose leaves each contain one edge, and whose inner nodes each contain the edges contained in their children. For each node with set of edges E we calculate the upper envelope of $E^{\#}$, as stated in Corollary 3.2. The tree with the upper envelope at each node uses $O(n^{2+\varepsilon})$ space and can be constructed in $O(n^{2+\varepsilon})$ time.

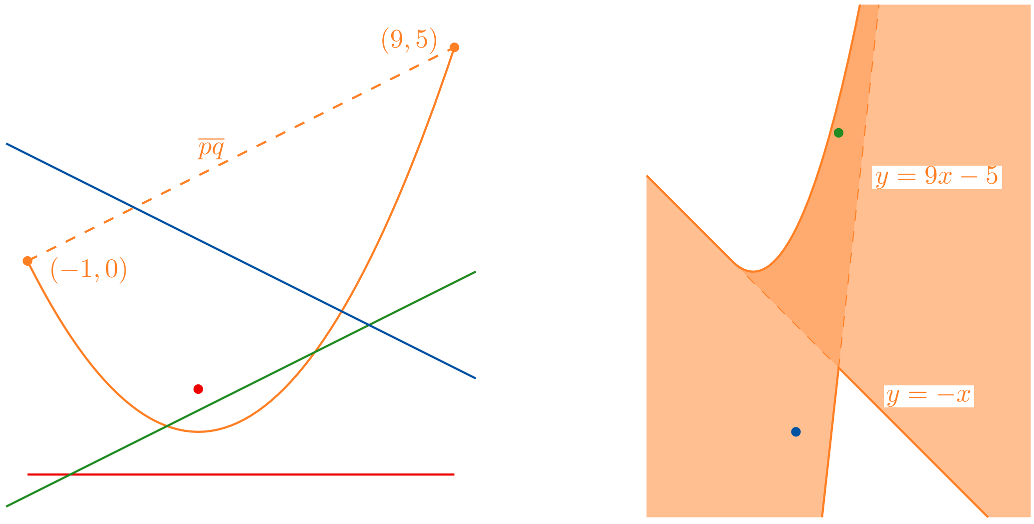
Then we recurse for each cell. That is, for each cell, if it is intersected by $k > 0$ lines, we again cut it into $O(r^2)$ cells where each cell is intersected by at most k/r lines. We again determine E_∇^{int} for each cell ∇ using the conflict list. Also, for each cell ∇ with parent cell ∇', we have $E_\nabla^{\mathrm{elem}} = \{e \in E_{\nabla'}^{\mathrm{int}} \mid \nabla \subseteq e^*\}$. We compute the above mentioned balanced binary search tree T_∇ and upper envelope datastructure on the edges E_∇^{elem}. In the end, the space needed is $S(n) = cr^2(n^{2+\varepsilon} + S(n/r))$, for some constant c. For a sufficiently large constant r, we get $S(n) \in O(n^{2+\varepsilon})$. The same holds for the construction time.

When querying with a line segment b that has supporting line ℓ, we proceed as follows. Determine the cell ∇ containing ℓ^*. Recurse within ∇ to find the set C of all cells containing ℓ^*. Within each tree T_∇ with $\nabla \in C$, we compute the consecutive range of edges intersecting b. This gives us a set of $O(\log^2 n)$ nodes that together represent all edges from Vor_R intersected by b. For each of those nodes we query the height of the upper envelope at $(s,t) = \ell^*$. The result of the query is the maximum over those heights. This query takes $O(\log^3 n)$ time and returns the intersection distance $\textsc{IntDi}(b, R)$. We finally make two point location queries in the Voronoi Diagram — one for each of the endpoints of b — to determine $\textsc{EndDi}(b, R)$.

▶ **Theorem 3.3.** *Let R be a set of n points in $\mathbb{R}^2$. In $O(n^{2+\varepsilon})$ time we can build an $O(n^{2+\varepsilon})$ size data structure that can compute $\overrightarrow{\mathcal{D}}(b, R)$ for a query line segment b in $O(\log^3 n)$ time.*

4 A Datastructure for Red Segments

Now we consider the problem when R is a set of segments instead of a set of points and we again query with a segment b. We create the same data structure as described above in Section 3.2. The main difference is that the edges of the Voronoi Diagram Vor_R may be parabolic arcs. In the following we explain the implication of this change on the datastructure. The main question concerns the shape of the dual e^* of a parabolic arc



primal

dual

Figure 7 A Voronoi edge e and the corresponding dual pseudo-wedge e^*. A line ℓ intersecting e once also intersects the segment $\overline{pq}$ between the endpoints of the parabolic arc and thus $\ell^* \in \overline{pq}^*$. A line intersecting e twice has a dual point between the wedge and a conic.

e. Furthermore we are interested in the shape of the lifted surface $e^\#$. To be precise, we define the *pseudo-wedge* e^* as the set of all points ℓ^*, where ℓ intersects the edge e, and $e^\# = \{(s, t, \max_{a \in \ell \cap e} d^2(a, p_e)) \mid \ell^* = (s,t) \in e^*\}$. See Figure 7 for an example. The following two lemmata determine the shape of e^*.

▶ **Lemma 4.1.** *The points dual to the tangents of a parabola form a nondegenerate conic.*

A conic is a curve described by an equation of the form $ax^2 + bxy + cy^2 + dx + ey + f = 0$, for some $a, b, c, d, e, f \in \mathbb{R}$. A conic is *nondegenerate* if it is a circle, ellipse, parabola, or hyperbola, that is, if it contains at least two points and no three collinear points.

▶ **Lemma 4.2.** *For an edge e, the pseudo-wedge e^* is an area bounded by three algebraic curves each of degree at most two.*

Sketch. Intuitively, a line ℓ whose dual is on the boundary of e^* is tangent to the convex hull of the parabolic arc e. Thus, ℓ either intersects one of the two endpoints of e or ℓ is tangent to e. The curve of the dual points of lines tangent to e^* is a conic by Lemma 4.1. Therefore, e^* is bounded by two lines and a conic. ◀

From Lemma 4.2 we deduce that the arrangement of pseudo-wedges E_R^* has complexity $O(n^2)$. Furthermore, the (vertical decomposition of the) arrangement of a random sample of $O(r^2 \log^2 r)$ such pseudo-wedges is expected to be an $1/r$-cutting of E_R^* [7]. It then follows we can compute an $1/r$-cutting of size $O(r^2)$ in expected $O(nr)$ time [3]. We now use the same approach as in Section 3.2; that is, we recursively build $1/r$-cuttings (for some constant r), each cell storing a binary search tree whose nodes store an upper envelope of $e^\#$ functions. As we show next, each $e^\#$ is an algebraic terrain of constant degree, which means their upper envelope can be stored for $O(\log n)$ point location queries using $O(n^{2+\varepsilon})$ space [9]. As before, we obtain an $O(n^{2+\varepsilon})$ size data structure that can be queried in $O(\log^3 n)$ time.

▶ **Lemma 4.3.** *The surface $e^\#$ is an algebraic terrain of degree at most eight or the maximum over two algebraic terrains each of degree at most eight.*

▶ **Theorem 4.4.** *Let R be a set of n disjoint line segments in $\mathbb{R}^2$. In expected $O(n^{2+\varepsilon})$ time we can store R in a data structure of size $O(n^{2+\varepsilon})$ such that given a query line segment b we can compute $\overrightarrow{\mathcal{D}}(b, R)$ in $O(\log^3 n)$ time.*

5 A Space-Time Tradeoff

Let $k \in [1 \dots n]$ be a parameter. We describe how to adapt our data structure to reduce the space used to $O(nk^{1+\varepsilon})$, at the cost of increasing the query time to $O((n/k)\,\text{polylog}\,k)$. The main idea is to partition $\mathbb{R}^2$ into $O(n/k)$ cells, such that in each cell ∇, there are only $O(k)$ points or line segments from R that contribute to Vor_R. We then build our data structure from Theorem 4.4 on each such set. Unfortunately, a query segment b may now intersect all $O(n/k)$ cells, which ultimately yields a query time of $O((n/k)\log k + \log^3 k)$. We additionally store R to also compute $\overrightarrow{\mathcal{D}}(R, b)$ efficiently.

▶ **Theorem 5.1.** *Let R be a set of n disjoint line segments in $\mathbb{R}^2$, and let $k \in [1 \dots n]$ and $\delta \in (0, 1)$ be two parameters. In expected $O(nk^{1+\varepsilon} + n^{1+\delta})$ time we can store R in a data structure of size $O(nk^{1+\varepsilon} + n^{1+\delta})$ such that given a query line segment b we can compute $\mathcal{D}(b, R)$ in $O((n/k)\log k + \log^3 k + 2^{1/\delta}\log n)$ time.*

Acknowledgments. Jérôme Urhausen and Jordi L. Vermeulen were supported by the Dutch Research Council (NWO) under project no. 612.001.651.

References

1 Helmut Alt, Peter Braß, Michael Godau, Christian Knauer, and Carola Wenk. Computing the Hausdorff distance of geometric patterns and shapes. In *Discrete & Computational Geometry*, pages 65–76. Springer, 2003.

2 Mikhail J. Atallah. A linear time algorithm for the Hausdorff distance between convex polygons. *Information Processing Letters*, 17(4):207–209, 1983. doi:10.1016/0020-0190(83)90042-X.

3 Mark de Berg and Otfried Schwarzkopf. Cuttings and applications. *International Journal of Computational Geometry & Applications*, 05(04):343–355, 1995. doi:10.1142/S0218195995000210.

4 Karl Bringmann and Bhaskar Ray Chaudhury. Polyline simplification has cubic complexity. In *35th International Symposium on Computational Geometry (SoCG)*, volume 129, pages 18:1–18:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.SoCG.2019.18.

5 Karl Bringmann and Bhaskar Ray Chaudhury. Polyline simplification has cubic complexity. *Journal of Computational Geometry*, 11(2):94–130, 2021.

6 Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9(2):145–158, 1993.

7 Sariel Har-Peled. *Geometric Approximation Algorithms*, volume 173. American Mathematical Society Boston, 2011.

8 Abhinandan Nath and Erin Taylor. k-median clustering under discrete Fréchet and Hausdorff distances. In *36th International Symposium on Computational Geometry (SoCG)*, volume 164, pages 58:1–58:15. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.SoCG.2020.58.

9 Micha Sharir. Almost tight upper bounds for lower envelopes in higher dimensions. *Discrete & Computational Geometry*, 12:327–345, 1994. doi:10.1007/BF02574384.

10 Marc van Kreveld, Maarten Löffler, and Lionov Wiratma. On optimal polyline simplification using the Hausdorff and Fréchet distance. *Journal of Computational Geometry*, 11(1):1–25, 2020.

Watchman Routes on Line Segments

Shahin John J S[1], Remi Raman[1], Subashini R[1], and Subhasree Methirumangalath[1]

1 **National Institute of Technology, Calicut**
`ivinjohn98@gmail.com,{remi_p170027cs,suba,subha}@nitc.ac.in`

Abstract

In this paper, we focus on the parameterized complexity of Watchman Route problem on arrangement of line Segments (WRS). We show that WRS is Fixed Parameter Tractable with respect to the parameter k being the number of faces in the connected arrangement of line segments in a plane. The proposed algorithm, which runs in time $O^*(5^{2k}3^k)$, transforms the connected arrangement of line segments $\mathcal{L}$ into a plane straight-line graph G, performs a series of preprocessing steps and uses a constraint propagation technique to find the optimal watchman route.

1 Introduction

The Watchman Route Problem (WRP) asks for a shortest closed path in the given polygon so that every point in the polygon is visible from at least one point on the path. The WRP was first introduced by Chin and Ntafos [2], as a variation of the art gallery problem. Dumitrescu and Toth [6] proved that WRP is NP-hard if the polygon contains holes. The following variants of WRP were considered in the literature [4, 5, 7, 8, 9, 10, 11]: for *simple polygons*, for *polygonal domains*, and for *arrangement of lines and line segments*. The WRP on an arrangement of line segments[1] in a plane (WRS) is a special case of the WRP on polygonal domains as one can consider a line segment as an alley or corridor of width zero. Xu [12] showed that the WRS is NP-hard. Later, Dumitrescu, Mitchell and Zylinski [5] proposed a simpler NP-hard proof for WRS. They added that the problem remains NP-hard even for axis-aligned line segments, and they presented an $O(log^3(n))$-approximation algorithm also.

Parameterized complexity offers a framework for solving NP-hard problems by measuring their running time in terms of one or more parameters, in addition to the input size. A problem with input size n, and a non-negative integer parameter k, is fixed-parameter tractable (FPT), if it can be solved by an algorithm that runs in O$(f(k)n^c)$-time or $O^*(f(k))$-time, where f is a computable function depending only on k, and c is a constant independent of k. We recommend interested readers to [3] for more details on the topic.

In this paper, we focus on the parameterized complexity of WRS, with the parameter k being the number of faces. The parameterized version of the problem is as follows.

▶ **Definition 1.1.** *k*-**WRS (*k*-Watchman route problem on line segments)**
Input: A connected arrangement of line segments $\mathcal{L}$ with k faces.
Parameter: The integer k
Output: A minimum length closed walk contained in the union of the line segments in $\mathcal{L}$ such that every line segment is visited (intersected) by the walk.

[1] Consecutive coinciding line segments are considered as degenerate cases for L.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

2 k-WRS is Fixed Parameter Tractable

In this section, we present our parameterized result for k-WRS. Figure 1 shows an example of an arrangement $\mathcal{L}$ of line segments in a plane and its corresponding optimal closed walk intersecting all line segments in $\mathcal{L}$.

2.1 Transforming $\mathcal{L}$ into a plane straight-line graph

To start with, we transform the instance $\mathcal{L}$ into an equivalent plane straight-line graph PSLG.

Input: $\mathcal{L}$, a connected arrangement of line segments.

Output: G, a plane straight-line graph (PSLG) corresponding to $\mathcal{L}$.

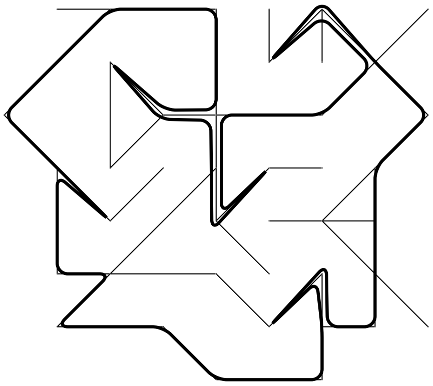


Figure 1 $\mathcal{L}$ and its corresponding optimal walk (shown in bold lines)

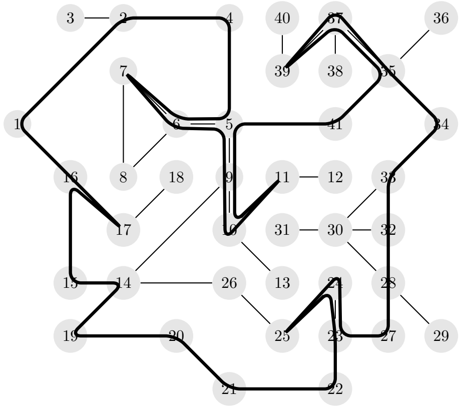


Figure 2 PSLG G transformed from $\mathcal{L}$ and *Opt-WRG* (shown in bold lines)

Each vertex $v_i \in V$ of G represents either an endpoint of a line segment or the intersection point of two line segments. There exists an edge $e_i = (u, v) \in E$ between two vertices $u, v \in V$, if the corresponding points of u and v have a line segment passing through them without a vertex in between. The positions of vertices and edges in the plane are preserved in order to fit the visibility and Euclidean distance criteria. We define *visibility* in PSLG as follows.

▶ **Definition 2.1.** A vertex $v_j \in V$ or an edge $e_j \in E$ is *visible* to a vertex $v_i \in V$ if there exists an alternating sequence of vertices and edges $v_i, e_{i+1}, v_{i+1}, e_{i+2}, \ldots, e_j, v_j$, such that $slope(e_r) = slope(e_s)$ for every $i < r, s \leq j$.

Alternatively, we say a walk *covers* a set of vertices or edges if the set of vertices or edges are visible from at least one vertex in the walk. The k-WRS is translated into k-WRG, which aims to find an optimal watchman route (*Opt-WRG*) in a PSLG G with k faces. An example input and output with respect to PSLG is given in Figure 2. We denote *Opt-Walk* to be a walk on G, which we intend to build through the subsequent sections.

2.2 Identification of required portion of *Opt-WRG*

In this section, we apply a preprocessing on G and then identify a few required portions of *Opt-WRG* on the preprocessed G.

Leaf vertices for which the slope of the edge between the leaf and the parent vertex is the same as the slope of some other edge incident with the parent vertex can be removed from the graph, as these leaf-parent edges will always be covered when the edge with the same slope incident with the parent is covered. Next, we use Lemma 2.2 and 2.3 for identifying a few walks that are required portions of *Opt-WRG*.

▶ **Observation 2.2.** *A cut-vertex is a required portion of Opt-WRG on G.*

▶ **Observation 2.3.** *An edge $e_i = (u, v)$, where u and v are cut-vertices and e_i is a cut-edge (bridge), is a required portion of any Opt-WRG on G and is to be traversed back and forth.*

In the trivial case, if G is a tree, then a closed walk by a depth-first traversal, on the above identified vertices (shown as boxes in Figure 3) and edges gives *Opt-WRG*. For G, when $|F| > 1$, a collection of open walks can be created where each *walk* is a depth-first traversal on these vertices and edges (Figure 3). Consequently, we remove all cut-edges (bridges) and then the isolated vertices from G, which gives us a 2-connected PSLG G (Figure 4). Details of this approach is omitted due to space limitations.

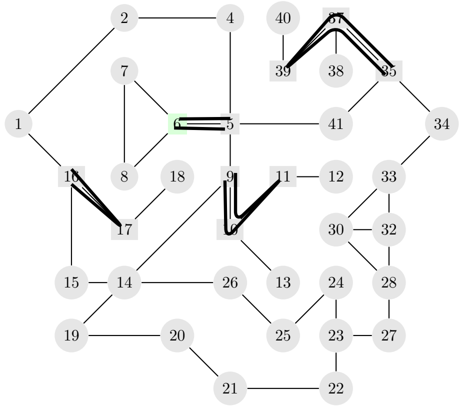


Figure 3 Collection of *walks* (*Opt-Walk*)

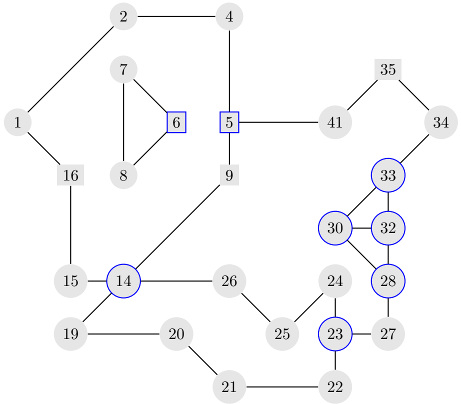


Figure 4 Resultant 2-connected PSLG G

These collection of walks is then used to build *Opt-Walk*, which is discussed in the next section.

2.3 Design of *Opt-Walk* on G

This section describes two structures that make up the 2-connected PSLG: cyclic strip and path strip (collectively called *strips*).

▶ **Definition 2.4.** A *cyclic* strip is a cycle $s = (v_0, e_1, v_1, e_2, \ldots, v_{n-1}, e_n, v_n = v_0)$ such that for $1 \leq i \leq n-1$, $degree(v_i) = 2$ in G.

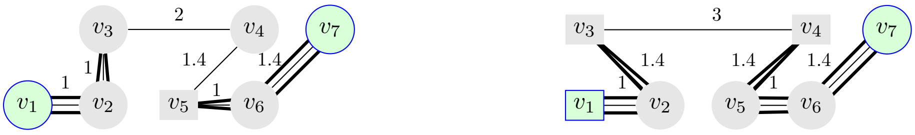
▶ **Definition 2.5.** A *path* strip is a maximal path $s = (v_0, e_1, v_1, e_2, \ldots, v_{n-1}, e_n, v_n)$ such that for $1 \leq i \leq n-1$, $degree(v_i) = 2$ in G.

To denote a strip, we use the notation (v_i, v_j)-strip, where vertices $v_i, v_j \in V$ denote the end vertices of the sequence s. We call these end vertices *strip-vertices* (blue colored vertices in Figure 4). For $|F| > 2$, strip-vertices have degree at least three ($|F| = 2$ is a trivial case). In addition, let S denote a set of strips and SV denote a set of strip-vertices. As with a graph $G = (V, E)$, we define a strip graph $SG = (SV, S)$ as a pair of sets SV and S, where strip $s \in S$ connects any two $v_i, v_j \in SV$. We use the notation $SG \setminus s_i$ to represent $SG(SV, S \setminus \{s_i\})$, and $SG \setminus v_i$ to represent $SG(SV \setminus \{v_i\}, S \setminus \{(v_i, v_j)\text{-}strip \in S\})$.

In our approach, strips are crucial, since they are a part of the 2-connected PSLG, along which watchman routes can interact only in a finite number of ways. A watchman is positioned on a strip-vertex v_i and he attempts to determine the optimal walk on the strip. The four walks explained in the next two sections illustrate his choices.

Walk that traverses both ends of a strip. We present two archetypes for an open walk on a (v_i, v_j)-strip under the constraint (C_1) that the walk traverses both vertex v_i and vertex v_j of the strip (for example: green vertices in Figure 5 are vertices to be traversed).

- If walk from v_i is to end at v_j, then the optimal walk $(walk_1)$ is $v_i v_{i+1} \dots v_{j-1} v_j$
- If walk from v_i is to end at v_i, then two walks $v_i v_{i+1} \dots v_{p-2} v_{p-1} v_{p-2} \dots v_{i+1} v_i$ and $v_j v_{j-1} \dots v_{p+2} v_{p+1} v_{p+2} \dots v_{j-1} v_j$ that avoid the longest visible sub-string $v_{p-1} v_p v_{p+1}$ constitutes the optimal walk $(walk_2)$ (bold lines in Figure 5).



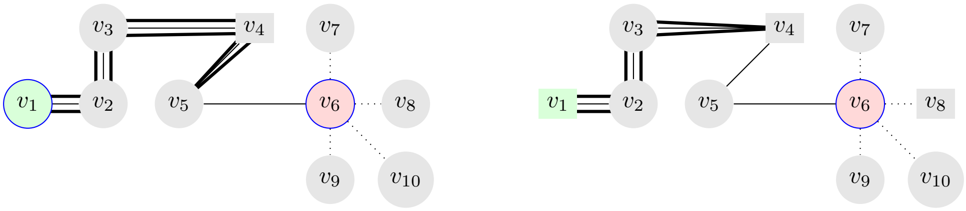
(a) Maximal visible sub-string is v_3, v_4, v_5.

(b) Maximal visible sub-string is v_3, v_4.

Figure 5 Examples for $walk_2$ on (v_1, v_7)-strip. In Figure 5b, since v_3 and v_4 are cut-vertices, they have to be traversed by $walk_2$.

Walk that avoids visiting an end of a strip. We present two archetypes for an open walk on a (v_i, v_j)-strip with the constraint (C_2) that the walk traverses vertex v_i and avoids traversing vertex v_j (for example: Red colored vertices in figure 6b). Let $S' = S \setminus \{(v_i, v_j)\text{-strip}\}$.

- If no walks on neighbouring strips (S') cover the edge of the (v_i, v_j)-strip incident with v_i, then the optimal walk $(walk_3)$ on the strip is $v_i v_{i+1} \dots v_{j-1}, v_{j-2} \dots v_{i+1} v_i$ (Figure 6a).
- If a walk on neighbouring strips (S') covers the edge of the (v_i, v_j)-strip incident with v_i, then the optimal walk $(walk_4)$ on the strip is $v_i v_{i+1} \dots v_{j-2} v_{j-3} \dots v_{i+1} v_i$ (Figure 6b).



(a) $walk_3$ has to traverse v_5 as no other walk will cover edge (v_5, v_6)

(b) $walk_4$ has to traverse only till v_4 as edge (v_5, v_6) will be covered by some other walk

Figure 6 Example for $walk_3$ (6a) and $walk_4$ (6b) on (v_1, v_6)-strip represented using bold lines.

Figure 7 shows an approach that limits the number of possible walks on a strip according to the constraints imposed on its strip-vertices.

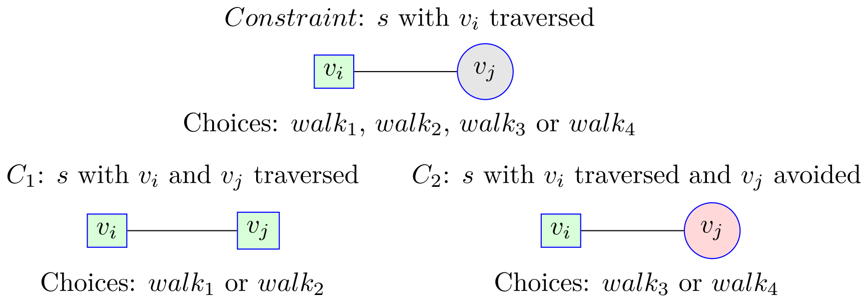

Constraint: s with v_i traversed
v_i
v_j
Choices: $walk_1$, $walk_2$, $walk_3$ or $walk_4$
C_1: s with v_i and v_j traversed
v_i
v_j
Choices: $walk_1$ or $walk_2$
C_2: s with v_i traversed and v_j avoided
v_i
v_j
Choices: $walk_3$ or $walk_4$

Figure 7 Determining walks based on constraints on the strip-vertices of a (v_i, v_j)-strip

2.4 FPT Algorithm for k-WRG

This section presents our FPT algorithm for k-WRG using the four *walks* ($walk_1$, $walk_2$, $walk_3$, $walk_4$), along with the constraints used to design them. We first show how to compute optimum walks on cyclic strips and then we propose an algorithm for calculating optimum walks on path strips.

Assigning walks for cyclic strips. We observe that strip-vertex of a cyclic strip s is a cut-vertex and is to be traversed. Hence, either $walk_1$ or $walk_2$, whichever is of minimum walk length, is the optimal walk on s. Therefore, for each cyclic strip $s \in S$, we assign the optimal walk and do the operations $SG \setminus s$ and $k \leftarrow k - 1$ recursively. As a result, SG simply contains path strips.

Assigning walks for path strips. Suppose $walk_1$ or $walk_2$ is chosen for a (v_i, v_j)-strip, then this choice constraints v_j to be traversed by $Opt\text{-}Walk$. Similarly, if $walk_3$ or $walk_4$ is chosen for a (v_i, v_j)-strip, then it constraints v_j to be avoided by $Opt\text{-}Walk$. In this way a choice of a *walk* for a strip imposes constraints to be propagated to neighbouring strip-vertices of strips. A Eulerian Path is a path in a graph that visits every edge exactly once. A Eulerian Circuit is an Eulerian Path that starts and ends on the same vertex. Combining these ideas, Algorithm 1 finds a minimum length closed walk starting from a strip-vertex v_i which covers G. Based on what we have discussed in previous sections, these are the rules we will follow when constructing the walk from our algorithm:

1. Each assigned walk for a strip should comply with all constraints in the strip.
2. A walk constructed should attach with cut-vertices in a strip and should connect with previously computed walks originating from these cut-vertices.

▶ **Lemma 2.6.** *Algorithm 1 always returns an $Opt\text{-}WRG$ on G given the strip-vertex v_i is traversed by $Opt\text{-}WRG$ and runs in $O^*(5^{2|S|/3}3^{|S|/3})$-time, where S is the set of strips.*

Proof. Algorithm 1 recursively builds a collection of walks where each walk is assigned to a strip in SG that starts from strip-vertex v_i. There are five ways to traverse the strip: $walk_1$, $walk_2$, $walk_3$, $walk_4$, and two $walk_1$ for traversing to and fro. When a walk complies with the constraints in the strip, it will be added to the collection $Opt\text{-}Walk$. Also, when a walk is added to the collection, all the edges covered by the walk are removed from G. A collection is finalized if all strips attached to traversed strip-vertices are exhausted. Each collection of walks can be visualized as a directed graph. Each of the walks $walk_2$,$walk_3$,$walk_4$ starts and ends at the same strip-vertex v_i, thus creating a self-loop edge. $walk_1$ forms a single outgoing

Algorithm 1: Finding optimal k-WRG starting from a strip-vertex

```
Input   : < SG, v_i, Opt-Walk, G >, a preprocessed strip-graph SG, a strip-vertex v_i,
          a partially computed Opt-Walk, and preprocessed PSLG G whose edges
          are not covered
Output : Opt-Walk', a minimum length closed walk that covers the preprocessed
          2-connected PSLG G.
Assumption: strip-vertex v_i is traversed by Opt-WRG.
Procedure Opt-Walk-Recursive(SG, v_i, Opt-Walk, G)
    if Eulerian Circuit exists for Opt-Walk and G has no edges then
        return a Eulerian Circuit on Opt-Walk;
    else
        Opt-Walk' ← a walk of infinite length;
        s ← select a strip in S(SG) which is incident with strip-vertex v_i;
        for each walk_i which is either walk_1,walk_2,walk_3,walk_4, or the to and fro
         walk_1 that meets the constraints for strip s do
            Set-Walks ← Opt-Walk ∪ {walk_i};
            next-v_i ← Select a traversed strip-vertex that is attached to at least one
             strip in SG;
            Temp-Opt-Walk_i ← Opt-Walk-Recursive(SG \ s, next-v_i, Set-Walks, G\
             edges covered by walk_i);
            if |Temp-Opt-Walk_i| < |Opt-Walk'| then
                Opt-Walk' ← Temp-Opt-Walk;
            end
        end
        return Opt-Walk';
    end
end
```

edge from v_i to an adjacent strip-vertex. In the base case, we check if the directed graph formed by an *Opt-Walk* has an Eulerian circuit, and that the walks within the *Opt-Walk* collection cover G. If the conditions are satisfied, we return an Eulerian circuit as a closed connected walk covering G. By comparing the length of each walk, the algorithm returns the shortest walk, thus returning *Opt-WRG*.

The algorithm essentially finds unique permutations of walks on the $|S|$ number of strips. With five choices per strip we have $5^{|S|}$ possible permutations. Given an undirected graph, Eulerian circuits can be found in $O(|SV| + |S|)$-time.

Optimization using Constraint propagation. For each walk assigned to an $s \in S$, the strip-vertices of s get constrained to be either traversed or avoided. The maximum number of walk choices on a strip constrained by constraint C_1 is three, and the maximum number of walk choices on a strip constrained by constraint C_2 is two. Let a face be circumscribed by vertices and strips in the order $v_0, s_1, v_2, .., v_{m-1}, s_{m-1}, v_m = v_0$. For any arrangement of walks on m strips, at least one strip will have both its strip-vertices traversed or avoided. In the m strip graph, adding a face can create at most three additional strips, which is the result of connecting two non-strip-vertices from two different strips. Suppose we compute the choices of walks in the resultant graph. There would be two strips with five walk choices and one strip with three choices. Hence, the marginal increase in time complexity is at most $5^{2/3}3^{1/3}$ and the Algorithm 1 runs in $O^*(5^{2|S|/3}3^{|S|/3})$ using the constraint propagation technique. ◀

▶ **Lemma 2.7.** *At least one strip-vertex in SG is to be traversed by Opt-WRG.*

Proof. Assume none of the strip-vertices are traversed by an *Opt-WRG*, on SG having $S \neq \emptyset$, then the walk must traverse within the strips without traversing v_i or v_j of a (v_i, v_j)-strip in SG. Since v_i is connected to at least three strips, SG must have at least three strips. A walk without visiting at least one strip-vertex cannot cover the other two strips. Thus, at least one strip-vertex in SG is to be traversed by *Opt-WRG*. ◀

▶ Remark. For every $v_i \in SV$, v_i is of degree at least three, hence $2|S| \geq 3|SV|$. It is easy to see that SG is also a planar graph. Thus, by extending Euler's formula for SG, we get $|SV| - |S| + |F| = 2$ [1]. Therefore, the number of strips in SG is at most $3(|F| - 2)$.

To obtain the optimal watchman route, we invoke Algorithm 1 on every vertex $v_i \in SV$, compare the walk lengths of *Opt-$Walk'$* obtained from each function call, and choose the shortest *Opt-$Walk'$*. Theorem 2.8 summarizes our result.

▶ **Theorem 2.8.** *Opt-WRG for G is obtained in $O^*(5^{2k}3^k)$-time.*

Proof. From Lemma 2.6 and Lemma 2.7, it is clear that our final computation which iterates through each strip-vertex in SV to find the minimum closed walk always return an *Opt-WRG* for an SG. As the Algorithm 1 is invoked $|SV|$ number of times, the total running time becomes $O^*(5^{2k}3^k)$. ◀

The following corollary on k-WRS is a direct consequence of Theorem 2.8.

▶ **Corollary 2.9.** *k-WRS is Fixed Parameter Tractable on the connected arrangement of line segments $\mathcal{L}$ with the parameter k being the number of faces $|F|$ and runs in $O^*(5^{2k}3^k)$-time.*

References

1 Martin Aigner and Günter Ziegler. *Three applications of Euler's formula*, pages 75–80. Springer, Berlin, Heidelberg, 01 2010. doi:10.1007/978-3-642-00856-6_12.

2 Wei-pang Chin and Simeon Ntafos. Optimum watchman routes. *Information Processing Letters*, 28:39–44, 01 1986. doi:10.1016/0020-0190(88)90141-X.

3 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Daniel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2015.

4 Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph Mitchell. Touring a sequence of polygons. *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, 04 2003. doi:10.1145/780542.780612.

5 Adrian Dumitrescu, Joseph S.B. Mitchell, and Paweł Żyliński. Watchman routes for lines and line segments. *Computational Geometry*, 47(4):527 – 538, 2014. doi:10.1016/j.comgeo.2013.11.008.

6 Adrian Dumitrescu and Csaba D. Tóth. Watchman tours for polygons with holes. In *Computational Geometry : Theory and Applications*, volume 45, pages 326–333, 2012. doi:10.1016/j.comgeo.2012.02.001.

7 Cristian S. Mata and Joseph S. B. Mitchell. Approximation algorithms for geometric tour and network design problems. In *SCG '95*, pages 360–369, 01 1995. doi:10.1145/220279.220318.

8 Joseph S. B. Mitchell. Approximating watchman routes. In Sanjeev Khanna, editor, *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, pages 844–855. SIAM, 2013. doi:10.1137/1.9781611973105.60.

9 Xuehou Tan. Approximation algorithms for the watchman route and zookeeper's problems. *Discrete Applied Mathematics*, 136(2):363 – 376, 2004. The 1st Cologne-Twente Workshop on Graphs and Combinatorial Optimization. doi:10.1016/S0166-218X(03)00451-7.

10 Xuehou Tan. A linear-time 2-approximation algorithm for the watchman route problem for simple polygons. *Theoretical Computer Science*, 384(1):92 – 103, 2007. Theory and Applications of Models of Computation. doi:10.1016/j.tcs.2007.05.021.

11 Xuehou Tan and Bo Jiang. An improved algorithm for computing a shortest watchman route for lines. *Information Processing Letters*, 131:51–54, 2018. doi:https://doi.org/10.1016/j.ipl.2017.11.011.

12 Ning Xu. Complexity of minimum corridor guarding problems. *Information Processing Letters*, 112(17):691 – 696, 2012. doi:10.1016/j.ipl.2012.06.003.

Orientation type of convex sets*

Péter Ágoston[1], Gábor Damásdi[1], Balázs Keszegh[1,2], and Dömötör Pálvölgyi[1]

1 **ELTE Eötvös Loránd University, MTA-ELTE Lendület Combinatorial Geometry Research Group, Budapest, Hungary**
agostonp@cs.elte.hu,gabor.damasdi@gmail.com,domotorp@gmail.com

2 **Alfréd Rényi Institute of Mathematics**
keszegh.balazs@renyi.hu

Abstract

We introduce a definition of orientation for intersecting planar convex sets and study its properties.

1 Introduction

A family is *intersecting* if any two members of the family intersect, and it is *3-intersection-free* if no three members of the family have a common intersection. These were studied by Jobson et al. [12] (see also Lehel and Tóth [14] and related recent results in extremal combinatorics [16]) who showed that if three compact convex planar sets, A, B, C, form an intersecting and 3-intersection-free family, then $\mathbb{R}^2 \setminus (A \cup B \cup C)$ has exactly one bounded component, called the *hollow* of ABC, which we will denote as ⅄(ABC) (see Figure 1). They have also shown that the convex hull of this hollow is a triangle with sides a, b, c, such that (apart from its endpoints) side a is contained in $A \setminus (B \cup C)$, side b in $B \setminus (A \cup C)$, and side c in $C \setminus (A \cup B)$. We may refer to the vertices of this triangle as the *vertices of the hollow*, but note that since the hollow is open, its vertices are not a part of it, only of its closure.

From now on whenever we refer to a convex set, it is always assumed to be compact.

The following lemma is a straightforward consequence of Lemma 1 in [12].

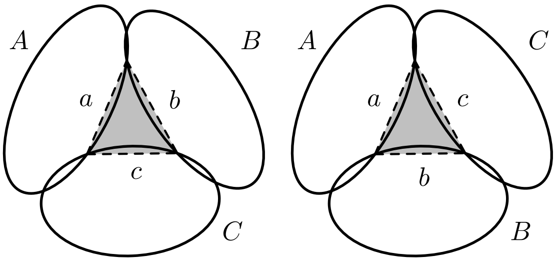


Figure 1 Three convex sets, A, B and C with negative (left) and positive (right) orientation and their hollow, ⅄(ABC).

* Research supported by the Lendület program of the Hungarian Academy of Sciences (MTA), under the grant LP2017-19/2017. Research of the 3rd author is also supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences, by the National Research, Development and Innovation Office – NKFIH under the grant K 132696 and FK 132060 and by the ÚNKP-20-5 New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

▶ **Lemma 1.1** (Jobson-Kézdy-Lehel-Pervenecki-Tóth [12]). *Three pairwise intersecting compact convex sets, A, B, C, that do not have a common point, enclose a hollow* ⅄(ABC)*, and the following four properties hold.*

(a) ⅄(ABC) *is a simply connected region.*
(b) The boundary of ⅄(ABC) *has exactly one arc from each of the boundaries of A, B and C.*
(c) The closure of the convex hull of ⅄(ABC) *is a triangle with sides a, b, c such that (apart from its endpoints) side a is contained in $A \setminus (B \cup C)$, side b in $B \setminus (A \cup C)$, and side c in $C \setminus (A \cup B)$.*
(d) For any $x \in B \cap C$, $y \in A \cap C$ and $z \in A \cap B$ the orientation of the xyz triangle is the same and agrees with the orientation of abc.

Define the *orientation* of ABC as the orientation of the triangle with sides a, b, c: if the sides along the boundary of the triangle follow each other in a counterclockwise direction as abc, then we define the orientation of ABC as *positive*, otherwise as *negative*. We also define the orientation of three convex sets with a common intersection as *zero*. This way we can assign an orientation to any three members of an intersecting family of convex sets in the plane. To simplify notation, we assign a value from $\{\pm 1, 0\}$ to each ordered triple according to their orientation and write (omitting the function assigning the value in notation) $ABC = +1$, $ABC = -1$, $ABC = 0$, respectively, for positive, negative, zero orientations.[1] From the definitions, it follows that $ABC = CAB = BCA = -ACB = -BAC = -CBA$. We will call $\{\pm 1, 0\}$ value assignments to all triples of some base set satisfying the previous equalities *cyclic partial orientations*, and if the zero value is not allowed, *cyclic orientations*. Different orientations of interest are compared later in Figure 5.

Helly's theorem says that if for some $n \geq 3$ planar convex sets $A_1, \ldots, A_n$ we have $A_i A_j A_k = 0$ for any $i < j < k$, then $\cap_{i=1}^{n} A_i \neq \emptyset$; we will abbreviate this condition as $A_1 \ldots A_n = 0$. We will also apply this shorthand notation for orientations of points, so for example $abcd = +1$ means that the points a, b, c, d are the vertices of a convex quadrangle, in this counterclockwise order. With this notation, Lemma 1.1(d) states $xyz = ABC$. Lemma 1.1(d) also implies the following.

▶ **Corollary 1.2.** *If convex sets A, B, C enclose a hollow and convex sets $A' \subset A, B' \subset B, C' \subset C$ are pairwise intersecting, then $A'B'C' = ABC$.*

▶ Remark. Our definition only allows us to define an orientation for pairwise intersecting triples of convex sets. This is unlike the situation in the case of the (quite different) definition in [3, 4, 5] by Bisztriczky and Fejes Tóth (later also investigated in [6, 7, 11, 17, 19, 20, 21]) which primarily focused on Erdős-Szekeres type theorems.[2] In these papers the condition on the family of convex sets is that they are pairwise disjoint, or in later papers that they are non-crossing. Such a family is in convex position if no set is covered by the convex hull of the rest. In this case the orientation of ABC is determined by any points $a \in A, b \in B, c \in C$ chosen from the boundary of $conv(A \cup B \cup C)$. This definition appeared explicitly in [11] and is implicitly in earlier works—we will refer to it as the Bisztriczky-Fejes Tóth type orientation. Note that if A, B, C are in addition also intersecting but 3-intersection-free, then

[1] It might seem counterintuitive that the intersecting case is assigned 0 but this is the natural choice in some cases; see also [18, Section 4].

[2] For intersecting families, an Erdős-Szekeres type theorem with our definition of orientation follows directly from Ramsey's theorem.

the Bisztriczky-Fejes Tóth type definition gives the same orientation as the one used in this paper. But such families can contain at most four connected sets, as K_5 is non-planar.

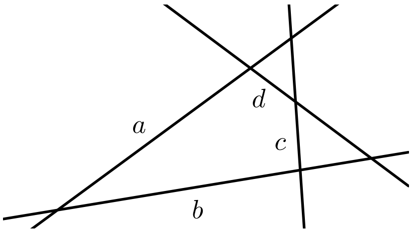


Figure 2 The following triples have positive orientation: abc, abd, adc, bdc

If a family of convex sets in the plane is intersecting and 3-intersection-free, we call it *holey.* For example, any collection of lines in general position is holey, and the orientation of any triple is determined by their slopes (see Figure 2). This orientation for lines is not to be confused with the much studied arrangement types of lines which were shown by Goodman and Pollack [8] to be the duals of order types of points. However, they also made the following simple observation about the orientations of triples of lines, which is relevant for us.

▶ **Observation 1.3** (Goodman-Pollack [9]). *If a holey family consists of lines $\ell_1, \ldots, \ell_n$, ordered according to their slopes in clockwise circular order, then the orientation of their triples is the same as the orientation of the triangles of n points $p_1, \ldots, p_n$ in convex position, ordered in counterclockwise order.*

Our main motivation to study holey families is that it can be the first step to improve our understanding of the intersection structure of planar convex sets, which can potentially lead to improved weak ε-nets [1] and (p,q)-theorems [2]. The question is, what abstract properties of the underlying geometric 3-hypergraphs are useful to derive interesting results.

In the following, we will call an abstract system of orientations of triples that can be derived from a holey family of planar convex sets a *C-3OSET* (Convex Triple Orientations) and its superfamily where the sets are only required to be pairwise intersecting a *C-3POSET* (Convex Triple Partial Orientations). So a C-3OSET is roughly like an order type of points in general position (sometimes this is called *simple* order type), while a C-3POSET would correspond to an order type where we allow more than two points to be collinear (sometimes this is called order type)—we will refer to this latter notion as *partial* order type (see Figure 5). In Knuth [13] these are referred to as partial signings that can be completed to form order types.

Our results. In Section 2 we show that C-3POSETs satisfy a natural interiority condition, and compare them with other well-studied cyclic orientations.
Due to space restrictions, the following topics will appear in the full version of the paper.
We give an example for a holey family that cannot be extended by adding another convex set to it.
We examine which small configurations are realizable as a C-3OSET, and we find that up to five elements, the single condition that the configuration satisfies Lemma 2.1, is sufficient.
On the other hand, we show that there is a five element configuration that corresponds to a five-point partial order type but cannot be realized as a C-3POSET.
Finally, we study what happens if a C-3POSET also has the (4,3) property, that is, among

any four convex sets there are three that have a common point. We derive some new abstract properties, however, we also show that on their own they are not yet sufficient to prove a (p, q) theorem.

2 Interiority

▶ **Lemma 2.1** (Interiority Lemma). *If A, B, C, O is an intersecting family of convex sets and $ABO = BCO = CAO = 1$, then $ABC = 1$.*

Proof. For a contradiction, suppose first $ABC = 0$. Fix some $w \in A \cap B \cap C$, and take any $a \in A \cap O$, $b \in B \cap O$ and $c \in C \cap O$ and check the orientations of the triples of w, a, b, c using Lemma 1.1(d). It follows that $w \in conv(a, b, c) \subset O$, contradicting that $ABO = 1$.

Now suppose $ABC = -1$. Take any $a \in A \cap O$, $b \in B \cap O$, $c \in C \cap O$, $z \in A \cap B$, $x \in B \cap C$ and $y \in A \cap C$. We can assume that these six points are in general position, otherwise we could slightly perturb them, along with the convex sets containing them, if necessary, without introducing a triple intersection. The conditions and Lemma 1.1(d) imply that $abz = bcx = cay = -1$ and $xyz = -1$. Also, as there is no triple intersection, we know that $x, y, z \notin conv(abc)$, $b, c, x \notin conv(ayz)$, $a, c, y \notin conv(bxz)$, $a, b, z \notin conv(cxy)$. We will deal with two cases, depending on the orientation of abc. The lines ab, bc, ca divide the plane into seven regions: a bounded triangle $conv(abc)$, three unbounded cones, which we denote by V_a, V_b, V_c, respectively, indexed by their apexes, and three unbounded regions sharing a side each with the triangle $conv(abc)$, which we denote by U_{ab}, U_{bc}, U_{ac}, respectively, indexed by the adjacent side of the triangle.

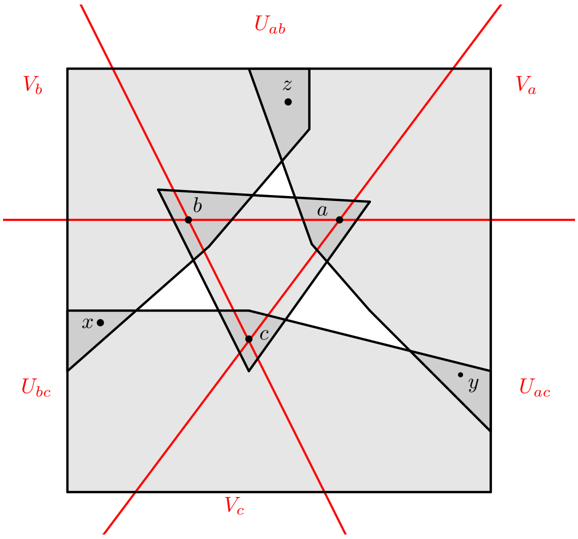


Figure 3 Case 1 of the proof of Lemma 2.1. Beware that in the figure $xyz = 1$ while in the proof $xyz = -1$ but we could find no better way to depict contradicting assumptions.

Case 1: $abc = 1$ (see Figure 3).
The orientation conditions and $x, y, z \notin conv(abc)$ imply that $x \in V_b \cup U_{bc} \cup V_c$, $y \in V_c \cup U_{ac} \cup V_a$, $z \in V_a \cup U_{ab} \cup V_b$.
Since $xyz = -1$, two of x, y, z must fall in the same cone V_i. Without loss of generality, assume that $x, y \in V_c$. As $c \notin conv(ayz)$, and a is to the right of the directed line yc, z must either lie to the right of line yc or to the left of the line ac. Since z lies to right of the line

ab, if it lies to the left of ac then it is in V_a. Hence z must lie to the right of yc. Similarly z must lie to the left of xc. But this implies $z \in U_{ab}$ and $xyz = 1$, a contradiction.

Case 2: $abc = -1$.
The orientation conditions and $x, y, z \notin conv(abc)$ imply that $x \in U_{ab} \cup V_a \cup U_{ac}$, $y \in U_{bc} \cup V_b \cup U_{ab}$, $z \in U_{ac} \cup V_c \cup U_{bc}$.
If any of x, y, z fall in a cone V_i, e.g., x falls in V_a then $y, z \in U_{a,c}$ and we can finish with a similar argument as in the previous case.
Otherwise, say that a has an *opposite* point, if $y \in U_{bc}$ or $z \in U_{bc}$ and, similarly, b has an opposite point, if $x \in U_{ac}$ or $y \in U_{ac}$ and c has an opposite point, if $x \in U_{ab}$ or $y \in U_{ab}$. If a does not have an opposite point, then $y \in U_{ab}$ and $z \in U_{ac}$, which implies that both b and c have an opposite point. Therefore, at least two of a, b, c have an opposite point, say, b and c. But then the segments connecting b and c to their opposite points intersect inside $conv(abc)$, which gives a triple intersection, contradicting our assumptions. ◀

If $ABO = BCO = CAO = 1$ or $ABO = BCO = CAO = -1$ for some intersecting family of convex sets, then we will write $O \in conv(ABC)$. Note that the order of A, B, C is irrelevant in the notation. This, however, can be quite misleading, as this notion of convexity does not have many natural properties, as we will see. We say that the containment $O \in conv(ABC)$ is *regular* if $O \cap \partial \text{⅄}(ABC) \cap A$, $O \cap \partial \text{⅄}(ABC) \cap B$ and $O \cap \partial \text{⅄}(ABC) \cap C$ are connected, and we say that the containment $O \in conv(ABC)$ is *irregular* if one of them has more than one connectivity component (see Figure 4). If $O \in conv(ABC)$ is regular, then $O \cap \text{⅄}(ABC) \cap \partial A$, $O \cap \text{⅄}(ABC) \cap \partial B$ and $O \cap \text{⅄}(ABC) \cap \partial C$ are a connected curves.

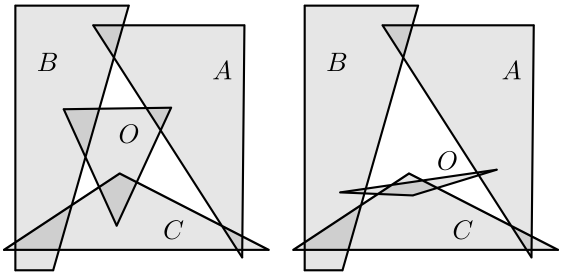


Figure 4 Regular and irregular containment $O \in conv(ABC)$.

Knuth [13] studied cyclic orientations that satisfy Lemma 2.1 under the name *interior triple system*, according to Knuth “for want of a better name.” We want a better name, so we will refer to such an orientation as a *3OSET* (Triple Orientations), while if zero-orientations are also allowed, then we call such a system a *3POSET* (Triple Partial Orientations). We believe these names are better as they are similar to POSETs, which would be a 2POSET (Pair Partial Order) in our language.

Lemma 2.1 implies that the orientation of the triples of any holey family is a 3OSET. To the best of our knowledge, such systems have not been studied anywhere except [13, Chapter 3], where the main result is that there are $2^{\Omega(n^3)}$ different 3OSETs over n elements.

If we add another property (the definition of which we omit here), then we get a much better studied notion, known under the names of CC systems, pseudoline arrangements, rank 3 oriented matroids. For order types and for pseudoline arrangements, see [10, Chapter 5], while for oriented matroids, see [10, Chapter 6]. That property, however, is not satisfied by holey convex families. In fact, not even the following weaker condition, that we define below.

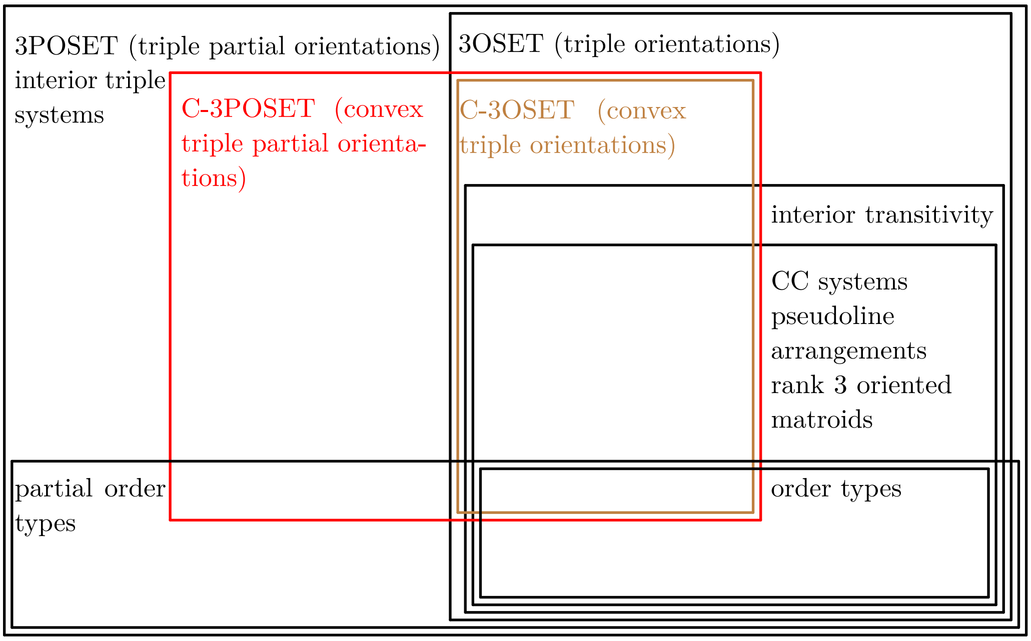

3POSET (triple partial orientations)
interior triple systems
3OSET (triple orientations)
C-3POSET (convex triple partial orientations)
C-3OSET (convex triple orientations)
interior transitivity
CC systems
pseudoline arrangements
rank 3 oriented matroids
partial order types
order types

Figure 5 A diagram illustrating the relationship of some related notions. A 3POSET is any cyclic partial orientation of triples satsifying $ABC = -ACB$ and Lemma 2.1. A 3OSET is a 3POSET such that no triple is zero-oriented. A C-3OSET (resp. C-3POSET) is a subset of these that is realizable with planar convex sets. C-3POSETs do not contain all partial order types, but we could not establish the respective statement for order types.[3]

Knuth [13, Chapter 2, (2.4)] defines the *interior transitivity* condition as follows: If $D \in conv(ABC)$ and $E \in conv(ABD)$, then $E \in conv(ABC)$. The interior transitivity condition is satisfied by the earlier mentioned CC systems, but it is strictly weaker than them. Indeed, the number of orientations of triples of n sets that satisfy the interior transitivity condition is $2^{\Omega(n^2 \log n)}$, while the number of CC systems is $2^{\Theta(n^2)}$, and the number of CC systems that are representable by planar point sets, known as stretchable arrangements/order types, is $2^{\Theta(n \log n)}$. We will see below that there are holey families that do not satisfy the interior transitivity condition. However, the following weaker statement is true.

▶ Claim 2.2. Suppose A, B, C, D and E are elements of a holey family. If $D \in conv(ABC)$ and $E \in conv(ABD)$, then $D \cap E \subset \text{⅄}(ABC)$.

For the proof we need the following simple observation.

▶ **Observation 2.3.** *Suppose A, B, C and O are elements of a holey family. Then $O \in conv(ABC)$ if and only if $\text{⅄}(ABO), \text{⅄}(BCO), \text{⅄}(CAO) \subset \text{⅄}(ABC) \cup A \cup B \cup C$.*

Proof of Claim 2.2. Since $D \cap E$ intersects $\partial \text{⅄}(ABD)$ which is contained in $\text{⅄}(ABC) \cup A \cup B \cup C$ by Observation 2.3, and $D \cap E$ cannot intersect $A \cup B \cup C$ as there are no triple intersections, we get that $D \cap E \subset \text{⅄}(ABC)$, as required. ◀

[3] For the Bisztriczky-Fejes Tóth type definition of order types of convex sets, any point order type is by definition realizable by convex sets, while in the other direction a configuration of convex sets whose order type is not realizable by points was given in [20] answering a question of Hubard and Montejano.

3 Discussion

Our definition of orientation can be generalized to intersecting pseudo-disk arrangements and to $d+1$ convex sets in $\mathbb{R}^d$ dimensions. We leave these for future research, just like the following two important questions left open in this paper.
Is there an order type of planar points that is not a C-3OSET, i.e., not realizable by holey planar convex sets?
What further properties of C-3POSET's (orientations of intersecting planar convex sets) are needed to obtain efficient (p, q) theorems?

Acknowledgments. We would like to thank Márton Naszódi for discussions during the entire project.

References

1 N. Alon, I. Bárány, Z. Füredi, and D. J. Kleitman. Point selections and weak ε-nets for convex hulls. *Combinatorics, Probability and Computing*, 1(3):189–200, 1992.
2 N. Alon and D. J. Kleitman. Piercing convex sets and the Hadwiger-Debrunner (p, q)-problem. *Advances in Mathematics*, 96(1):103–112, 1992.
3 T. Bisztriczky and G. Fejes Tóth. A generalization of the Erdös-Szekeres convex n-gon theorem. *Journal für die Reine und Angewandte Mathematik*, 395:167–170, 1989.
4 T. Bisztriczky and G. Fejes Tóth. Nine convex sets determine a pentagon with convex sets as vertices. *Geometriae Dedicata*, 31(1):89–104, 1989.
5 T. Bisztriczky and G. Fejes Tóth. Convexly independent sets. *Combinatorica*, 10(2):195–202, 1990.
6 M. G. Dobbins, A. Holmsen, and A. Hubard. The Erdös-Szekeres problem for non-crossing convex sets. *Mathematika*, 60(2):463–484, 2014.
7 M. G. Dobbins, A. Holmsen, and A. Hubard. Regular systems of paths and families of convex sets in convex position. *Transactions of the American Mathematical Society*, 368(5):3271–3303, 2016.
8 J. E. Goodman and R. Pollack. A theorem of ordered duality. *Geometriae Dedicata*, 12:63–74, 1982.
9 J. E. Goodman and R. Pollack. Semispaces of configurations, cell complexes of arrangements. *Journal of Combinatorial Theory. Series A*, 37:259–293, 1984.
10 Handbook of Discrete and Computational Geometry (edited by J. E. Goodman, J. O'Rourke and C. D. Tóth), Third Edition, CRC Press LLC, Boca Raton, FL 2017.
11 A. Hubard, L. Montejano, E. Mora, and A. Suk. Order types of convex bodies. *Order*, 28(1):121–130, 2011.
12 A. S. Jobson, A. E. Kézdy, J. Lehel, T. J. Pervenecki, and G. Tóth. Petruska's question on planar convex sets. *Discrete Mathematics*, 343(9):1–13, 2020.
13 D. E. Knuth. Axioms and hulls. Springer-Verlag, *Lecture Notes in Computer Science*, 1992
14 J. Lehel and G. Tóth. On the hollow enclosed by convex sets. *Geombinatorics*, 30(3):113–122, 2021.
15 D. McGinnis. A family of convex sets in the plane satisfying the (4,3)-property can be pierced by nine points, `https://arxiv.org/abs/2010.13195` 2020.
16 D. Nagy and B. Patkós. Triangles in intersecting families, `https://arxiv.org/abs/2201.02452` 2022.
17 J. Pach and G. Tóth. A generalisation of the Erdös-Szekeres theorem to disjoint convex sets. *Discrete & Computational Geometry*, 19(3):437–445, 1998.

18 J. Pach and G. Tardos. Forbidden paths and cycles in ordered graphs and matrices. *Israel Journal of Mathematics*, 155:359–380, 2006.

19 J. Pach and G. Tóth. Erdös-Szekeres-type theorems for segments and noncrossing convex sets. *Geometriae Dedicata*, 81(1-3):1–12, 2000.

20 J. Pach and G. Tóth. Families of convex sets not representable by points. *Indian Statistical Institute Platinum Jubilee Commemorative Volume–Architecture and Algorithms*, 43–53, 2009.

21 A. Suk. On order types of systems of segments in the plane. *Order*, 27(1):63–68, 2010.

A quality measure for Reeb graph drawings

Erin Chambers[1], **Elizabeth Munch**[2], and **Tim Ophelders**[3]

1 **Dept. Computer Science, Saint Louis University**
`erin.chambers@slu.edu`

2 **Dept. Computational Science, Mathematics, and Engineering, Michigan State University**
Dept. Mathematics, Michigan State University
`muncheli@msu.edu`

3 **Dept. Information and Computing Science, Utrecht University, the Netherlands**
Dept. Mathematics and Computer Science, TU Eindhoven, the Netherlands
`t.a.e.ophelders@uu.nl`

Abstract

Reeb graphs form a variant of level graphs that commonly arise in the field of topological data analysis. The vertices of a Reeb graph are associated with a real-valued level. In the context of topological data analysis, the level of a vertex usually corresponds to some imprecise measured quantity. Due to this imprecision, nearby points in a Reeb graph may represent the same point in the ground-truth.

We consider drawings of Reeb graphs in the plane, where the y-coordinate of a vertex is specified by its level, and edges are drawn as y-monotone paths. We introduce the *crossing radius* as a quality measure for Reeb graph drawings. Under this measure, level-planar drawings of Reeb graphs are optimal if they exist. In this regard, our measure coincides with existing quality measures that simply count the number of crossings. On the other hand, most Reeb graphs found in practice do not admit level-planar drawings. In contrast to measures that count the number of crossings, our measure associates a cost with each crossing, and returns the maximum cost over all crossings. The cost of a crossing intuitively quantifies the likelihood that the necessity of a crossing can be attributed to imprecise measurements. For this reason, crossing radius is a preferable measure in the context of topological data analysis. We also show that for a given Reeb graph, computing a drawing with optimal crossing radius is NP-hard.

1 Introduction

Reeb graphs have become an important tool in computational topology for the purpose of visualizing continuous functions on complex spaces as a simplified discrete structure. Essentially, Reeb graphs track for a given real-valued function on a topological space, how the connectivity of its level sets changes as one continuously increases the function value. Each connected component of a level set is represented as a point, which is either a vertex or a point on an edge. As one increases the function value, level sets change and can create, merge, split, or destroy components. Vertices correspond to such changes, and edges correspond to components of level sets that do not change their connectivity for a given interval of function values. (See Figure 1 for an illustration and Section 2 for a formal definition.) Reeb graphs were originally introduced in [14], and recent work on computing Reeb graphs efficiently [8, 13] has led to their increasing use in visualization and shape comparison [2, 15].

Despite their prevalence, surprisingly few tools from the graph drawing community are used to draw or compare Reeb graphs. The only prior work that we are aware of considers book embeddings of Reeb graphs [12]. Although of combinatorial interest, book embeddings

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

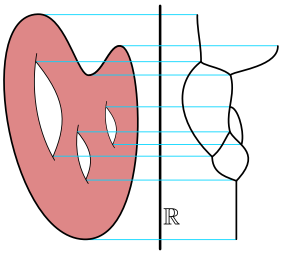


Figure 1 Left: A manifold with a function (given by height). Right: its Reeb graph.

seem less practical for visualizing larger Reeb graphs. Reeb graphs generalize level graphs, which are commonly studied in the context of level planarity [10, 9]. Several variations on level planarity are known to be NP-hard [11], and our main result is inspired by such reductions. In contrast to level graphs, vertices of Reeb graphs generally have a real-valued function value as opposed to an integer-valued one. Although the exact level of a vertex is irrelevant for most existing quality measures of graph drawings, the level of vertices carries a significant meaning in the context of topological data analysis.

Our work is motivated by tools from topological data analysis, which seek to quantify the "persistent" cycles and other topological features, in order to classify which ones are more important. Our work is particularly motivated by recent works on comparing Reeb graphs using the interleaving distance [5, 3]. At a high level, however, the interleaving distance disregards small cycles, as they are less persistent, and it disregards crossings entirely, as the embedding is not important for these distances. In contrast, traditional embedding algorithms for level planarity seek to minimize only crossings (see for example [1, 7]), which while important in our setup does not at all take the persistence of the crossing into account.

In this paper, we introduce a new comparison measure for Reeb (and hence also for level) graphs, which we call the crossing radius. This measure compares embeddings of Reeb graphs, but (in line with interleavings and in contrast to most graph drawing measures) it does not count crossings between "nearby" points, as they are more likely (in the context of Reeb graph) to be due to simple noise from the input data.

2 Reeb graphs and level planarity

For a graph G, its *geometric realization* $|G|$ is a topological space consisting of a distinct point for each vertex, and an interval (homeomorphic to $[0,1]$) for each edge (u,v), such that the endpoints of the interval are identified with the points corresponding to vertices u and v respectively, and the intervals corresponding to different edges are interior-disjoint.

Traditionally [14], Reeb graphs are constructed from a topological space Y with a real valued function $g\colon Y \to \mathbb{R}$ as follows. Define an equivalence relation on the points of Y by setting $y \sim y'$ if and only if for some value $a \in \mathbb{R}$, y and y' lie in the same path-connected component of $g^{-1}(a)$. The Reeb graph of (Y,g) is the quotient space $X = Y/{\sim}$. Notice that $g(y) = g(y')$ whenever $y \sim y'$, so the Reeb graph automatically inherits a function $f\colon X \to \mathbb{R}$ such that $f([y]) = g(y)$, where $[y]$ denotes the equivalence class of y.

When the starting space and function are well behaved (e.g. Morse functions on manifolds and generalizations [6, 5]), the resulting space can be represented as the geometric realization of a graph, and the inherited function value increases strictly along edges. The endpoints of an edge therefore necessarily have different function values, and we may assume that the function value increases linearly along edges. In this article, we consider such Reeb graphs as

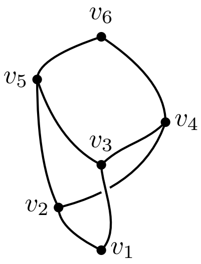


Figure 2 A Reeb graph that is not level planar. The function is given by $f(v_i) = i$.

our main object of study, rather than the spaces that give rise to Reeb graphs.

▶ **Definition 2.1.** A *Reeb graph* is a pair (G, f) consisting of a directed graph $G = (V, E)$ and a function $f\colon |G| \to \mathbb{R}$, such that $f(u) < f(v)$ for any $(u, v) \in E$, and f interpolates linearly along edges. Note that the function f is uniquely determined by its values on vertices. Unless otherwise noted, we assume that G is connected.

A *drawing* of a graph G is a continuous map $\phi\colon |G| \to \mathbb{R}^2$ that maps its geometric realization to the plane. Vertices are drawn at points in the plane, and each edge is drawn as a path connecting its vertices. As we often want to study the coordinates separately, we write $\phi =: (\phi_x, \phi_y)$. In the context of Reeb graphs, we are interested in drawings in which one of the coordinates represents the associated function. Specifically, a *level drawing* of a Reeb graph (G, f) is a drawing ϕ of G such that $\phi_y = f$. Therefore, a level drawing ϕ of a Reeb graph is uniquely determined by ϕ_x.

A drawing is *plane* if it is injective, and a Reeb graph is (level) *planar* if it admits a plane (level) drawing. Although the underlying graph G of a level planar Reeb graph (G, f) is necessarily planar, a Reeb graph (G, f) may not be level planar even if G is planar, see Figure 2.

3 Crossing radius

We will now introduce crossing radius as a quality measure of level drawings of Reeb graphs (G, f). Underlying this quality measure will be a family of metrics $d_y\colon f^{-1}(y) \times f^{-1}(y) \to \mathbb{R}$. For two points of $|G|$ with the same function value y, their distance under d_y is the minimum value such that $|G|$ contains a path from p to q whose function values deviates at most r from the value y (at its endpoints), see Figure 3. Equivalently, for two points $p, q \in |G|$ with

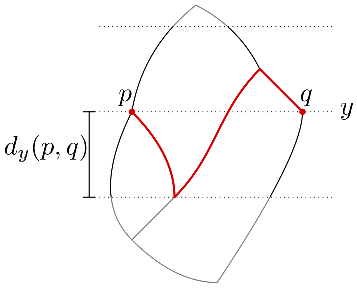


Figure 3 The distance between p and q under d_y is given by the length of the marked interval.

$f(p) = f(q) = y$, we define $d_y(p, q)$ to be the minimum radius r such that p and q lie in the same connected component of $f^{-1}([y - r, y + r]) \subseteq |G|$.

For a given level drawing $\phi\colon |G| \to \mathbb{R}^2$ of (G, f), we define its *crossing radius* to be the maximum value $d_y(p, q)$ over all crossings $\phi(p) = \phi(q) =: (x, y)$ between distinct points p and q. We define the *crossing radius* of a Reeb graph to be the minimum crossing radius over all its level drawings.

4 NP-Hardness

We show by reduction from planar monotone 3-SAT that it is NP-hard to find a level drawing with optimal crossing radius. The *Planar monotone 3-SAT* problem is NP-hard [4] and asks whether a given formula ψ with the following properties is satisfiable. The formula ψ has n Boolean variables $x_1, \ldots, x_n$ and is of the form $\bigwedge_{i=1}^{m} C_i$, where each C_i is a clause. Each clause has three variables, and is either positive (of the form $x_i \vee x_j \vee x_k$) or negative (of the form $\neg x_i \vee \neg x_j \vee \neg x_k$). Additionally, there is a restriction in terms of the graph whose vertices correspond to variables and clauses, and whose edges represent the containment of variables in clauses. The formula has the property that this graph has a planar layout in which all variables lie on a horizontal line, all positive clauses lie above that line, all negative clauses lie below that line, and no edge crosses that horizontal line, see Figure 4 (left). For our purposes, it will be more useful to derive a different layout of the same graph, in which variables lie on a vertical segment, and all positive (resp. negative) clauses lie above and to right (resp. left) of this segment, where again no edge crosses the vertical line through the vertices, see Figure 4 (right). We may assume this layout to be part of the input. Call ψ a YES instance if it is satisfiable, and a NO instance otherwise.

For brevity, we will call a level drawing of a Reeb graph *cheap* if it has crossing radius at most r, and *expensive* otherwise. We show that deciding whether a Reeb graph has a cheap drawing is NP-hard. For this, we construct, given Planar monotone 3-SAT formula ψ, a Reeb graph (G, f) that has crossing radius at most r if and only if ψ is a YES instance. Figure 5 shows the Reeb graph corresponding to Figure 4. On a global level, our Reeb graph will mimic the graph and layout accompanying the planar monotone 3-SAT instance. Vertices are replaced by clause gadgets and variable gadgets, and edges by wires and split gadgets.

An important and reoccurring gadget in the construction is a *switch* gadget, see Figure 6(a).

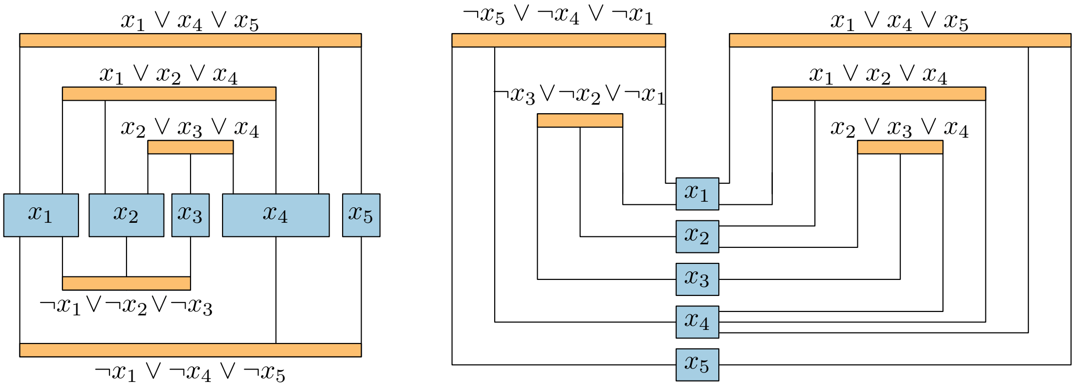


Figure 4 (left) The planar graph corresponding to an example instance of planar monotone 3-SAT. Vertices (clauses and variables) are shaded. (right) A different layout of the same graph that will be more useful for our purposes.

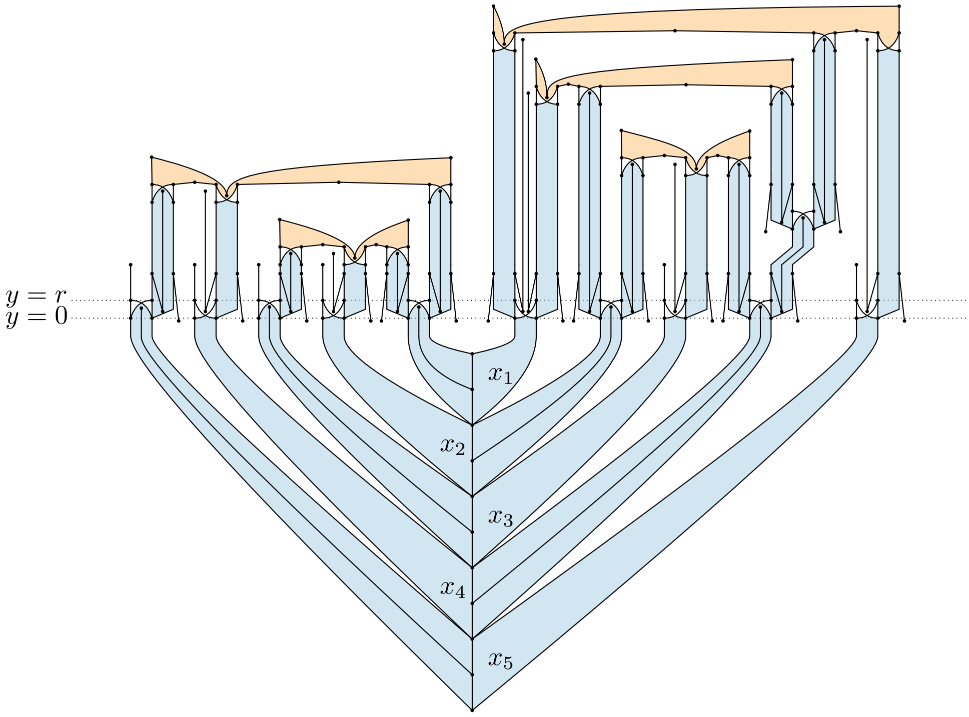


Figure 5 The reduction for a YES instance with satisfying assignment $(x_1, \neg x_2, x_3, \neg x_4, x_5)$.

A switch gadget S has two vertices at some level y and two vertices at level $y + r$. Both vertices at level y have edges to both vertices at level $y+r$. By itself, any drawing of a switch gadget will have crossing radius at most r, and for any YES instance, the Reeb graph $|G|$ will admit a drawing in which every crossing pair of points lies on a switch gadget (such as in Figure 5). A switch gadget has two drawings that may be desirable, which we call the up and the down positions. In both positions, S has a vertical left and right edge, and the other two edges have a crossing near level $y + r$ (in the up position) or y (in the down position).

Roughly speaking, in the up position, there will be space for part of $|G|$ to be drawn in the bottom part of the switch gadget (between its left and right edges), whereas in the down position, there will be space for part of $|G|$ to be drawn in the top part. More formally, we

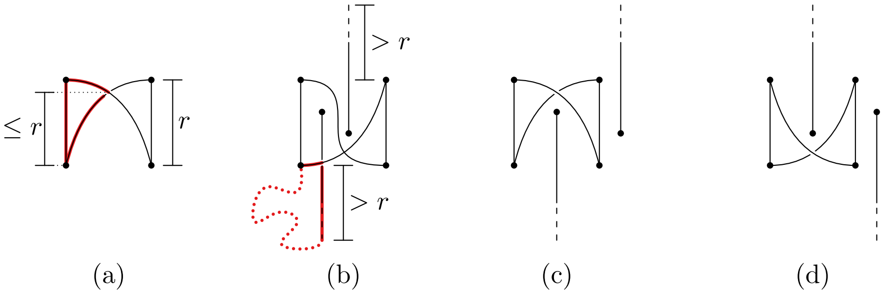


Figure 6 (a) A switch gadget with a cheap drawing (and the corresponding path of radius $\leq r$ marked in red). (b) A drawing with an expensive crossing, as indicated by the red path of radius greater than r. (c/d) a switch gadget drawn in the up/down position allows something to be drawn inside it on the bottom/top.

say that for some drawing, some subspace H of $|G|$ with $f(H) \subseteq [y, y + r]$ is drawn *inside* S if all of H lies between the left and right envelopes[1] of S.

We can now describe the behavior of S in detail. Let U and D be nonempty subspaces of $|G|$, such that for any $s \in S$ and any p in U or D at the same level, we have $d_{f(s)}(p, s) > r$, so that no point of U or D may cross S in any cheap drawing. Assume that for each component C of U, we have $[y + r/2, y + r] \subseteq f(C) \subseteq (y, y + r]$, and for each component C of D, we have $[y, y + r/2] \subseteq f(C) \subseteq [y, y + r)$. There exist drawings of $S \cup U \cup D$ such that either all of U lies inside S or all of D lies inside S, and there are no crossings between different components, see Figure 6(c) and (d). However, there exists no cheap drawing in which U and D simultaneously lie inside S, see Figure 6(b).

Before elaborating on the remainder of the construction, we remark that although the existence of cheap drawings for YES instances will be fairly straightforward, establishing the non-existence of cheap drawings for NO instances is quite involved. We will therefore dedicate the remainder of this article to providing an intuition for the construction and the non-existence of cheap drawings for NO instances.

We refer back to the visual overview of Figure 5 to describe the various components of the construction. The variable, clause, and split gadgets in our construction always connect to each other using switch gadgets. The bottom half of the construction consists entirely of variable gadgets. The variable gadgets are labeled x_i and lie on a vertical path which we call the *spine*. The spine consists of $2n$ edges of length $2r$. This ensures that edges connected to the spine at different heights cannot cross each other in any cheap drawing. Edges connected at the same height can cross, but only near the point where they attach to the spine. Every variable gadget connects to two switch gadgets at $y = 0$. These switch gadgets cannot cross each other and ensure that the edges connected to the spine at the same height must essentially lie on different sides of the spine. From this, we can conclude that any cheap drawing will place the two switch gadgets of x_i between the switch gadgets of x_{i+1}. Every variable gadget x_i connects to an isolated vertex at $y = r/2$, that must lie inside one of the two attached switch gadgets, so nothing can enter that switch gadget from above. The n vertices at $y = r/2$ encode a Boolean assignment to the variables. If a variable occurs in k positive (resp. negative) clauses, then $k - 1$ split gadgets will propagate a negative (resp. positive) assignment of that variable to those clauses by preventing anything from entering the split gadgets from above. Each clause is attached from above to three split gadgets, and in any cheap drawing of the clause, at least one of the split gadget must allow something to be placed inside from above (see Figure 7).

Acknowledgments. The authors wish to thank the reviewers for their helpful comments.

[1] For standard drawings such as the up and down position, these left and right envelopes will simply be the left and right edges. However, to handle adversarial drawings, we define the inside using envelopes.

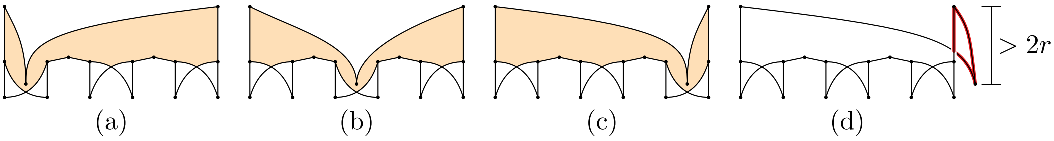


Figure 7 (a)-(c) Satisfying assignments of a clause gadget. (d) A high crossing radius in case of an unsatisfying assignment.

References

1 Christian Bachmaier, Hedi Buchner, Michael Forster, and Seok-Hee Hong. Crossing minimization in extended level drawings of graphs. *Discrete Applied Mathematics*, 158(3):159–179, 2010. doi:10.1016/j.dam.2009.09.002.

2 S. Biasotti, D. Giorgi, M. Spagnuolo, and B. Falcidieno. Reeb graphs for shape analysis and applications. *Theoretical Computer Science: Computational Algebraic Geometry and Applications*, 392(13):5 – 22, 2008. doi:10.1016/j.tcs.2007.10.018.

3 Erin Wolf Chambers, Elizabeth Munch, and Tim Ophelders. A Family of Metrics from the Truncated Smoothing of Reeb Graphs. In *proc. 37th International Symposium on Computational Geometry (SoCG)*, pages 22:1–22:17, 2021. doi:10.4230/LIPIcs.SoCG.2021.22.

4 Mark de Berg and Amirali Khosravi. Optimal binary space partitions in the plane. In *Computing and Combinatorics*, pages 216–225, 2010.

5 Vin de Silva, Elizabeth Munch, and Amit Patel. Categorified Reeb graphs. *Discrete & Computational Geometry*, pages 1–53, 2016. doi:10.1007/s00454-016-9763-9.

6 Herbert Edelsbrunner, John Harer, and Amit K. Patel. Reeb spaces of piecewise linear mappings. In *proc. 24th Annual Symposium on Computational Geometry (SoCG)*, pages 242–250, 2008. doi:10.1145/1377676.1377720.

7 Graeme Gange, Peter J. Stuckey, and Kim Marriott. Optimal k-level planarization and crossing minimization. In *proc. 18th International Symposium on Graph Drawing (GD)*, pages 238–249, 2010. doi:10.1007/978-3-642-18469-7_22.

8 William Harvey, Yusu Wang, and Rephael Wenger. A randomized $O(m \log m)$ time algorithm for computing Reeb graphs of arbitrary simplicial complexes. In *proc. 26th annual Symposium on Computational Geometry (SoCG)*, pages 267–276, 2010. doi:10.1145/1810959.1811005.

9 Michael Jünger and Sebastian Leipert. Level planar embedding in linear time. In *proc. 7th International Symposium on Graph Drawing (GD)*, pages 72–81, 1999.

10 Michael Jünger, Sebastian Leipert, and Petra Mutzel. Level planarity testing in linear time. In *proc. 6th International Symposium on Graph Drawing (GD)*, pages 224–237, 1998.

11 Boris Klemz and Günter Rote. Ordered level planarity and its relationship to geodesic planarity, bi-monotonicity, and variations of level planarity. *ACM Trans. Algorithms*, 15(4), 2019. doi:10.1145/3359587.

12 Vitaliy Kurlin. Book embeddings of Reeb graphs. *Preprint*, 2013. arXiv:1312.1725v1.

13 Salman Parsa. A deterministic $O(m \log m)$ time algorithm for the Reeb graph. In *proc. 28th annual Symposium on Computational geometry (SoCG)*, 2012.

14 Georges Reeb. Sur les points singuliers d'une forme de Pfaff complèment intégrable ou d'une fonction numérique. *Comptes rendus de l'Académie des Sciences*, 222:847–849, 1946.

15 Lin Yan, Talha Bin Masood, Raghavendra Sridharamurthy, Farhan Rasheed, Vijay Natarajan, Ingrid Hotz, and Bei Wang. Scalar field comparison with topological descriptors: Properties and applications for scientific visualization. *Preprint*, 2021. arXiv:2106.00157.

The Shortest Path with Increasing Chords in a Simple Polygon

Mart Hagedoorn[1] and Irina Kostitsyna[2]

1 TU Dortmund, Germany
mart.hagedoorn@tu-dortmund.de

2 TU Eindhoven, The Netherlands
i.kostitsyna@tue.nl

Abstract

We study the problem of finding the shortest path with *increasing chords* in a simple polygon. A path has increasing chords if and only if for any points a, b, c, and d that lie on the path in that order, $|ad| \geq |bc|$. In this paper we show that the shortest path with increasing chords is unique and present an algorithm to construct it.

Related Version arXiv:2202.12131

1 Introduction

An s-t path σ has *increasing chords* if and only if σ is a directed path from some point s to t and for any points a, b, c, and d that appear in that order on σ, the Euclidean distance between a and d is greater or equal to the Euclidean distance between b and c [8]. Figure 1 shows an example of a shortest s-t path with increasing chords in simple polygon P. Furthermore, a path has increasing chords if and only if the path is *self-approaching* in both directions. A path is self-approaching if, when traversing the path the Euclidean distance to any point on the remainder of the path is not increasing.

Paths with increasing chords are closely related to beacon and greedy routing applications. Path finding with beacon and greedy routing often results in paths that are directed curves such that the distance to the destination is never increasing [2, 1, 4]. Paths with this property are called *radially monotone paths.* If a path has increasing chords, then every subpath of that path is radially monotone in both directions.

Furthermore, self-approaching paths and paths with increasing chords have the property that the length of the path is bounded in comparison with the Euclidean distance between the start and destination of the path. This bounding factor of paths with increasing chords

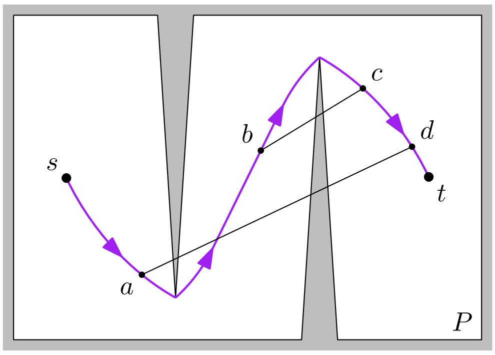


Figure 1 The shortest s-t path with increasing chords inside simple polygon P.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

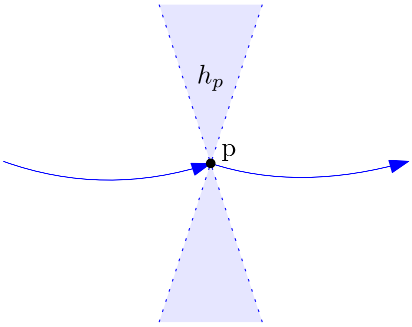


Figure 2 The normal h_p of p, where p is a bend point.

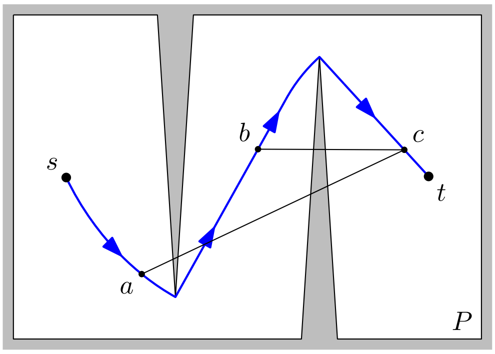


Figure 3 The shortest self-approaching s-t path inside simple polygon P.

is only $2\pi/3$, whereas the bounding factor of self-approaching paths is approximately 5.3331 [9, 7].

The results of this paper are further discussed in the master's thesis of Hagedoorn [6].

2 Preliminaries

An s-t path ζ is a directed curve that starts in point s and ends in point t. Moreover, path ζ must lie entirely inside of polygon P, i.e. $\zeta \subseteq P$. We define $\zeta(p, q)$ to be the subpath of ζ that lies between points p and q. Paths in a Euclidean space can make smooth turns and sharp turns. In order to differentiate between these types of turns, we use the standard notation of a *bend point* (Fig. 2). A *bend point* b of a piecewise smooth curve ζ is a point where the first derivative of ζ is discontinuous.

Furthermore, we define a *normal* h_p to path ζ at point p to be the line through $p \in \zeta$ such that h_p is perpendicular to the tangent of ζ in p. If p is a bend point, then we use the normal definition as proposed by Icking et al. [7]. This definition states that the normal to ζ at p is the set of lines that are included in the double wedge between the perpendicular lines to the tangents of the smooth pieces meeting at p (Fig. 2).

2.1 Self-approaching paths

Self-approaching paths were first described by Icking et al. [7]. A path π is self-approaching if and only if for any points a, b, and c that appear on π in that order, the Euclidean distance between a and c is greater or equal to the Euclidean distance between b and c (Fig. 3). Furthermore, Icking et al. [7] showed the following *normal property* of self-approaching paths.

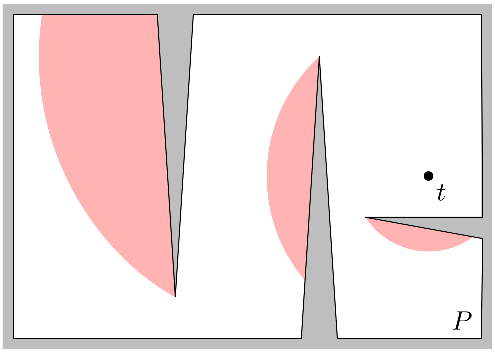


Figure 4 Dead region $\mathcal{D}_t$ (red) for some point t in simple polygon P.

▶ **Lemma 1** ([7]). *An s-t path π is self-approaching if and only if the normal to π at any point $p \in \pi$ does not intersect the subpath $\pi(p, t)$.*

Bose et al. [3] proposed an algorithm for finding the shortest *self-approaching s-t* path in a simple polygon P (Fig. 3). Their algorithm uses the notion of *dead regions*. A dead region $\mathcal{D}_t$ for point t is a set of points such that for any point $s \in \mathcal{D}_t$, no self-approaching s-t path exists (Fig. 4). They proved that the shortest self-approaching s-t path π is the s-t geodesic in $P \setminus \mathcal{D}_t$, i.e. the set difference of the polygon and the dead region for point t. Therefore, the shortest self-approaching s-t path consists of straight line segments and segments that are boundaries of $\mathcal{D}_t$. Moreover, they showed that if v is a point on π and v lies on a boundary of $\mathcal{D}_t$, then the normal h_v to π at v must *touch* the subpath $\pi(v, t)$. A line ℓ touches a curve γ if there exists some point p such that $\ell \cap \gamma = \{p\}$ and for the normal h_p to γ at point p, $\ell \in h_p$ or $\ell = h_p$ when p is a bend point or not, respectively.

The equations that define boundaries of $\mathcal{D}_t$ are transcendental equations and likely cannot be solved or evaluated analytically [3]. Therefore, the shortest self-approaching path can only be found if we assume these equations can be solved or that an approximation of the path will be calculated.

2.2 Paths with increasing chords

As mentioned before, a path σ has increasing chords if and only if σ is self-approaching in both directions. Furthermore, for paths with increasing chords we can again define the normal property:

▶ **Lemma 2.** *An s-t path σ has increasing chords if and only if the normal to σ at any point $p \in \sigma$ does not intersect the subpaths $\sigma(s, p)$ and $\sigma(p, t)$.*

Proof. A path σ with increasing chords is self-approaching from s to t and from t to s. Therefore, at any point $p \in \sigma$ the normal properties of the self-approaching s-t and t-s paths state that $\sigma(s, p)$ and $\sigma(p, t)$ cannot cross the normal to σ at point p. ◀

This property above can be reformulated in terms of the *negative* and *positive half-plane*. Using the definition from Bose et al. [3], the positive half-plane h_p^+ of path ζ at point p is the closed half-plane that is defined by normal h_p to ζ and contains points $q \in \mathbb{R}^2$ such that vector $\vec{pq}$ makes a non-negative dot product with the tangent vector at point p. Analogously, the negative half-plane h_p^- contains points $q \in \mathbb{R}^2$ such that vector $\vec{pq}$ makes a non-positive dot product with the tangent vector at point p.

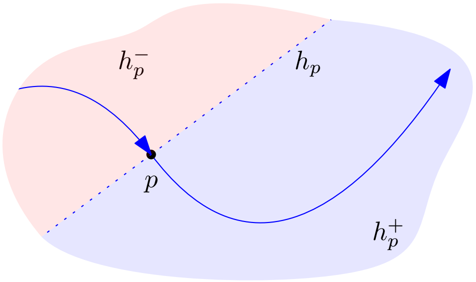


Figure 5 Example figure showing the normal h_p at point p and the corresponding half planes h_p^- and h_p^+.

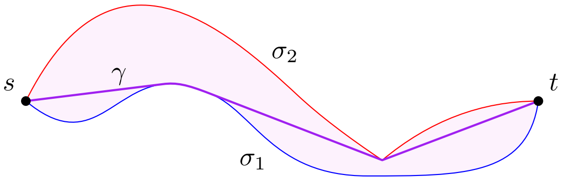


Figure 6 Geodesic γ (purple) that lies between paths σ_1 (blue) and σ_2 (red) with increasing chords.

▶ **Corollary 3.** *An s-t path σ has increasing chords if and only if, for any line h normal to σ at point $p \in \sigma$, the subpath $\sigma(s,p)$ lies completely in the negative half-plane h_p^- and the subpath $\sigma(p,t)$ lies completely in the positive half-plane h_p^+.*

3 Shortest Path with Increasing Chords

In this section we first show that a shortest s-t path with increasing chords in a simple polygon is unique. Therefore, we only need to search for one shortest s-t path with increasing chords. The following two proofs are extensions from the analogous propositions about self-approaching paths given by Bose et al. [3]. The proofs of Lemma 4 and Theorem 5 can be found in [5].

▶ **Lemma 4.** *A geodesic path γ between two distinct paths with increasing chords σ_1 and σ_2 also has increasing chords (Fig. 6).*

Using Lemma 4 we can prove the following Theorem.

▶ **Theorem 5.** *A shortest s-t path with increasing chords in a simple polygon is unique.*

In the next theorem we show that if we subtract the union of dead regions $\mathcal{D}_t$ and $\mathcal{D}_s$ from a simple polygon P, then the shortest s-t path in this space is also the shortest s-t path with increasing chords. This theorem is proven by contradiction. In more detail, we show that if there is a point where the normal property is violated, then no self-approaching path can exist.

▶ **Theorem 6.** *Let s and t be two points in a simple polygon P. The shortest path between s and t in $P \setminus (\mathcal{D}_t \cup \mathcal{D}_s)$ is the shortest s-t path with increasing chords in simple polygon P.*

Proof. Assume that σ does not have increasing chords, hence the normal property is violated at some point of σ. Let point $p \in \sigma$ be the last point on σ for which the normal property

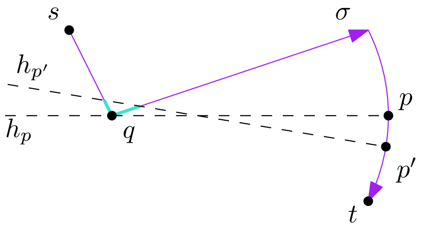


Figure 7 Shortest path σ (purple) where the normal property does not hold in p', normal h_p touches point q that lies on subpath ρ (blue) of σ.

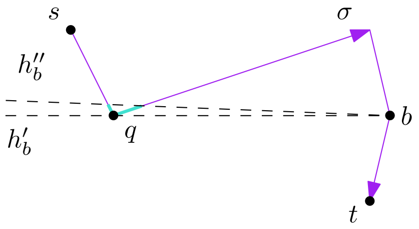


Figure 8 Shortest path σ (purple) where the normal property does not hold in p', normal h'_b touches point q that lies on subpath ρ (blue) of σ.

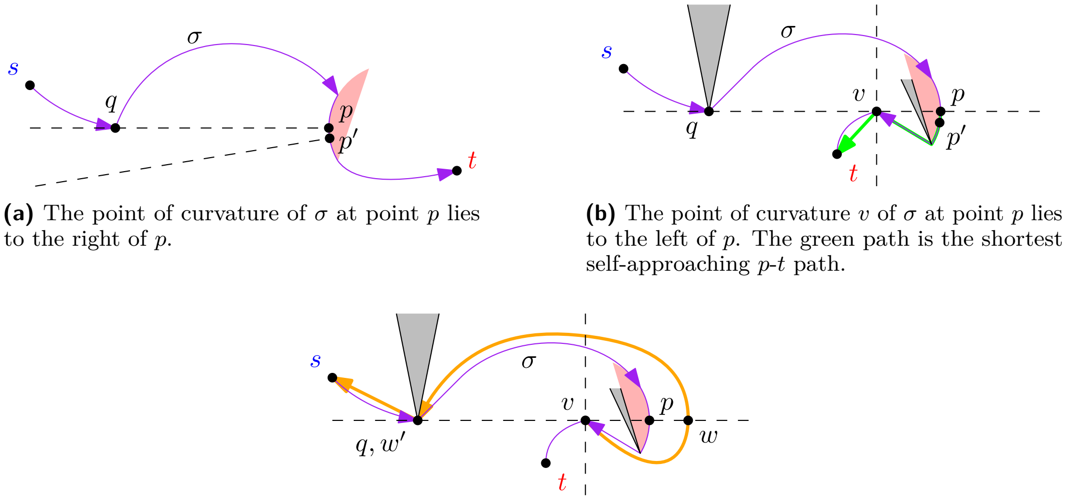
holds. Point p is not guaranteed to exist. However, if p exists, then there also exists point $p' \in \sigma(p,t)$ such that p' lies in the ϵ-neighborhood of p for arbitrary small ϵ and the normal property does not hold in p' (Fig. 7). Since in p' the normal property does not hold, there is some subpath ρ of $\sigma(s,p')$ or $\sigma(p',t)$ that lies in the positive or negative half-plane defined by the normal through h'_p, respectively. Furthermore, the normal h_p touches σ at some point $q \in \rho$. However, if p does not exist, there exists a bend point b such that there are two lines $h'_b, h''_b \in h_b$ among the set of lines in the normal h_b, such that h'_b touches σ at some point $q \in \sigma$ and h''_b intersects σ (Fig. 8). All cases where p does not exist are simply analogous to the cases where p exists. Therefore, we will only cover the cases where p exists.

We assume, without loss of generality, that line segment $\overline{pq}$ is horizontal, p lies to the right of q, and $\sigma(s,p)$ lies above h_p. Let e be the segment of σ containing p and f be the segment of σ containing q. We must consider the cases where $q \in \sigma(s,p)$ and $q \in \sigma(p,t)$. Furthermore, since σ is a geodesic in $P \setminus (\mathcal{D}_t \cup \mathcal{D}_s)$ the segments e and f can be straight line segments, or boundaries of a dead region of $\mathcal{D}_t$ or $\mathcal{D}_s$. If f is a straight line segment, then q must be an end point of f and thus be a vertex of polygon P. Here, we will only cover the case where $q \in \sigma(s,p)$, e is a boundary of $\mathcal{D}_t$, and q is a vertex of P. For each possible combination of the segment types of e and f and the location of q on the path σ we show that these cases are not possible by contradiction. In this paper we prove that one of the cases is not possible, the other possible cases follow similar ideas and can be found in [5].

Once we have shown that no point exists where the normal property is violated, σ indeed has increasing chords by Lemma 1. Furthermore, σ must be the shortest path with increasing chords. Any path that is shorter than σ must go through either of the dead regions $\mathcal{D}_t$ or $\mathcal{D}_s$. Therefore, any path shorter than σ cannot have increasing chords and σ must be the shortest path with increasing chords.

Point $q \in \sigma(s,p)$, e is a boundary of $\mathcal{D}_t$, and q is a vertex of P (Fig. 9)

The center of curvature of σ at p must lie to the left of p. Otherwise the normal $h_{p'}$ to σ cannot intersect $\sigma(s,p)$, as is depicted in Fig. 9a. Let π_{pt} be the shortest self-approaching path from p to t. Segment e is part of $\mathcal{D}_t$, therefore there must be a point $v \in \pi_{pt}$ touched by normal h_p (Fig. 9b). Because π_{pt} goes through v, h_v is perpendicular to h_p (if v is a bend point, then there must be a line which is perpendicular h_p in the set of normals h_v). Furthermore, $\pi_{pt}(v,t)$ lies completely in the positive half-plane h_v^+ by the half-plane property. We will now show that in the region between $\overline{pv}$ and $\pi_{pt}(p,v)$ there are vertices of P. The straight line segment $\overline{pv}$ concatenated to $\pi_{pt}(v,t)$ would be self-approaching. However, p' lies below h_p, thus polygon P must intersect with $\overline{vp}$. Therefore, the shortest self-approaching v-s path π_{vs} must first intersect or touch h_p to the right of v at point w and later to the



(a) The point of curvature of σ at point p lies to the right of p.

(b) The point of curvature v of σ at point p lies to the left of p. The green path is the shortest self-approaching p-t path.

(c) The point of curvature v of σ at point p lies to the left of p. The orange path is the shortest self-approaching v-s path, that intersects with h_p in w and w'.

Figure 9 Geodesic σ (purple) where point $q \in \sigma(s, p)$, p lies on a boundary of $\mathcal{D}_t$ (red), and q is a vertex of P.

left of v at point w' (Fig. 9c). Thus, $|vw'| < |ww'|$ which contradicts the self-approaching property of π_{vs}. Hence, this case is not possible if the geodesic between s and t exists. ◀

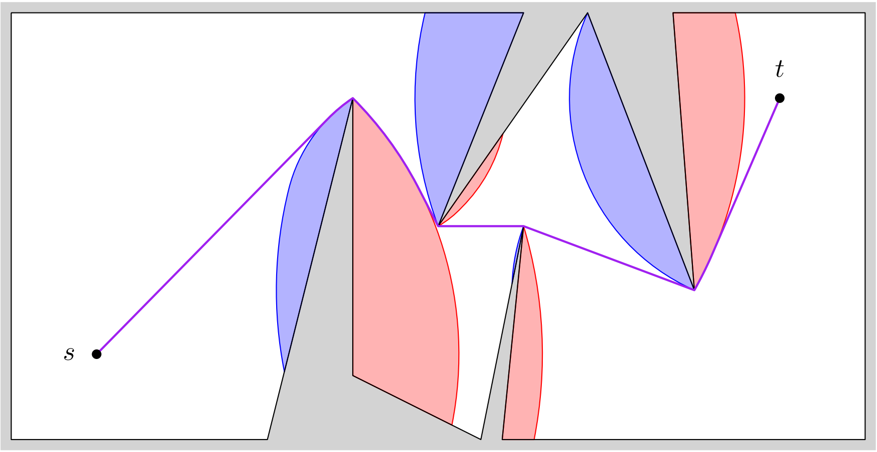


Figure 10 Polygon P with dead regions $\mathcal{D}_s$ (red) and $\mathcal{D}_t$ (blue), the geodesic (purple) of the remaining area is the shortest path with increasing chords.

Using Theorem 6, the shortest s-t path with increasing chords in a polygon P can be found by subtracting the dead regions $\mathcal{D}_t$ and $\mathcal{D}_s$ from polygon P (Fig. 10). Therefore, this shortest path with increasing chords can be found in a similar manner to finding the shortest self-approaching s-t path as described by Bose et al. [3]. As is the case for the algorithm for the shortest self-approaching path, the shortest path with increasing chords can only be found if we assume that transcendental equations can be solved or that an approximation of the path will be calculated.

4 Conclusions

In this paper, we showed that the shortest s-t path with increasing chords in a simple polygon is unique. Furthermore, we showed that the shortest s-t path in a polygon minus the dead regions of s and t is the shortest path with increasing chords. Therefore, the algorithm for finding the shortest path with increasing chords is similar to finding a shortest self-approaching path.

A natural direction for future research is the question of whether an efficient algorithm exists that can find a shortest s-t path with increasing chords in a polygon with holes. The possibility remains open that this problem is NP-hard.

References

1 M. Biro, J. Gao, J. Iwerks, I. Kostitsyna, and J.S.B. Mitchell. Beacon-based routing and coverage. In *21st Fall Workshop on Computational Geometry (FWCG 2011)*, 2011.

2 M. Biro, J. Gao, J. Iwerks, I. Kostitsyna, and J.S.B. Mitchell. Combinatorics of beacon-based routing and coverage. In *Proceedings of the 25th Canadian Conference on Computational Geometry (CCCG)*, pages 1–6, 2013. URL: https://cs.uwaterloo.ca/conferences/cccg2013/.

3 P. Bose, I. Kostitsyna, and S. Langerman. Self-approaching paths in simple polygons. *Computational Geometry*, 87, 2020. doi:10.1016/j.comgeo.2019.101595.

4 J. Gao and L. Guibas. Geometric algorithms for sensor networks. *Philosophical transactions. Series A, Mathematical, physical, and engineering sciences*, 370:27–51, 2012. doi:10.1098/rsta.2011.0215.

5 Mart Hagedoorn and Irina Kostitsyna. The shortest path with increasing chords in a simple polygon, 2022. arXiv:2202.12131.

6 M.H. Hagedoorn. Self-approaching paths and paths with increasing chords in polygonal domains. Master's thesis, TU Eindhoven, 2021.

7 C. Icking, R. Klein, and E. Langetepe. Self-approaching curves. *Mathematical Proceedings of the Cambridge Philosophical Society*, 125, 05 2002. doi:10.1017/S0305004198003016.

8 D.G. Larman and P. McMullen. Arcs with increasing chords. *Mathematical Proceedings of the Cambridge Philosophical Society*, 72(2):205–207, 1972. doi:10.1017/S0305004100047022.

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

Spanning ratio of shortest paths in weighted square tessellations*

Prosenjit Bose[1], Guillermo Esteban[1,2], David Orden[2], and Rodrigo I. Silveira[3]

1 **School of Computer Science, Carleton University, Canada**
`jit@scs.carleton.ca`

2 **Departamento de Física y Matemáticas, Universidad de Alcalá, Spain**
`{g.esteban, david.orden}@uah.es`

3 **Departament de Matemàtiques, Universitat Politècnica de Catalunya, Spain**
`rodrigo.silveira@upc.edu`

Abstract

Continuous 2-dimensional space is often discretized by considering a grid of weighted square cells. In this work we study how well a weighted tessellation approximates the space, with respect to shortest paths. In particular, we consider a shortest path $SP_w(s,t)$, which is a shortest path from s to t in the continuous weighted 2D-space, and a shortest grid path $SGP_w(s,t)$, which is a shortest path in the tessellated weighted 2D-space. Our main result is that the ratio $\frac{\|SGP_w(s,t)\|}{\|SP_w(s,t)\|}$ is at most $\frac{2}{\sqrt{2+\sqrt{2}}} \approx 1.08$, irrespective of the weight assignment.

1 Introduction

Geometric shortest path problems are a class of computational geometry problems where the goal is to find an optimal path between two points s and t in a certain setting. Many variations of shortest path problems exist, depending on the domain (e.g., discrete, continuous), the objective function (e.g., Euclidean metric, link-distance, geodesic distance), or specific domain constraints (e.g., obstacles in the plane, or holes in polygons).

An important shortest path problem is computing an optimal path in a geometric domain when the cost of traversing the domain varies depending on the region. That is, the domain consists of a weighted planar polygonal subdivision. Each region i of the subdivision has a weight ω_i, which represents the cost per unit of distance of traveling in that region. Thus, the cost of traversing a region is typically given by the Euclidean distance traversed in the region, multiplied by the corresponding weight. The resulting metric is often called the *weighted region metric*, and the problem of computing a (weighted) shortest path between two points under this metric is known as the *weighted region problem* (WRP) [19]. The WRP is very general, since it allows to model many well-known variants of geometric shortest path problems [6, 17].

Existing algorithms for the WRP are quite complex in design and implementation or have very high time and space complexities [1, 2, 3]. They are sophisticated methods that usually are based on variants of continuous Dijkstra, partitioning each edge of the subdivision in parts for which crossing shortest paths have the same combinatorial structure (e.g., [19]), or work by computing a discretization of the domain by carefully placing Steiner points

* P. B. is partially supported by NSERC. G. E., D. O. and R. I. S. are partially supported by H2020-MSCA-RISE project 734922 - CONNECT and project PID2019-104129GB-I00 funded by MCIN/AEI/10.13039/501100011033. G. E. and D. O. are also supported by PIUAH21/IA-062 and CM/JIN/2021-004. G. E. is funded by an FPU of the Universidad de Alcalá.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

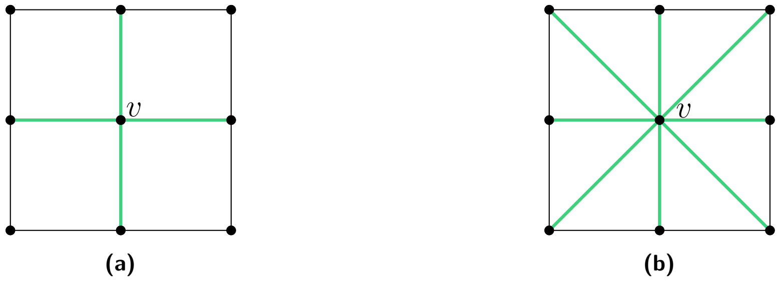


Figure 1 Vertex v is connected to its neighbors in $G_{4\text{corner}}$ (left) and in $G_{8\text{corner}}$ (right).

(e.g., see [10]). The lack of exact algorithms for WRP is probably justified by algebraic reasons: the problem has been recently proven to be impossible to solve in the Algebraic Computation Model over the Rational Numbers [11]. Efficient algorithms for WRP only exist for a few special cases, e.g., rectilinear subdivisions with L_1 metric [9], or weights restricted to $\{0, 1, \infty\}$ [14].

Real-time applications where the WRP arises, like robotics [13, 22, 23], gaming [16, 24] or geographic information science [12], feature increasingly large amounts of information. All these data are expected to be managed efficiently in terms of execution time and solution quality. This emphasizes the need for high-quality approximate paths instead of optimal paths, so, in practice, the problem is simplified in two ways. First, the domain is approximated by considering a (weighted) tessellation, a subdivision with a simpler structure. Secondly, optimal shortest paths in that simpler subdivision are approximated. The main technique for decomposing the continuous space $\mathbb{R}^2$ into cells is by using *navigation meshes* [25], subdivisions of the walkable space into polygonal regions that do not intersect. Regular grids, convex polygons, or disks—of different sizes—are among the most frequently used region shapes [25]. Navigation meshes allow efficient path planning in large environments as long as the region weights are limited to $\{1, \infty\}$ (i.e., obstacles only). The simplest cell decomposition technique to define and implement are *regular grids* since they are defined by the length of a side of a cell and a reference coordinate. In a 2-dimensional space, only three types of regular polygons can be used to tessellate continuous 2D environments, namely triangles, squares and hexagons. The drawback with a grid is that it suffers from digitization bias [18], which can be reduced by decreasing the size of the grid cells. However, this increases the number of cells or regions. Still, even in the more general case where each region has an assigned weight, regular grids are often used as navigation meshes, since they are easy to implement, are a natural choice for environments that are grid-based by design (e.g., many game designs), and popular shortest path algorithms such as A^* can be optimized for grids [20]. Thus, in practice, exact shortest paths are not computed: instead, an approximation is considered by computing a shortest path on a weighted graph associated to the grid, called *k-corner grid graph* ($G_{k\text{corner}}$) [5].

In a k-corner grid graph the vertex set is the set of corners of the tessellation, and each vertex is connected by an edge to a predefined set of k neighboring vertices, depending on the tessellation and other design decisions. See Figures 1a and 1b for the 4-corner and the 8-corner grid graph in a square tessellation, respectively. (Analogous k-corner grid graphs can be defined for triangular and hexagonal tessellations.) When the continuous space is tessellated, each cell S_i has a weight $\omega_i \in \mathbb{R}_{>0}$, and the cost of a segment π_i traversing cell S_i is given by $\omega_i\|\pi_i\|$, where $\|\cdot\|$ is the Euclidean norm. In the case where a segment π goes along the boundary of two cells S_j and S_k, the cost is $\min\{\omega_j, \omega_k\}\|\pi\|$.

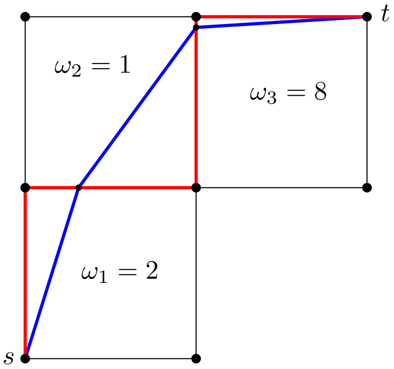


Figure 2 The cost of $SP_w(s,t)$ (blue) and a $SGP_w(s,t)$ (red) is 22.53, and 24, respectively, for a cell side length of 2 in $G_{4\text{corner}}$.

A shortest path in the continuous weighted space will be denoted $SP_w(s,t)$. A path in $G_{k\text{corner}}$ is called a *grid path*; a shortest grid path between the two vertices s and t will be denoted $SGP_w(s,t)$. See the blue path $SP_w(s,t)$, and the red path $SGP_w(s,t)$ in Figure 2 for a comparison between these two types of paths in $G_{4\text{corner}}$. Note that in all figures in this work, cells that are not depicted are considered to have infinite weight. Exact shortest paths for regions with weights in $\{0,1,\infty\}$ were studied in [17, 19]. Also, shortest grid paths for the case of weights of the cells being 1 or ∞ have been previously studied in [4].

Since $SGP_w(s,t)$ is considered as an alternative to $SP_w(s,t)$, the aim of this work is to quantify the relation between the weights of these two paths. In this paper we focus on $G_{8\text{corner}}$. In particular, we are interested in maximizing the ratio $R = \frac{\|SGP_w(s,t)\|}{\|SP_w(s,t)\|}$, since it indicates the worst possible approximation factor of a shortest grid path.

2 Previous results

Almost all previous bounds for the ratio R consider a limited set of weights for the cells. Nash [21] considered only weights in the set $\{1,\infty\}$ and proved that the weight of $SGP_w(s,t)$ in hexagonal $G_{6\text{corner}}$ and $G_{12\text{corner}}$, square $G_{4\text{corner}}$ and $G_{8\text{corner}}$, triangle $G_{6\text{corner}}$ and $G_{3\text{corner}}$ can be up to ≈ 1.15, ≈ 1.04, ≈ 1.41, ≈ 1.08, ≈ 1.15, and 2 times the weight of $SP_w(s,t)$, respectively. When the weights of the cells are allowed to be in $\mathbb{R}_{>0}$, the only previous results that we are aware of are a result by Jaklin [15] for square tessellations and another type of shortest path (with vertices at the center of the cells) showing that $R \le 2\sqrt{2}$, and our recent upper bound of $R = \frac{2}{\sqrt{3}}$ when weighted triangular cells are considered [7]. The main contribution of this paper is to apply a similar method as in [7] to a weighted square tessellation. However, in a square tessellation, we need a more exhaustive analysis since now we have more cases to analyze. Moreover, we have to consider other types of grid paths that help us to improve the worst-case ratio.

3 $\frac{\|SGP_w(s,t)\|}{\|SP_w(s,t)\|}$ ratio in $G_{8\text{corner}}$ for square cells

We are interested in maximizing the ratio $\frac{\|SGP_w(s,t)\|}{\|SP_w(s,t)\|}$ when each vertex of the graph is placed at the corners of the square cells, and is connected to 8 neighboring vertices, i.e., in $G_{8\text{corner}}$. We will assume that $SP_w(s,t)$ is unique. Furthermore, we will consider from now on that every cell has a positive real value associated to them. Due to space limitations, we defer to the full version of the paper the full explanation of this section.

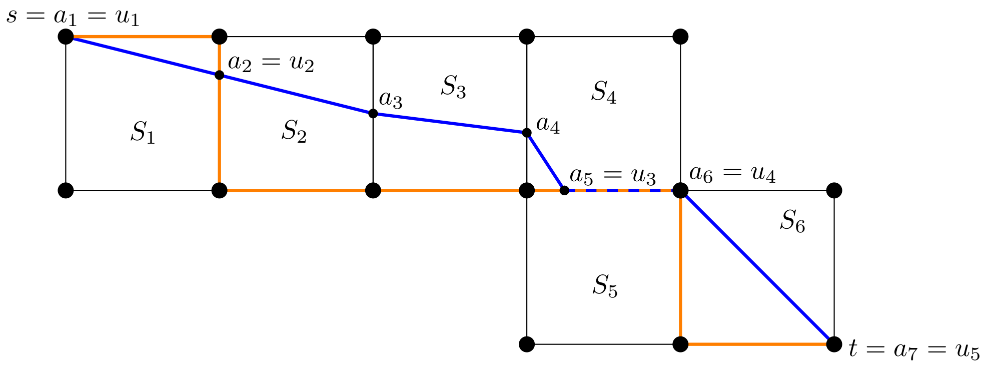


Figure 3 Shortest path $SP_w(s,t)$ (blue) and crossing path $X(s,t)$ (orange) from s to t.

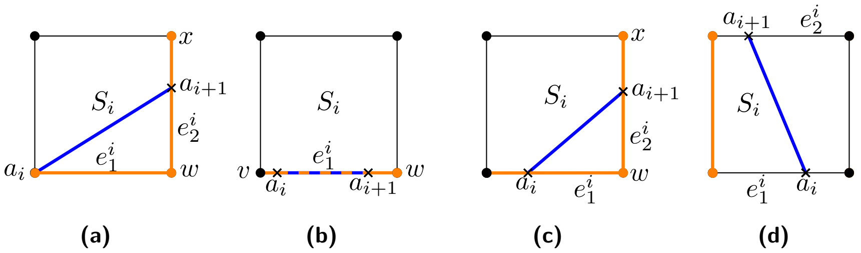


Figure 4 The sequence X_i of vertices that defines the crossing path $X(s,t)$ in a square cell S_i is depicted in orange. The path $SP_w(s,t)$ is shown in blue; and $X(s,t)$, in orange.

3.1 Crossing path and weakly simple polygons

$SGP_w(s,t)$ and $SP_w(s,t)$ can be very different in both shape and length. Thus, we need to define a restricted class of grid paths, called *crossing paths* $X(s,t)$), whose behavior will be easier to control. Since $X(s,t)$ is a grid path, its length is an upper bound for the length of the shortest grid path.

Let $(S_1,\dots,S_n)$ be the ordered sequence of consecutive cells intersected by $SP_w(s,t)$ in a square tessellation $\mathcal{S}$. Let a_i and a_{i+1} be, respectively, the points where $SP_w(s,t)$ enters and leaves S_i, for $i \in \{1,\dots,n\}$, see Figure 3. Then, the crossing path $X(s,t)$ from a vertex s to a vertex t in $\mathcal{S}$ is defined by the sequence $(X_1,\dots,X_n)$, where X_i is a sequence of up to three vertices of S_i, determined by the pair (a_i, a_{i+1}). The key property of $X(s,t)$ is that we want it to traverse only the edges of the cells that $SP_w(s,t)$ traverses. In order to guarantee that, the exact sequence of vertices depends on a number of cases that are determined in the full version of the paper. Figure 4 depicts some of the sequences X_i.

Applying the mediant inequality to $X(s,t)$ and $SP_w(s,t)$, we first observe that the ratio $\frac{\|X(s,t)\|}{\|SP_w(s,t)\|}$ of the whole path can be upper-bounded by the maximum among all the ratios $\frac{\|X(u_i,u_{i+1})\|}{\|SP_w(u_i,u_{i+1})\|}$, where u_i and u_{i+1} are two consecutive points where $X(s,t)$ and $SP_w(s,t)$ coincide. In case $X(s,t)$ and $SP_w(s,t)$ share one or more segments, we define the corresponding points as the endpoints of each of these segments, see u_3 and u_4 in Figure 3.

Let $(s=u_1, u_2, \dots, u_\ell = t)$ be a sequence of consecutive points where $X(s,t)$ and $SP_w(s,t)$ coincide. The union of $SP_w(s,t)$ and $X(s,t)$ between two points u_j and u_{j+1}, for $1 \le j < \ell$, induces a weakly simple polygon (see [8] for a formal definition). We distinguish two types of weakly simple polygons. In the first type, called P_k^1, the points u_j and u_{j+1} belong to the same cell, and $X(u_j,u_{j+1})$ and $SP_w(u_j,u_{j+1})$ intersect $k+1$ different edges of the tessellation,

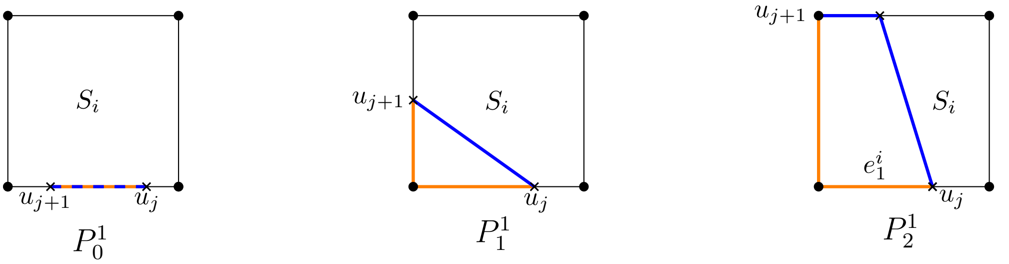


Figure 5 Weakly simple polygons P_k^1, for $k = \{0, 1, 2\}$, and the subpaths $SP_w(u_j, u_{j+1})$ (blue) and $X(u_j, u_{j+1})$ (orange) in a square tessellation.

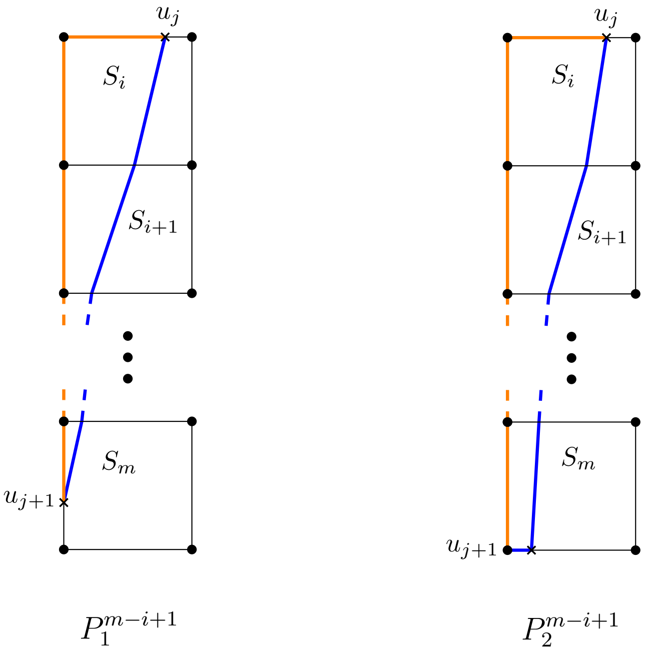


Figure 6 Weakly simple polygons P_1^{m-i+1} and P_2^{m-i+1}, and the subpaths of $SP_w(s, t)$ (blue) and $X(s, t)$ (orange) between u_j and u_{j+1} in a square tessellation.

for $0 \leq k \leq 2$, see Figure 5. In a square tessellation, it is possible for two consecutive crossing points to belong to cells that are not adjacent, see u_j and u_{j+1} in Figure 6. Hence, we need to define another type of weakly simple polygon, called P_k^{m-i+1}, where $m-i+1$ is the number of cells intersected by $SP_w(u_j, u_{j+1})$, and $k+1$ is the number of different edges of S_m intersected by $X(u_j, u_{j+1})$ and $SP_w(u_j, u_{j+1})$, for $k \in \{1, 2\}$, see Figure 6.

The full definition of the weakly simple polygons is deferred to the full version of the paper. However, one can show that, by the definition of the crossing path $X(s, t)$, these are the only weakly simple polygons that can arise. Thus, our goal will be to maximize the ratio $\frac{\|X(u_j,u_{j+1})\|}{\|SP_w(u_j,u_{j+1})\|}$, for all $j \in \{1, \ldots, \ell-1\}$, in each of the weakly simple polygons.

3.2 Bounding the ratio for weakly simple polygons

Before maximizing the ratio $\frac{\|X(u_j,u_{j+1})\|}{\|SP_w(u_j,u_{j+1})\|}$ in each of the weakly simple polygons, we will reduce the number of polygons. In this way, we will reduce the number of calculations. First, we can see that $\frac{\|X(u_j,u_{j+1})\|}{\|SP_w(u_j,u_{j+1})\|} = 1$ in P_0^1. Also, P_2^k is a particular case of P_1^{k+1}. Analogously,

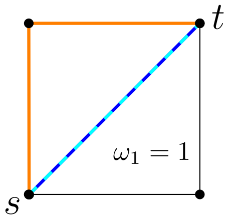


Figure 7 The ratio between the weight of $X(s,t)$ (orange) and the weight of $SP_w(s,t)$ (navy blue) is $\sqrt{2}$. The ratio between the weight of the cyan grid path and $SP_w(s,t)$ is 1.

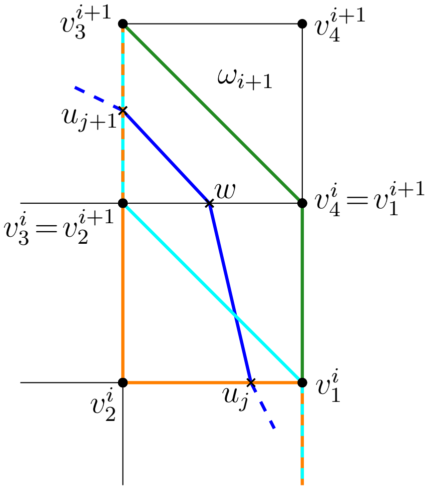


Figure 8 Subpaths of $SP_w(s,t)$ (blue), $X(s,t)$ (orange), $\Pi_i^1(s,t)$ (cyan), and $\Pi_i^2(s,t)$ (green). By definition of the shortcut paths, $X(s,v_1^i) = \Pi_i^1(s,v_1^i) = \Pi_i^2(s,v_1^i)$, $X(v_3^i,t) = \Pi_i^1(v_3^i,t)$, $X(v_3^{i+1},t) = \Pi_i^2(v_3^{i+1})$.

after some calculations, we are able to prove that the ratio in a weakly simple polygon of type P_1^k, for $k \geq 2$, is maximized when $k = 2$. Thus, using this result, we get that the only relevant spanning ratios are those in the polygons of type P_1^1, and P_1^2.

The edges of the crossing path $X(s,t)$ are edges of the tessellation. However, a grid path in $G_{8\text{corner}}$ can also use diagonals. Furthermore, $X(s,t)$ is defined based on the points where $SP_w(s,t)$ intersects the edges of the square cells, so it might not be a shortest grid path. Hence, the ratio $\frac{\|X(u_j,u_{j+1})\|}{\|SP_w(u_j,u_{j+1})\|}$ could be larger than the ratio $\frac{\|SGP_w(s,t)\|}{\|SP_w(s,t)\|}$, see Figure 7.

Thus, we define two additional classes of grid paths, called *shortcut paths* $\Pi_i^1(s,t)$ and $\Pi_i^2(s,t)$, see Figure 8. These paths intersect almost all the edges intersected by $X(s,t)$, and imply a better fit in case $SP_w(s,t)$ intersects two parallel edges or is close to the diagonal of a cell. Thus, they allow us to find a tighter upper bound for the ratio $\frac{\|SGP_w(s,t)\|}{\|SP_w(s,t)\|}$.

$\Pi_i^1(s,t)$ is defined when $X(s,t)$ intersects a P_1^1, or a P_1^2, while $\Pi_i^2(s,t)$ is just defined when $X(s,t)$ intersects a P_1^2. So, let $(S_k,\ldots,S_m)$ be the sequence of consecutive cells in $\mathcal{S}$ for which there exists a shortcut path $\Pi_i^1(s,t)$, $\Pi_i^2(s,t)$, $i \in \{k,\ldots,m\}$. The key idea to prove the upper bound in each of the weakly simple polygons, is to obtain a relation between the weights of $(S_k,\ldots,S_m)$. So, we prove that if the weights of $X(s,t)$, $\Pi_i^1(s,t)$ and $\Pi_i^2(s,t)$ are not equal, then the weights of some cells can be modified so that the ratio between the shortest grid path and the shortest path increases. Hence, the ratio $\frac{\|SGP_w(s,t)\|}{\|SP_w(s,t)\|}$ is maximized when $\|X(s,t)\| = \|\Pi_i^1(s,t)\| = \|\Pi_i^2(s,t)\|$ in $G_{8\text{corner}}$. Thus, using this fact, we proceed to calculate an upper bound on the ratio $\frac{\|X(u_j,u_{j+1})\|}{\|SP_w(u_j,u_{j+1})\|}$ in weakly simple polygons of type P_1^1 and P_1^2 when $\|X(s,t)\| = \|\Pi_i^1(s,t)\| = \|\Pi_i^2(s,t)\|$. However, even with this simplification, we need to split each ratio into different subcases to prove that $\frac{\|X(u_j,u_{j+1})\|}{\|SP_w(u_j,u_{j+1})\|} \leq \frac{2}{\sqrt{2+\sqrt{2}}}$. Finally, since $\|SGP_w(s,t)\| \leq \|X(s,t)\|$, and using the mediant inequality, we obtain our main result.

▶ **Theorem 3.1.** *In $G_{8corner}$, an upper bound on the ratio $\frac{\|SGP_w(s,t)\|}{\|SP_w(s,t)\|}$ is $\frac{2}{\sqrt{2+\sqrt{2}}} \approx 1.08$.*

4 Conclusions

We proved an upper bound for the ratio between the lengths of a shortest grid path in the 8-corner grid graph and a shortest path. Following an analogous procedure we can obtain an upper bound for the ratio $\frac{\|SGP_w(s,t)\|}{\|SP_w(s,t)\|}$ in $G_{4\text{corner}}$, as well as for hexagonal tessellations. The main differences lie in the number of weakly simple polygons to analyze. Also, we have to take into account a different number of shortcut paths.

References

1 L. Aleksandrov, M. Lanthier, A. Maheshwari, and J. R. Sack. An ε-approximation algorithm for weighted shortest paths on polyhedral surfaces. In *Scandinavian Workshop on Algorithm Theory*, pages 11–22. Springer, 1998.

2 L. Aleksandrov, A. Maheshwari, and J. R. Sack. Approximation algorithms for geometric shortest path problems. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 286–295, 2000.

3 L. Aleksandrov, A. Maheshwari, and J. R. Sack. Determining approximate shortest paths on weighted polyhedral surfaces. *Journal of the ACM (JACM)*, 52(1):25–53, 2005.

4 A. Ammar, H. Bennaceur, I. Chăari, A. Koubăa, and M. Alajlan. Relaxed Dijkstra and A^* with linear complexity for robot path planning problems in large-scale grid environments. *Soft Computing*, 20(10):4149–4171, 2016.

5 J. Bailey, C. Tovey, T. Uras, S. Koenig, and A. Nash. Path planning on grids: The effect of vertex placement on path length. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 11, 2015.

6 M. De Berg and M. van Kreveld. Trekking in the alps without freezing or getting tired. *Algorithmica*, 18(3):306–323, 1997.

7 P. Bose, G. Esteban, D. Orden, and R. I. Silveira. On approximating shortest paths in weighted triangular tessellations. *arXiv preprint arXiv:2111.13912*, 2021.

8 H. C. Chang, J. Erickson, and C. Xu. Detecting weakly simple polygons. In *Proceedings of the twenty-sixth annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1655–1670. SIAM, 2014.

9 D. Z. Chen, K. S. Klenk, and H. Y. Tu. Shortest path queries among weighted obstacles in the rectilinear plane. *SIAM J. Comput.*, 29(4):1223–1246, 2000. doi:10.1137/S0097539796307194.

10 S. W. Cheng, J. Jin, and A. Vigneron. Triangulation refinement and approximate shortest paths in weighted regions. In Piotr Indyk, editor, *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1626–1640. SIAM, 2015. doi:10.1137/1.9781611973730.108.

11 J. L. de Carufel, C. Grimm, A. Maheshwari, M. Owen, and M. Smid. A note on the unsolvability of the weighted region shortest path problem. *Computational Geometry*, 47(7):724–727, 2014.

12 L. de Floriani, P. Magillo, and E. Puppo. Applications of computational geometry to geographic information systems. *Handbook of computational geometry*, 7:333–388, 2000.

13 D. Gaw and A. Meystel. Minimum-time navigation of an unmanned mobile robot in a 2-1/2D world with obstacles. In *Proceedings. 1986 IEEE International Conference on Robotics and Automation*, volume 3, pages 1670–1677. IEEE, 1986.

14 L. Gewali, A. C. Meng, J. S. B. Mitchell, and S. C. Ntafos. Path planning in $0/1/\infty$ weighted regions with applications. *INFORMS J. Comput.*, 2(3):253–272, 1990. doi:10.1287/ijoc.2.3.253.

15 N. S. Jaklin. *On Weighted Regions and Social Crowds: Autonomous-agent Navigation in Virtual Worlds.* PhD thesis, Utrecht University, 2016.

16 A. Kamphuis, M. Rook, and M. H. Overmars. Tactical path finding in urban environments. In *First International Workshop on Crowd Simulation.* Citeseer, 2005.

17 J. Mitchell. Shortest paths among obstacles, zero-cost regions, and roads. Technical report, Cornell University Operations Research and Industrial Engineering, 1987.

18 J. Mitchell and D. M. Keirsey. Planning strategic paths through variable terrain data. In *Applications of Artificial Intelligence I*, volume 485, pages 172–179. SPIE, 1984.

19 J. Mitchell and C. Papadimitrou. The weighted region problem: Finding shortest paths through a weighted planar subdivision. *Journal of the ACM*, 38(1):18–73, 1991.

20 B. N. Nagy. Shortest paths in triangular grids with neighbourhood sequences. *Journal of Computing and Information Technology*, 11(2):111–122, 2003.

21 A. Nash. *Any-Angle Path Planning.* PhD thesis, University of Southern California, 2012.

22 N. C. Rowe and R. S. Ross. Optimal grid-free path planning across arbitrarily contoured terrain with anisotropic friction and gravity effects. *IEEE Transactions on Robotics and Automation*, 6(5):540–553, 1990.

23 M. Sharir and S. Sifrony. Coordinated motion planning for two independent robots. *Annals of Mathematics and Artificial Intelligence*, 3(1):107–130, 1991.

24 N. R. Sturtevant, D. Sigurdson, B. Taylor, and T. Gibson. Pathfinding and abstraction with dynamic terrain costs. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 15, pages 80–86, 2019.

25 W. van Toll, R. Triesscheijn, M. Kallmann, R. Oliva, N. Pelechano, J. Pettré, and R. Geraerts. A comparative study of navigation meshes. In *Proceedings of the 9th International Conference on Motion in Games*, pages 91–100, 2016.

Flipping Plane Spanning Paths*

Oswin Aichholzer†1, Kristin Knorr‡2, Maarten Löffler3, Zuzana Masárová4, Wolfgang Mulzer§2, Johannes Obenaus¶2, Rosna Paul‖1, and Birgit Vogtenhuber1**

1 **Institute of Software Technology, Graz University of Technology, Austria**
`{oaich,ropaul,bvogt}@ist.tugraz.at`

2 **Department of Computer Science, Freie Universität Berlin, Germany**
`{first name}.{family name}@fu-berlin.de`

3 **Utrecht University, the Netherlands**
`m.loffler@uu.nl`

4 **Institute of Science and Technology Austria**
`zuzana.masarova@ist.ac.at`

Abstract

Let S be a planar point set in general position, and let $\mathcal{P}(S)$ be the set of all plane (straight-line) spanning paths for S. A flip in a path $P \in \mathcal{P}(S)$ is the operation of removing an edge $e \in P$ and replacing it with a new edge f on S such that the resulting graph is again a path in $\mathcal{P}(S)$. Towards the question whether any two plane spanning paths of $\mathcal{P}(S)$ can be transformed into each other by a sequence of flips, we give positive answers if S is a wheel set, an ice cream cone, or a double chain. On the other hand, we show that in the general setting, it is sufficient to prove the statement for plane spanning paths with fixed first edge.

Related Version A full version of this paper is available at `https://arxiv.org/abs/2202.10831`

1 Introduction

Let S be a set of n points in the plane such that no three points in S are collinear (this property is called *general position* of S). Let $\mathcal{P}(S)$ be the set of all plane (i.e., crossing-free), straight-line spanning paths for S. A *flip* on a path $P \in \mathcal{P}(S)$ is the operation of removing one edge $e \in P$ and replacing it with a new edge f on S such that the resulting graph is again a plane spanning path form $\mathcal{P}(S)$ (note that e and f might cross). Unless stated otherwise, all paths in this paper are plane, spanning, and straight-line.

The question we consider is the following. Given two paths $P_s, P_t \in \mathcal{P}(S)$, can we always transform the starting path P_s into the target path P_t by a sequence of flips? Or, to phrase it in a more graph-theoretic manner: the *flip-graph* (on $\mathcal{P}(S)$) is defined to have vertex set $\mathcal{P}(S)$ and two vertices form an edge if and only if the corresponding paths differ by a single flip. Then, we are concerned with the question, whether the flip-graph is connected for any point set S.

[*] This work was initiated at the 2nd Austrian Computational Geometry Reunion Workshop in Strobl, June 2021. We thank all participants for fruitful discussions, especially Eva-Maria Hainzl, Guangping Li, Irene Parada, Daniel Perz, Josef Tkadlec, and Alexandra Weinberger.

[†] Partially supported by the Austrian Science Fund (FWF): W1230 and the European Union H2020-MSCA-RISE project 73499 - CONNECT.

[‡] Supported by DFG within the Research Training Group GRK 2434 *Facets of Complexity*.

[§] Supported in part by ERC StG 757609.

[¶] Supported by ERC StG 757609.

[‖] Supported by the Austrian Science Fund (FWF): W1230.

[**] Partially supported by Austrian Science Fund within the collaborative DACH project *Arrangements and Drawings* as FWF project I 3340-N35.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

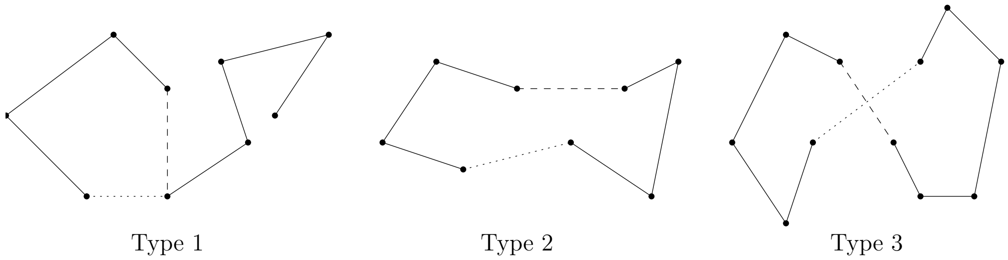


Figure 1 For Type 1 flips, the two involved edges share a common endpoint. For Type 2, the union of both paths is a plane spanning cycle (note that Type 2 flips can be simulated by a sequence of Type 1 flips). For Type 3, the two involved flipping edges cross, while the rest of the cycle is plane.

Flips in geometric graphs are a local but very powerful operation, see the survey of Bose and Hurtado [3]. On page 71, Bose and Hurtado write: "We are unaware of any progress for the same problem (obtaining a connected flip-graph for Hamiltonian crossing-free paths) on generic point sets."

Akl, Islam, and Meijer [2] showed that the flip graph is connected with diameter at most $2n-5$ for any $n \geq 3$ points in convex position, and for any $n \leq 8$ points in general position. Slightly later, tight bounds were derived by Chang and Wu [4]: if S is in convex position, then the diameter of the flip graph of $\mathcal{P}(S)$ is exactly $2n-5$, for $n = 3, 4$, and exactly $2n-6$, for $n \geq 5$.

There are different types of possible flips in a plane spanning path $P \in \mathcal{P}(S)$, but here we will mostly focus on *Type 1 flips* (see Figure 1 for other types of flips): enumerate the vertices of P as $p_1, \ldots, p_n$. Then, a *Type 1* flip consists of replacing an edge $p_{i-1}p_i$ of P, $i > 2$, by the edge p_1p_i. It results in the path $p_{i-1}, \ldots, p_1, p_i, \ldots, p_n$ (of course, the flip is only valid if the resulting path is still plane). In other words, a Type 1 flip inverts a contiguous chunk from one of the two ends of P.[1]

Our Results. First, we verify by a computer assisted proof with the help of the order type database [1] that the flip graph is connected for any set of $n \leq 10$ points.

For the general setting, we pursue two directions. On the one hand, we extend the proof of Akl et al. [2] to point sets in wheel, ice cream cone, and double chain configuration, the result for the double chains being the main contribution (see Theorem 6 in Section 2).

On the other hand, we show that it is sufficient to consider the flip-graph for paths where the first edge is fixed. More precisely: for distinct $p, q \in S$, let $\mathcal{P}(S, p)$ be the set of all plane spanning paths for S that start at p, and let $\mathcal{P}(S, p, q)$ be the set of all plane spanning paths for S that start at p and have pq as their first edge. We conjecture:

▶ **Conjecture 1** ([2]). *For any finite set $S \subset \mathbb{R}^2$ in general position and any two paths $P_s, P_t \in \mathcal{P}(S)$, there is a sequence of flips transforming P_s into P_t.*

▶ **Conjecture 2.** *For any finite set $S \subset \mathbb{R}^2$ in general position, any $p \in S$, and any two paths $P_s, P_t \in \mathcal{P}(S, p)$, there is a sequence of flips transforming P_s into P_t such that all intermediate paths are in $\mathcal{P}(S, p)$.*

[1] The corresponding flip at the other end of the path replaces an edge of the form p_jp_{j+1}, $j < n-1$ by the edge p_jp_n, resulting in the path $p_1, \ldots, p_j, p_n, \ldots, p_{j+1}$.

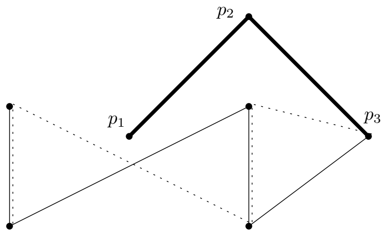


Figure 2 Example where the flip graph is disconnected if the first three points of the paths are fixed. The solid path cannot be flipped, but there is at least one other path (dotted) with the same three starting points.

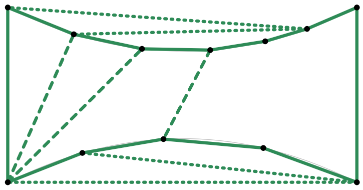


Figure 3 A double chain. Boundary edges are solid, bridge edges dashed, and chordal edges dotted (not all edges are drawn).

▶ **Conjecture 3.** *For any finite set $S \subset \mathbb{R}^2$ in general position, any distinct $p, q \in S$, and any two paths $P_s, P_t \in \mathcal{P}(S, p, q)$, there is a sequence of flips transforming P_s into P_t such that all intermediate paths are in $\mathcal{P}(S, p, q)$.*

We show (Lemmas 8 and 9 in Section 3) that for any fixed n, a positive answer to Conjecture 3 implies a positive answer to Conjecture 2, and similarly a positive answer to Conjecture 2 implies a positive answer to Conjecture 1.

Given Conjectures 1–3, one might think that an analogous statement for paths with a common starting sequence $p_1, p_2, \ldots, p_k$ of $k \geq 3$ points might also hold. Figure 2, however, shows a counter-example with 7 points for $k = 3$.

2 Special classes of point sets

We prove the connectedness of the flip-graph for wheel sets, ice cream cones, and double chains. Due to space constraints, however, we focus only on our main result, namely double chains. Our strategy is always to transform some path to a *canonical* path (consisting only of certain edges).

A *double chain* consists of two convex chains (each containing at least two points) with opposed concavity such that (i) the convex hull forms a quadrilateral (where the left and right endpoints of upper and lower chain form the *extreme vertices*) and (ii) no line through two points of the same chain separates the other chain (see e.g. [5, 3]). We classify the edges as follows: *boundary* edges are the edges between consecutive points on the upper and the lower chain, as well as the two *special* boundary edges between the two left and between the two right extreme points; *bridge* edges are the edges that connect the upper and the lower chain (except for the leftmost and rightmost such edge); all the other edges are *chordal* edges. A crucial property of double chains is the fact that boundary edges are uncrossed. We denote the class of double chains by DC (see Figure 3 for an illustration).

We define a (combinatorial) *distance* on the boundary of S, the plane cycle formed by the

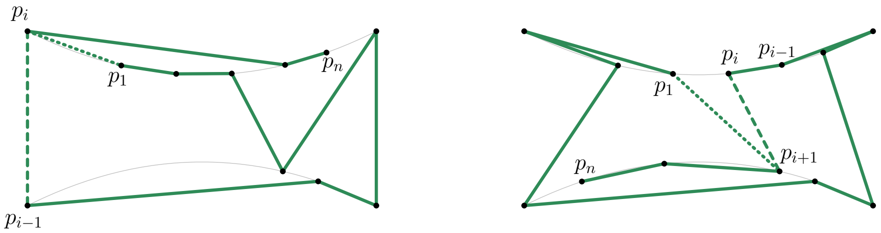


Figure 4 Illustration of Observation 4 (left) and Observation 5 (right). Replacing the dashed edge by the dotted forms a valid flip.

boundary edges[2]: let $S \in \mathsf{DC}$, and let $p, q \in S$ be two points on the boundary of S.[3] Further, let $o \in \{\text{cw}, \text{ccw}\}$ be an *orientation*. We define the *distance* between p and q in direction o, denoted by $d^o(p, q)$, as the number of boundary edges along the boundary that lie between p and q in direction o. Also, let the *distance* between p and q be

$$d(p, q) = \min\{d^{\text{cw}}(p, q), d^{\text{ccw}}(p, q)\}.$$

Note that neighboring vertices (along the boundary) have distance 1. Associating the pairs of vertices with an edge, we may also speak of the distance of an edge, i.e., the distance of an edge is just the distance between its endpoints. The *total* or *overall* distance (of a plane spanning path) is just the sum of all distances of its edges.

Let $S \in \mathsf{DC}$, and let $P = p_1, \dots, p_n \in \mathcal{P}(S)$. For $i = 1, \dots, n-1$, we call the two vertices p_i, p_{i+1} *consecutive* along P, and we say that p_i is the *predecessor* of p_{i+1} and that p_{i+1} is the *successor* of p_i. We emphasize that the terms *consecutive*, *predecessor*, and *successor* are reserved for the order along paths, whereas the terms *neighboring* and *neighbors* always refer to vertices that are incident to a common boundary edge of S.

The following observations will be useful to verify the validity of a flip (the first holds because no boundary edge is crossed by another edge on S), see also Figure 4:

▶ **Observation 4.** *Let $S \in \mathsf{DC}$, and let $P = p_1, \dots, p_n \in \mathcal{P}(S)$ be a plane spanning path on S. Let p_i, $i \neq 2$, be a neighbor of p_1. Then, the edge $p_{i-1}p_i$ can be flipped to the edge p_1p_i, i.e., replacing $p_{i-1}p_i$ by p_1p_i results in a valid plane spanning path for S.*

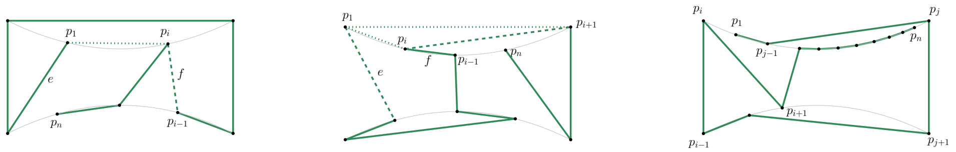
▶ **Observation 5.** *Let $S \in \mathsf{DC}$, and let $P = p_1, \dots, p_n \in \mathcal{P}(S)$ be a plane spanning path on S. Let p_1, p_i, $i \neq n$, be neighbors on the same chain. Then, the only edge of P that p_1p_{i+1} may cross is $p_{i-1}p_i$. In particular, if $p_{i-1}p_i$ is a boundary edge, replacing p_ip_{i+1} by p_1p_{i+1} forms a valid flip.*

The following theorem constitutes the main result of this section. We illustrate the main ideas and structure of the proof, but postpone the most involved cases to the full version.

▶ **Theorem 6.** *Let $S \in \mathsf{DC}$, and let $P, Q \in \mathcal{P}(S)$ be two plane spanning paths on S. Then, P can be transformed to Q in $O(n^2)$ flips.*

Proof. Let $P = p_1, \dots, p_n \in \mathcal{P}(S)$, and consider the edge $e = p_1p_2$. Let $p_i, i \neq 2$ be a neighbor of p_1 and whenever we have the choice, we pick p_i to be a neighbor such that p_1p_i does not form a special boundary edge (if both neighbors fulfill this property, pick one arbitrary). We denote $f = p_{i-1}p_i$. We describe a process where in each iteration we either:

[2] We emphasize that the boundary of S is distinct from the convex hull of S.
[3] Note, in the setting of double chains, any vertex is on the boundary.



(a) Case 1 **(b)** Case 2.1 **(c)** Case 2.2

Figure 5 Illustration of the three cases of Theorem 6. The solid paths together with the dashed edges form the initial path. Then, the dashed edges are replaced by the dotted (in (b), pay attention to the order of the flips). (c) illustrates some of the intricacies, if the starting edge is a boundary edge. None of the flips in the previous cases are valid here (no matter from which endpoint the path is viewed).

(i) increase the number of boundary edges (while not increasing the overall distance of P), or
(ii) decrease the overall distance of P (while not decreasing the number of boundary edges).

We can assume, w.l.o.g., that the endpoints of P are not neighbors, since otherwise we add the edge p_1p_n and remove an arbitrary (non-boundary) edge. We distinguish the following cases:

Case 1 f is not a boundary edge.
Then, we can simply replace f by p_1p_i (forming a proper flip by Observation 4). This increases the number of boundary edges and decreases the overall distance (recall that boundary edges have distance one and all other edges distance at least two).

Case 2 f is a boundary edge.
Then, the edge p_ip_{i+1} is not a boundary edge, since p_i already has the two neighbors p_1 and p_{i-1}.

Case 2.1 e is not a boundary edge.
Note that, since e is not a boundary edge, p_1p_i is not a special boundary edge.
We apply the following flips:

- replace p_ip_{i+1} by p_1p_{i+1} and
- replace e by p_1p_i.

The first flip is valid by Observation 5 and the second flip by Observation 4. The first flip may increase the overall distance by at most one, but the second flip decreases the overall distance by at least one. Hence, the overall distance does not increase. On the other hand, we increase the number of boundary edges.

Case 2.2 e is a boundary edge.
The case where e and f are both boundary edges is surprisingly intricate (especially when p_1 and p_i lie on different chains). It is easy to see that either $d(p_1, p_{i+1}) < d(p_i, p_{i+1})$ or $d(p_{i-1}, p_{i+1}) < d(p_i, p_{i+1})$ holds and our goal is to perform the corresponding flip that decreases the distance. However, if p_1 and p_i lie on different chains, we need to be very careful in order to preserve planarity (the details can be found in the full version of this paper).

Recursively applying above process, we will eventually transform P to a canonical path that consists only of boundary edges (the only paths with minimum overall distance). Doing the same for Q and noting that any pair of canonical paths can be transformed into each other by a single flip, the connectedness of the flip-graph follows.

Concerning the required number of flips, note that any edge has distance at most $\frac{n}{2} - 1$ and the path has $n - 1$ edges. Hence, the total number of iterations to transform P into a canonical path is at most

$$\left((n-1) \cdot \left(\frac{n}{2} - 2\right) + (n-1)\right) \in O(n^2)$$

Furthermore, any iteration requires at most two flips and hence, the total number of flips to transform P into Q is still in $O(n^2)$. ◀

3 A sufficient condition

In this section, we prove the sufficient condition of considering only paths with a fixed starting edge (recall that we consider point sets in general position now). We need one preliminary lemma, whose proof can be found in the full version of this paper:

▶ **Lemma 7.** *For any two points p_1 and p_2 of S there exists a path $P \in \mathcal{P}(S)$ which has p_1 as starting and p_2 as target point.*

▶ **Lemma 8.** *A positive answer to Conjecture 2 implies a positive answer to Conjecture 1.*

Proof. Let P_s and P_t be the two paths of Conjecture 1. If they have a common endpoint, we can directly use Conjecture 2 and the statement follows. So assume that P_s has the endpoints p_a and p_b, and P_t has the endpoints p_c and p_d, which are all distinct. By Lemma 7 there exists a path P_m having the two endpoints p_a and p_c. By Conjecture 2 there is a flip sequence from P_s to P_m with the common endpoint p_a, and again by Conjecture 2 there is a further flip sequence from P_m to P_t with the common endpoint p_c. This implies the statement. ◀

▶ **Lemma 9.** *A positive answer to Conjecture 3 implies a positive answer to Conjecture 2.*

The general strategy to prove Lemma 9 is similar to the one of Lemma 8, but of course more involved as we also need to handle the position of the common starting point (again, we defer the details to the full version).

4 Conclusion

In this paper, we made progress towards a positive answer of Conjecture 1, though it still remains open. A natural way to prove Conjecture 1 would be to prove Conjecture 3 by induction. We can assume all three conjectures to hold for all sets of size at most $n - 1$ and only need to show that Conjecture 3 holds for n.

Concerning the approach of special classes of point sets, of course one can try to further adapt the ideas to other classes.

Lastly, there are several other directions for further research conceivable, e.g. considering simple drawings (or other types of drawings) instead of straight-line drawings.

References

1 Oswin Aichholzer, Franz Aurenhammer, and Hannes Krasser. Enumerating order types for small point sets with applications. *Order*, 19:265–281, 2002. URL: https://link.springer.com/article/10.1023/A:1021231927255, doi:https://doi.org/10.1023/A:1021231927255.

2 Selim G. Akl, Md. Kamrul Islam, and Henk Meijer. On planar path transformation. *Information Processing Letters*, 104(2):59–64, 2007. URL: `https://www.sciencedirect.com/science/article/pii/S0020019007001366`, `doi:https://doi.org/10.1016/j.ipl.2007.05.009`.

3 Prosenjit Bose and Ferran Hurtado. Flips in planar graphs. *Computational Geometry*, 42(1):60–80, 2009. URL: `https://www.sciencedirect.com/science/article/pii/S0925772108000370`, `doi:https://doi.org/10.1016/j.comgeo.2008.04.001`.

4 Jou-Ming Chang and Ro-Yu Wu. On the diameter of geometric path graphs of points in convex position. *Information Processing Letters*, 109(8):409–413, 2009. URL: `https://www.sciencedirect.com/science/article/pii/S0020019008003827`, `doi:https://doi.org/10.1016/j.ipl.2008.12.017`.

5 Alfredo García, Marc Noy, and Javier Tejel. Lower bounds on the number of crossing-free subgraphs of K_n. *Computational Geometry*, 16(4):211–221, 2000.

Complexity Results on Untangling Planar Rectilinear Red-Blue Matchings

Arun Kumar Das*[1], Sandip Das*[2], Guilherme D. da Fonseca†[3], Yan Gerard†[4], and Bastien Rivier†[5]

1 **Indian Statistical Institute, Kolkata, India**
arund426@gmail.com

2 **Indian Statistical Institute, Kolkata, India**
sandipdas@isical.ac.in

3 **Aix-Marseille Université and LIS, France**
guilherme.fonseca@lis-lab.fr

4 **Université Clermont Auvergne and LIMOS, France**
yan.gerard@uca.fr

5 **Université Clermont Auvergne and LIMOS, France**
bastien.rivier@uca.fr

Abstract

Given a rectilinear matching between n red points and n blue points in the plane, we consider the problem of obtaining a crossing-free matching through flip operations that replace two crossing segments by two non-crossing ones. We first show that (i) it is NP-hard to α-approximate the shortest flip sequence, for any constant α. Second, we show that when the red points are colinear, (ii) given a matching, a flip sequence of length at most $\binom{n}{2}$ always exists, and (iii) the number of flips in any sequence never exceeds $\binom{n}{2}\frac{n+4}{6}$. Finally, we present (iv) a lower bounding flip sequence with roughly $1.5\binom{n}{2}$ flips, which disproves the conjecture that $\binom{n}{2}$, reached in the convex case, is the maximum. The last three results, based on novel analyses, improve the constants of state-of-the-art bounds.

Related Version arxiv.org/abs/2202.11857

1 Introduction

We consider the problem of untangling a planar rectilinear red-blue matching. We are given a set of $2n$ points in the plane, partitioned into a set R of n red points, and a set B of n blue points, in general position (no three colinear points, unless they have the same color).

A *configuration* is a set of n line segments where each point of R is matched to exactly one point of B, i.e. a perfect rectilinear red-blue matching. A flip is a combinatorial operation changing a configuration into another [7, 16]. In our case, a *flip* replaces two crossing segments by two non-crossing ones (Figure 1).

The *reconfiguration graph* of R, B is the directed simple graph whose vertices $\mathcal{V}$ are the configurations, and such that there is a directed edge from a configuration M_1 to another one M_2 whenever a flip transforms M_1 into M_2. Note that the reconfiguration graph is acyclic [6]. Let $\mathcal{S} \subseteq \mathcal{V}$ be the set of sinks, which corresponds to the crossing-free configurations. Given two configurations $u, v \in \mathcal{V}$, let $\mathcal{P}(u, v)$ be the set of directed paths from u to v. Given a path P, let the *length* of P, denoted $|P|$, be the number of edges in P. The *distance* from u

* This work is partially supported by the IFCAM project "Applications of graph homomorphisms" (MA/IFCAM/18/39).

† This work is supported by the French ANR PRC grant ADDS (ANR-19-CE48-0005).

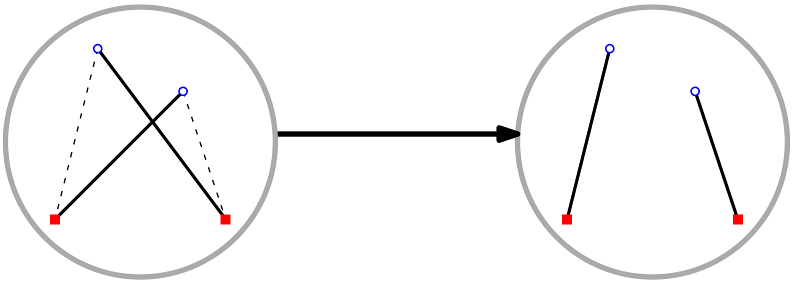


Figure 1 A flip. Red points are represented by solid squares and blue points by hollow circles.

to v, denoted $d(u,v)$, is the minimum path length from u to v. The *distance* from u to $\mathcal{S}$, $d(u,\mathcal{S})$, also abbreviated as $d(u)$, is the minimum path length from u to a configuration in $\mathcal{S}$. We are interested in two parameters of this reconfiguration graph:

$$\mathbf{d}(R,B) = \max_{u\in\mathcal{V}} \min_{v\in\mathcal{S}} \min_{P\in\mathcal{P}(u,v)} |P| \quad \text{and} \quad \mathbf{D}(R,B) = \max_{u\in\mathcal{V}} \max_{v\in\mathcal{S}} \max_{P\in\mathcal{P}(u,v)} |P|\,.$$

This leads to the definitions of $\mathbf{d}(n)$ and $\mathbf{D}(n)$ respectively as the maximum of $\mathbf{d}(R,B)$ and $\mathbf{D}(R,B)$ with $|R| = |B| = n$. An *untangle sequence* is a path in the reconfiguration graph ending in $\mathcal{S}$. Intuitively, $\mathbf{d}$ corresponds to the minimum length of an untangle sequence in the worst case, while $\mathbf{D}$ corresponds to the longest untangle sequence.

We also consider a more specific version of the problem where the red points are colinear [4], say, on the x-axis. As the flips on each half-plane defined by the x-axis are independent, we additionally suppose all blue points to lie on the upper half-plane without loss of generality. The matchings in this case are called *red-on-a-line* matchings.

Related work. The parameters $\mathbf{d}$, $\mathbf{D}$ have been studied in several different contexts with similar definitions of a flip, but considering other configurations.

In 1981, an $O(n^3)$ upper bound on $\mathbf{D}(n)$ was stated in the context of optimizing a TSP tour [23] (the configurations are polygons). This upper bound should be compared to the exponential lower bound on $\mathbf{D}(n)$ when the flips are not restricted to crossing segments, as long as they decrease the Euclidean length of the tour [10]. The convex case (i.e. the case where the points are in convex position) has been studied in [20, 25].

In the non-bipartite version of the rectilinear perfect matching problem, there are two possible pairs of segments to replace a crossing pair. This additional choice yields an $n^2/2$ upper bound on $\mathbf{d}(n)$ [6].

It is also possible to relax the flip definition to all operations that replace two segments by two others with the same four endpoints, whether they cross or not, and generalize the configurations to multigraphs with the same degree sequence [12, 13, 16]. In this context, finding the shortest path from a given configuration to another in the reconfiguration graph is NP-hard, yet 1.5-approximable [2, 3, 11, 24]. If we additionally require the configurations to be connected graphs, the same problem is NP-hard and 2.5-approximable [8].

Reconfiguration problems in the context of triangulations are widely studied [19]. A flip consists of removing one edge and adding another one while preserving a triangulation. It is know that $\Theta(n^2)$ flips are sufficient and sometimes necessary to obtain a Delaunay triangulation [14, 17]. Determining the flip distance between two triangulations of a point set [18, 21] and between two triangulations of a simple polygon [1] are both NP-hard.

Considering perfect matchings of an arbitrary graph (instead of the complete bipartite graph on R, B), a flip amounts to exchanging the edges in an alternating cycle of length

Table 1 Lower and upper bounds on $\mathbf{d}(n)$ and $\mathbf{D}(n)$ for red-blue matchings.

	$\mathbf{d}(n)$ bounds		$\mathbf{D}(n)$ bounds	
	lower	upper	lower	upper
general	$1.4n^{(a)}$, Thm. 5.2	$\binom{n}{2}(n-1)$, [6, 23]	$\frac{3}{2}\binom{n}{2} - \frac{n}{4}^{(b)}$, Thm. 5.1	$\binom{n}{2}(n-1)$, [6, 23]
convex	$1.4n^{(a)}$, Thm. 5.2	$2n-2$, [4]	$\binom{n}{2}$, [6]	$\binom{n}{2}$, [4]
red-on-a-line	$n-1$, [6]	$\binom{n}{2}$, Thm. 3.1	$\frac{3}{2}\binom{n}{2} - \frac{n}{4}^{(b)}$, Thm. 5.1	$\binom{n}{2}\frac{n+4}{6}$, Thm. 4.1

(a) For n multiple of 20.
(b) For even n.

four. It is then PSPACE-complete to decide whether there exists a path from a configuration to another [5]. There is, actually, a wide variety of reconfiguration contexts derived from NP-complete problems where this same accessibility problem is PSPACE-complete [15]. Many other reconfiguration problems are presented in [22].

Getting back to our context of rectilinear red-blue matchings, the values of $\mathbf{d}$ and $\mathbf{D}$ have been determined almost exactly in the convex case (see Table 1). Notice that the $n-1$ lower bound on $\mathbf{d}(n)$ carries to both the general and red-on-a-line cases [6]. It is notable that the upper bound on $\mathbf{D}(n)$ is also the best known bound on $\mathbf{d}(n)$ and has not been improved since 1981 [23].

Contributions. We show in Section 2 that it is NP-hard to α-approximate the shortest untangle sequence starting at a given matching, for any fixed $\alpha \geq 1$.

The following results are summarized in Table 1. An improved lower bound on $d(n)$ in the convex case is presented in Section 5.2. The remainder of the paper considers the red-on-a-line case. In Section 3, we slightly improve the former $\binom{n+1}{2}$ upper bound on $\mathbf{d}(n)$ [4], using a simpler algorithm and a novel analysis. In Section 4, we asymptotically divide by 6 the historical $\binom{n}{2}(n-1)$ upper bound on $\mathbf{D}(n)$ [6, 23], using a different potential argument.

In Section 5.1, we present a counter-example to the intuitive conjecture that the longest untangle sequence is attained in the convex case (where the number of crossings is maximal). We take advantage of points that are not in convex position to increase the lower bound by a factor of $\frac{3}{2}$. This red-on-a-line lower bound on $\mathbf{d}(n)$ carries over to the general case (and even to the case of general perfect matchings without color distinction among the points). The weaker conjecture that $\mathbf{D}(n)$ is quadratic [6] still holds, though.

2 NP-Hardness

In this section, we sketch the reduction of a known NP-complete problem, called rectilinear planar monotone (*RPM*) 3-SAT [9], to the following problem. The full proof is presented in the ArXiv version.

▶ Problem 1. Let $\alpha \geq 1$ be a constant.
Input: M, a red-blue matching.
Output: An untangle sequence starting at M of length at most α times $d(M)$.

▶ **Theorem 2.1.** *Problem 1 is NP-hard for all* $\alpha \geq 1$.

In *RPM 3-SAT*, the *graph* of a CNF formula is the bipartite graph with the variables and clauses as vertices, and where there is an edge between a variable and a clause if and only if the clause contains the variable. A CNF formula is *monotone* if each clause contains either only positive or only negative variables. An *RPM 3-CNF* formula is a monotone formula whose graph can be drawn with no intersection, and with the three following conventions (Figure 2). (i) The variables and the clauses are represented by axis-parallel rectangles. (ii) The variable rectangles lie on the x-axis. (iii) The positive clause rectangles are above the x-axis, the negative ones, below. We call such a drawing the *planar embedding* of Φ.

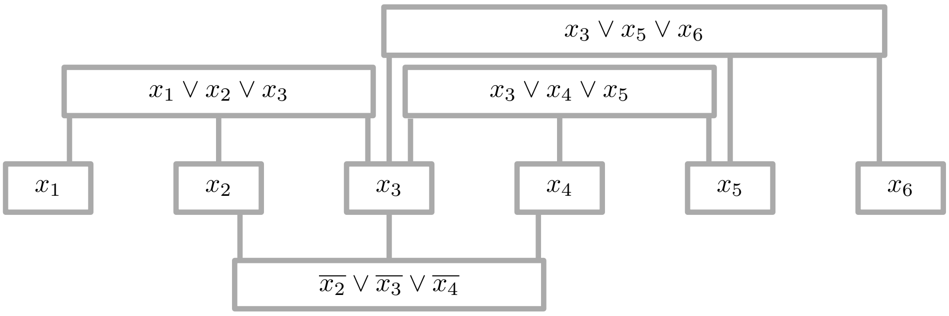


Figure 2 A planar embedding of an RPM 3-CNF formula.

The idea of the reduction is that, given an RPM 3-CNF formula Φ, we draw a rectilinear red-blue matching M_Φ of polynomial size such that all the untangle sequences starting at M_Φ are of length at most k_1 if Φ is satisfiable, and of length at least k_2 if Φ is not satisfiable.

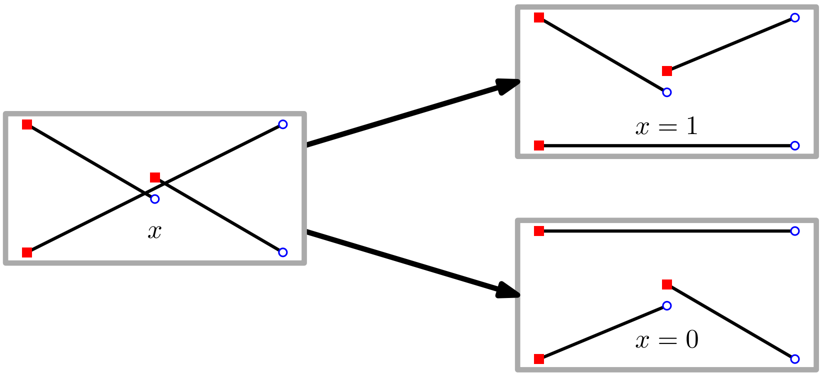


Figure 3 A variable gadget.

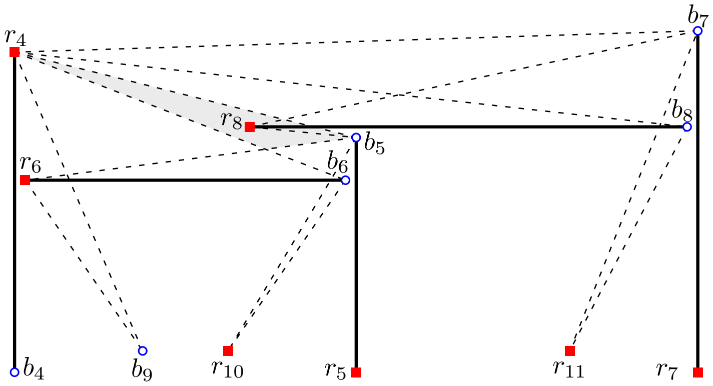


Figure 4 A clause gadget.

The aforesaid matching M_Φ is built upon the planar embedding of Φ. The variable rectangles are replaced by variable gadgets (Figure 3). The clause rectangles together with the corresponding edges are replaced with clause gadgets (Figure 4). A clause gadget consists of two OR gadgets, working like OR gates, and is connected to a padding gadget (Figure 5). If a clause is satisfied, then any untangle sequence of the two OR gadgets will end without creating any crossing in the padding gadget. If a clause is not satisfied, then any untangle sequence of the two OR gadgets will end creating a crossing in the padding gadget, which will trigger an arbitrary long series of flips, thus ensuring an arbitrary gap $k_2 - k_1$.

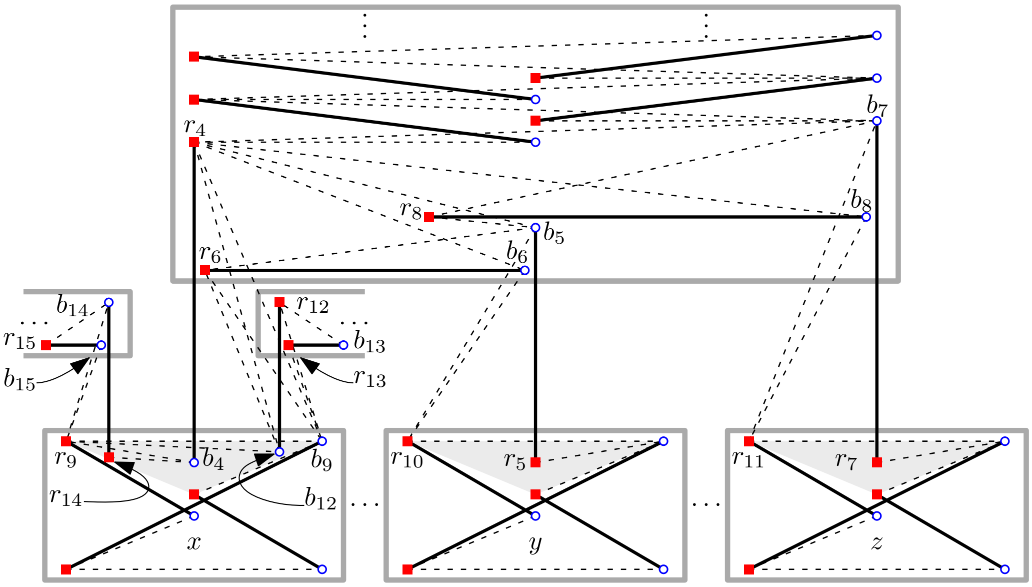


Figure 5 A clause gadget with padding connected to its variable gadgets, with branching on x.

3 Upper Bound on $\mathbf{d}(n)$

In this section, we give some insight into the proof of the following upper bound.

▶ **Theorem 3.1.** *In the red-on-a-line case,* $\mathbf{d}(n) \leq \binom{n}{2}$.

The proof consists of the analysis of the number of flips performed by the following recursive algorithm. We assume general position (no two blue points at same height). Let the *top* segment of a red-on-a-line matching be the segment with the topmost blue endpoint.

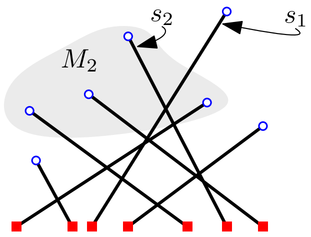


Figure 6 A red-on-a-line matching with s_1 as the top segment. The top segment of M_2 is s_2.

Algorithm 1:

```
Input  : $M$, a red-on-a-line matching.
Output : An untangle sequence starting at $M$.
0 If $R = B = \emptyset$, then stop.
1 Let $M_2$ be the set of segments crossing $s_1$, the top segment of $M$ (Figure 6). If $M_2$ is
    not empty, flip $s_1$ and $s_2$, the top segment of $M_2$, and repeat Step 0.
2 Recursively call the algorithm on the sub-matchings on both sides of the updated top
    segment of $M$.
```

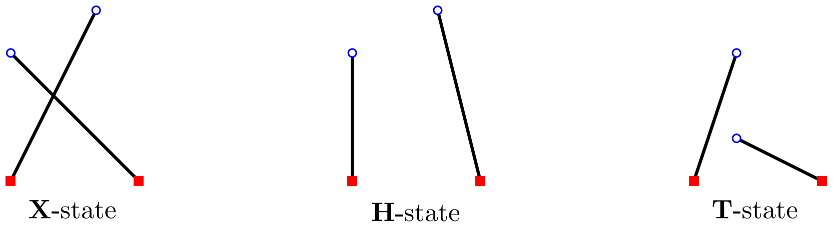
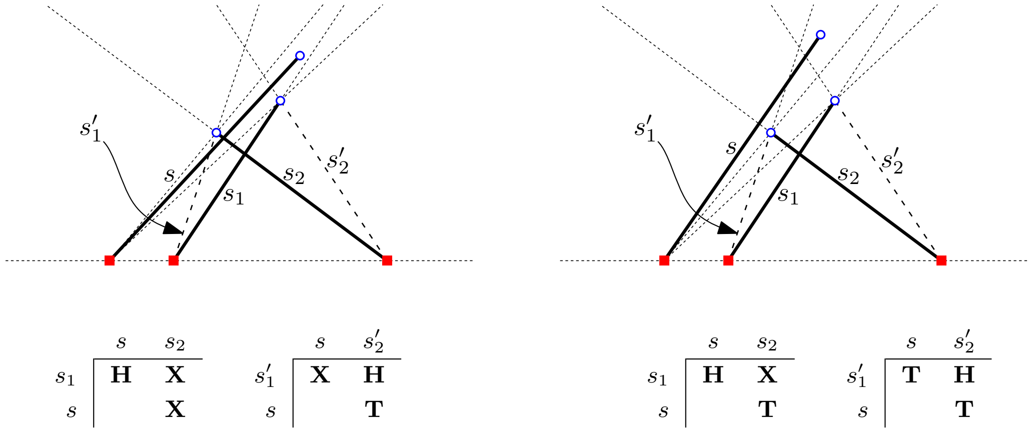


Figure 7 The three different states of pairs of segments.

The idea behind Algorithm 1 stems from the following observations. We define three states for a pair of segments: state **X**, when the segments are crossing, state **H**, when the segments are not crossing and their endpoints are in convex position, and state **T**, when the endpoints are not in convex position (Figure 7). In the convex case, a flip increases the number of **H**-pairs of at least 1 unit, providing the $\binom{n}{2}$ upper bound on $\mathbf{D}(n)$. However, the number of **H**-pairs may not increase in the general case. Figure 8 shows two such situations where there is one **H**-pair involving the segment s before the flip, and none after the flip. Algorithm 1 avoids these situations by choosing to flip top segments. The full proof, presented in the ArXiv version, involves state tracking, a novel approach to analyse flip sequences.



	s	s_2
s_1	**H**	**X**
s		**X**

	s	s'_2
s'_1	**X**	**H**
s		**T**

	s	s_2
s_1	**H**	**X**
s		**T**

	s	s'_2
s'_1	**T**	**H**
s		**T**

Figure 8 Two cases where the number of **H**-pairs does not increase. The flipped pair is s_1, s_2.

4 Upper Bound on $\mathrm{D}(n)$

In this section, we sketch the proof of the following upper bound.

▶ **Theorem 4.1.** *In the red-on-a-line case,* $\mathbf{D}(n) \leq \binom{n}{2}\frac{n+4}{6}$.

Let $r_1, \ldots, r_n$ be the red points, ordered from left to right. Theorem 4.1 is a corollary of the following bound on the number of flips involving r_k.

▶ **Lemma 4.2.** *In the red-on-a-line case, the number of flips involving the red point* r_k *is at most* $(k-1)(n-k)+n-1$.

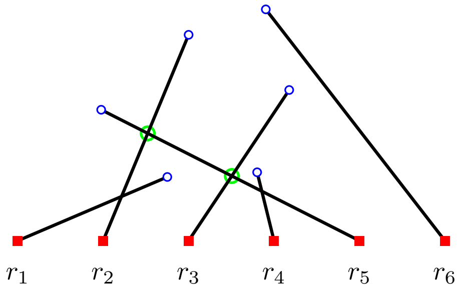


Figure 9 The two crossing pairs that may undergo a 3-flip ($k = 3$) immediately are circled.

The upper bound of Theorem 4.1 is obtained by computing the sum $\sum_{k=1}^{n}(k-1)(n-k)+n-1$ of the number of flips involving each red point, and then dividing this sum by 2, since each flip is counted twice (once for each red point).

The proof of Lemma 4.2 comes from a stronger lemma bounding the number of k-flips by $(k-1)(n-k)+n-1$, where a k-flip is a flip of a pair of segments $r_i b, r_j b'$, with $i \leq k \leq j$ (see Figure 9, where $k = 3$). The proof is fully presented in the ArXiv version.

5 Lower Bounds

In this section, we sketch the proof of the following lower bounds.

▶ **Theorem 5.1.** *In the red-on-a-line case, for even* n, $\mathbf{D}(n) \geq \frac{3}{2}\binom{n}{2} - \frac{n}{4}$.

▶ **Theorem 5.2.** *In the convex case, for* n *multiple of* 20, $\mathbf{d}(n) \geq 1.4 \cdot n$.

5.1 Lower Bound on $D(n)$

In order to define the starting configurations of lower bounding untangle sequences, we first provide some ad hoc definitions. We call a red-on-a-line convex matching an *n-star* when the maximum crossing number is attained, i.e. all the $\binom{n}{2}$ pairs of segments are crossing. For convenience, we say that an n-star *looks* at a point p if its blue points are all on a common line, and if p is the intersection of this line with the line on which the red points lie. We also say that two red-blue point sets R, B and R', B' are *fully crossing* if all the pairs of segments of the form $rb, r'b'$ are crossing, where $(r, b, r', b') \in R \times B \times R' \times B'$. Two matchings are fully crossing if their underlying red-blue point sets are fully crossing.

An *m-butterfly* is a red-on-a-line matching consisting of two fully crossing m-stars both looking at the same point p, where p is a median of the $2m$ red points (Figure 10). The existence of an untangle sequence of length $\frac{3}{2}\binom{2m}{2} - \frac{m}{2}$, starting at an m-butterfly is presented in the ArXiv version.

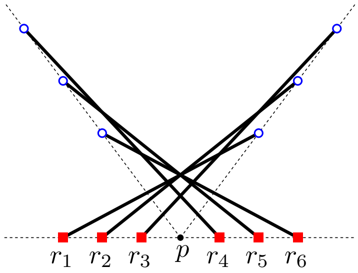


Figure 10 The 3-butterfly used to lower bound $\mathbf{D}(6)$.

5.2 Lower Bound on $\mathrm{d}(n)$

An improved lower bound on $\mathbf{d}(n)$ in the convex case comes from running a breadth-first search on the 20-segment configuration in Figure 11 and finding a minimum untangle sequence length of 24. Arranging multiple copies of this configuration, we get $\mathbf{d}(n) \geq 1.4 \cdot n$ for n multiple of 20. The source code is available on github.com/gfonsecabr/untangling.

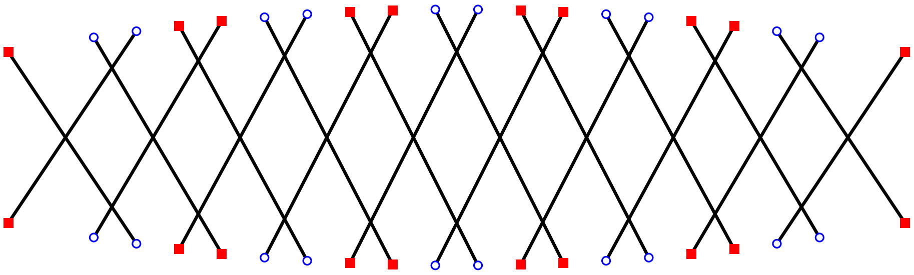


Figure 11 The convex configuration used to show that $\mathbf{d}(20) \geq 28$.

6 Concluding Remarks

Untangle sequences of TSP tours have been investigated since the 80s, when a cubic upper bound on $\mathbf{D}(n)$ has been discovered [23]. This bound also holds for matchings and has not been improved ever since. Except for the convex case, there are big gaps between the lower and upper bounds, as can be seen in Table 1. Experiments on tours and matchings have shown that, in all cases tested, the cubic upper bound is not tight and the lower bounds seem to be asymptotically tight.

Untangle sequences have many unexpected properties which make the problem harder than it seems at first sight. The following questions remain open.

1. If we add a new segment to a crossing-free matching, what is the maximum length of an untangle sequence? Notice that an $o(n^2)$ bound would lead to an $o(n^3)$ bound for $\mathbf{d}(n)$.
2. Is it always possible to find an untangle sequence that does not flip the same pair of segments twice? Using a balancing argument, we can show that the number of *distinct* flips in any untangle sequence is $O(n^{8/3})$.
3. What is the maximum number of flips involving a given point? The classic potential [23] provides a quadratic bound which leads again to $\mathbf{D}(n) = O(n^3)$.
4. Is there a potential that provides better bounds?

We proved the NP-hardness of computing the shortest untangle sequence for a red-blue matching. What is the complexity of computing the shortest untangle sequence for a TSP tour, for a red-on-a-line matching, or even for a convex instance? What about the longest untangle sequences?

References

1 Oswin Aichholzer, Wolfgang Mulzer, and Alexander Pilz. Flip distance between triangulations of a simple polygon is NP-complete. *Discrete & Computational Geometry*, 54(2):368–389, 2015.

2 Sergey Bereg and Hiro Ito. Transforming graphs with the same degree sequence. In *Computational Geometry and Graph Theory*, pages 25–32, 2008.

3 Sergey Bereg and Hiro Ito. Transforming graphs with the same graphic sequence. *Journal of Information Processing*, 25:627–633, 2017.

4 Ahmad Biniaz, Anil Maheshwari, and Michiel Smid. Flip distance to some plane configurations. *Computational Geometry*, 81:12–21, 2019.

5 Marthe Bonamy, Nicolas Bousquet, Marc Heinrich, Takehiro Ito, Yusuke Kobayashi, Arnaud Mary, Moritz Mühlenthaler, and Kunihiro Wasa. The perfect matching reconfiguration problem. In *44th International Symposium on Mathematical Foundations of Computer Science*, volume 138 of *LIPIcs*, pages 80:1–80:14, 2019.

6 Édouard Bonnet and Tillmann Miltzow. Flip distance to a non-crossing perfect matching. *Computing Research Repository*, abs/1601.05989, 2016.

7 Prosenjit Bose and Ferran Hurtado. Flips in planar graphs. *Computational Geometry*, 42(1):60–80, 2009.

8 Nicolas Bousquet and Alice Joffard. Approximating shortest connected graph transformation for trees. In *Theory and Practice of Computer Science*, pages 76–87, 2020.

9 Mark De Berg and Amirali Khosravi. Optimal binary space partitions for segments in the plane. *International Journal of Computational Geometry & Applications*, 22(03):187–205, 2012.

10 Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst case and probabilistic analysis of the 2-Opt algorithm for the TSP. *Algorithmica*, 68(1):190–264, 2014.

11 Péter L Erdős, Zoltán Király, and István Miklós. On the swap-distances of different realizations of a graphical degree sequence. *Combinatorics, Probability and Computing*, 22(3):366–383, 2013.

12 Seifollah Louis Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph. i. *Journal of the Society for Industrial and Applied Mathematics*, 10(3):496–506, 1962.

13 Seifollah Louis Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph ii. uniqueness. *Journal of the Society for Industrial and Applied Mathematics*, 11(1):135–147, 1963.

14 Ferran Hurtado, Marc Noy, and Jorge Urrutia. Flipping edges in triangulations. *Discrete & Computational Geometry*, 22(3):333–346, 1999.

15 Takehiro Ito, Erik D. Demaine, Nicholas J.A. Harvey, Christos H. Papadimitriou, Martha Sideri, Ryuhei Uehara, and Yushi Uno. On the complexity of reconfiguration problems. *Theoretical Computer Science*, 412(12):1054–1065, 2011.

16 Alice Joffard. *Graph domination and reconfiguration problems*. PhD thesis, Université Claude Bernard Lyon 1, 2020.

17 Charles L Lawson. Transforming triangulations. *Discrete Mathematics*, 3(4):365–372, 1972.

18 Anna Lubiw and Vinayak Pathak. Flip distance between two triangulations of a point set is NP-complete. *Computational Geometry*, 49:17–23, 2015.

19 Naomi Nishimura. Introduction to reconfiguration. *Algorithms*, 11(4), 2018.

20 Yoshiaki Oda and Mamoru Watanabe. The number of flips required to obtain non-crossing convex cycles. In *Kyoto International Conference on Computational Geometry and Graph Theory*, pages 155–165, 2007.

21 Alexander Pilz. Flip distance between triangulations of a planar point set is apx-hard. *Computational Geometry*, 47(5):589–604, 2014.

22 Jan van den Heuvel. The complexity of change. *Surveys in Combinatorics*, 409:127–160, 2013.

23 Jan van Leeuwen. Untangling a traveling salesman tour in the plane. In *7th Workshop on Graph-Theoretic Concepts in Computer Science*, 1981.

24 Todd G Will. Switching distance between graphs with the same degrees. *SIAM Journal on Discrete Mathematics*, 12(3):298–306, 1999.

25 Ro-Yu Wu, Jou-Ming Chang, and Jia-Huei Lin. On the maximum switching number to obtain non-crossing convex cycles. In *26th Workshop on Combinatorial Mathematics and Computation Theory*, pages 266–273, 2009.

On Some Relations Between Optimal TSP Solutions and Proximity Graphs in the Plane*

Logan D. Graham[1], Joseph S. B. Mitchell[1], Gaurish Telang[1], and Sam van der Poel[1]

1 **Department of Applied Mathematics and Statistics, Stony Brook University**
{logan.graham,joseph.mitchell,sam.vanderpoel}@stonybrook.edu,
gaurish108@gmail.com

Abstract

Relations between the Euclidean traveling salesman problem (TSP) and proximity graphs have primarily been explored in the context of TSP heuristics. We present a set of theoretical results, a computer-assisted proof, and extensive experimental findings that reveal a subtle relationship between the TSP and proximity graphs. Carefully constructed examples show that the TSP does not necessarily contain a nearest neighbor graph (NNG) edge. We show that the TSP must contain an edge of the order-k Delaunay for constant $k \in \mathbb{N}$. We devised and implemented an enumeration algorithm that allows us to prove that the TSP tour on point sets of size $n \leq 9$ necessarily contains an NNG edge. Large-scale simulations and an application to geometric deep learning reinforce the strong ties between the TSP and proximity graphs that are observed in practice.

1 Introduction

The Euclidean traveling salesman problem (TSP), seeking the shortest tour or path through n points in the Euclidean plane, is NP-hard and one of the most intensively studied problems in computer science.

The length-minimizing quality of the TSP suggests that the TSP may necessarily contain edges of proximity graphs such as the (undirected) nearest neighbor graph (NNG), minimum spanning tree (MST), k-Gabriel graphs, or k-Delaunay graphs. We refute some of this intuition by providing examples of TSP solutions that "avoid" having edges of some proximity graphs. The point sets in Section 2 serve to preclude certain claims concerning the intersection of the TSP with proximity graphs, akin to those of Dillencourt [6, 7], who showed that the Delaunay triangulation is not always Hamiltonian. We show, for instance, that the Euclidean TSP does not necessarily contain a nearest neighbor graph edge.

A positive result in Section 2, along with extensive empirical findings in Section 4, and a computer-assisted proof in Section 3 reaffirm the strong ties between the TSP and proximity graphs. We show that, experimentally, close to 99% of TSP edges tend to be Delaunay edges for large point sets sampled from distributions considered by Bentley [3], and an enumeration algorithm shows that small TSP instances ($n \leq 9$) necessarily contain NNG edges. These results further support the use of TSP heuristics inspired by proximity graphs [16, 17, 22, 24].

* This work has been partially supported by NSF (CCF-2007275). The authors would like to thank Stony Brook Research Computing and Cyberinfrastructure, and the Institute for Advanced Computational Science at Stony Brook University for access to the high-performance SeaWulf computing cluster, which was made possible by a $1.4M National Science Foundation grant (#1531492). The authors would also like to acknowledge Omrit Filtser for insight regarding the structure of the point set in Fig. 1, Hugo Mainguy for the initial idea that led to the point sets in Fig. 5, others who participated in Stony Brook's Computational Geometry Group research meetings, and the anonymous reviewers.

38th European Workshop on Computational Geometry, Perugia, Italy, March 14–16, 2022.
This is an extended abstract of a presentation given at EuroCG'22. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

The code used to obtain the included results, the data used in the experiments, and a GUI to study and visualize the TSP and proximity graphs are provided at `https://github.com/samvanderpoel/TSP-vs-Graphs`, a code base that is user-friendly and easily extensible.

1.1 Proximity Graph Definitions

Fix a finite point set S in the Euclidean plane. We consider weighted graphs on S whose edge weights are given by Euclidean distance. The *k-nearest neighbor graph (k-NNG)* contains an (undirected) edge between two vertices u, v if and only if the distance between u and v is among the k shortest distances either from u to any $p \in S \setminus \{u\}$ or from v to any $p \in S \setminus \{v\}$. References to the NNG are to be understood as those to the 1-NNG. The *minimum spanning tree (MST)* is a connected graph on S with lowest aggregate edge weight. The *k-Delaunay* contains an edge between two vertices u, v if and only if there exists a closed disk containing u, v, and at most k points in $S \setminus \{u, v\}$. The *k-Gabriel graph* contains an edge between two vertices u, v if and only if the closed disk with diametrical chord (u, v) contains at most k points in $S \setminus \{u, v\}$. We write simply *Delaunay* and *Gabriel* to mean 0-Delaunay and 0-Gabriel, respectively. For points in general position, the *Urquhart graph* is that which results from removing the longest edge from each triangle in the Delaunay triangulation.

2 Theoretical Results

In what follows, all graphs are considered to be undirected. All instances of the TSP were verified with the Concorde TSP Solver [1]. The existence of the well-known nearest neighbor TSP heuristic prompts the question of whether the Euclidean TSP tour and path each necessarily contain an NNG edge. We answer this in the negative by presenting Figure 1, a point set whose (unique) Euclidean TSP tour and NNG are disjoint.

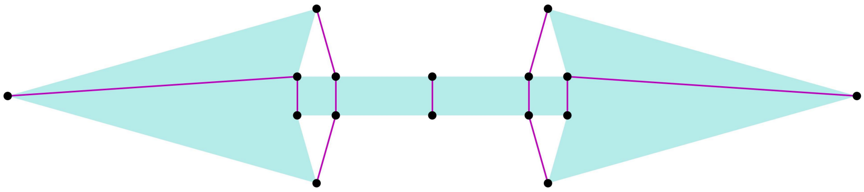


Figure 1 The Euclidean TSP tour (filled in blue) and NNG (purple) are disjoint.

There exists a similar point set, omitted here, demonstrating that the TSP tour and NNG may be disjoint for L^p metrics, $2 \leq p \leq 100$. The TSP path and NNG of the point set in Figure 2 are disjoint for L^p metrics, $2 \leq p \leq 6$.

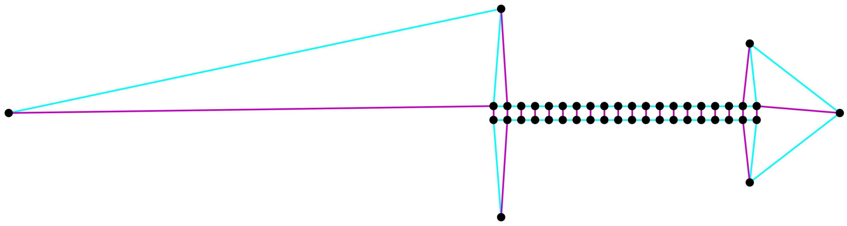


Figure 2 The TSP path (blue) and NNG (purple) are disjoint for L^p metrics, $2 \leq p \leq 6$.

The NNG is one of a set of proximity graphs that we consider here; they form the following inclusions with the k-Delaunay being the densest (having most edges) graph.

$$\text{MST} \subseteq \text{Urquhart} \subseteq \text{Gabriel} \subseteq \text{Delaunay Triangulation} \subseteq k\text{-Delaunay} \tag{1}$$

Further, the inclusion NNG $\subseteq$ MST holds for points in general position.

It is natural to conjecture that the TSP tour contains a constant fraction of edges of the Gabriel graph or one of its subgraphs. This is refuted by the family ($n \geq 3$) of point sets in Figure 3 each of whose TSP tour and Gabriel graph share only two edges.

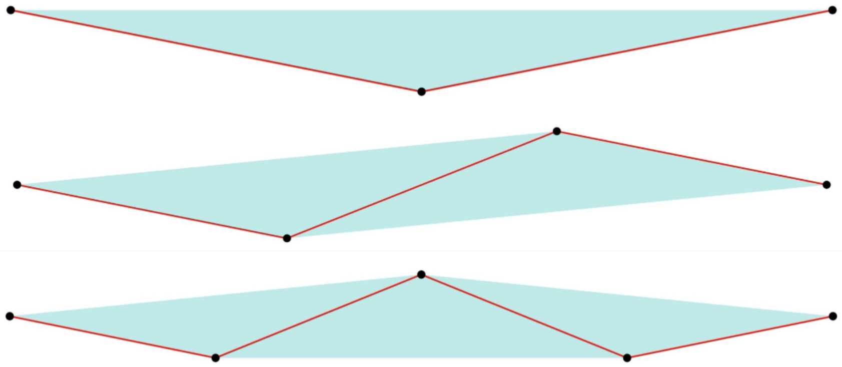


Figure 3 A family ($n \geq 3$) whose TSP tours (filled in blue) contain only two Gabriel edges.

In the family of point sets in Figure 3, the MST coincides with the Gabriel graph, and the MST has vertices of degree at most two. It is worth asking whether MST vertices of degree three or more must be incident to a TSP edge that also belongs to the MST. The example of Figure 4 shows that this is not the case: there are three MST vertices of degree three, and none of the MST edges incident to these vertices belong to the TSP tour.

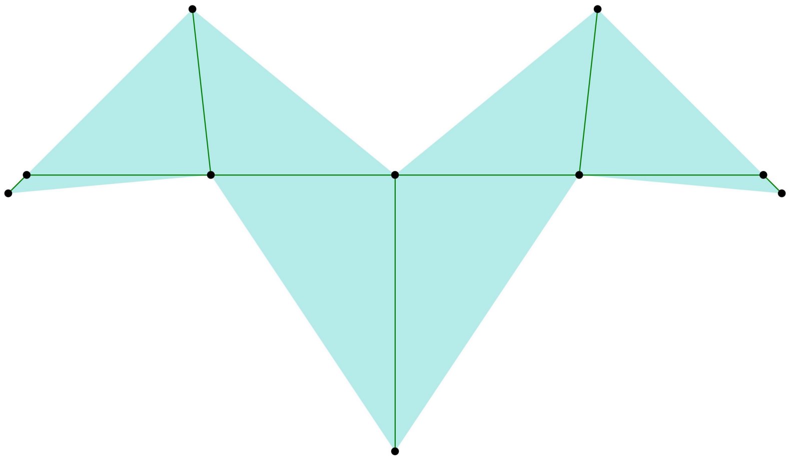


Figure 4 No MST edges (green) incident to degree-three MST vertices are TSP tour edges.

Optimality of the TSP tour implies that it forms a simple polygonal cycle in the Euclidean plane. To show that a TSP tour necessarily includes a Delaunay edge (a conjecture we are actively pursuing), it is natural to consider whether any simple polygon on a set of points necessarily uses at least one Delaunay edge. Figure 5 shows that this is not so: there exist point sets that admit simple polygonalizations, none of whose edges belong to the Delaunay.

Another approach to showing that the TSP tour must contain an edge of the Delaunay consists in arguing that at least one out of every k successive edges of a TSP tour must be Delaunay. This argument, too, is cast in doubt by Figures 6, 7, and 8, showing point sets whose TSP tours contain two, three, and four consecutive non-Delaunay edges, respectively.

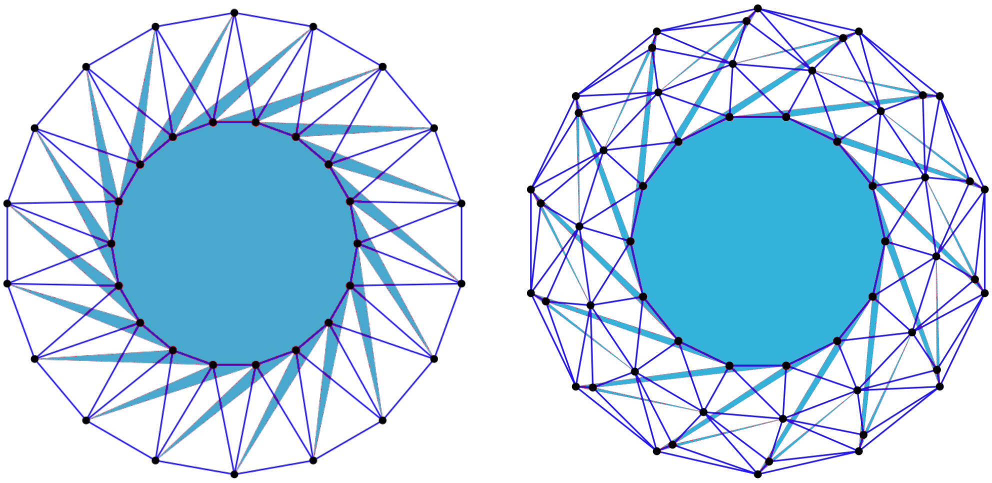


Figure 5 Point sets with polygonalizations (light) edge-disjoint from Delaunay graphs (dark).

In constructing the point sets of Figures 6, 7, and 8, we noticed a strong sensitivity of the TSP tour to the precise placement and density of points. Nevertheless, we believe that the theme of these point sets can be generalized and applied for any given $k \in \mathbb{N}$.

▶ **Conjecture 1.** *For any $k \in \mathbb{N}$ there exists a point set S in the Euclidean plane such that k consecutive edges of the TSP on S are non-Delaunay.*

Figures 6, 7, and 8, show that despite the prevalence of Delaunay edges (>99%) in the TSP tours on randomly generated point sets (Section 4), Delaunay edges are not "so" prevalent as to ensure that one must exist among every $k \leq 4$ consecutive edges of the TSP.

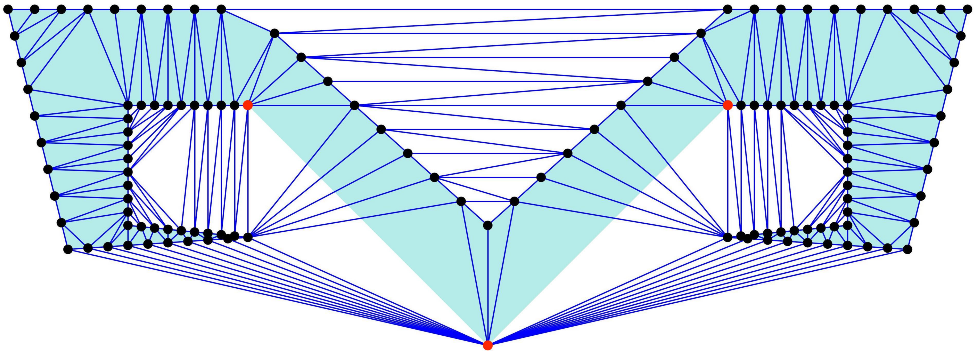


Figure 6 Two successive edges of the TSP, incident to the red vertices, are non-Delaunay.

In addition, Figure 6 shows an example where both of the TSP cycle edges that are incident on a convex hull vertex can be non-Delaunay.

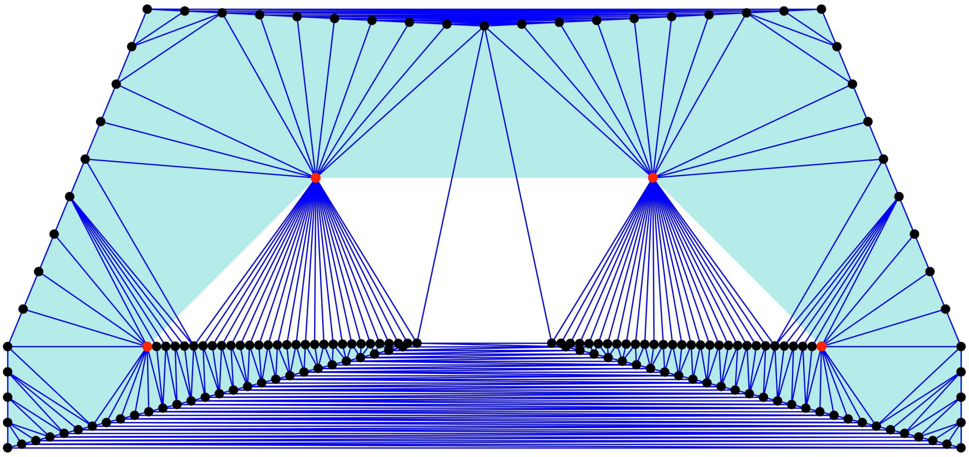


Figure 7 Three successive edges of the TSP tour, incident to the red vertices, are non-Delaunay.

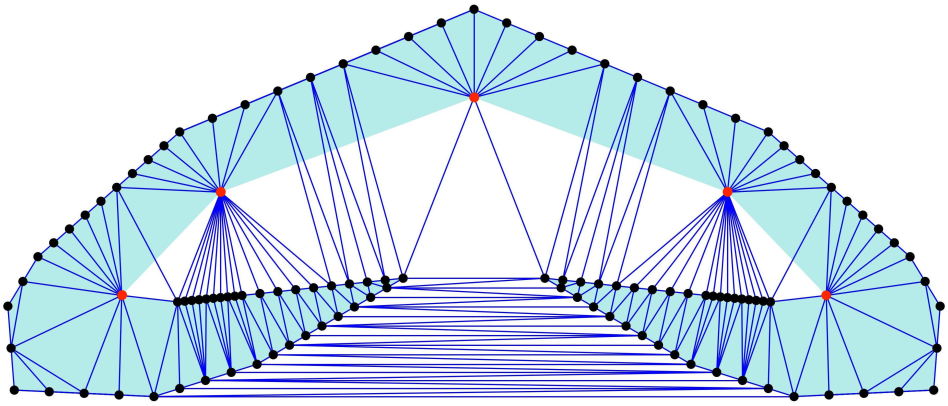


Figure 8 Four successive edges of the TSP tour, incident to the red vertices, are non-Delaunay.

It was shown in [12] that the 10-Gabriel graph – and hence also the 10-Delaunay – is Hamiltonian. Theorem 2 supplements this result, showing that there does not exist an integer $k \geq 0$ such that the k-Gabriel graph or k-Delaunay always contains the Euclidean TSP tour or path.

▶ **Theorem 2.** *For every integer $k \geq 0$, there exist point sets S_1 and S_2 in the Euclidean plane such that the TSP tour of S_1 and the TSP path of S_2 are not subsets of the respective k-Delaunay graphs of S_1 and S_2.*

▶ **Theorem 3.** *There exists a constant k such that, for every finite point set in $\mathbb{R}^2$, the Euclidean k-Gabriel graph contains a shortest edge in an optimal Euclidean TSP.*

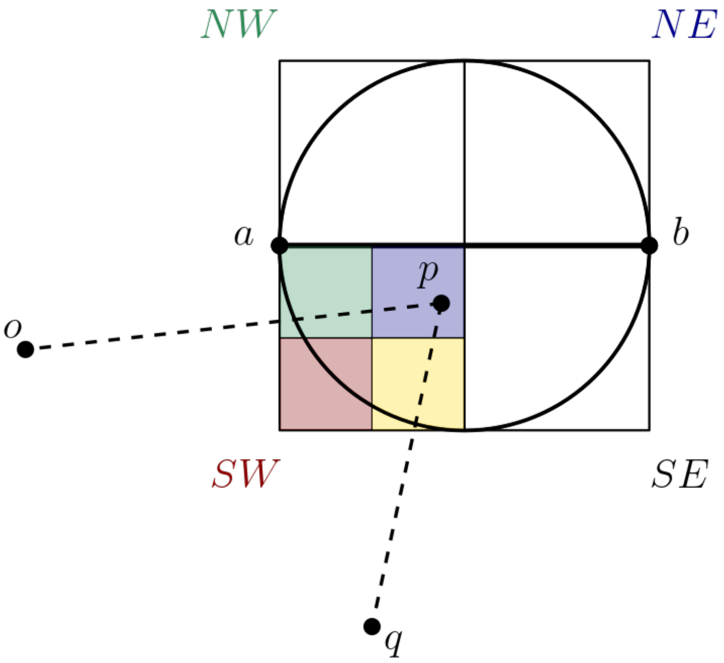


Figure 9 Proof of Theorem 3: only $O(1)$ points can lie in the diametrical disk defined by ab.

Proof sketch. Fix a finite point set in the Euclidean plane. Denote by ab a shortest edge in this point set's TSP. Consider the bounding square of the diametrical disk defined by ab, and partition it into 16 subsquares (four are shown in Fig. 9). Pick a subsquare and a point p therein. Using extremality, we reason about p and its neighbors o and q along an optimal TSP tour, invoking a geometric separator theorem from [5] (adapted from [13, 23]) to show that there are $O(1)$ points in each subsquare. Applying this reasoning to all subsquares, we conclude that there are only $O(1)$ points in the diametrical disk characterized by ab. ◀

3 TSP–NNG Intersection Algorithm

For small point sets, it is possible to directly search the set of all possible NNGs (under some criteria) to determine whether a TSP tour or path necessarily uses an NNG edge. This is made precise in the discussion below for the case of TSP tour, leading to an enumeration algorithm, the output of which justifies the following theorem.

▶ **Theorem 4.** *Let G be an undirected weighted graph on $n \leq 9$ vertices. Suppose that H is a Hamiltonian cycle on the complement of the NNG of G. Then the total weight of H can be reduced, maintaining its Hamiltonicity, by a set of edge exchanges only with the NNG.*

Fix an undirected weighted graph $G = (V, E)$ on n vertices $v_0, \ldots, v_{n-1}$ and suppose there is a Hamiltonian cycle $H = (v_{i_0}, v_{i_1}, \ldots, v_{i_{n-1}}, v_{i_0})$ that is disjoint from the NNG of G (nearest neighbors may be non-unique). Let $NN(v_i)$ denote the set of nearest-neighbors of v_i. Define the set $\mathcal{N}(G) = NN(v_0) \times \cdots \times NN(v_{n-1})$, and call its elements *representatives* of the NNG of G; fix a representative $r = (r_0, \ldots, r_{n-1}) \in \mathcal{N}(G)$. For each i_j, since r_{i_j} is a nearest neighbor of v_{i_j}, exchanging either of the edges $(v_{i_j}, v_{i_{(j-1) \bmod n}})$ or $(v_{i_j}, v_{i_{(j+1) \bmod n}})$ in H for (v_{i_j}, r_{i_j}) shortens the total weight of H as a weighted graph. Depending on whether (v_{i_j}, r_{i_j}) substitutes $(v_{i_j}, v_{i_{(j-1) \bmod n}})$, $(v_{i_j}, v_{i_{(j+1) \bmod n}})$, or neither, these exchanges can be encoded with the integers -1, 1, or 0, respectively. Applying this encoding of edge exchanges to all $i_j = 0, \dots, n-1$, the set of all possible nontrivial edge exchanges between H and a representative of the NNG is given by the set

$$\mathscr{E}(n) = \{(e_0, \dots, e_{n-1}); e_i \in \{-1, 0, 1\}\} \setminus \{(0, \dots, 0)\}. \tag{2}$$

If, after application of an edge exchange $E \in \mathscr{E}(n)$ to H, the resulting graph is still a cycle, then we will have shortened the Hamiltonian cycle H via an edge exchange with a representative of the NNG. By a direct search of $\mathscr{E}(n)$ it can be determined whether a Hamiltonian cycle admits a shortening via an edge exchange with a given representative r.

Without reference to a specific graph G, and knowing only that H contains no NNG edges, we may iterate over the set of all possible representatives of the NNG, given by

$$S_{NNG}(n) = \Big\{(\sigma_0, \dots, \sigma_{n-1}) : \sigma_i \in \{0, \dots, n-1\} \setminus \{j \bmod n : j = i-1, i, i+1\}\Big\}. \tag{3}$$

The meaning of $s = (\sigma_0, \dots, \sigma_{n-1}) \in S_{NNG}(n)$ is that σ_i is the index of a nearest neighbor of v_i. The brute-force iteration over all of $S_{NNG}(n)$ and, for each $s \in S_{NNG}(n)$, over $\mathscr{E}(n)$, checking whether any edge exchange yields a Hamiltonian cycle, comprises an algorithm whose output proves that the TSP tour must contain an NNG edge for $n \leq 9$, as verified by our implementation on Stony Brook University's high-performance computing cluster.

4 Empirical Findings

We now present experimental results concerning the intersection of the TSP and proximity graphs. Table 1 shows edgewise comparisons between the TSP tour and proximity graphs for several Euclidean TSPLIB [21] instances.

Table 1 Percentages of TSP tour edges in proximity graphs for Euclidean TSPLIB instances.

Instance	1-NNG	2-NNG	MST	Urquhart	Gabriel	Delaunay
rat575	58.96	85.57	74.43	86.09	96.17	99.48
p654	50.61	82.11	69.88	92.05	95.72	99.24
d657	58.75	81.89	75.19	81.89	92.69	99.09
u724	58.84	83.70	74.31	83.01	96.13	99.45
rat783	59.26	82.50	76.37	85.31	94.89	100.00

Extensive experiments were conducted on Stony Brook University's high-performance computing cluster using the Concorde TSP Solver [1] along with Python and its scientific computing libraries. Point sets of cardinalities 10 to 1660 were sampled in 50-point intervals from 7 probability distributions, with 50 point sets sampled at each cardinality. Six of the 7 distributions we consider were studied by Bentley [3]; we present results of 2 distributions here and make the others available in the GitHub repository. For each point set sampled, fractions of TSP edges belonging to different proximity graphs were calculated, the results of which constitute the below plots. The error bands represent ± one standard deviation at each point set size.

Concorde computes optimal tours; we used it to compute optimal TSP paths by adding a dummy vertex that is equidistant from every point of the given point set.

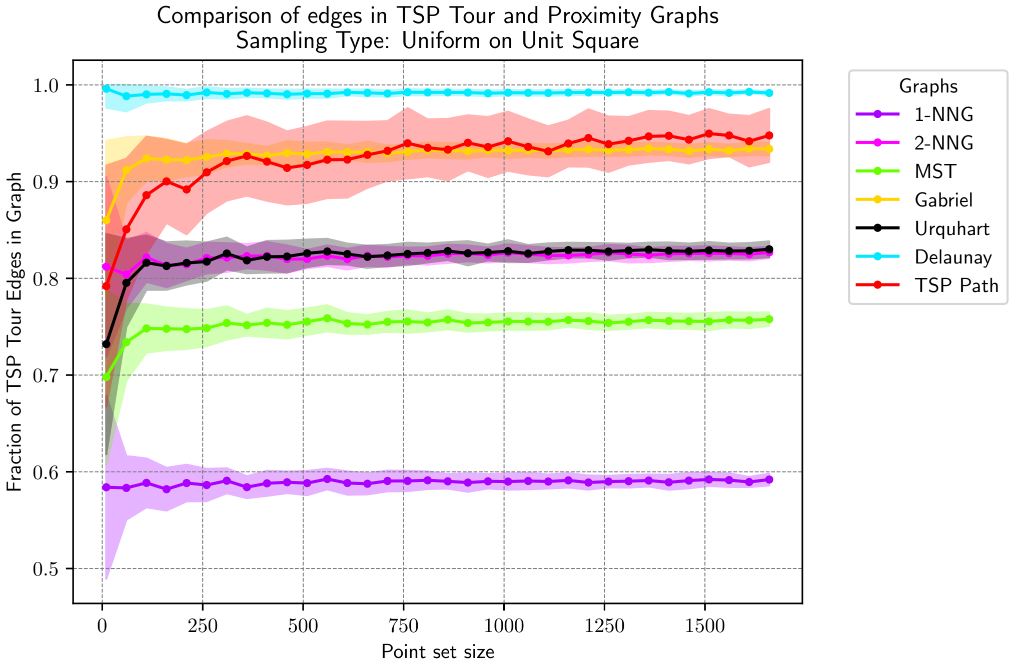


Figure 10 Comparisons of the edges in the TSP tour versus those of proximity graphs when point clouds were sampled uniformly from $[0, 1] \times [0, 1]$.

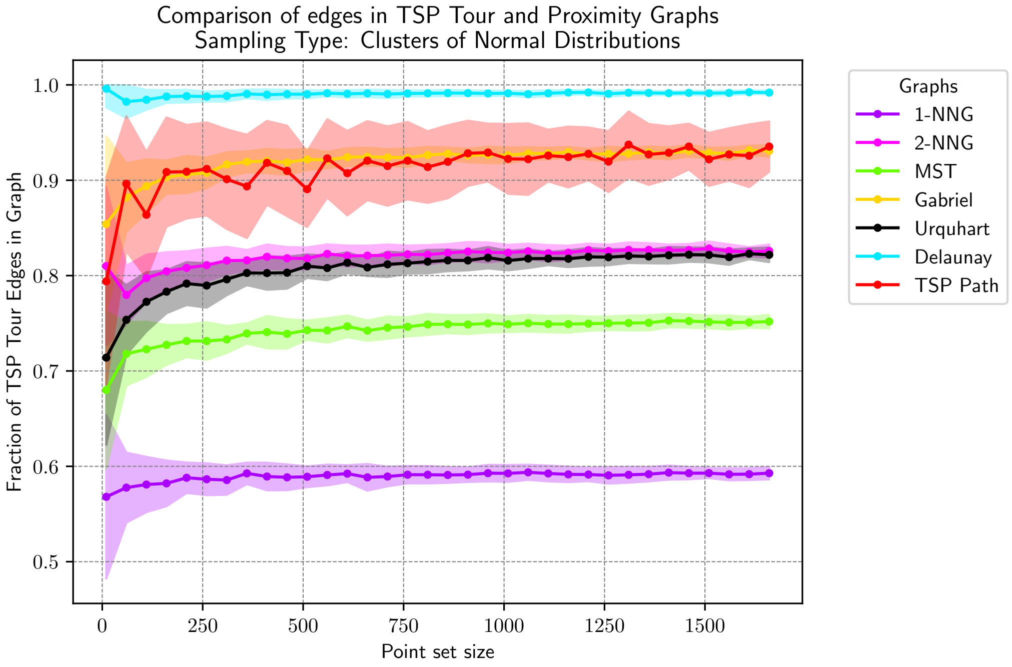


Figure 11 Comparison when points are sampled from a set of 10 $N(\sigma = 0.05)$ distributions, each one centered at one of 10 points selected uniformly from the unit square.

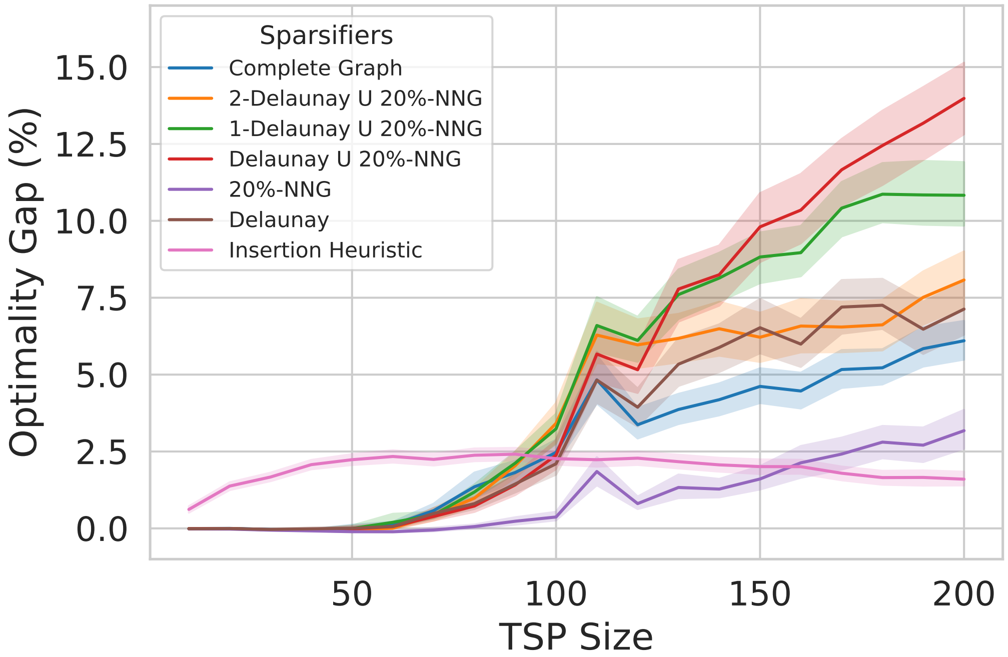


Figure 12 Optimality gaps for GNN approximators on spokes TSP instances.

The resemblance between Figures 10 and 11 is noteworthy, as the asymptotic intersection of the TSP tour with proximity graphs seems to be the same despite the difference in sampling method.

5 Applications to Geometric Deep Learning TSP Approximators

Recent work in the geometric deep learning community has emphasized *end-to-end* graph neural network (GNN) approximators for the Euclidean TSP [2, 4, 9, 10, 11, 14, 15, 19, 20, 25]. In this setting, the objective is not to outperform TSP solvers (e.g., Concorde), but to use the TSP as a difficult benchmark with which to evaluate competing GNN architectures [8]. While state-of-the-art GNN approximators for the TSP achieve an optimality gap of $< 1\%$ for instances smaller than 60 points, they struggle with larger instances [9, 18, 19, 20].

Researchers have primarily considered TSP instances whose point clouds are sampled uniformly over $[0, 1]^2$. Using our experimental pipeline, we investigate GNN performance on point sets sampled from a richer set of distributions, endowing the TSP benchmark problem with a healthier diversity of geometric structure. We observe that current state-of-the-art GNN approximators perform well on Bentley's *spokes* point clouds, maintaining an optimality gap of less than 1% on instances with up to 100 points, as shown in Fig. 12. These GNNs, trained on TSP instances with between 20 and 50 points, were able to accurately generalize in a *zero-shot* fashion to larger spokes point sets. This level of effective generalization has not yet been observed [9, 18, 19, 20], and it suggests that GNNs can learn to identify and exploit the highly-structured geometric properties of spokes point sets.

As a pre-processing step, *sparsifying* the input yields improvements over un-sparsified input (a weighted, complete graph). As GNNs, trained on Uniform $[0, 1]^2$ point sets of sizes varying from 20 to 50, are forced to generalize in a zero-shot fashion to point sets of sizes

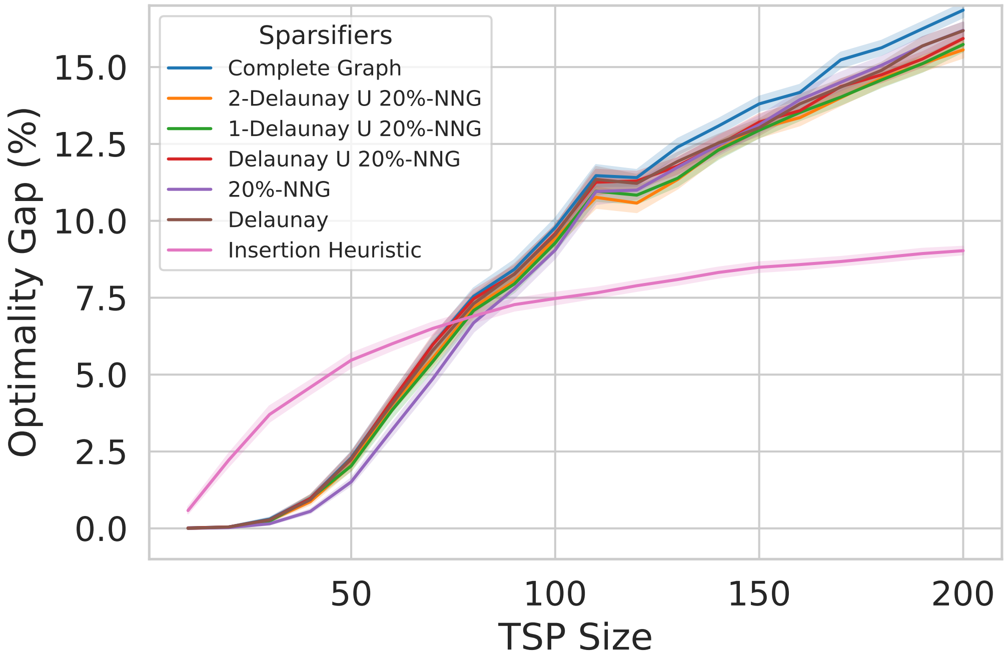


Figure 13 Optimality gaps for GNN approximators on Uniform $[0,1]^2$ TSP instances.

between 100 and 200, Delaunay-inspired sparsifiers indicate marginal improvements over state-of-the-art 20%-NNG sparsifiers [9]. For Uniform $[0,1]^2$ point sets of size 150 or more, the $\Theta(n)$-space Delaunay sparsifier performs equally as well as the $\Theta(n^2)$-space 20%-NNG sparsifier. For spokes point sets, the 20%-NNG sparsifier yields the strongest performance.

In future work, we will utilize the central theme of our empirical results – that various proximity graphs are closely related to the TSP – to inform more powerful architectures for GNN approximators. We will also further investigate the generalization capabilities afforded by Delaunay-inspired sparsifiers, leveraging our experimental pipeline.

References

1 David L Applegate, Robert E Bixby, Vašek Chvátal, William Cook, Daniel G Espinoza, Marcos Goycoolea, and Keld Helsgaun. Certification of an optimal TSP tour through 85,900 cities. *Operations Research Letters*, 37(1):11–15, 2009.

2 Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 2020.

3 Jon Louis Bentley. Fast algorithms for geometric Traveling Salesman Problems. *ORSA Journal on computing*, 4(4):387–411, 1992.

4 Xavier Bresson and Thomas Laurent. The transformer network for the Traveling Salesman Problem. *arXiv preprint arXiv:2103.03012*, 2021.

5 Mark de Berg, Hans L. Bodlaender, Sándor Kisfaludi-Bak, and Sudeshna Kolay. An ETH-tight exact algorithm for Euclidean TSP. In *IEEE 59th Annual Symposium on Foundations of Computer Science*, pages 450–461. IEEE, 2018.

6 Michael B Dillencourt. A non-Hamiltonian, nondegenerate Delaunay triangulation. *Information Processing Letters*, 25(3):149–151, 1987.

7 Michael B Dillencourt. Traveling salesman cycles are not always subgraphs of Delaunay triangulations or of minimum weight triangulations. *Information Processing Letters*, 24(5):339–342, 1987.

8 Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

9 Chaitanya K. Joshi, Quentin Cappart, Louis-Martin Rousseau, and Thomas Laurent. Learning TSP requires rethinking generalization. In *27th International Conference on Principles and Practice of Constraint Programming*, 2021.

10 Chaitanya K. Joshi, Thomas Laurent, and Xavier Bresson. An efficient graph convolutional network technique for the Travelling Salesman Problem. *arXiv preprint arXiv:1906.01227*, 2019.

11 Chaitanya K. Joshi, Thomas Laurent, and Xavier Bresson. On learning paradigms for the Travelling Salesman Problem. *arXiv preprint arXiv:1910.07210*, 2019.

12 Tomáš Kaiser, Maria Saumell, and Nico Van Cleemput. 10-Gabriel graphs are Hamiltonian. *Information Processing Letters*, 115(11):877–881, 2015.

13 Viggo Kann. *On the approximability of NP-complete optimization problems.* PhD thesis, Royal Institute of Technology Stockholm, 1992.

14 Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *Advances in Neural Information Processing Systems*, 30:6348–6358, 2017.

15 Wouter Kool, Herke van Hoof, and Max Welling. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2018.

16 Natalio Krasnogor, Pablo Moscato, and Michael G Norman. A new hybrid heuristic for large geometric Traveling Salesman Problems based on the Delaunay triangulation. In *Anales del XXVII Simposio Brasileiro de Pesquisa Operacional*, pages 6–8. Citeseer, 1995.

17 Adam N Letchford and Nicholas A Pearson. Good triangulations yield good tours. *Computers & operations research*, 35(2):638–647, 2008.

18 Qiang Ma, Suwen Ge, Danyang He, Darshan Thaker, and Iddo Drori. Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. *arXiv preprint arXiv:1911.04936*, 2019.

19 Wenbin Ouyang, Yisen Wang, Shaochen Han, Zhejian Jin, and Paul Weng. Improving generalization of deep reinforcement learning-based TSP solvers. *arXiv preprint arXiv:2110.02843*, 2021.

20 Wenbin Ouyang, Yisen Wang, Paul Weng, and Shaochen Han. Generalization in deep RL for TSP problems via equivariance and local search. *arXiv preprint arXiv:2110.03595*, 2021.

21 Gerhard Reinelt. TSPLIB-A Traveling Salesman Problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

22 Gerhard Reinelt. Fast heuristics for large geometric Traveling Salesman Problems. *ORSA Journal on computing*, 4(2):206–217, 1992.

23 Warren D. Smith and Nicholas C. Wormald. Geometric separator theorems and applications. In *IEEE 39th Annual Symposium on Foundations of Computer Science*, pages 232–243. IEEE, 1998.

24 WR Stewart. Euclidean Traveling Salesman Problems and Voronoi diagrams. *School of Business Administration, College of William and Mary*, 1997.

25 Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. *Advances in Neural Information Processing Systems*, 28:2692–2700, 2015.

SPONSORS

iTS UMBRIA SMART ACADEMY

slope®

ORDINE DEGLI INGEGNERI PROVINCIA DI PERUGIA

BAZUEL.com

algorithms